



An Algorithmic Framework for Making Use of Negative Learning in Ant Colony Optimization

A dissertation presented in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science

by

TEDDY NURCAHYADI

Supervisor:

DR. CHRISTIAN BLUM

Tutor:

DR. FELIP MANYÀ



**Universitat Autònoma
de Barcelona**

**Institut d'Investigació en Intel·ligència Artificial, CSIC
Programa de Doctorat en Informàtica
Universitat Autònoma de Barcelona**

October, 2022

ABSTRACT

Ant Colony Optimization (Aco) is a metaheuristic that took its inspiration from nature, particularly from the foraging behavior of ant colonies. This algorithm imitates the ants' capability in finding short routes between their nest and food sources by using two mechanisms: *solution construction* and *pheromone update*. Similar to the real ants that choose their route based on its *pheromone concentration*, the Aco algorithm makes use of *pheromone values*, in addition to *greedy information*, to construct several solutions at each iteration. Subsequently, the algorithm uses some of the good ones of the constructed solutions to update its *pheromone model* as a way to emulate the dynamics of the pheromone trails during the ants' foraging activities. This pheromone model will bias the search towards areas in the proximity of the solutions used for updating. Hence, we can say that this optimization algorithm uses a *positive learning* mechanism, as most metaheuristics do.

Observations showed that *negative learning*, in addition to the positive learning mechanism, also plays a vital role in animal swarm behavior, the species and inter-species evolution, as well as the human history. Metaheuristic methods such as evolutionary algorithms, extremal optimization, particle swarm optimization, and the opposition-based learning algorithm have already exploited this feature. Several works also tried to improve existing Aco algorithms by integrating negative learning mechanisms. However, most of these implementations did not produce any consistent improvement over the standard Aco algorithm.

Our negative learning Aco proposal employs several features not considered in previous implementations. Unlike the previous approaches that identify the negative learning information by a function that searches for worse solutions or simply by choosing the worst solution in an iteration for updating the pheromone values, our negative learning Aco identifies bad solutions (or bad solution components) by comparing the solutions produced in an Aco iteration to the solution generated by an additional algorithmic component. Any solution component found in an Aco iteration but not present in the solution generated by the additional algorithmic component is considered a bad solution component. As a consequence, its *negative pheromone* value is increased. Consequently, this component will be less likely to be selected in the next iterations. We developed another novel feature to support this negative information identification mechanism. The *baseline* Aco algorithm generates a *sub-instance* that consists of solution components found in each of its iterations. Subsequently, the baseline algorithm feeds the sub-instance to the additional

algorithmic component. Hence, the algorithmic component works on a problem instance that is smaller in size and consists of presumably high-quality solution components.

We successfully used several optimization algorithms (C_{PLEX}, *MAX-MIN* Ant System, and the MaxSAT solvers *SATLike* and *SLSMcs*) as options for the additional algorithmic component in our negative learning Aco applications to several optimization problems (Multi Dimensional Knapsack Problem, Minimum Dominating Set problem, Maximum Satisfiability problem, Capacitated Minimum Dominating Set problem, and Minimum Positive Influence Dominating Set problem). The results show that our negative learning Aco variants improve consistently over the baseline Aco algorithm and over the other optimization algorithms used internally in our negative learning Aco variants. Clearly, this is an achievement that no other negative learning Aco proposal obtained in their implementations. We also compared them to four representative negative learning Aco proposals from the literature in our application to the Multi Dimensional Knapsack Problem and Minimum Dominating Set problem. The comparison shows that, globally, our negative learning Aco variants perform significantly better than these competitors. Moreover, we found that $\text{Aco}_{\text{neg}}^+$ —our overall-best negative learning Aco variant—always performs competitively with the state-of-the-art algorithms in each considered optimization problem. In fact, $\text{Aco}_{\text{neg}}^+$ is currently the state-of-the-art algorithm for the Minimum Positive Influence Dominating Set problem. This way, we proved the general applicability and the effectiveness of our negative learning Aco proposal.

Keywords: *Ant Colony Optimization, Negative Learning, Multi Dimensional Knapsack Problem, Minimum Dominating Set, Maximum Satisfiability, Capacitated Minimum Dominating Set, Minimum Positive Influence Dominating Set, C_{PLEX}, MMAS, SATLike, SLSMcs.*

RESUMEN

La *Optimización con colonias de hormigas* (Aco) es una metaheurística que se inspira en la naturaleza; más concretamente, en el comportamiento que siguen las hormigas cuando buscan alimentos. Este algoritmo hace uso de la capacidad que tienen las hormigas para encontrar rutas cortas, entre el hormiguero y la fuente de alimentos, utilizando dos mecanismos: la *construcción de soluciones* y la *actualización de feromonas*. Al igual que las hormigas eligen la ruta en función de la *concentración de feromonas*, el algoritmo Aco utiliza *valores de feromonas*, además de *información greedy*, para construir varias soluciones en cada iteración. Posteriormente, el algoritmo usa algunas de las mejores soluciones construidas para actualizar su *modelo de feromonas* como una forma de emular la dinámica de los rastros de feromonas de las hormigas durante la búsqueda de alimentos. Este modelo de feromonas conduce la búsqueda hacia áreas cercanas a la solución utilizada para la actualización. Por lo tanto, podemos decir que este algoritmo de optimización utiliza un mecanismo de *aprendizaje positivo*, como lo hacen la mayoría de metaheurísticas.

Se sabe por experiencia que el *aprendizaje negativo*, junto con el mecanismo de aprendizaje positivo, juega un papel vital en el comportamiento de los enjambres de animales y en la evolución dentro de las especies y entre especies, así como en la historia humana. Los métodos metaheurísticos como los algoritmos evolutivos, el método *extremal optimization*, la *optimización con enjambres de partículas* y el algoritmo de aprendizaje basado en oposición han explotado el aprendizaje negativo. Además, varios trabajos han intentado mejorar los algoritmos Aco existentes integrando mecanismos de aprendizaje negativo. Sin embargo, la mayoría de estas implementaciones no han producido ninguna mejora significativa sobre el algoritmo Aco estándar.

Nuestra propuesta de aprendizaje negativo para Aco emplea varias características no contempladas en implementaciones anteriores. A diferencia de los enfoques anteriores que identifican la información de aprendizaje negativo mediante una función que busca las peores soluciones o simplemente elige la peor solución en una iteración para actualizar los valores de las feromonas, nuestro aprendizaje negativo Aco identifica soluciones malas (o componentes de una solución mala) comparando las soluciones generadas en una iteración Aco con la solución generada por un componente algorítmico adicional. Cualquier componente de una solución que se encuentre en una iteración de Aco pero que no esté presente en la solución generada por el componente algorítmico adicional se considera un componente de solución de baja calidad y su valor

de *feromona negativo* se incrementa. Por tanto, es menos probable que este componente se seleccione en las iteraciones sucesivas. También desarrollamos otra característica novedosa para ayudar a este mecanismo de identificación de información negativa. El algoritmo *baseline* Aco genera una *subinstancia* que consta de los componentes comunes encontrados en todas las iteraciones. Posteriormente, el algoritmo de base pasa esta subinstancia al componente algorítmico adicional. Por lo tanto, el componente algorítmico considera una instancia más pequeña formada por componentes que, presumiblemente, son de alta calidad.

En esta tesis, utilizamos con éxito varios algoritmos de optimización (Cplex, MAX-MIN Ant System, SATLike y SLSMcs) como opciones para el componente algorítmico adicional en nuestras implementaciones Aco con aprendizaje negativo y resolvemos varios problemas de optimización (el problema de la mochila multidimensional, el problema del conjunto dominante mínimo, el problema de la máxima satisfactibilidad, el problema del conjunto dominante mínimo con capacidades y el problema del conjunto dominante mínimo con influencia positiva). Los resultados experimentales muestran que nuestras variantes Aco con aprendizaje negativo son mejores que el algoritmo Aco de base y los otros algoritmos de optimización utilizados internamente en nuestras variantes Aco con aprendizaje negativo. Claramente, este es un logro que ninguna otra implementación de aprendizaje negativo Aco ha obtenido anteriormente. Por otro lado, comparamos nuestras variantes con cuatro propuestas Aco con aprendizaje negativo, representativas de la literatura, para el problema de la mochila multidimensional y el problema del conjunto dominante mínimo. La comparación muestra que, globalmente, nuestras variantes Aco con aprendizaje negativo funcionan significativamente mejor que estos competidores. Además, encontramos que Aco_{neg}^+ —nuestra mejor variante Aco con aprendizaje negativo en general—siempre es competitiva con los mejores algoritmos para cada uno de los problemas de optimización considerados. De hecho, Aco_{neg}^+ es actualmente el algoritmo más competitivo para el problema del conjunto dominante mínimo con influencia positiva. De esta manera, hemos demostrado la aplicabilidad general y la efectividad de nuestra propuesta Aco con aprendizaje negativo.

RESUM

L'*optimització amb colònies de formigues* (Aco) és una metaheurística que s'inspira en la naturalesa; més concretament, en el comportament que segueixen les formigues quan busquen aliments. Aquest algorisme fa ús de la capacitat que tenen les formigues per a trobar rutes curtes, entre el formiguer i la font d'aliments, utilitzant dos mecanismes: la *construcció de solucions* i l'*actualització de feromones*. Igual que les formigues trien la ruta en funció de la *concentració de feromones*, l'algorisme Aco fa servir *valors de feromones*, a més d'*informació greedy*, per a construir diverses solucions en cada iteració. Posteriorment, l'algorisme fa servir algunes de les millors solucions construïdes per a actualitzar el seu *model de feromones* com una manera d'emular la dinàmica dels rastres de feromones de les formigues durant la cerca d'aliments. Aquest model de feromones condueix la cerca cap a àrees pròximes a la solució utilitzada per a l'actualització. Per tant, podem dir que aquest algorisme d'optimització utilitza un mecanisme d'*aprenentatge positiu*, com ho fan la majoria de metaheurístiques.

Se sap per experiència que l'*aprenentatge negatiu*, juntament amb el mecanisme d'aprenentatge positiu, juga un paper vital en el comportament dels eixams d'animals i en l'evolució dins de les espècies i entre espècies, així com en la història humana. Els mètodes metaheurístics com els algorismes evolutius, el mètode *extremal optimization*, l'*optimització amb eixams de partícules* i l'algorisme d'aprenentatge basat en oposició han explotat l'aprenentatge negatiu. A més, diversos treballs han intentat millorar els algorismes Aco existents integrant mecanismes d'aprenentatge negatiu. No obstant això, la majoria d'aquestes implementacions no han produït cap millora significativa sobre l'algorisme Aco estàndard.

La nostra proposta d'aprenentatge negatiu per a Aco empra diverses característiques no contemplades en implementacions anteriors. A diferència dels enfocaments anteriors que identifiquen la informació d'aprenentatge negatiu mitjançant una funció que busca les pitjors solucions o simplement tria la pitjor solució en una iteració per a actualitzar els valors de les feromones, el nostre aprenentatge negatiu Aco identifica solucions dolentes (o components d'una solució dolenta) comparant les solucions generades en una iteració Aco amb la solució generada per un component algorítmic addicional. Qualsevol component d'una solució que es trobi en una iteració de Aco però que no sigui present en la solució generada pel component algorítmic addicional es considera un component de solució de baixa qualitat i el seu valor de *feromona negatiu* s'incrementa. Per tant, és menys probable que aquest component se seleccioni

en les iteracions successives. També desenvolupem una altra característica nova per a ajudar a aquest mecanisme d'identificació d'informació negativa. L'algorisme *baseline* Aco genera una *subinstància* que consta dels components comuns trobats en totes les iteracions. Posteriorment, l'algorisme de base passa aquesta subinstància al component algorítmic addicional. Per tant, el component algorítmic considera una instància més petita formada per components que, presumiblement, són d'alta qualitat.

En aquesta tesi, utilitzem amb èxit diversos algorismes d'optimització (Cplex, *MAX-MIN* Ant System, *SATLike* i SlsMcs) com a opcions per al component algorítmic addicional en les nostres implementacions Aco amb aprenentatge negatiu i resollem diversos problemes d'optimització (el problema de la motxilla multidimensional, el problema del conjunt dominant mínim, el problema de la màxima satisfactibilitat, el problema del conjunt dominant mínim amb capacitats i el problema del conjunt dominant mínim amb influència positiva). Els resultats experimentals mostren que les nostres variants Aco amb aprenentatge negatiu són millors que l'algorisme Aco de base i els altres algorismes d'optimització utilitzats internament en les nostres variants Aco amb aprenentatge negatiu. Clarament, aquesta és una fita que cap altra implementació d'aprenentatge negatiu Aco ha assolit anteriorment. D'altra banda, comparem les nostres variants amb quatre propostes Aco amb aprenentatge negatiu, representatives de la literatura, per al problema de la motxilla multidimensional i el problema del conjunt dominant mínim. La comparació mostra que, globalment, les nostres variants Aco amb aprenentatge negatiu funcionen significativament millor que aquests competidors. A més, trobem que Aco_{neg}^+ —la nostra millor variant Aco amb aprenentatge negatiu en general—sempre és competitiva amb els millors algorismes per a cadascun dels problemes d'optimització considerats. De fet, Aco_{neg}^+ és actualment l'algorisme més competitiu per al problema del conjunt dominant mínim amb influència positiva. D'aquesta manera, hem demostrat l'aplicabilitat general i l'efectivitat de la nostra proposta Aco amb aprenentatge negatiu.

ACKNOWLEDGEMENT

I am glad that I finally got to this point in my Ph.D. study. It was an incredible journey for me. I admit that it was not easy, particularly at the start. I am deeply grateful to my Supervisor, Dr. Christian Blum, who guided me very well in this challenging endeavor. He understood that I had a different background and had a lot of things to learn in this new field. I am indebted to his precious time, efforts, and encouragement in guiding me through that very steep learning curve. He helped me to strategically plan my Ph.D. study, and he consistently paid a great deal of attention to my progress. He gave an excellent thesis supervision throughout my Ph.D. journey, and he was always concerned about its successful completion. Out of his kindness, he also helped me solve many problems that were not directly related to my academic work, but critical to me. All of this has made my studies here much more manageable. I am lucky to be one of his Ph.D. students. Thank you very, very much.

This endeavor would also not have been possible without the support of my Tutor, Dr. Felip Manyà. He helped me take care of many academic tasks and gave me a lot of motivation. I always received words of optimism and encouragement every time I meet him. I learned a lot from him during our collaboration on MaxSAT, and he gave numerous feedback that significantly improved my work. Thank you so much.

I would also like to express my deepest gratitude to my thesis committee members.

Special thanks to Albert López Serrano and Dr. Salim Bouamama for the collaboration in applying our negative learning Aco to the Minimum Positive Influence Dominating Set problem.

I am deeply grateful to Beasiswa Pendidikan Pascasarjana Luar Negeri (Indonesian Ph.D. scholarship) from the Directorate General of Higher Education, Research, and Technology of the Ministry of Education, Culture, Research, and Technology of the Republic of Indonesia. This Ph.D. study would not have been possible without their support.

Universitas Muhammadiyah Yogyakarta (UMY), my home university in Indonesia, provided me with many training and facilities for my doctoral studies. This has made my travel and my stay Catalonia feel effortless, which I sincerely appreciate; my deepest gratitude for the commitment and support of UMY to my Ph.D. studies.

I am grateful to the Spanish Ministry of Science and Innovation, which funded all research in this thesis through project MCIN/AEI/10.13039/501100011033

(Grants PID2019-104156GB-I00 and PID2019-111544GB-C21).

The Artificial Intelligence Research Institute (IIIA-CSIC) provided an outstanding working environment during my Ph.D. studies. I am genuinely grateful for the access provided to the High-Performance Computer Cluster Ars Magna, which I used intensively for experiments in this thesis. I witnessed people's dedication, passion, and humbleness reflected in their daily activities and work in this Institute. Thanks to them for providing me with a friendly and encouraging atmosphere.

I enjoyed working in the student rooms on the first and second floors of the Institute. It was the best working environment that I ever had. Many thanks to all fellow students in these rooms.

Finally, I would like to thank my wife, Elis, and our three children, Azizah, Adly, and Adlyna. It was not always easy for us to go through this great adventure, but I am glad that we were together to experience this fantastic journey. Thanks for all your love and support.

CONTENTS

ABSTRACT	i
ACKNOWLEDGEMENT	vii
LIST OF FIGURES	xiv
LIST OF TABLES	xvii
LIST OF ALGORITHMS	xxi
1 INTRODUCTION	1
1.1 Methods for Combinatorial Optimization	2
1.2 Metaheuristics	4
1.2.1 Greedy Randomized Adaptive Search Procedure	4
1.2.2 Iterated Greedy Algorithms	5
1.2.3 Iterated Local Search	5
1.2.4 Simulated Annealing	5
1.2.5 Tabu Search	5
1.2.6 Variable Neighborhood Search	6
1.2.7 Evolutionary Algorithms	6
1.2.8 Particle Swarm Optimization	7
1.2.9 Extremal Optimization	7
1.2.10 Opposition Based Learning	8
1.2.11 Ant Colony Optimization	8
1.3 Negative Learning in Metaheuristics	12
1.4 Negative Learning in Ant Colony Optimization	13
1.5 General Ideas	18
1.5.1 The Storage and Update of the Negative Learning Information	19
1.5.2 The Way of Deriving the Negative Learning Information	20
1.6 Lessons Learned and Resulting Proposal	21

1.7	Thesis Contributions	23
1.8	The Organization of This Thesis	25
1.9	Publications Derived from this Thesis	28
2	GENERAL DESCRIPTION OF THE ALGORITHMIC FRAMEWORK	31
2.1	Introduction	31
2.2	<i>MMAS</i> : The Baseline Algorithm	31
2.2.1	Solution Construction	33
2.2.2	Pheromone Update	33
2.2.3	Convergence Factor	34
2.3	Adding Negative Learning to <i>MMAS</i>	35
2.3.1	Information Maintenance	35
2.3.2	Information Generation and Update	35
2.3.3	Information Use	36
3	APPLICATION TO THE MULTI DIMENSIONAL KNAPSACK PROBLEM	39
3.1	Introduction	39
3.2	The Multi Dimensional Knapsack Problems	39
3.2.1	ILP Model for the MDKP	40
3.3	<i>MMAS</i> Implementation to the MDKP	40
3.3.1	Solution Construction	41
3.3.2	Pheromone Update and Convergence Factor	41
3.4	Adding Negative Learning to <i>MMAS</i>	42
3.5	Proposals from the Literature	44
3.5.1	Subtractive Anti-Pheromone (SAP)	45
3.5.2	Explorer Ants (EA)	46
3.5.3	Preferential Anti-Pheromone	46
3.5.4	Second-Order Swarm Intelligence	47
3.6	Summary of the Tested Algorithms	47
3.7	Experimental Evaluation	49

3.7.1	Algorithm Tuning	49
3.7.2	Results	50
3.7.3	Search Trajectory Network Analysis	55
3.7.4	Comparison to the State-of-the-Art	59
3.8	Conclusions	61
4	APPLICATION TO THE MINIMUM DOMINATING SET PROBLEM	75
4.1	Introduction	75
4.2	The Minimum Dominating Set Problem	75
4.2.1	ILP Model for the MDS Problem	76
4.3	<i>MMAS</i> Implementation to the MDS	76
4.3.1	Solution Construction	77
4.3.2	Pheromone Update and Convergence Factor	78
4.4	Adding Negative Learning to <i>MMAS</i>	79
4.5	Proposals from the Literature	81
4.6	Summary of the Tested Algorithms	83
4.7	Experimental Evaluation	84
4.7.1	Algorithm Tuning	84
4.7.2	Results	85
4.7.3	Search Trajectory Network Analysis	89
4.7.4	Comparison to the State of the Art	98
4.8	Conclusions	99
5	APPLICATION TO THE MAXIMUM SATISFIABILITY PROBLEM	119
5.1	Introduction	119
5.2	The Maximum Satisfiability Problem	120
5.2.1	ILP Model for the MaxSAT	120
5.2.2	Existing Approaches to MaxSAT	121
5.3	Negative Learning Aco for MaxSAT	123
5.3.1	Solution Construction	125
5.3.2	Pheromone Update and Convergence Factor	126

5.4	Experimental Evaluation	128
5.4.1	Problem instances	128
5.4.2	Algorithm tuning and test settings	129
5.4.3	Results	130
5.4.4	Search Trajectory Network Analysis	134
5.5	Conclusions	138
6	APPLICATION TO THE MINIMUM CAPACITATED DOMINATING SET PROBLEM	151
6.1	Introduction	151
6.2	The CapMDS Problem	151
6.2.1	ILP Model for the CAPMDS Problem	152
6.3	<i>MMAS</i> Implementation to the CapMDS	153
6.4	Adding Negative Learning to <i>MMAS</i>	155
6.5	Experimental Evaluation	157
6.5.1	Algorithm Tuning	157
6.5.2	Numerical Results	158
6.6	Conclusions	160
7	APPLICATION TO THE MINIMUM POSITIVE INFLUENCE DOMINATING SET PROBLEM	161
7.1	Introduction	161
7.2	The Minimum Positive Influence Dominating Set Problem	162
7.2.1	ILP Model for the MPIDS	162
7.3	Negative Learning Aco for MPIDS Problem	162
7.4	Experimental Evaluation	164
7.4.1	Problem instances	164
7.4.2	Algorithm tuning and test settings	164
7.4.3	Results	165
7.5	Discussion and Conclusions	165

8	ADDITIONAL WORK:	
	<i>MMAS</i> APPLICATION TO THE MULTI-HEAD WEIGHER	
	MACHINES PROBLEM	169
8.1	Introduction	169
8.2	The Multi-head Weigher Machine Problem	171
	8.2.1 ILP Model for the MWM	171
	8.2.2 Existing Approaches to the MWM problem	172
8.3	<i>MMAS</i> for MWM	172
8.4	Experimental Evaluation	173
	8.4.1 Problem instances	174
	8.4.2 Algorithm tuning and test settings	174
	8.4.3 Results	175
8.5	Conclusions	180
9	CONCLUSIONS AND OUTLOOK	181
9.1	Conclusions	181
9.2	Outlook	186
	REFERENCES	188

List of Figures

1.1	The basic concept of Aco	2
1.2	The existing strategies for storing and updating the negative learning information	19
1.3	Passive method for deriving negative learning information	20
1.4	Active method for deriving negative learning information	21
1.5	Our method for deriving negative learning information	22
3.1	Illustrative example of negative learning Aco applied to the MDKP	43
3.2	Critical difference plot concerning all MDKP instances	50
3.3	Critical difference plots for MDKP instance groups based on their densities	52
3.4	Critical difference plots for MDKP instance groups based on their number of resources	53
3.5	Heatmaps concerning the results for the MDKP	54
3.6	Search trajectory network of Aco, Aco _{neg} , and Aco ⁺ _{neg} applied to problem instance cb_5_500.0	55
3.7	Search trajectory network of Aco ⁺ , Aco _{neg} , and Aco ⁺ _{neg} applied to problem instance cb_5_500.0	57
3.8	Search trajectory network of Aco-Aco _{neg} , Aco-Aco ⁺ _{neg} , and Aco ⁺ _{neg} applied to problem instance cb_5_500.0	57
3.9	Search trajectory network of Aco-EA, Aco-SAP, and Aco ⁺ _{neg} applied to problem instance cb_5_500.0	58
3.10	Search trajectory network of Aco ^{2o} , Aco-PAP, and Aco ⁺ _{neg} applied to problem instance cb_5_500.0	58
4.1	Illustrative example of the negative learning mechanism for the MDS problem	79

4.2	Critical difference plot for all MDS instances	86
4.3	Critical difference plots concerning different graph types	87
4.4	Heatmaps concerning the results for the MDS problem	88
4.5	Search trajectory network with the trajectories of Aco , Aco_{neg} , and Aco_{neg}^+ applied to problem instance 5000.rgg.0.1.0	90
4.6	Search trajectory network with the trajectories of Aco^+ , Aco_{neg} , and Aco_{neg}^+ applied to problem instance 5000.rgg.0.1.0	90
4.7	Search trajectory network with the trajectories of $Aco-Aco_{neg}$, $Aco-Aco_{neg}^+$, and Aco_{neg}^+ applied to problem instance 5000.rgg.0.1.0	91
4.8	Search trajectory network with the trajectories of $Aco-EA$, $Aco-SAP$, and Aco_{neg}^+ applied to problem instance 5000.rgg.0.1.0	92
4.9	Search trajectory network with the trajectories of Aco^{2o} , $Aco-PAP$, and Aco_{neg}^+ applied to problem instance 5000.rgg.0.1.0	93
4.10	Search trajectory network with the trajectories of Aco , Aco_{neg}^+ , and Aco_{neg} applied to problem instance 5000.rgg.1.0.1	94
4.11	Search trajectory network with the trajectories of Aco^+ , Aco_{neg}^+ , and Aco_{neg} applied to problem instance 5000.rgg.1.0.1	95
4.12	Search trajectory network with the trajectories of $Aco-Aco_{neg}$, $Aco-Aco_{neg}^+$, and Aco_{neg} applied to problem instance 5000.rgg.1.0.1	96
4.13	Search trajectory network with the trajectories of $Aco-EA$, $Aco-SAP$, and Aco_{neg} applied to problem instance 5000.rgg.1.0.1	97
4.14	Search trajectory network with the trajectories of Aco^{2o} , $Aco-PAP$, and Aco_{neg} applied to problem instance 5000.rgg.1.0.1	97
5.1	Illustrative example of the negative learning Aco approach for the MaxSAT problem	125
5.2	Critical difference plot concerning the results of the test on Villagra and Baran's instances	132
5.3	Critical difference plot concerning the results of the tests for the MSE 2020 and 2016 instances	134
5.4	Search trajectory networks concerning Aco , Aco_{neg} , and Aco_{neg}^+ applied to problem instance HG-4SAT-V150-C1350-100	135

5.5	Search trajectory networks concerning Aco_{neg}^+ , $Aco-Sls_{neg}^+$, and $Aco-Sat_{neg}^+$ applied to problem instance HG-4SAT-V150-C1350-100 .	136
5.6	Search trajectory networks concerning Aco , Aco_{neg} , and Aco_{neg}^+ applied to problem instance scplr12_maxsat	137
5.7	Search trajectory networks concerning Aco_{neg}^+ , $Aco-Sat_{neg}^+$, and $Aco-Sls_{neg}^+$ applied to problem instance scplr12_maxsat	137
6.1	Negative learning Aco for CapMDS	155
6.2	Critical difference plots	160
8.1	A schema of a single layer MWM	170
8.2	Illustrative example of problem instance and solution of configuration number 1 in Table 8.5	177
8.3	Illustrative example of problem instance and solution of configuration number 4 in Table 8.5	177

List of Tables

2.1	Values for weights κ_{ib} , κ_{rb} , and κ_{bsf} which depend on cf and bs_update	34
3.1	Summary of the parameters that arise in the considered algorithms, together with their description and the domains considered for parameter tuning.	48
3.2	Summary of the parameters that arise in each algorithm.	49
3.3	Parameter values for all algorithms for solving the MDKP	50
3.4	Summarized comparison between ACO_{neg}^+ and the state-of-the-art algorithms for the MDKP	60
3.5	Best results of all algorithms tested on OR-LIB instances with 5 resources	62
3.6	Best results of all algorithms tested on OR-LIB instances with 10 resources	63
3.7	Best results of all algorithms tested on OR-LIB instances with 30 resources	64
3.8	Average results of all algorithms tested on OR-LIB instances with 5 resources	65
3.9	Average results of all algorithms tested on OR-LIB instances with 10 resources	66
3.10	Average results of all algorithms tested on OR-LIB instances with 30 resources	67
3.11	Average computation time of all algorithms tested on OR-LIB instances with 5 resources	68
3.12	Average computation time of all algorithms tested on OR-LIB instances with 10 resources	69

3.13	Average computation time of all algorithms tested on OR-LIB instances with 30 resources	70
3.14	Comparison of ACO_{neg}^+ , ACO_{neg} , and ACO with the current state of the art on OR-LIB instances with 5 resources.	71
3.15	Comparison of ACO_{neg}^+ , ACO_{neg} , and ACO with the current state of the art on OR-LIB instances with 10 resources.	72
3.16	Comparison of ACO_{neg}^+ , ACO_{neg} , and ACO with the current state of the art on OR-LIB instances with 30 resources.	73
4.1	Features comparison between our negative learning ACO variants	84
4.2	Parameter values for all algorithms for solving the MDS problem	85
4.3	MDS problem: summarized comparison to the state of the art. Competitor names are accompanied by publication year and the reference.	98
4.4	Best results of all algorithms tested on MDS random graphs with 5000 vertices	100
4.5	Best results of all algorithms tested on MDS random geometric graphs with 5000 vertices	101
4.6	Best results of all algorithms tested on MDS random graphs with 10000 vertices	103
4.7	Best results of all algorithms tested on MDS random geometric graphs with 10000 vertices	104
4.8	Average results of all algorithms tested on MDS random graphs with 5000 vertices	106
4.9	Average results of all algorithms tested on MDS random geometric graphs with 5000 vertices	107
4.10	Average results of all algorithms tested on MDS random graphs with 10000 vertices	109
4.11	Average results of all algorithms tested on MDS random geometric graphs with 10000 vertices	110
4.12	Average computation time of all algorithms tested on MDS random graphs with 5000 vertices	112

4.13	Average computation time of all algorithms tested on MDS random geometric graphs with 5000 vertices	113
4.14	Average computation time of all algorithms tested on MDS random graphs with 10000 vertices	115
4.15	Average computation time of all algorithms tested on MDS random geometric graphs with 10000 vertices	116
5.1	Parameter values obtained for all Aco algorithms concerning the Pinto et al. instances	129
5.2	Parameter values obtained for all Aco algorithms concerning the Villagra and Baran instances	130
5.3	Parameter values obtained for $ACO-SAT_{neg}^+$, MSE 2016 and 2020 instances	131
5.4	Parameter values obtained for $ACO-SLS_{neg}^+$, MSE 2016 and 2020 instances	131
5.5	Parameter values obtained for ACO_{neg}^+ , MSE 2016 and 2020 instances	131
5.6	Parameter values obtained for ACO_{neg} , MSE 2016 and 2020 instances	131
5.7	Parameter values obtained for Aco, MSE 2016 and 2020 instances .	132
5.8	Comparative performance of our negative learning Aco variants with their individual algorithmic components	133
5.9	Average results for Pinto's instances	140
5.10	Summary of results for Villagra and Baran's instances	140
5.11	Average results of all algorithms tested on Villagra and Baran's instances	140
5.12	Best results of all algorithms tested on MSE 2020 and 2016 instances	142
5.13	Average results of all algorithms tested on MSE 2020 and 2016 instances	146
6.1	Parameter values obtained for solving CapMDS instances	158
6.2	Results for CapMDS graphs with uniform capacity.	159
6.3	Results for CapMDS graphs with variable capacity.	159
7.1	Tuning results obtained by irace for small/medium size networks	165
7.2	Tuning results obtained by irace for large size networks	165

7.3	Numerical results of all algorithm tested on MPIDS instances	167
8.1	MWM test case configurations with target a weight of 500 grams . .	174
8.2	MWM operational configurations	175
8.3	ACO parameter values obtained by irace	175
8.4	C _{PLEX} and ACO results and execution times for the MWM packing problem with one product type	176
8.5	C _{PLEX} and ACO results and execution times for the MWM packing problem with two product types	176
8.6	C _{PLEX} and ACO results and execution times for the MWM packing problem with three product types	179

List of Algorithms

2.1	MMAS in the hypercube framework (the baseline algorithm)	32
4.1	MDS solution construction	78
5.1	Negative Learning Aco for unweighted MaxSAT	124

CHAPTER 1

INTRODUCTION

Ant Colony Optimization (Aco) [1] is a *metaheuristics* optimization technique [2] that was inspired by the foraging behaviour of ant colonies in nature. This algorithm has widely been used for solving numerous *Combinatorial Optimization* (CO) problems such as scheduling [3], routing [4], transportation [5], and feature selection [6]. Aco, as shown in Fig. 1.1, consists of two main algorithmic components: *probabilistic solution construction* and *pheromone value update*. Based on the considered CO problem, the Aco algorithm first defines a finite set of solution components and a parameterized probabilistic model, which is called *pheromone model*. At each of its iterations, Aco constructs several solutions in a probabilistic way by making use of *greedy information* and *pheromone values*. Good solutions found at each iteration will be used to update the pheromone values. These two algorithmic components are executed so that new solutions with similar components to the ones from the previous best solutions will have a higher probability of being used in subsequent iterations. This way, Aco makes use of its best results in the past as the base for improving its results in the future. Therefore, it can also be said that the algorithm utilizes *positive learning*.

Since its first introduction by Dorigo et al. in 1991 [7, 8], Aco has evolved into several major variants [9–14] that gave significant improvements to the original variant by implementing a better utilization of the positive learning mechanism. Nature, however, provides numerous examples which show that *negative learning* mechanisms are also integral parts of the communication and coordination systems in various social insects [15–19]. There are also several examples of negative learning in species and inter-species evolution, animal swarm behavior, as well as in human history. These examples had already been adopted in several metaheuristic techniques such as *evolutionary algorithm* [20–22], *extremal optimization* [23–27], *particle swarm optimization* [28–32], and *the opposition based learning algorithm* [33, 34]. The Aco research community had also identified this potential and produced several relevant works. Most of them, however, had a limited success.

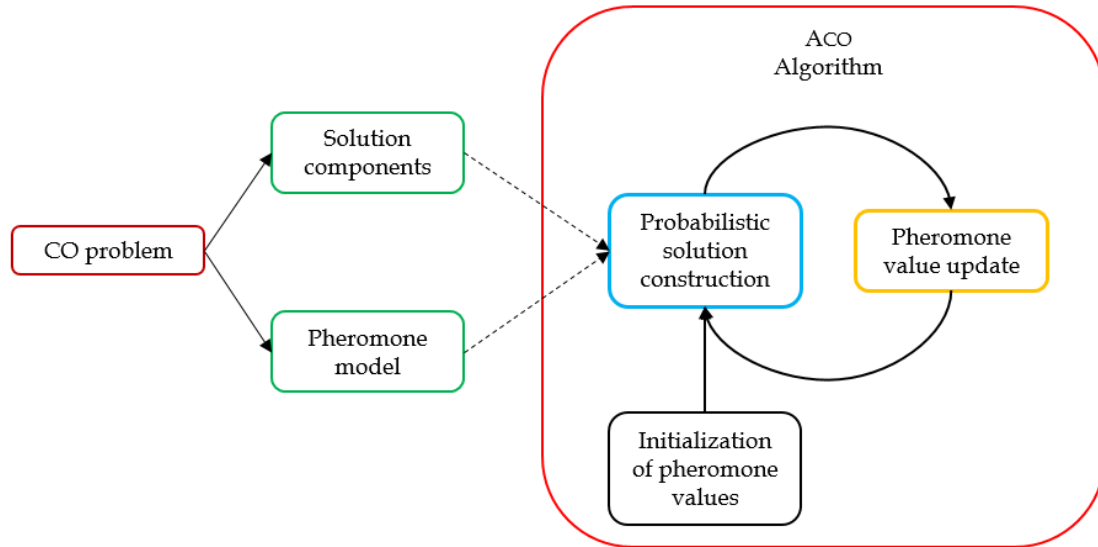


Fig. 1.1 The basic concept of Aco

This chapter reviews the classification of optimization methods and the essential characteristics of several metaheuristic methods, including the Aco algorithm. Subsequently, examples of the current implementations of negative learning in several metaheuristic techniques, including Aco, are provided. Based on this review, we describe our evaluation of the current negative learning implementations and propose our ideas for their improvements. Thereafter, we describe the application of our negative learning Aco to several CO problems.

1.1 METHODS FOR COMBINATORIAL OPTIMIZATION

A *Combinatorial Optimization* (CO) problem is defined by means of a finite set of objects and an objective function that assigns a non-negative cost value to each of the objects [35]. The goal is then to find an object with minimum (or maximum) cost value. Many important real-life problems—such as vehicle routing, scheduling, network design, and bioinformatics—can be represented in terms of a CO problem. Even a slightly improved solution to any of these problems can have a substantial impact. CO problems can be solved by using *exact* or *approximate* techniques. Exact techniques guarantee to find optimal solutions for every finite-size instance of a CO problem in bounded time. However, the time needed to obtain these optimal solutions is not guaranteed to stay within a reasonable computation time, especially for certain CO problems with complex structures and/or problem instances of large size. On the other hand, approximate methods sacrifice the guarantee of finding optimal solutions for

the sake of being able to produce high-quality solutions in practically acceptable computational time.

The group of exact optimization techniques consist of several methods. Some of the most well known ones are *Tree Search Methods* [36, 37]. They consider the search space of the tackled CO problem in form of a tree. The most simple tree search technique consists in the *exhaustive enumeration* of all valid solutions. This enumeration technique, however, is prone to excessive running times when dealing with relatively large search spaces. More clever tree search methods—such as, for example, *branch and bound* methods—divide the search space iteratively into distinct sub-spaces. Moreover, they employ *pruning* techniques for being able to discard complete sub-problems in which an optimal solution cannot appear. Employing similar mechanisms as Tree Search Methods, *Dynamic Programming* (DP) [38, 39] recursively partitions the search space of a CO problem into simpler sub-problems. In addition to that, DP uses *memoization* techniques [40, 41] for storing results of sub-problems in order to use them back whenever necessary. With this mechanism, DP can be quite efficient, especially when dealing with CO problems that have the so-called *overlapping sub-problems characteristic*.

Mixed Integer Linear Programming (MIP) [42] requires a combinatorial optimization problem to be modelled by means of a linear objective function and a set of linear constraints on continuous and/or integral variables. MIP techniques are very popular because coming up with such valid mathematical models of CO problems is frequently possible. Generic MIP Solvers (IBM ILOG CPLEX [43, 44], GUROBI [45], SCIP [46], or XPRESS [47]) are available today and they can be used to solve MIP models by using, for example, *branch and bound* methods combined with other advanced linear programming techniques. Similar to MIP, *Constraint Programming* (CP) is also employed to solve mathematical models of CO problems. CP techniques generally work in two phases. In the constraint propagation phase, the intention is to reduce the variable domains, that is, to reduce the search space. In the labelling phase, CP techniques search for valid solutions in the remaining search space. These techniques are usually applied to problems for which finding feasible solutions is already a difficult task on its own. The objective function in such problems is either not given or has only a secondary role. Several generic CP solvers (Geocode, Choco [48], and IBM CP Optimizer [49]) are available today.

The class of approximate optimization techniques consists of numerous methods. The most basic ones are *heuristics*, either *constructive heuristics* or *Local Search* (LS) methods. As the name suggests, constructive heuristics generate a solution from scratch by adding solution components until a valid solution is

obtained. The most well-known example of a constructive heuristic is a *greedy heuristic* which uses a greedy function—a measure for the goodness of solution components—to select a solution component from the available options at each step of the solution construction process. In contrast to constructive heuristics, LS methods [50] work by generating, at each step, a group of neighboring solutions of the incumbent solution based on a function called the *neighborhood function*. Then, at each step, a solution better than the incumbent solution is chosen for being the new incumbent solution. This is generally done until the incumbent solution does not have any improving neighbor. *Variable neighborhood descent* (VND) [51], an extended version of LS methods, makes a structured, deterministic use of several neighborhood functions at the same time.

1.2 METAHEURISTICS

Metaheuristics are approximate algorithms that combine constructive heuristics and/or LS methods with other ideas in a framework that aims at a better exploration of the search space for finding an optimal or near-optimal solution. As opposed to heuristics, the application of metaheuristics is not limited to a particular CO problem. Moreover, all types of metaheuristics incorporate mechanisms for escaping from local optima; thus, they can be considered the extensions of constructive heuristics and LS methods that aim to explore the search space in less limited ways. In the following sections, several examples of metaheuristic methods are presented.

1.2.1 Greedy Randomized Adaptive Search Procedure

Greedy Randomized Adaptive Search Procedure (GRASP) [52] is a metaheuristic method that integrates the use of greedy *randomized* constructive heuristics with the subsequent application of LS methods. At each iteration, a solution is constructed by using a greedy heuristic in a randomized way. In fact, instead of always selecting the best solution component, a so-called candidate list is determined and one of the solution components from the candidate list is chosen at random. Longer candidate lists lead to a more diversified solution construction. LS methods are used in the second phase of GRASP. Apart from using simple LS methods or VND, other metaheuristics such as *iterated local search* or *tabu search* can be used in this step.

1.2.2 Iterated Greedy Algorithms

Iterated Greedy Algorithms (IG) [53] work by generating new solutions based on partially destroyed solutions. Hence, the main feature of this algorithm is the exploitation of good parts of the incumbent solution as the starting point for obtaining better solutions. The selection of these good parts is made in a probabilistic way, potentially guided by some measure of the usefulness of solution components. There is an optional LS method that can be used after the generation of a new solution, but it was rarely implemented in the published literature due to the fact that the most desired characteristics of this algorithm lie in its high computational efficiency as a result of its simplicity.

1.2.3 Iterated Local Search

Iterated Local Search (ILS) [54] shares with IG algorithms that the general principle is quite simple. At each iteration, the algorithm constructs a solution by *perturbing* the incumbent solution randomly. This strategy aims at moving the search to a different basin of attraction while still remaining close to the location of the incumbent solution. The perturbation is done in a non-deterministic way and possibly depending on the search history. The strength of this perturbation is important to ILS since it will determine whether this algorithm can escape the basin of attraction of the incumbent solution.

1.2.4 Simulated Annealing

Simulated Annealing (SA) [55–57] borrowed its main idea from the physical annealing process of metal and glass in which heating and cooling rates hold a significant role in the final quality of the material's internal structure. SA is probably the oldest metaheuristic method and the first one that explicitly implemented a strategy for escaping from local minima. The mechanism—which is called *uphill move*—is carried out by allowing the selection of a solution that has an objective function value worse than the current one. A temperature parameter T_k is used to calculate the probability of accepting an uphill move during the execution of the algorithm. The value of T_k decreases as the algorithm's running time increases. This way, the probability of accepting a worse solution will also decrease during the run of the algorithm.

1.2.5 Tabu Search

Tabu Search (TS) [58–60] explicitly uses search history for escaping from local optima and for exploring the search space further. This algorithm stores attributes

of recently visited solutions in a short-term memory called *tabu list* (TL) in order to exclude them from the neighborhood of the current solution. This is done at each iteration until the tabu list reaches its maximum capacity. At this condition, the oldest attribute will be removed from the TL, and it will be available again for the search process. The maximum capacity of TL is called *tabu tenure*, and it has an important role not only in controlling the memory of the search process but also for the robustness of the TS algorithm. In fact, there is a variant of TS—which is called *Reactive TS* [61]—which is based on controlling the tabu tenure value in a dynamic way.

1.2.6 Variable Neighborhood Search

Variable Neighborhood Search (VNS) [51] is an algorithm that adds a probabilistic search component to the deterministic VND. This metaheuristic implements a swapping strategy between neighborhood functions for diversifying the search and escaping from local optima. By using an initial solution S and a set of neighborhood functions $\{N_1, \dots, N_{k_{max}}\}$, VNS works in three phases at each of its iterations: *shaking*, *local search*, and *move*. A solution S' from the k -th neighborhood of the current solution S is chosen randomly in the shaking phase. A local search phase is applied to S' resulting in an improved solution S'' . Subsequently, S'' is evaluated against the incumbent solution S in the move phase. If it is better than S , S'' is assigned as the incumbent solution, and the algorithm proceeds with the first neighborhood function. Otherwise, the algorithm will use the $k + 1$ -th neighborhood function in the shaking phase of the subsequent iteration. Here we can see that the special feature of the VNS algorithm rests in the implementation of the shaking phase. Note also that there is a *generalized VNS* version [62] that utilizes a VND with its own set of neighborhood functions as its advanced local search mechanism.

1.2.7 Evolutionary Algorithms

Inspired by natural evolution, basic *Evolutionary Algorithms* make use of three mechanisms: *selection*, *recombination*, and *mutation* [20]. The initial population consists of diverse candidate solutions which may be generated in a random way. At each iteration this metaheuristic selects a subset of solutions—called *parents*—from the current *population*. The selection process is based on a measure called *fitness* [21] which is generally related to the objective function, results of simulation or experiments, or some other quality measures. The parent population P is then subject to a randomized reproduction process that creates an

offspring population P' through recombination of the parent solutions. Mutation operators that cause small self-adaptive modifications are then applied to the offspring population, which will transform P' into P'' . Eventually, the population for the next iteration is selected from the old population P and the offspring population P'' . Evolutionary algorithms are commonly known to have good exploration capabilities, but they are sometimes said to have their weakness in fine-tuning solutions. Therefore EAs are often hybridized with LS to locally improve some or all of the generated offspring in a variant called *memetic algorithms* [63, 64].

1.2.8 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a metaheuristic that was inspired by internal interactions in a moving flock of birds or a fish school. This metaheuristic distributes and moves candidate solutions—called *particles*—around the search space for obtaining the best possible solution [65]. It was originally proposed for function optimization [66], but later it was also employed in discrete optimization; see, for example, in the works of Zhi et al. [67] and Shi et al. [68]. Particles in PSO move with the guidance of a simple mechanism influenced by their own best-known positions and the best-known position known by particles in their neighborhood. The discovery of a better position, i.e., a better solution, by any of the particles will be used as a guide for the next movement of the entire neighborhood of that particle. Several parameters hold a significant role in PSO performance. The optimum size of the population seems to be dependent on the dimension and difficulty level of the problem instance; however, the expected values are said to be in the range of twenty to fifty [69]. Early variants of PSO regulated the particles' response to the newly found best location by using *acceleration coefficients* and *maximum velocity*. Later, Shi and Eberhart [70] proposed the use of an *inertia weight* as a better way of controlling the movements of the particles. It is said that a higher inertia weight value is suitable at the beginning of the search since it simulates more dynamic particles movements in a less viscous medium; thus, the search characteristics are more exploratory. The opposite is true near the end of the search process; the lower inertia weight value makes the search characteristics to be more exploitative.

1.2.9 Extremal Optimization

Inspired by natural occurrences of optimized configurations, Boettcher et al. [23] proposed *Extremal Optimization* (EO) algorithms. The main idea of EO centers

around a concept called *Self-Organized Criticality* (SOC) [71, 72] and a simplified representation of an *ecosystem* which is called *Bak-Sneppen model* [73]. This algorithm treats a solution S to a CO problem as an ecosystem and each member of the solution $x_i \in X$ as a *species* [74]. A *local fitness value* λ_i —that represents the adaptation ability of the species—is associated with each x_i . The total cost of a solution $C(S)$ is an accumulation of the λ_i values, which are regarded as individual cost contributions. Typically, the value of λ_i depends on the state of x_i in relation with other variables that connect to it.

The standard EO algorithm [23, 24] starts by generating an initial solution S . At each iteration, λ_i of each x_i in S is evaluated. Next, a solution component x_j —with the worst λ value—is selected. A local search in the neighborhood of the initial solution $N(S)$ is implemented by selecting a random solution S' such that x_j in S' no longer has the worst λ value. If $C(S')$ is better than $C(S)$, the algorithm stores S' as its best solution S^{best} . Finally, S' is assigned as the current solution S in the next iteration. A later version of EO—called τ -EO—modified the procedure for selecting x_j [26, 27, 75]. In this version, x_j is not selected based on the worst λ value; instead, it is selected stochastically based on a probability distribution over the rank order.

1.2.10 Opposition Based Learning

Opposition Based Learning (OBL) was originally introduced by Tizhoosh [33] as an alternative to optimization algorithms inspired by natural phenomena. Tizhoosh regarded a social revolution as an effort towards establishing *opposite circumstances*. Bringing this analogy to the context of optimization algorithms, OBL proposes the *counter-balance* mechanism for confining the solutions search space. By using this concept, for every *estimate* there is a *counter-estimate*, for every *weight* there is a *counter-weight*, and for every *action* there is a *counter-action*. Tizhoosh integrated the OBL concept for the first time in extended versions of *Genetic Algorithms* (GA), *Reinforcement Learning* (RL), and *Artificial Neural Networks* (ANN). The results show that the use of OBL in these algorithms may result in advantages at the early stages of the search. However, maintaining the OBL mechanism throughout the rest of the search will no longer benefit the search.

1.2.11 Ant Colony Optimization

At each iteration Aco algorithms construct solutions, step by step, in a probabilistic way, making use of two types of information: (1) *greedy information* and (2) *pheromone values*. Pheromone utilization and the pheromone update

mechanism—two special features of Aco—have undergone several improvements since this algorithm’s first introduction. The original variant of Aco is called *Ant System* (AS) [7, 8] and was introduced for the first time in the context of the *Travelling Salesman Problem* (TSP). This variant has three sub-variants: (1) *ant-density*, (2) *ant-quantity*, and (3) *ant-cycle*. The pheromone update in all three sub-variants is performed by all ants employed in the execution of the algorithm. The difference between the three sub-variants lies in the timing of the update and in the calculation of the amount of pheromone to be deposited. In *ant-cycle*, the update is implemented after constructing a complete solution, while in *ant-density* and *ant-quantity*, the update is applied at every step of the solution construction process.

The original AS algorithm developed into several variants, the first of which is called *Elitist Ant System* (EAS) [9]. The main idea proposed by this Aco variant is about giving preference to the ants that found the best solutions—which are called *elite ants*—to deposit more pheromone than any other ant. This idea was extended by Bullnheimer et al. [10, 11] who proposed *Rank-based Ant System* (AS_{rank}) which allows not only the elite ants to deposit pheromone, but also to a range of other ants that found good solutions. Another important variant called *Ant Colony System* (ACS) [12] works by combining the *local update* done by all ants in every step of the solution construction with the *global update* done only by the best ant after the completion of an iteration.

Stützle and Hoos [13] proposed *MAX-MIN Ant System* (MMAS) to improve the performance of previous ACO algorithm versions which they considered as not competitive when compared to the state-of-the-art algorithms developed explicitly for specific CO problems. The source of this weakness was said to be the lack of exploitation of the search history for directing the search process. MMAS algorithms are characterized by a stronger exploitation of the search history by giving privilege strictly to the best ants to do the pheromone update. Stützle and Hoos indicated that the two ACO versions ACS and AS_{rank} are able to perform better than the original AS because they carry out more intense exploitation of the best-found solutions. Furthermore, they conducted a *search space analysis* [76, 77] on several sample instances of the TSP and the *Quadratic Assignment Problem* (QAP) to show the characteristics of the search landscapes and their correlation with search strategies. The results of the search space analysis suggested that the TSP sample instances have a stronger relationship between the local optima and the global optima; hence, in the context of this CO problem more benefit is obtained from a stronger exploitation of the search history. In the case of the QAP, however, the locations of local optima are more scattered; thus, a stronger search

space exploration would be more beneficial for the search process for this CO problem. Based on those results, Stützle and Hoos asserted that a mechanism for avoiding premature stagnation should be provided to counterbalance a stronger exploitation of the search history.

In the quest for a well-working balance between a stronger exploitation and search stagnation avoidance, Stützle and Hoos proposed the following scheme for selecting between the *iteration-best solution* S^{ib} and the *best-so-far solution* S^{gb} for the pheromone update during the execution of the algorithm. As a default, S^{ib} is used for the pheromone update, being substituted by the use of S^{gb} only at every fixed number of iterations. Additionally, Stützle and Hoos also proposed a dynamic strategy in which the frequency of using S^{gb} for updating the pheromone trails increases as the number of iterations grows. This dynamic strategy, however, is still depending on a static arrangement for determining the update frequencies.

As another source for search stagnation Stützle and Hoos identified situations in which the amount of pheromone on a solution component becomes too low or too high. For this reason, $MMAS$ uses an upper limit τ_{max} and a lower limit $\tau_{min} > 0$ for all pheromone values. However, Stützle and Hoos argued that this measure is not enough. A further search diversification can be achieved through a pheromone value re-initialization. Implementing this strategy, a new variant called $MMAS+ri$ was invented. The pheromone re-initialization in this variant is carried out whenever the search reaches a condition called *convergence* i.e., when pheromone values on solution components other than those in the best-so-far solution S^{gb} are very close to the value of τ_{min} . The search history exploitation is then counterbalancing the search diversification mechanism by using S^{gb} to perform a dynamic pheromone update. Arguing that performing a dynamic pheromone update by using the *restart-best solution* S^{rb} followed later by S^{gb} would provide a more intense search space exploration, Stützle and Hoos devised another variant called $MMAS+rs$. Their experimental results showed that this last variant has a better performance than the standard $MMAS$ and the $MMAS+ri$ variant.

Blum and Dorigo [14] improved the performance and robustness of $MMAS$ by limiting the pheromone values to the interval $[0,1]$ and by employing a more sophisticated pheromone update mechanism. This development was motivated by the fact that the standard ACO algorithms are unable to deliver consistent performance when dealing with *isomorphic* problems that differ only in the scale of the objective function values. This drawback existed due to the ACO algorithms' difficulty in scaling the influences of solutions used for pheromone update at each stage of an algorithm run. The timing at which a certain solution is used

to update the pheromone values holds an important role in determining the correct implementation of exploration or exploitation during the search process. Assigning S^{gb} too early for updating the pheromone values, for example, could make the search being trapped in a local optimum. On the contrary, giving S^{ib} too much time for updating the pheromone values will cause the search to take too long to reach convergence. The standard \mathcal{MMAS} determines the pheromone update schedule based on merely static ranges of iteration numbers; hence, this mechanism will not be able to provide precise update assignments on accurate timings since different sizes of instances will be less likely to cause the algorithm to behave in the same way along the iterations.

Addressing this problem, Blum and Dorigo proposed a variant called \mathcal{MMAS} in the *Hyper Cube Framework* (\mathcal{MMAS}_{HCF}), which established an advanced scheduling mechanism that assigns the corresponding type of solutions for updating the pheromone values more accurately to the corresponding stages of the search. They introduced a parameter called *convergence factor* cf , where $0 \leq cf \leq 1$, as a way to categorize the different stages of the search process, rather than just an iteration count as in the standard \mathcal{MMAS} . In this algorithmic framework, the pheromones are initialized to 0.5. As the search progresses, the pheromones values will continue to change to either higher or lower values until each of them has either the highest value τ_{max} or the lowest value τ_{min} , i.e., when the search is said to have reached *convergence*. As an example, a schedule can be established in \mathcal{MMAS}_{HCF} as follows. When the stage of the search is in a condition in which the cf value is below 0.4, \mathcal{MMAS}_{HCF} will assign the iteration-best solution S^{ib} to do the pheromone update. Unlike the standard \mathcal{MMAS} that allows only one type of solution to do the update at each search stage, \mathcal{MMAS}_{HCF} allows both S^{ib} and S^{rb} to do the update simultaneously. For example, when $0.4 \leq cf < 0.6$ the influence of S^{ib} on the pheromone update might be at $2/3$, while the influence of S^{rb} is at $1/3$. With an increasing convergence factor value ($0.6 \leq cf < 0.8$) this influence may shift in the following way. Now, S^{ib} might only have $1/3$ of the influence, while S^{rb} now has $2/3$ of the influence. Finally, in the range of $0.8 \leq cf < 0.999$, S^{rb} is given the total influence to update the pheromone values. When the value of cf exceeds 0.999, its value is re-initialized to 0, and the full weight of influence is given to S^{gb} for updating the pheromone trails until the value of cf exceeds 0.999 again. This is done in order to try to find an even better solution in the search space between S^{rb} and S^{gb} . Afterwards, the search is restarted again by re-initializing all the pheromones values and solution S^{rb} .

1.3 NEGATIVE LEARNING IN METAHEURISTICS

Negative learning mechanisms have been employed to work together with positive learning in several variants of metaheuristic algorithms. In evolutionary algorithms [20–22], for example, positive learning is used by assigning a higher probability to individuals with a higher fitness to be selected as parent individuals allowed to do reproduction. The negative learning mechanism supports the positive learning mechanism in the next phase of the evolutionary algorithms selection process by not allowing individuals to do reproduction if they have a fitness lower than a certain limit. This way, negative learning mechanisms in evolutionary algorithms serve as instruments for *avoiding* certain regions of the search space that lack promising candidate solutions. In contrast to evolutionary algorithms that avoid search space in the proximity of bad solutions, simulated annealing [55–57] intentionally searches for good solutions that may be present in the proximity of a bad solution. Simulated annealing implements this by using low-quality solutions as the base for generating new neighborhoods of solutions. Thus, both algorithms use negative learning as a mechanism to escape from *local minima* but with different assumptions. The evolutionary algorithm considers that the search space in the proximity of a low-quality solution should be avoided altogether. On the other hand, simulated annealing assumes the search process has to pass through regions of low-quality solutions in order to finally reach areas of the search space with high-quality solutions.

Cooperation between positive and negative learning mechanisms can also be observed in several variants of particle swarm optimization algorithms. This algorithm implements positive learning by using the particles' *local-best* or *global-best* positions to determine the search direction [69]. Negative learning is added to the PSO variants from the works of Cooren et al. [30–32] by dividing the swarm into several sub-populations called *tribes*. Individual particles in these tribes are evaluated periodically to determine whether they are *good* or *bad*, based on the improvement of their best positions. The tribes with a larger number of good particles are considered the better tribes than those with fewer good particles. The negative learning mechanism is then implemented by removing the worst particle in the best tribe with the intention to reduce the number of evaluations of the objective function. On the other hand, the worst tribe of the swarm is then given the privilege to generate particles that will form a new tribe which will establish a connection that is expected to improve the original tribe's position in the following search process. Hence, the negative learning in these PSO variants serves the same purpose as the one in the SA algorithm, that is, for

escaping local minima and diversifying new solutions.

As in SA algorithms, negative learning mechanisms are also used for improving the exploration capability in extremal optimization algorithms. In these algorithms, the species—i.e. the component of a solution—that has the worst local fitness value in an ecosystem—i.e. the solution—is punished by forcing it to be the base for generating a new ecosystem neighborhood in which at least one better ecosystem must exist [74]. In another variant of EO, which is called τ -EO [25, 26], the selection of this particular species is done stochastically according to a probability distribution over the rank order. Opposition-based learning algorithms take a similar approach in utilizing negative learning mechanisms. OBL considers negative learning as a way to perform a simultaneous search in two opposite directions. This idea is based on the assumption that the chance of finding an optimal solution will be higher when the algorithms does the search in directions opposite to each other [34].

1.4 NEGATIVE LEARNING IN ANT COLONY OPTIMIZATION

The Aco research community has produced several works for incorporating negative learning mechanisms in Aco. In this section, we will review several examples of these efforts. Maniezzo [78] initiated the first of these efforts in an application that was aimed at solving the *Quadratic Assignment Problem* (QAP). This approach is characterized by utilizing *lower bound* values to estimate the attractiveness of choices during the QAP solution construction process. At each phase of the solution construction process, the attractiveness of each move is evaluated by a lower bound value for the cost of the completion of a partial solution. If the value is greater than the one of the current best solution, the move is discarded because it can not lead to a better solution. The results show that the proposal outperforms *Robust Tabu Search* (RTS) [79] and GRASP [80] as the state-of-the-art algorithms for solving QAP at that time.

Observing the similarity between the Aco algorithm and *Population-Based Incremental Learning* (PBIL) [81] from the field of evolutionary algorithms, Cordon et al. [82] proposed a negative learning strategy called *Best-Worst Ant System* (BWAS). This approach was applied originally to the TSP, but later, they also implemented it for the QAP with the addition of a new variant which is called *Best-Worst Ant Colony System* (BWACS) [83]. BWAS and BWACS work with three special features, the first two of which are inspired by PBIL: (1) best-worst pheromone update, (2) pheromone trail mutation, and (3) restart of the search

when it gets stuck. The pheromone update in these two algorithm variants is implemented according to the procedure described by Bullnheimer et al. [10, 11] in which only a certain number of ants that found the best solutions at the current iteration may add pheromone to the pheromone trails according to their rank. A negative learning concept is adopted in this pheromone update mechanism by making the worst ant—the ant that found the worst solution of the iteration—to reduce the pheromone on the solution components included in its solution. This way, the components that constitute the worst solution are expected to have a lower probability of being chosen in the next iterations. The authors compared all of their algorithm variants to the standard AS and ACS algorithms and showed that both BWAS and BWACS can outperform the standard algorithms. Note, however, that the authors acknowledged that the utilization of negative learning turned out to be the least important component of the improvement of BWAS and BWACS over the original algorithm.

Montgomery and Randall [84] proposed three negative learning strategies that were partially inspired by the work of Iredi et al. [85] which made use of several types of pheromones. In the first approach—which is called *Subtractive Anti Pheromone* (SAP)—the pheromone values of solution components that belong to the worst solution at each iteration are decreased. The second approach—which is called *Preferential Anti Pheromone* (PAP)—makes explicit use of negative pheromones in addition to the standard pheromone values. Each ant in PAP has a specific bias towards each pheromone type. Finally, the third approach—which is called *Explorer Ants* (EA)—uses a certain number of ants to build solutions consisting of solution components with lower pheromone values without introducing dedicated anti-pheromones. Unfortunately, their experimental evaluation did not allow clear conclusions about a potential advantage of any of the three strategies over standard Aco. Different extensions of the approaches from Montgomery and Randall [84] were explored by Simons and Smith [86]. The authors, however, admitted that nearly all of their approaches proved to be counter-intuitive. The only idea that showed to be useful to some extent was to make use of a rather high amount of anti-pheromone at the early stages of the search process. Another application to the TSP was proposed by Ramos et al. [87]. They proposed a method that uses a second-order co-evolved compromise between positive and negative feedback. According to the authors, their method achieves better results than the single positive feedback systems in the context of the TSP.

Malisia and Tizhoosh [88] integrated opposition based learning to a standard Ant Colony System approach for solving TSP problems. Following the

philosophy of OBL, they made use of *opposite-ants* in addition to the standard ones. Opposite-ants work in pair with the normal ants—which are called *leading-ants*—of the original ACS by using several different schemes. The first scheme is called *synchronous opposition* where the leading-ant and the opposite-ant begin the search at cities assigned to each of them randomly. The opposite-ant *mimics* the decisions of the leading ant in choosing the next city. Specifically, when the leading ant has chosen its next city, the opposite-ant will select its next city that has a similar rank to the one chosen by the leading-ant. The decision-making process will be totally different, however, when both ants are in the same current city. In this situation, the opposite-ant will choose the next city that has an opposite rank of the one chosen by the leading ant. The second scheme called *free opposition* still retained the same selection mechanism when both ants are in the same city. In case the opposite-ant is in a different city, it will choose the next city randomly instead of following the choice of the leading ant. In the last scheme called *free quasi-opposition*, the opposite-ant will choose the next city randomly, whether it is currently in the same city or not as the leading-ant. The opposite-ant, however, is forbidden from choosing the same next city when both ants are currently in the same city.

Malisia and Tizhoosh proposed two additional schemes—called *opposite-pheromone* strategies—which are named as follows: *Opposite-Pheromone per Edge* (OPE) and *Opposite-Pheromone per Node* (OPN). In contrast to the opposite-ant schemes, ants in these strategies choose the next city either by using normal pheromone or opposite-pheromone. A parameter—which is called *opposite-rate*—regulates the frequency at which each of these pheromone types is used during the solution construction process. The difference between OPE and OPN rests in the placement of pheromone; in OPE the pheromone is placed on edges while in OPN it is placed on nodes. Testing all their negative learning Aco variants on only four TSP instances, the authors reported that ACS with the OPN scheme is the only variant that performs better than the original algorithm. A more recent approach that also integrates OBL to Aco was implemented by Zhang et al. [89] for solving TSP problems. Building their proposals upon an Ant System algorithm, the authors developed three variants that differ in how each generates opposite paths. In contrast to the state-of-the-art Aco algorithms, these OBL Aco variants allow all ants to do the pheromone update instead of just a few best ants. Furthermore, the pheromone update is implemented by using solutions which are built by combining solutions from normal and opposite ants. The authors suggest that even though the use of this opposite information may reduce the convergence speed at the early stage of the search, the provided

additional information could increase the search accuracy in the later stage. The results show that at least one of their OBL Aco variants performs better than AS for each of the considered TSP instances.

Rojas-Morales et al. proposed a negative learning approach—which is called *Cooperative Opposition-Inspired Strategy for Ants* (COISA) [90]—for solving the *Multi Dimensional Knapsack Problem* (MDKP). Modifying an Aco variant with *Opposition-Inspired Learning* (OIL) [91], the main idea proposed by the authors is on providing negative feedback prior to the execution of the search. For this purpose, the authors proposed three mechanisms for obtaining negative feedback information. The first of them—which is called *Soft Opposite Learning* (SOL)—works similar to SAP by Montgomery and Randall [84] in that penalties are given to the components of the worst solution found in an iteration. The second mechanism—which is called *Worst Opposite Learning* (WOL)—follows the same working principle of the EA variant used by Montgomery and Randall [84], which intentionally searches in the opposite direction of the optimization objective instead of just looking for the worst result in the normal search scheme. The third mechanism that was proposed by Rojas-Morales et al. for providing the negative feedback information is called *Half Opposite Learning* (HOL). Different from the two previous mechanisms, HOL is problem-specific. It means that the negative feedback mechanism tries to identify bad solution components by using a greedy function of the particular CO problem. In the context of the MDKP, the specific greedy function is calculated by dividing the item's profit by the sum of the remaining capacities instead of the item's resource consumption in each dimension. SOL, WOL, and HOL work simultaneously in the first phase of the algorithm and combine their results to compose the negative feedback information given to the main Aco algorithm in the second phase of COISA. In their experiment, COISA is tested against *Ant Knapsack* (AK) [92]. Unfortunately, no consistent improvement over Ant Knapsack could be observed in the results. Furthermore, the authors admitted that running all three negative feedback mechanisms prior to the main search mechanism is time-consuming.

Ye et al. [93] proposed *Ant-Colony system with negative feedback mechanism* (ACON) for solving *Constraint Satisfaction Problems* (CSPs). The authors suggested that for improving the diversity of solutions, the area of low-quality solutions must be identified in advance. Subsequently, this area must be avoided so that the search process is directed to more promising regions. ACON identifies these low-quality areas using the same mechanism as the one employed in the SAP variant of Montgomery and Randall [84], i.e., it simply penalizes the worst assignment found in an iteration. ACON, however, utilizes a dedicated

negative pheromone model for storing the negative feedback information rather than reducing the values of the standard pheromones on the components of the worst solution as implemented in the SAP variant from Montgomery and Randall. Furthermore, ACON combines the use of the normal and the negative pheromone values in its assignment generation function in a way such that it causes the probability to visit unexplored paths to be higher than the probability to visit worse paths as the iteration number increases. Ye et al. tested and compared the ACON variant with the *ACO with smallest-domain-first heuristics* (ACOS) and *ACO with dynamic random heuristics* (ACOD) variants based on Aco algorithms from the work of Gonzalez and Camacho [94]. In addition to these Aco algorithm variants, the authors also compare the performance of ACON to the *Pure Random Walk* (PRW) algorithm [95] which is based on a pure stochastic mechanism. The results show that ACON performs better than the rest of the algorithm variants on several large-size problem instances. However, in small size problem instances, the superiority over ACOS and ACOD is not noticeable. Moreover, for several problem instances, the performance of ACON is inferior to the one of ACOS and ACOD.

Masukane and Mizuno proposed *cunning Ant System with NEgative Pheromone* (cASNEP) [96] which is also implemented for solving CSPs. The authors highlighted the need to speed up the search process by introducing bias on the pheromone trail as early as possible. For achieving this goal, an additional negative pheromone and the corresponding negative feedback mechanism are employed. This scheme is deployed in the *cunning Ant System* (cAS) [97], an *MMAS* based Aco variant that employs a mechanism in which a *cunning ant* borrows several solution components produced by a *donor ant*—usually the best ant—from the previous iteration. These borrowed solution components are then extended into a complete solution by the cunning ant by making use of pheromone and greedy information. The negative learning information in cASNEP is derived in a similar manner as in the SAP variant of Montgomery and Randall [84], in a sense that it does not employ a specific function other than the normal solution construction mechanism to search bad assignments; instead, it simply identifies the worst solution in an iteration, i.e., the assignment that has the highest number of constraint violations. For taking profit from this negative feedback, however, cASNEP uses a dedicated negative pheromone model, as in the PAP variant by Montgomery and Randall [84].

In their follow-up work, Masukane and Mizuno [98] suggested that even in the best-so-far solution, there are bad components that need to be identified and thus should be less considered in the subsequent construction of assignments.

Following this assumption, the authors proposed a further mechanism for introducing bias on the pheromone trails by allowing the negative pheromone to reduce the value of the normal pheromone of the components of the best-so-far solution during the update process. This is achieved by multiplying the normal pheromone value with a weighted parameter $w_{i,j}$ whose value is in the interval $[0,1]$ and is influenced by ϑ , which is the ratio between the component's negative pheromone value $\tau_{i,j}^{neg}$ and the maximum negative pheromone value τ_{max}^{neg} . Masukane and Mizuno proposed four pattern functions for determining $w_{i,j}$. The first of these pattern functions defines a continuous change of $w_{i,j}$ depending on the change of ϑ , while the remaining pattern functions define discrete changes of $w_{i,j}$, in varying paces, again depending on the change of ϑ . In these so-called *Ant Colony with DUal Pheromone Trails* (ADUPT) variants, they tested the impact of the $w_{i,j}$ change patterns on the performance of the algorithms. The results show that the variants whose $w_{i,j}$ change discretely can outperform the standard cAS and the variant whose $w_{i,j}$ change continuously to the change of ϑ .

1.5 GENERAL IDEAS

Section 1.2 shows that most metaheuristic techniques that make use of learning are based on positive learning. Aco algorithms, in particular, heavily rely on the utilization of positive learning. In fact, exhaustive works on balancing search exploitation and exploration based on positive learning mechanisms were the key to the Aco algorithm's improvement over the last three decades. In Section 1.3, we indicated that negative learning has already been employed in several variants of other metaheuristics such as evolutionary algorithms, simulated annealing, particle swarm optimization, extremal optimization, and opposition based learning. Negative learning mechanisms in evolutionary algorithm variants are basically used to prevent the algorithm from visiting search regions that lack promising candidate solutions. In SA, PSO, EO, and OBL algorithms, however, negative learning mechanisms are used to escape from local minima and diversify the generation of new solutions. All in all, negative learning mechanisms in these metaheuristics are used for improving their exploration capabilities.

The original version of Aco [7], as well as all major variants that followed [9–14], do not make use of negative learning mechanisms. In Section 1.4, we outlined all strategies for negative learning that we could find in several later versions of Aco. In the following sub-sections, we compare the two main features of these

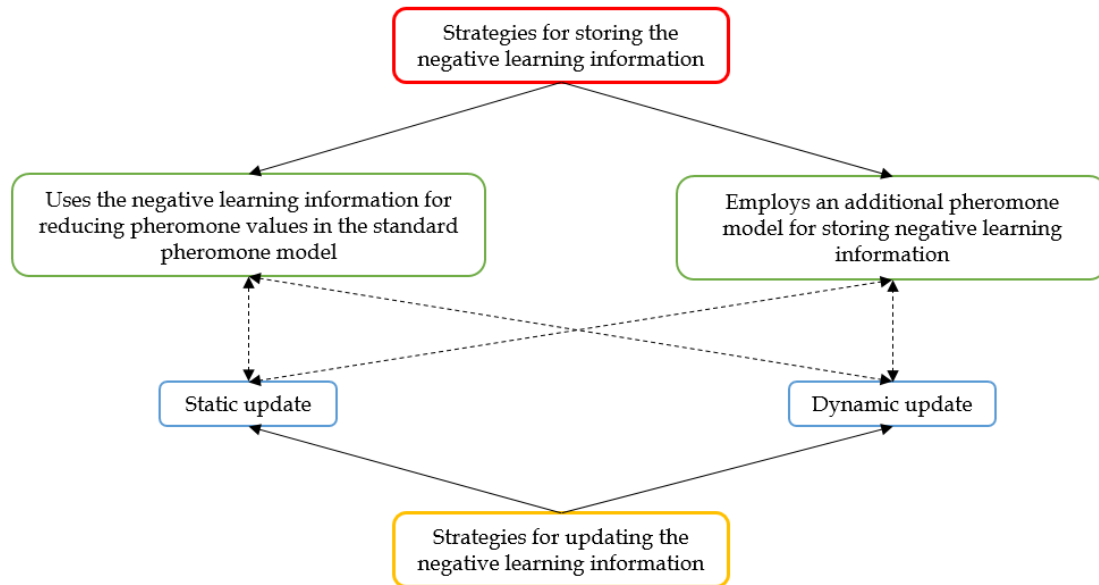


Fig. 1.2 The existing strategies for storing and updating the negative learning information

strategies and describe the ones we consider for comparison purposes in this thesis.

1.5.1 The Storage and Update of the Negative Learning Information

The first distinct feature that can be observed in the existing proposals for negative learning in Aco is the strategy for storing and updating the negative learning information. Figure 1.2 shows the current implementations of this feature. Most of the variants described in Section 1.4 do not store their negative pheromone in a dedicated pheromone model. Negative learning information in these variants is simply used to increase pheromone evaporation on the standard pheromone trails. Moreover, the pheromone trails are updated statically in most of them. This means that the pheromone evaporation rate due to negative learning information does not change at any moment of the algorithm runs. Among this group of negative learning Aco variants, however, the one from Malisia and Tizhoosh [88] already implemented a dynamic update strategy by changing the evaporation rate randomly during the search. This strategy is in accordance with the finding of Simon and Smith [86] and Zhang et al. [89] which suggested that the amount of pheromone reduction seems to have a different impact at different states of the search.

The examples of negative learning Aco variants that use a dedicated pheromone model can be found in the PAP variant from the work of Montgomery and Randall [84], the ACON variant from the work of Ye et al. [93], as well as the

cASNEP and ADUPT variants from the work of Masukane and Mizuno [96, 98]. The PAP, ACON, and cASNEP variants, however, still perform their negative pheromone update statically. Moreover, despite of having their own dedicated negative pheromone models, each of them is still updated with the same parameter setting as the one used in the standard pheromone update. The ADUPT variant, on the contrary, has already a separate and dynamic update mechanism for its negative pheromone values by using four mechanisms as described in Section 1.4. Employing a dedicated pheromone model and its update mechanism will most likely be more advantageous for negative learning implementations. By doing so, the learning rates (resp. evaporation rates) of standard pheromone and negative pheromone can be adjusted independently to suit the characteristics of the tackled problem instances. Accordingly, in the proposal of this thesis we employ a dedicated negative pheromone model with its own update mechanism.

1.5.2 The Way of Deriving the Negative Learning Information

Based on how the negative learning information is derived, there are two basic strategies: passive and active methods. The passive method, shown conceptually in Fig. 1.3, is implemented by adding a function to the baseline Aco algorithm. This function identifies the worst solution at each Aco iteration and then reduces its components' pheromone in the standard pheromone model. Examples of passive strategies can be found in the SAP version of Montgomery and Randall [84] as well as in the majority of negative learning Aco variants [82, 83, 86, 87, 93, 96, 98] described in Section 1.4.

The active method, shown conceptually in Fig. 1.4, is implemented by

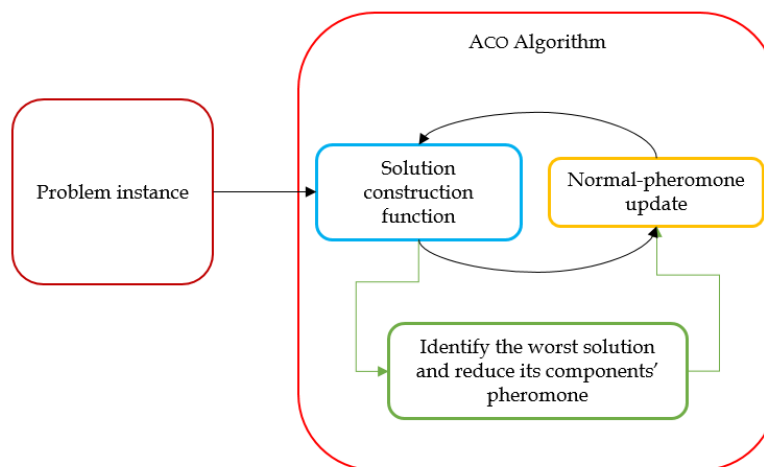


Fig. 1.3 Passive method for deriving negative learning information

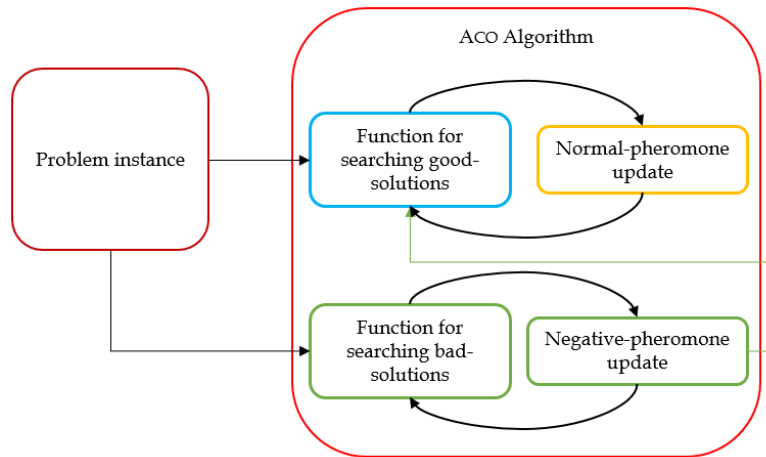


Fig. 1.4 Active method for deriving negative learning information

employing a function that actively tries to generate low-quality solutions in addition to the existing function that tries to generate good solutions in the baseline Aco algorithm. The worst solution found by this additional function is then used to update the negative pheromone model, as implemented in the PAP variant of Montgomery and Randall [84]. Alternatively, the worst solution is used to induce further pheromone evaporation at the corresponding solution components in the standard pheromone model, as implemented in the EA variant of Montgomery and Randall. More implementation examples of this active strategy can be found in the works of Malisia and Tizhoosh [88], Zhang et al. [89], and Rojas-Morales et al. [90]. Most of these active strategies employ only one function at a time to search for low-quality solutions, except for the variant in the work of Rojas-Morales et al. [90] that implements three simultaneous mechanisms for identifying low-quality solutions. The authors, however, admitted that simultaneous execution of these mechanisms could be time-consuming and thus requires additional resources.

1.6 LESSONS LEARNED AND RESULTING PROPOSAL

The following conclusion can be drawn from the previous proposals found in the literature. Active derivation of negative learning information by allocating a dedicated function to search for low-quality solutions will not benefit the search process if the goal is to avoid low-quality solutions. The baseline Aco algorithm is already able to avoid these low-quality solutions. However, it is likely to be more beneficial if the goal is to explore good solutions in the proximity of these low-quality solutions, in the line of what has already been implemented

in the works of Malisia and Tizhoosh [88], Zhang et al. [89], and Rojas-Morales et al. [90]. Passive derivation of negative learning information, on the contrary, would be beneficial if the objective is to avoid low-quality solutions that may be present around a good solution, as already implemented in the works of several authors [82–84, 86, 87, 93, 96, 98].

Based on these lessons learned, in this thesis we propose a novel method, shown in Fig. 1.5, to identify the negative learning information. This method is almost similar to the active derivation of negative learning information from Fig. 1.4 in that an active effort is allocated to identify the negative learning information. However, our proposed mechanism does not directly search for bad solution components as the active derivation of negative learning information method do. Instead, it compares the solution components generated during an iteration of the baseline Aco algorithm to those generated by an additional algorithmic component. Any solution component generated in an Aco iteration that is not chosen in the solution generated by the additional algorithmic component is considered a bad solution component. Subsequently, the negative pheromone value of this component is increased, because each solution component has both a standard and a negative pheromone value. Hence, it will lower the component’s probability of being selected as a part of solutions in subsequent iterations. Considering the important role of the additional algorithmic component in this proposal, we strongly believe that it should consist of another optimization algorithm that is powerful enough to generate high-quality solutions. Moreover, the additional algorithmic component is designed to work only on a reduced problem instance, or the *sub-instance*, which is generated by merging the components of solutions found

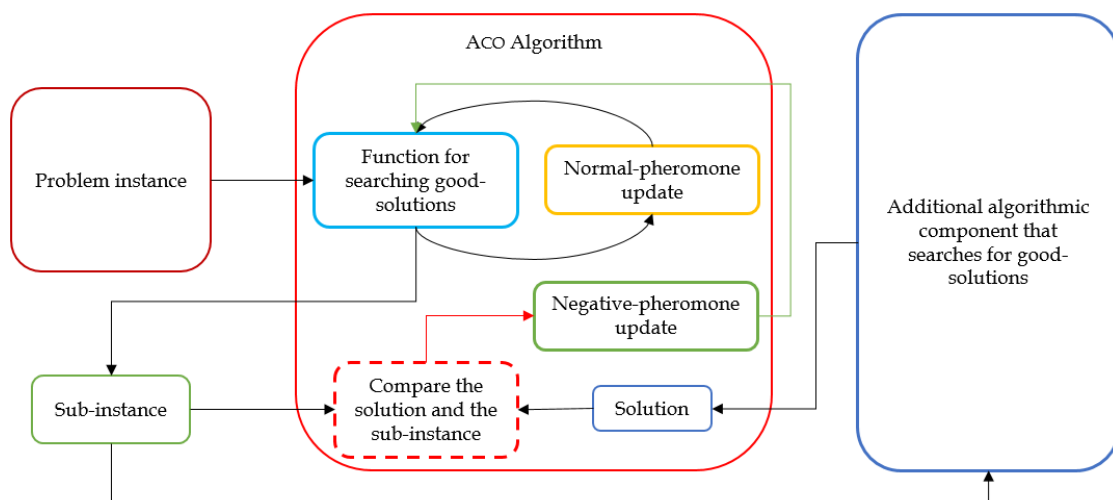


Fig. 1.5 Our method for deriving negative learning information

in an iteration of the baseline Aco algorithm. We believe this arrangement could result in an excellent cooperation between the baseline Aco and the additional algorithmic component. On the one hand, the baseline Aco could benefit from the high-quality feedback from the additional algorithmic component. On the other hand, the additional algorithmic component could benefit from the sub-instance with a smaller size consisting of high-quality solution components provided by the baseline Aco algorithm.

1.7 THESIS CONTRIBUTIONS

After designing the proposal outline above in more detail, we tested our negative learning Aco approach on several CO problems. Each of these problems requires its specific pheromone model and update system as well as its sub-instance and solutions generation mechanisms. In our negative learning Aco approach, the result from the additional algorithmic component can be utilized in two ways. It can be used to identify low-quality solution components and increase their negative pheromone values, as described in Section 1.5.2. However, it can also be used to reinforce the positive learning mechanism of the main Aco algorithm. Accordingly, we developed and tested several algorithm variants to evaluate the relationship between their configurations and their performances. In addition, we also experimented with different types of algorithmic components that is used for providing negative feedback information. Accordingly, the results of these experiments can be used for evaluating the applicability and effectiveness of our negative learning Aco approach not only to a wide range of CO problems but also to the use of different types of algorithmic components. The summary of these results is as follows:

- Our first test case [99] for this negative learning Aco strategy was the *Capacitated Minimum Dominating Set* (CapMDS) problem. For solving this optimization problem, we developed two negative learning Aco variants: (1) ACO_{neg} that uses the results of the additional algorithmic component only for updating the negative pheromone values and (2) ACO_{neg}^+ that uses the results for updating both the negative pheromone values and the best-so-far solution. In this first application, we used IBM ILOG CPLEX [43, 44] as the additional algorithmic component that provides negative feedback to the main Aco algorithm. The results of our experiments showed that our negative learning Aco variants significantly improve over the standard Aco algorithm. Compared to the state-of-the-art approach for the CapMDS

problem [100], one of the variants— $\text{Aco}_{\text{neg}}^+$ —improves the best-known results in 10 out of 36 problem instances. This work was presented and received the *Best Paper Award* in the ANTS 2020 Twelfth International Conference on Swarm Intelligence October 26-28, 2020 in Barcelona (<https://iridia.ulb.ac.be/ants2020/>). Subsequently, the paper was published in ANTS 2020 Swarm Intelligence, Lecture Notes in Computer Science, vol 12421 (https://doi.org/10.1007/978-3-030-60376-2_2).

- Our second implementation [101] of negative learning Aco was for the *Multi Dimensional Knapsack Problems* (MDKP). In this application, again, our negative learning Aco variants perform better than the standard Aco. Moreover, one of these variants— $\text{Aco}_{\text{neg}}^+$ —performs competitively with the state-of-the-art approaches for MDKP [102, 103]. This work was published in the proceedings of the 2021 5th International Conference on Intelligent Systems, Metaheuristics and Swarm Intelligence (<https://doi.org/10.1145/3461598.3461602>).
- In our third implementation [104], we developed two additional negative learning Aco variants: (1) Aco-Aco_{neg} and (2) Aco-Aco_{neg}⁺. Each of these variants uses an Aco algorithm as the additional algorithmic component that provides negative feedback to the baseline Aco algorithm. This implies that Aco is used twice in these algorithm variants: one Aco algorithm is the baseline algorithm and the other one acts as the additional algorithmic component that provides negative feedback to the baseline algorithm. Note, however, that since the parameter values of the two Aco are independent, they behave differently according to their own roles. In this work, we also re-implemented three negative learning Aco variants from the work of Montgomery and Randall [84] and one variant from the work of Ramos et al. [87]. After the application of our negative learning Aco variants to the MDKP and *Minimum Dominating Set* (MDS) problems, the results show that all our negative learning Aco variants improve over the standard Aco as well as over the negative learning Aco variants from other authors. Moreover, the $\text{Aco}_{\text{neg}}^+$ variant can perform competitively with the state-of-the-art approaches for the MDKP [102, 103] and the MDS [105–109]. This work was published in the Mathematics Journal Volume 9 Issue 4 2021 (<https://doi.org/10.3390/math9040361>).
- Our fourth test case [110] for the Aco_{neg} and Aco_{neg}⁺ variants was the *Minimum Positive Influence Dominating Set* (MPIDS) problem. The results show that these negative learning variants significantly improve over

the standard Aco algorithm. Moreover, the results also show that our negative learning Aco variants outperform all competitors from the literature [111, 112]. This work was published in the Proceedings of the Genetic and Evolutionary Computation Conference Companion, July 2021, Pages 1974–1977, (<https://doi.org/10.1145/3449726.3463130>).

- Exploring further the general applicability of our negative learning Aco strategy, we tested our approaches on the *Maximum Satisfiability* (MaxSAT) problem [113, 114], which differs substantially from the previously considered optimization problems. We developed two new variants in this work, $\text{ACO-SAT}_{\text{neg}}^+$ and $\text{ACO-SLS}_{\text{neg}}^+$, in addition to the existing ACO_{neg} and $\text{ACO}_{\text{neg}}^+$ variants. Each of these new variants uses MaxSAT solvers, *SATLike-c(w)* [115] and *SLSMcs* [115], as the additional algorithmic component that provides negative feedback to the main Aco algorithm. The results show that all negative learning Aco variants improve over the standard Aco algorithm. Moreover, each of the new variants, $\text{ACO-SAT}_{\text{neg}}^+$ and $\text{ACO-SLS}_{\text{neg}}^+$, improves over the corresponding MaxSAT solver, *SATLike-c(w)* and *SLSMcs*. This work was submitted to the *International Journal of Computational Intelligence Systems* and is currently in the state of minor revision.

Using an additional algorithmic component that works on a sub-instance for providing negative feedback to the main Aco algorithm is a novel approach for the implementation of negative learning in Aco. None of the previous approaches had implemented this idea. Our experiments show that this strategy works very well with *Cplex*, *Aco*, *SATLike-c(w)*, and *SLSMcs* as the additional algorithmic components. Moreover, this approach has shown to be able to solve numerous optimization problems effectively. Several algorithm variants— $\text{ACO}_{\text{neg}}^+$, $\text{ACO-SAT}_{\text{neg}}^+$, and $\text{ACO-SLS}_{\text{neg}}^+$ —can even compete with the state-of-the-art approaches for the corresponding optimization problems.

1.8 THE ORGANIZATION OF THIS THESIS

This thesis report is organized as follows:

- Chapter 1 explains the general characteristics of metaheuristics methods, the existing implementation of negative learning in metaheuristics (including Aco), and the main ideas of our own proposal for the implementation of negative learning in Aco.

- Chapter 2 provides a general description of our negative learning Aco proposal. This explanation is conducted in the context of *sub-set selection problems*, which represent all CO problems to which we applied our negative learning approach. In this context, we describe the baseline Aco algorithm that we used and the mechanism with which the negative learning is incorporated into this baseline algorithm. Despite having been described in this chapter, several important aspects of the algorithmic framework are re-introduced in subsequent chapters when describing its application to the specific CO problem that were considered.
- Chapter 3 describes our negative learning Aco implementation to the MDKP, a CO problem that is often used as a test case for new algorithmic proposals. In this work, we tested eleven negative learning Aco variants. Six of these variants are from our own proposal, and the rest of the variants are our re-implementation of negative learning approaches from the literature [84, 87]. In the six negative learning Aco variants of our own proposal, we experimented with two additional algorithmic components for providing negative feedback to the main Aco algorithm: (1) ILP solver CPLEX and (2) *MAX-MIN* Ant System (*MMAS*). Moreover, we also experimented with the way in which the result of the additional algorithmic component is used: (1) for reinforcing both the positive and negative learning mechanism, (2) for reinforcing only the negative learning mechanism, and (3) for reinforcing only the positive learning mechanism. Regarding the re-implementations of the four negative learning approaches from the literature, we describe their adaptation in the context of the baseline algorithm (*MMAS*) and the adaptation of their specific negative learning features to the baseline algorithm. In addition to these five negative learning Aco variants from the literature, we also compare our best-performing variant— $\text{Aco}_{\text{neg}}^+$ —to the state-of-the-art approaches for the MDKP [102, 103].
- Chapter 4 provides a detailed description of the implementation of our negative learning Aco to the MDS problem. As in the case of the MDKP, this CO problem is often used as a test case for new algorithmic proposals. Similar to our work for the MDKP, we tested six negative learning Aco variants from our own proposal against five variants from the literature. Moreover, we also compare our best-performing variant— $\text{Aco}_{\text{neg}}^+$ —to the state-of-the-art approaches for the MDS problem [105–109].
- Chapter 5 describes the application of our negative learning Aco to the

MaxSAT problem. This CO problem is a perfect test case for evaluating the general applicability of our negative learning approach due to its different characteristics compared to the rest of the CO problems to which we applied our negative learning approach in this work. Moreover, none of the existing negative learning Aco variants has been used so far to solve this CO problem. Despite of being a popular metaheuristic, Aco itself has been applied just a few times to MaxSAT. In this work, we also developed two new variants in addition to our previous negative learning Aco variants, which use CPLEX as the algorithmic component that provides negative feedback to the main Aco algorithm. Each of them makes use of a high-performance MaxSAT solver—SAT*Like-c(w)* [115] and SLSMcs [115]—for this purpose.

- Chapter 6 provides a detailed description of our negative learning Aco proposal for the CapMDS problem, which is the first CO problem on which we tested our negative learning Aco. In this work, we used the ILP solver CPLEX as the additional algorithmic component that provides negative feedback to the baseline Aco algorithm. Moreover, the results of our negative learning Aco are compared to the ones from the state-of-the-art approach for the CapMDS problem [100].
- Chapter 7 describes our negative learning Aco application to the MPIDS problem, which is an NP-hard combinatorial optimization problem with applications in social networks. Again, we tested our negative learning Aco variants using CPLEX as the additional algorithmic component that provides negative feedback to the main Aco algorithm. The obtained results of this experiment were compared to the one of the competitors from the literature [111, 112].
- Chapter 8 concerns the application of the *MMAS* algorithm to the *Multi-head Weigher Machine* (MWM) problem, which is a CO problem that has a wide application in food packaging industries. This work is an initial study that serves as an entry point for a wider application of our negative learning Aco approach to CO problems from mechanical engineering, which is the scientific background of the author of this thesis.
- Chapter 9 summarizes and evaluates our negative learning Aco approach. Subsequently, a plan for future works is provided.

1.9 PUBLICATIONS DERIVED FROM THIS THESIS

Most of the findings in this thesis have been published, as shown in the following list.

1. Teddy Nurcahyadi and Christian Blum. A new approach for making use of negative learning in ant colony optimization. In *Proceedings of the ANTS 2020 Swarm Intelligence, Lecture Notes in Computer Science*, vol 12421, pages 16–28, 2020. (https://doi.org/10.1007/978-3-030-60376-2_2).
2. Teddy Nurcahyadi and Christian Blum. Negative learning in ant colony optimization: Application to the multi dimensional knapsack problem. In *Proceedings of the 2021 5th International Conference on Intelligent Systems, Metaheuristics and Swarm Intelligence*, pages 22–27, 2021. (<https://doi.org/10.1145/3461598.3461602>).
3. Teddy Nurcahyadi and Christian Blum. Adding negative learning to ant colony optimization: A comprehensive study. *Mathematics*, 9(4):361, 2021. (<https://doi.org/10.3390/math9040361>).
4. Albert López Serrano, Teddy Nurcahyadi, Salim Bouamama, and Christian Blum. Negative learning ant colony optimization for the minimum positive influence dominating set problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1974–1977, 2021. (<https://doi.org/10.1145/3449726.3463130>).
5. (In the state of minor revision) Teddy Nurcahyadi, Christian Blum, and Felip Manyà. Negative learning ant colony optimization for MaxSAT. *International Journal of Computational Intelligence Systems*.
6. (Accepted for publication) Teddy Nurcahyadi and Christian Blum. Ant colony optimization for packing process optimization in multihead weigher machines. In *Proceedings of the International Symposium on Advances and Innovation in Mechanical Engineering 2021*.

Among these publications, the paper that reports our first application of the negative learning Aco to the CapMDS problem [99] received the Best Paper Award at the ANTS 2020 Twelfth International Conference on Swarm Intelligence October 26-28, 2020 in Barcelona (<https://iridia.ulb.ac.be/ants2020/>). Another paper that presents our negative learning Aco application to the MDKP and MDS problems [104], was published in the *Mathematics Journal* which is a

Q1 Journal with an impact factor of 2.258. Moreover, our paper on the negative learning Aco application to MaxSAT is currently in the state of *minor revision* at the International Journal of Computational Intelligence Systems which is a Q3 Journal with impact factor 1.736.

In addition to our publications in the scientific journals and conferences, several news outlets have covered our works. The following list provides the links to the media coverage.

1. <https://www.lavanguardia.com/natural/fauna-flora/20201222/6141796/diminuta-hormiga-faraon-inspira.html>
2. <https://www.lainformacion.com/management/aprendizaje-automatico-hormigas-feromonas-camino-negativo/2827153/>
3. <https://gacetamedica.com/investigacion/ia-un-algoritmo-en-busqueda-de-farmacos-es-inspirado-por-hormigas-faraon/>
4. <https://proceso.hn/investigadores-del-csic-estudian-las-hormigas-faraon-para-mejorar-algoritmos/>
5. <https://elglobal.es/industria/las-hormigas-faraon-inspiran-la-mejora-de-un-algoritmo-de-ia-desarrollado-por-el-csic/>
6. <https://www.diariosigloxxi.com/texto-ep/mostrar/20201222120814/hormigas-faraon-inspiran-algoritmo-aplicable-busqueda-farmacos-optimizacion-logistica>
7. <https://www.dicyt.com/viewNews.php?newsId=43334>
8. https://www.mundo-geo.es/conocimiento/hormigas-faraon-maestras-inteligencia-artificial_224853_102.html
9. https://www.elplural.com/leequid/ciencia/algoritmo-aprende-hormigas-faraon_256081102
10. <https://www.pcdemano.com/sc/14796/>
11. <https://losenlacesdelavida.fundaciondescubre.es/noticias/un-algoritmo-inspirado-en-los-caminos-descartados-por-las-hormigas/>

CHAPTER 2

GENERAL DESCRIPTION OF THE ALGORITHMIC FRAMEWORK

2.1 INTRODUCTION

This chapter provides a general description of the framework of our negative learning Aco. The description in this chapter is provided in the context of *subset selection problems*, which represents all CO problems that we used for testing our negative learning approaches. A subset selection problem is an optimization problem that can be modeled as follows. Given is a complete set C of solution components and an objective function $F()$ (to be maximized or minimized). Solutions S are subsets of C , that is, $S \subseteq C$. However, only those subsets S of C that fulfill all the problem constraints are valid solutions. Note that a wide range of well-known optimization problems can be phrased as subset selection problems. An example is the travelling salesman problem (TSP). Moreover, many of these problems can be modelled by *Integer Linear Programs* (ILPs).

2.2 *MMAS*: THE BASELINE ALGORITHM

Many of the negative learning approaches for Aco cited in Section 1.4 were introduced for different Aco variants. We add our negative learning proposal to a standard Aco algorithm: *MAX-MIN* Ant System (*MMAS*) in the hypercube framework [14], which is one of the most-used Aco versions from the last decade. This section first describes the standard *MMAS* algorithm in the hypercube framework for subset selection problems. This will be our baseline algorithm. Subsequently, we describe how the negative learning proposal is added to this baseline algorithm in Section 2.3.

The pheromone model \mathcal{T} in the context of subset selection problems consists of a pheromone value $\tau_i \geq 0$ for each solution component $c_i \in C$. The *MMAS* algorithm maintains three solutions throughout a run:

1. $S^{ib} \subseteq C$: the best solution generated at the current iteration, also called the *iteration-best* solution.
2. $S^{rb} \subseteq C$: the best solution generated since the last restart of the algorithm, also called the *restart-best* solution.
3. $S^{bsf} \subseteq C$: the *best-so-far* solution, that is, the best solution found since the start of the algorithm.

Moreover, the algorithm makes use of a Boolean control variable `bs_update` $\in \{\text{TRUE}, \text{FALSE}\}$ and the convergence factor $cf \in [0, 1]$ for deciding on the pheromone update mechanism and on the decision whether or not to restart the algorithm. At the start of the algorithm, solutions S^{bsf} and S^{rb} are initialized to `NULL`, the convergence factor is set to zero, `bs_update` is set to `FALSE` and the pheromone values are all initialized to 0.5 in function `InitializePheromoneValues(\mathcal{T})`; see lines 2 and 3 of Algorithm 2.1. Then, at each iteration, n_a solutions are probabilistically generated in function `Construct_Solution(\mathcal{T})`, based on pheromone information and on greedy

Algorithm 2.1 MMAS in the hypercube framework (the baseline algorithm)

```

1: input: a problem instance
2:  $S^{bsf} := \text{NULL}$ ,  $S^{rb} := \text{NULL}$ ,  $cf := 0$ , bs_update := FALSE
3: InitializePheromoneValues( $\mathcal{T}$ )
4: while termination conditions not met do
5:    $S^{\text{iter}} := \emptyset$ 
6:   for  $k = 1, \dots, n_a$  do
7:      $S^k := \text{Construct\_Solution}(\mathcal{T})$ 
8:      $S^{\text{iter}} := S^{\text{iter}} \cup \{S^k\}$ 
9:   end for
10:   $S^{ib} :=$  best solution from  $S^{\text{iter}}$ 
11:  if  $S^{ib}$  better than  $S^{rb}$  then  $S^{rb} := S^{ib}$ 
12:  if  $S^{ib}$  better than  $S^{bsf}$  then  $S^{bsf} := S^{ib}$ 
13:  ApplyPheromoneUpdate( $\mathcal{T}$ ,  $cf$ , bs_update,  $S^{ib}, S^{rb}, S^{bsf}$ )
14:   $cf := \text{ComputeConvergenceFactor}(\mathcal{T})$ 
15:  if  $cf > 0.999$  then
16:    if bs_update = TRUE then
17:       $S^{rb} := \text{NULL}$ , and bs_update := FALSE
18:      InitializePheromoneValues( $\mathcal{T}$ )
19:    else
20:      bs_update := TRUE
21:    end if
22:  end if
23: end while
24: output:  $S^{bsf}$ , the best solution found by the algorithm

```

information. A general explanation of the solution construction in the context of subset selection problems is presented in Section 2.2.1. The generated solutions are stored in set S^{iter} , and the best one from S^{iter} is stored as S^{ib} ; see lines 5–10 of Algorithm 2.1. Then, the restart-best and best-so-far solutions— S^{rb} and S^{bsf} —are updated with S^{ib} , if appropriate; lines 11 and 12. Afterward, the pheromone update is conducted in function `ApplyPheromoneUpdate(\mathcal{T} , cf , bs_update , S^{ib}, S^{rb}, S^{bsf})` and the new value for the convergence factor cf is computed in function `ComputeConvergenceFactor(\mathcal{T})`; lines 13 and 14. Detailed descriptions of these two functions are provided in Section 2.2.2 and 2.2.3. Finally, based on the values of cf and bs_update , the algorithm might be restarted. Such a restart consists in re-initializing all pheromone values, in setting the restart-best solution S^{rb} to `NULL`, and bs_update to `TRUE`.

2.2.1 Solution Construction

The construction of a solution in function `Construct_Solution(\mathcal{T})` starts with an empty solution $S = \emptyset$. Then, at each step, exactly one solution component is selected from a set $C(S) \subseteq C$. Hereby, $C(S)$ contains those solution components that can be added to S in a way such that the possibility of generating a complete, valid solution is maintained. The selection of the next solution component is done as follows. First, a probability $\mathbf{p}(c_i)$ is calculated for each $c_i \in C(S)$:

$$\mathbf{p}(c_i) := \frac{\eta(c_i) \cdot \tau(c_i)}{\sum_{c_k \in C(S)} \eta(c_k) \cdot \tau(c_k)} \quad (2.1)$$

Hereby, $\eta(c_i)$ in Eqn. (2.1) is the greedy information. Then, a random number $r \in [0, 1]$ is drawn. If $r \leq d_{\text{rate}}$, c_j (to be added to S) is selected such that $\mathbf{p}(c_j) \geq \mathbf{p}(c_i)$ for each $c_i \in \{0, 1\}$. Otherwise, c_j is chosen by *roulette-wheel-selection* based on the calculated probabilities. Note that d_{rate} is an important parameter of the algorithm. These solution construction steps are repeated until a valid solution S that fulfils all of the problem constraints is obtained. For each specific CO problem (MDKP, MDS, MaxSAT, CapMDS, and MPIDS) to which we applied our negative learning Aco approaches, the detail description of the solution construction is presented in Chapters 3, 4, 5, 6, and 7.

2.2.2 Pheromone Update

The pheromone update implemented by `ApplyPheromoneUpdate(\mathcal{T} , cf , bs_update , S^{ib}, S^{rb}, S^{bsf})` of Algorithm 2.1 is the same as in any other *MMAS* algorithm in the hypercube framework. First, the three solutions S^{ib} , S^{rb} , and

Table 2.1 Values for weights κ_{ib} , κ_{rb} , and κ_{bsf} which depend on cf and bs_update .

	bs_update = FALSE				bs_update = TRUE
	$cf < 0.4$	$cf \in [0.4, 0.6)$	$cf \in [0.6, 0.8)$	$cf \geq 0.8$	
κ_{ib}	1	$\frac{2}{3}$	$\frac{1}{3}$	0	0
κ_{rb}	0	$\frac{1}{3}$	$\frac{2}{3}$	1	0
κ_{bsf}	0	0	0	0	1

S^{bsf} receive weights κ_{ib} , κ_{rb} and κ_{bsf} , respectively. A standard setting of these weights, depending on cf and bs_update , is provided in Table 2.1. It always holds that $\kappa_{ib} + \kappa_{rb} + \kappa_{bsf} = 1$. After having determined the solution weights, each pheromone value τ_i is updated by using Eqn. (2.2).

$$\tau_i := \tau_i + \rho \cdot (\xi(c_i) - \tau_i) \quad , \quad (2.2)$$

where:

$$\xi(c_i) := \kappa_{ib} \cdot \Delta(S^{ib}, c_i) + \kappa_{rb} \cdot \Delta(S^{rb}, c_i) + \kappa_{bsf} \cdot \Delta(S^{bsf}, c_i) \quad (2.3)$$

Hereby, $\rho \in [0, 1]$ is the so-called learning rate, and function $\Delta(S, c_i)$ evaluates to 1 if solution S contains solution component c_i . Otherwise, the function evaluates to 0. Finally, after conducting this update, those pheromone values that exceed $\tau_{\max} = 0.999$ are set to τ_{\max} , and those values that have dropped below $\tau_{\min} = 0.001$ are set to τ_{\min} . Note that, in this way, a complete convergence of the algorithm is avoided. Finally, note that the learning mechanism represented by this pheromone update can clearly be labeled positive learning, because it makes use of the best solutions found for updating the pheromone values.

2.2.3 Convergence Factor

Just like the pheromone update mechanism, the computation of the convergence factor by the `ComputeConvergenceFactor(\mathcal{T})` function is a standard procedure that works in the same way for all *MMAS* algorithms in the hypercube framework, as presented in Eqn. (2.4)

$$cf := 2 \left(\left(\frac{\sum_{\tau_i \in \mathcal{T}} \max\{\tau_{\max} - \tau_i, \tau_i - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right) \quad (2.4)$$

Hereby, \mathcal{T} in Eqn. (2.4) stands for the set of all pheromone values, that is, the set of values τ_i for all $c_i \in C$. Accordingly, the value of cf is zero in the case when

all pheromone values are set to 0.5. The other extreme case is represented by all pheromone values having either value τ_{\min} or τ_{\max} . In this case, cf evaluates to one. Otherwise, cf has a value between 0 and 1. Herewith the description of all components of the baseline algorithm is completed.

2.3 ADDING NEGATIVE LEARNING TO \mathcal{MMAS}

In each negative learning mechanism there are two fundamental questions to be answered: (1) how is the negative information generated, maintained and updated, and (2) how is this information being used. This Section describes how these aspects are implemented in our negative learning proposal.

2.3.1 Information Maintenance

The proposed algorithmic framework maintains the information derived from negative learning by means of a second pheromone model \mathcal{T}^{neg} , which consists of a pheromone value τ_i^{neg} for each solution component $c_i \in C$. We henceforth refer to these values as the *negative pheromone values*. Whenever the pheromone values are (re-)initialized, the negative pheromone values are set to τ_{\min} , which is in contrast to the standard pheromone values, which are set to 0.5 (see Section 2.2).

2.3.2 Information Generation and Update

The generation of the information for negative learning is done by two new instructions (Eqn. (2.5) and Eqn. (2.6)), which are introduced between lines 9 and 10 of the baseline \mathcal{MMAS} algorithm (Algorithm 2.1).

$$\mathcal{S}^{\text{sub}} := \text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf) \quad (2.5)$$

$$\mathcal{S}^{\text{iter}} := \mathcal{S}^{\text{iter}} \cup \{\mathcal{S}^{\text{sub}}\} \quad (2.6)$$

The first instruction (Eqn. (2.5)) consists in executing function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$. This function merges all solutions from $\mathcal{S}^{\text{iter}}$, resulting in a subset $C' \subseteq C$ of solution components. The detailed mechanism on how a sub-instance to the original problem instance is generated based on C' for each specific CO problem (MDKP, MDS, MaxSAT, CapMDS, and MPIDS) is provided in Chapters 3, 4, 5, 6, and 7. Solving a sub-instance is done by the application of an additional algorithmic component (Cplex, Aco, or MaxSAT solvers) to find the best-possible solution that only consists of solution components from C' . Trying to solve the sub-instance for an allotted

computation time, the additional algorithmic component produces a solution which is called S^{sub} . The computation time (t^{sub} CPU seconds) for the additional algorithmic component is calculated on the basis of a pre-defined computation time limit and the current value of the convergence factor, which is passed to function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ as a parameter. In particular, the allowed computation time (in seconds) is $(1 - cf)t^{\text{sub}} + 0.1cf$. This means that the available computation time for solving the sub-instance that was created on the basis of C' decreases with an increasing convergence factor value. The rationale behind this setting is that, when the convergence factor is low, the variance between solutions in $\mathcal{S}^{\text{iter}}$ is rather high and C' is therefore rather large, which means that more time is necessary to explore the sub-instance created on the basis of C' .

The last action in function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ is the update of the negative pheromone values based on solution S^{sub} . This update only concerns the negative pheromone values of those components that form part of C' and is implemented by using Eqn. (2.7).

$$\tau_i^{\text{neg}} := \tau_i^{\text{neg}} + \rho^{\text{neg}} \cdot (\xi(c_i)^{\text{neg}} - \tau_i^{\text{neg}}) \quad \text{for all } c_i \in C' \quad (2.7)$$

Hereby the parameter ρ^{neg} in Eqn. (2.7) is the *negative learning rate*. Moreover, the parameter $\xi(c_i)^{\text{neg}} = 1$ if $c_i \in S^{\text{sub}}$ and respectively $\xi(c_i)^{\text{neg}} = 0$ otherwise.

The second instruction added to the baseline \mathcal{MMAS} (see Eqn. (2.6)) adds solution S^{sub} to the $\mathcal{S}^{\text{iter}}$. Subsequently, an iteration-best solution S^{ib} is selected among the solutions from $\mathcal{S}^{\text{iter}}$, as shown in line 10 of Algorithm 2.1. Note that, if S^{sub} is better than all other solutions in $\mathcal{S}^{\text{iter}}$, it will be used for both updating the negative pheromone values (see the description of function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$) and the iteration-best solution S^{ib} of the main algorithm. Moreover, using S^{sub} for updating S^{ib} implies that S^{sub} is also used for updating the standard pheromone values of the main algorithm (see lines 11 – 13 of Algorithm 2.1 and the description of the standard pheromone update in \mathcal{MMAS} in Section 2.2.2).

2.3.3 Information Use

The negative pheromone values are used in the context of the construction of solutions. In particular, Eqn. (2.1) is replaced by Eqn. (2.8).

$$\mathbf{p}(c_i) := \frac{\eta(c_i) \cdot \tau_i \cdot (1 - \tau_i^{\text{neg}})}{\sum_{c_k \in C(S)} \eta(c_k) \cdot \tau_k \cdot (1 - \tau_k^{\text{neg}})} \quad (2.8)$$

In this way, those solution components that have accumulated a rather high

negative pheromone value have a decreased probability to be chosen for solutions in the current iteration. Herewith the description of our negative learning Aco is completed. Further detail on how this mechanism is implemented for each specific CO problem is provided in Chapters [3](#), [4](#), [5](#), [6](#), and [7](#).

CHAPTER 3

APPLICATION TO THE MULTI DIMENSIONAL KNAPSACK PROBLEM

3.1 INTRODUCTION

This chapter describes the application of our negative learning Aco for the well-known *Multi Dimensional Knapsack Problem* (MDKP) [116]. Some parts of this chapter were also presented in our papers [101, 104] that were published in the proceedings of ISMSI 2021: 2021 5th International Conference on Intelligent Systems, Metaheuristics and Swarm Intelligence (<https://doi.org/10.1145/3461598.3461602>) and in the Mathematics Journal (<https://doi.org/10.3390/math9040361>). The obtained results show (1) that our negative learning mechanism significantly improves over standard Aco, and (2) that our approach obtains results that are comparable to the current state of the art for the MDKP. Moreover, for comparison purposes we implement several negative learning approaches introduced for Aco in the related literature. The obtained results show that our mechanism outperforms all of them with statistical significance.

3.2 THE MULTI DIMENSIONAL KNAPSACK PROBLEMS

The MDKP is a popular *NP*-hard combinatorial optimisation problem which is often used as a test case for new algorithmic proposals (see, for example, [117–119]). The problem can technically be defined as follows. Given is (1) a set $C = \{c_1, \dots, c_n\}$ of n items and (2) a number of m resources. The availability of each resource k is limited by $\text{cap}_k > 0$, which is also called the capacity of resource k . Moreover, each item $c_i \in C$ consumes a fixed amount $r_{i,k} \geq 0$ from each resource $k = 1, \dots, m$ (*resource consumption*). Additionally, each item $c_i \in C$ comes with a profit $p_i > 0$.

Any subset $S \subseteq C$ is a candidate solution to the MDKP. However, such a

candidate solution S is only valid if and only if, concerning all resources, the total amount consumed by the items in S does not exceed the resource capacities. In other words, it is required that $\sum_{c_i \in S} r_{i,k} \leq \text{cap}_k$ for all $k = 1, \dots, m$. Moreover, a valid solution S is called non-extensible, if no $c_i \in C \setminus S$ can be added to S without destroying its property of being a valid solution. The aim is to find a valid solution S of maximum total profit ($\sum_{c_i \in S} p_i$).

3.2.1 ILP Model for the MDKP

The MDKP can in the following way be expressed by means of an integer linear program (ILP).

$$\text{maximize } \sum_{c_i \in C} p_i \cdot x_i \quad (3.1)$$

subject to:

$$\sum_{c_i \in C} r_{i,k} \cdot x_i \leq \text{cap}_k \quad \forall k = 1, \dots, m \quad (3.2)$$

$$x_i \in \{0, 1\} \quad \forall c_i \in C \quad (3.3)$$

This model is based on a binary variable x_i for each item from $c_i \in C$. Inequalities (3.2) are called the *knapsack constraints*. In general, the literature offers very successful exact solution techniques; see, for example, [120–122]. However, devising heuristic solvers still remains to be a challenge. Among numerous metaheuristic proposals for the MDKP problem, the currently best performing ones are the DQPSO algorithm from [103] and the TPTEA algorithm from [102].

3.3 MMAS IMPLEMENTATION TO THE MDKP

We are aware of the fact that a lot of text in Section 3.3 will be a repetition from Section 2.2. We decided for this option in order to avoid that a reader has to go back to that section. As in the general description of the MMAS algorithm to subset selection problems in Section 2.2, we keep three different solutions at any time: (1) the best solution generated at the current iteration, that is, the *iteration-best* solution S^{ib} , (2) the best solution generated since the last restart of the algorithm, that is, the *restart-best* solution S^{rb} , and (3) the best solution generated overall, that is, the *best-so-far* solution S^{bsf} . At the start of the algorithm, both S^{bsf} and S^{rb} are initialized to empty sets (the worst solutions possible); see line 2 of

Algorithm 2.1. Moreover, a Boolean control variable (`bs_update`) that is used to control the pheromone update is initialized to `FALSE`.

For the MDKP, we used a pheromone model \mathcal{T} which consists of a pheromone value τ_i for each item $c_i \in C$. All pheromone values from \mathcal{T} are initialized to 0.5 by function `InitializePheromoneValues(\mathcal{T})` of Algorithm 2.1. Then, at each iteration n_a solutions are generated based on greedy information and on pheromone information by function `Construct_Solution(\mathcal{T})` of Algorithm 2.1. For the MDKP, this function is described in Sub-Section 3.3.1. Subsequently, all solutions generated in the current iteration—that is, all solutions from S^{iter} —are merged, resulting in a subset $C' \subseteq C$. After updating solutions S^{ib} , S^{rb} and S^{bsf} in lines 10–12 of Algorithm 2.1, the pheromone update is conducted in function `ApplyPheromoneUpdate(\mathcal{T} , cf , bs_update, S^{ib}, S^{rb}, S^{bsf})`, which is explained in Sub-Section 3.3.2. Finally, the value of the convergence factor is determined in function `ComputeConvergenceFactor(\mathcal{T})`, and—in case $cf > 0.999$ and `bs_update = TRUE`—the algorithm is restarted (see lines 15–22 of Algorithm 2.1).

3.3.1 Solution Construction

Function `Construct_Solution(\mathcal{T})` of Algorithm 2.1 is implemented in the case of the MDKP as follows. It starts with an empty solution $S := \emptyset$, and at each construction step exactly one item c_j is selected from a set $\bar{C} \subseteq C$. The definition of \bar{C} is as follows. An item $c_k \in C$ forms part of \bar{C} if and only if (1) $c_k \notin S$, and (2) $S \cup \{c_k\}$ is a valid solution. The probability $\mathbf{p}(c_i)$ for an item $c_i \in \bar{C}$ to be chosen at the current construction step is calculated by Eqn. (3.4).

$$\mathbf{p}(c_i) := \frac{\eta_i \cdot \tau_i}{\sum_{c_k \in \bar{C}} \eta_k \cdot \tau_k} \quad (3.4)$$

Hereby, η_i is the so-called *utility ratio* of item c_i and its definition is provided in Eqn. (3.5).

$$\eta_i := \frac{p_i}{\sum_{k=1}^m r_{i,k} / \text{cap}_k} \quad \forall c_i \in C \quad (3.5)$$

Subsequently, a random number $r \in [0, 1]$ is drawn. If $r \leq d_{\text{rate}}$, c_j (to be added to S) is selected such that $\mathbf{p}(c_j) \geq \mathbf{p}(c_i)$ for all $c_i \in \bar{C}$. Otherwise, c_j is chosen by roulette-wheel-selection based on the calculated probabilities.

3.3.2 Pheromone Update and Convergence Factor

The pheromone update in the case of the MDKP is implemented by function `ApplyPheromoneUpdate(\mathcal{T} , cf , bs_update, S^{ib}, S^{rb}, S^{bsf})` of Algorithm 2.1. This

update is implemented in the same way as in any MMAS algorithms implemented in the hypercube framework, making use of solutions S^{ib} , S^{rb} , and S^{bsf} in the following way. The weight of each solution on the pheromone update is calculated on the basis of the convergence factor (cf) and the value of bs_update (see Table 2.1). Each pheromone value τ_i is updated by using Eqn (3.6).

$$\tau_i := \tau_i + \rho \cdot (\xi_i - \tau_i) \quad (3.6)$$

Hereby $\rho \in [0, 1]$ in Eqn (3.6) is the learning rate and ξ_i in the same equation is defined in Eqn (3.7).

$$\xi_i := \kappa_{ib} \cdot \Delta(S^{ib}, c_i) + \kappa_{rb} \cdot \Delta(S^{rb}, c_i) + \kappa_{bs} \cdot \Delta(S^{bsf}, c_i) \quad (3.7)$$

Note that in Eqn (3.7), κ_{ib} is the weight of solution S^{ib} , κ_{rb} the one of solution S^{rb} , and κ_{bs} the one of solution S^{bsf} . Moreover, $\Delta(S, c_i)$ evaluates to 1 if and only if $c_i \in S$. Otherwise, the function evaluates to 0. Note also that (according to Table 2.1) $\kappa_{ib} + \kappa_{rb} + \kappa_{bs} = 1$. After this pheromone update, those values that exceed $\tau_{\max} = 0.999$ are set back to τ_{\max} , and those values that have dropped below $\tau_{\min} = 0.001$ are set back to τ_{\min} . This prevents the algorithm from reaching the state of complete convergence.

The value of the convergence factor cf is computed in a standard way on the basis of the pheromone values by function `ComputeConvergenceFactor(\mathcal{T})` of Algorithm 2.1. For the MDKP, cf is calculated by using Eqn (3.8).

$$cf := 2 \left(\left(\frac{\sum_{\tau_i \in \mathcal{T}} \max\{\tau_{\max} - \tau_i, \tau_i - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right) \quad (3.8)$$

Hereby, \mathcal{T} in Eqn. (3.8) stands for the set of all τ_i values. Accordingly, the value of cf results in zero, when all pheromone values are set to 0.5. In contrast, when all pheromone values have either value τ_{\min} or τ_{\max} , the value cf evaluates to one. In all other cases, cf has a value between 0 and 1.

3.4 ADDING NEGATIVE LEARNING TO MMAS

As already explained in Chapter 2, a negative pheromone model \mathcal{T}^{neg} is used for storing the negative learning information. For the MDKP, this pheromone model consists of a negative pheromone τ_i^{neg} for each item $c_i \in C$. Each of these negative pheromone values is initialized or re-initialized to $\tau_{\min} = 0.001$, which is

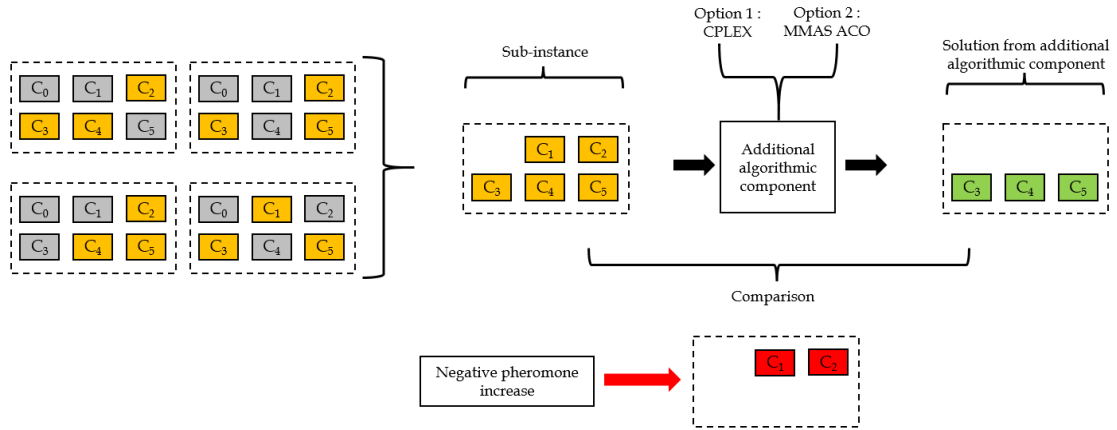


Fig. 3.1 Illustrative example of negative learning Aco applied to the MDKP

in contrast to the standard pheromone values, which are set to 0.5 (see Section 3.3). As described in Section 2.3.2, the information for negative learning is generated by two new instructions $S^{\text{sub}} := \text{SolveSubinstance}(S^{\text{iter}}, cf)$ (Eqn. (2.5)) and $S^{\text{iter}} := S^{\text{iter}} \cup \{S^{\text{sub}}\}$ (Eqn. (2.6)) which are introduced between lines 9 and 10 of the baseline \mathcal{MMAS} algorithm (Algorithm 2.1).

Figure 3.1 provides an illustrative example on how the negative learning is added to the baseline Aco in the MDKP case. The set C in this example consists of six items, and the baseline Aco algorithm—in this example—produces four solutions in one iteration. Function $\text{SolveSubinstance}(S^{\text{iter}}, cf)$ in Eqn. (2.5) merges all four solutions from S^{iter} , resulting in a subset $C' \subseteq C$. Notice that from six items available in the set C , the sub-instance C' in Fig 3.1 consists of only five items found in S^{iter} . Then an optimization algorithm is applied to find the best-possible solution that only consists of items from C' .

In this work, we have experimented with two options as the optimization algorithm employed for solving the sub-instance C' :

1. **Option 1:** Application of the ILP solver CPLEX 12.10. In the MDKP problem, the ILP model from sub-Section 3.2.1 is used after adding an additional constraint $x_i = 0$ for all $c_i \in C \setminus C'$.
2. **Option 2:** Application of the baseline \mathcal{MMAS} algorithm (Algorithm 2.1). This application of the baseline \mathcal{MMAS} only considers items from C' for the construction of solutions. Moreover, this \mathcal{MMAS} application uses its own pheromone values, parameter settings, etc. Finally, the best-so-far solution of this (inner) Aco is initialized with S^{ib} .

In any of these options, the optimization algorithm returns a solution S^{sub} after the allotted computation time of $(1 - cf)t^{\text{sub}} + 0.1cf$ CPU second is used up. Hereby

t^{sub} is the maximum computation time, which is subject to parameter tuning. This setting implies that the available computation time for solving the sub-instance C' decreases with an increasing convergence factor value. The rationale behind this setting is that, when the convergence factor is high, the variance between solutions in $\mathcal{S}^{\text{iter}}$ is rather low and C' is therefore rather small, which means that less time is needed to explore sub-instance C' .

In addition to solving the sub-instance C' , the function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ also performs the update of the negative pheromone values, as indicated in Fig 3.1. This update is done by comparing the components in solution S^{sub} to those in the sub-instance C' . Any component of the sub-instance C' that does not form part of S^{sub} is considered a low-quality solution component, at least in comparison to the others from C' . Notice in Fig 3.1 that there are five items in the sub-instance C' and only three of these items appear in solution S^{sub} . This way, the last two items that remain in the sub-instance C' are marked as low-quality solution components. Subsequently, their negative pheromone value is increased. For each item in set C' , its negative pheromone value is updated by using Eqn. (3.9).

$$\tau_i^{\text{neg}} := \tau_i^{\text{neg}} + \rho^{\text{neg}} \cdot (\xi_i^{\text{neg}} - \tau_i^{\text{neg}}) \quad , \quad (3.9)$$

Hereby ρ^{neg} in Eqn. (3.9) is the negative learning rate and $\xi_i^{\text{neg}} = 1$ if $c_i \notin S^{\text{sub}}$, resp. $\xi_i^{\text{neg}} = 0$ otherwise. After the execution of function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ (Eqn. (2.5)), the algorithm then adds the S^{sub} to $\mathcal{S}^{\text{iter}}$ (Eqn. (2.6)), which contains all solutions from the current iteration.

The negative pheromone values are used for the solutions construction in the negative learning Aco algorithm by replacing the standard \mathcal{MMAS} calculation in Eqn. (3.4) with the one in Eqn. (3.10).

$$\mathbf{p}(c_i) := \frac{\eta_i \cdot \tau_i \cdot (1 - \tau_i^{\text{neg}})}{\sum_{c_k \in \bar{C}} \eta_k \cdot \tau_k \cdot (1 - \tau_k^{\text{neg}})} \quad (3.10)$$

In this way, those items that have accumulated a rather high negative pheromone value have a decreased probability to be chosen for solutions.

3.5 PROPOSALS FROM THE LITERATURE

As mentioned in the introduction of this chapter, we compared our negative learning strategies to some of the ones available in the existing literature. These

negative learning proposals were introduced in the context of several different Aco versions. In order to ensure a fair comparison, we re-implemented those proposals that we chose for comparison in the context of the baseline *MMAS* algorithm. In particular, we implemented four different approaches, which all share the following common feature. In addition to the iteration-best solution (S^{ib}), the restart-best solution (S^{rb}) and the best-so-far solution (S^{bsf}), these extensions of the baseline *MMAS* algorithm maintain the *iteration-worst* solution (S^{iw}), the *restart-worst* solution (S^{rw}) and the *worst-so-far* solution (S^{wsf}). As in the case of S^{rb} and S^{bsf} , solutions S^{rw} and S^{wsf} are initialized to NULL at the start of the algorithm. Then, the following three lines are introduced between lines 12 and 13 of Algorithm 2.1:

$$\begin{aligned}
 & S^{iw} := \text{worst solution from } S^{\text{iter}} \\
 & \text{if } S^{iw} \text{ worse than } S^{rw} \text{ then } S^{rw} := S^{iw} \\
 & \text{if } S^{iw} \text{ worse than } S^{wsf} \text{ then } S^{wsf} := S^{iw}
 \end{aligned}$$

The way in which these three additional solutions are used differs among the four implemented approaches.

3.5.1 Subtractive Anti-Pheromone (SAP)

This idea is adopted from the work of Montgomery and Randall [84], but has already been used in similar form in the work of Maniezzo [78] and in the work of Cordon et al. [82]. Our implementation of this idea is as follows. After the standard pheromone update of the baseline *MMAS* algorithm (see line 13 of Algorithm 2.1), the following is done. First, a set B is generated by joining the items in solutions S^{iw} , S^{rw} and S^{wsf} , that is, $B := S^{iw} \cup S^{rw} \cup S^{wsf}$. Then, all those items for which the corresponding pheromone value receives an update from at least one of the solutions S^{ib} , S^{rb} , or S^{bsf} in the current iteration are removed from B . That is:

$$\begin{aligned}
 & \text{if } \kappa_{ib} > 0 \text{ then } B := B \setminus S^{ib} \\
 & \text{if } \kappa_{rb} > 0 \text{ then } B := B \setminus S^{rb} \\
 & \text{if } \kappa_{bsf} > 0 \text{ then } B := B \setminus S^{bsf}
 \end{aligned}$$

Afterward, the following additional update is applied by using Eqn. (3.11).

$$\tau_i := \gamma \cdot \tau_i \quad \forall c_i \in B \quad (3.11)$$

In other words, the pheromone values of all those components that appear in “low-quality” solutions, but who do not form part of “good” solutions, are subject to a pheromone value decrease depending on the *reduction rate* γ . Finally, note

that the solution construction procedure in this variant—which is henceforth labeled ACO-SAP—is exactly the same as in the baseline *MMAS* algorithm.

3.5.2 Explorer Ants (EA)

The explorer ants approach from Montgomery and Randall [84]—henceforth labeled ACO-EA—is very similar to the previously presented ACO-SAP approach. The only difference is in the construction of solutions. This approach has an additional parameter: $p_{\text{exp}_a} \in [0, 1]$, the proportion of explorer ants. Given the number of ants (n_a) and p_{exp_a} , the number of explorer ants n_a^{exp} is calculated by using Eqn. (3.12).

$$n_a^{\text{exp}} := \max\{1, \lfloor p_{\text{exp}_a} \cdot n_a \rfloor\} \quad (3.12)$$

At each iteration, $n_a - n_a^{\text{exp}}$ solution constructions are performed in the same way as in the baseline *MMAS* algorithm. The remaining n_a^{exp} solution constructions make use of Eqn. (3.13) (instead of Eqn. (3.4)) for calculating the probabilities:

$$\mathbf{p}(c_i) := \frac{\eta_i \cdot (1 - \tau_i)}{\sum_{c_k \in \bar{C}} \eta_k \cdot (1 - \tau_k)} \quad (3.13)$$

In other words, explorer ants make use of the opposite of the pheromone values for constructing solutions.

3.5.3 Preferential Anti-Pheromone

Like our own negative learning proposal, the preferential anti-pheromone approach from the work of Montgomery and Randall [84] makes use of an additional set \mathcal{T}^{neg} of pheromone values. Remember that \mathcal{T}^{neg} contains a pheromone value τ_i^{neg} for each item $c_i \in C$. These negative pheromone values are initialized at the start of the algorithm, as well as when the algorithm is restarted, to a value of 0.5. Moreover, after the update of the standard pheromone values in line 13 of the baseline *MMAS* algorithm, exactly the same update is conducted for the negative pheromone values by using Eqn. (3.14).

$$\tau_i^{\text{neg}} := \tau_i^{\text{neg}} + \rho^{\text{neg}} \cdot (\xi_i^{\text{neg}} - \tau_i^{\text{neg}}) \quad , \quad (3.14)$$

where:

$$\xi_i^{\text{neg}} := \kappa_{ib} \cdot \Delta(S^{iw}, c_i) + \kappa_{rb} \cdot \Delta(S^{rw}, c_i) + \kappa_{bsf} \cdot \Delta(S^{wsf}, c_i) \quad (3.15)$$

Hereby, $\rho^{\text{neg}} \in [0, 1]$ in Eqn. (3.14) is the negative learning rate, and function $\Delta(S, c_i)$ evaluates to 1 if and only if item c_i forms part of solution S . Moreover, values κ_{ib} , κ_{rb} and κ_{bsf} are the same as the ones used for the update of the standard

pheromone values. This means that the learning of the negative pheromone values is dependent on the dynamics of the learning of the standard pheromone values.

The standard pheromone values and the negative pheromone values are used as follows for the construction of solution. The probabilities for the a -th solution construction—where $a = 1, \dots, n_a$ —are determined by using Eqn. (3.16).

$$\mathbf{p}(c_i) := \frac{\eta_i \cdot (\lambda\tau_i + (1 - \lambda)\tau_i^{\text{neg}})}{\sum_{c_k \in \bar{C}} \eta_k \cdot (\lambda\tau_k + (1 - \lambda)\tau_k^{\text{neg}})} \quad , \quad (3.16)$$

where:

$$\lambda := \frac{a - 1}{n_a - 1} \quad (3.17)$$

Accordingly $\lambda = 0$ for the first solution construction, which means that only the negative pheromones values are used. In the other extreme, it holds that $\lambda = 1$ for the n_a -th solution construction, that is, only the standard pheromone values are used. All other solution constructions combine both pheromone types at different rates. Note that this preferential anti-pheromone approach is henceforth labeled ACO-PAP.

3.5.4 Second-Order Swarm Intelligence

Our implementation of the second-order swarm intelligence approach from the work of Ramos et al. [87] works exactly like the ACO-PAP approach from the previous section for what concerns the definition and the update of the negative pheromone values. However, the way in which they are used is different. The item probabilities for the construction of solutions is calculated by using Eqn. (3.18).

$$\mathbf{p}(c_i) := \frac{\eta_i \cdot (\tau_i)^\alpha \cdot (\tau_i^{\text{neg}})^{(\alpha-1)}}{\sum_{c_k \in \bar{C}} \eta_k \cdot (\tau_k)^\alpha \cdot (\tau_k^{\text{neg}})^{(\alpha-1)}} \quad , \quad (3.18)$$

Hereby $\alpha \in [0, 1]$ is a parameter of the algorithm. Note that this approach is henceforth labeled Aco^{2o}.

3.6 SUMMARY OF THE TESTED ALGORITHMS

In addition to the baseline *MMAS* algorithm (henceforth simply labeled ACO), and the four approaches from the literature (ACO-SAP, ACO-EA, ACO-PAP and Aco^{2o}) we test the following six versions of the negative learning mechanism proposed in this thesis:

1. $\text{Aco}_{\text{neg}}^+$: The mechanism described in Section 3.4 using option 1 (CPLEX) for solving sub-instances.
2. Aco_{neg} : This algorithm is the same as $\text{Aco}_{\text{neg}}^+$, with the exception that the instruction $\mathcal{S}^{\text{iter}} := \mathcal{S}^{\text{iter}} \cup \{S^{\text{sub}}\}$ (Eqn. (2.6)) is not performed. This means that the algorithm does not make use of solution S^{sub} for additional positive learning. Studying this variant will show if, by solely adding negative learning, the algorithm improves over the baseline Aco.
3. Aco^+ : This algorithm is the same as $\text{Aco}_{\text{neg}}^+$, apart from the fact that the update of the negative pheromone values is not performed. In this way, the algorithm only makes use of the additional positive learning mechanism obtained by adding solution S^{sub} to $\mathcal{S}^{\text{iter}}$.

The remaining three algorithm variants are $\text{Aco-Aco}_{\text{neg}}^+$, $\text{Aco-Aco}_{\text{neg}}$ and Aco-Aco^+ . These algorithm variants are the same ones as $\text{Aco}_{\text{neg}}^+$, Aco_{neg} and Aco^+ , except that they make use of option 2 (baseline Aco algorithm) for solving the corresponding sub-instances at each iteration.

A summary of the parameters that arise in these 11 algorithms is provided in Table 3.1, together with a description of their function and the parameter value domains that were used for parameter tuning (which will be described in Section 3.7.1). Moreover, an overview on the parameters that are involved in each of the 11 algorithms is provided in Table 3.2.

Table 3.1 Summary of the parameters that arise in the considered algorithms, together with their description and the domains considered for parameter tuning.

Parameter	Description	Considered Domain
n_a	Number of solution constructions per iteration	{3, 5, 10, 20}
ρ	Learning rate	{0.1, 0.2, ..., 0.4, 0.5}
d_{rate}	Determinism rate for solution construction	{0.0, 0.1, ..., 0.8, 0.9}
ρ^{neg}	Negative learning rate	{0.1, 0.2, ..., 0.4, 0.5}
γ	Reduction rate for negative pheromone values	{0.1, 0.2, ..., 0.8, 0.9}
p_{exp_a}	Proportion of explorer ants	{0.1, 0.2, ..., 0.4, 0.5}
α	Exponent for the pheromone values	{0.01, ..., 0.99}
t^{sub}	Maximum computation time (seconds) for sub-instance solving	{1, 2, ..., 9, 10}
n_a^{sub}	Number of solution constructions in the inner application of the baseline ACO algorithm (option 2 for solving sub-instances)	{3, 5, 10, 20}
ρ^{sub}	Learning rate in the inner application of the baseline ACO algorithm (option 2 for solving sub-instances)	{0.1, 0.2, ..., 0.4, 0.5}
$d_{\text{rate}}^{\text{sub}}$	Determinism rate for solution construction in the inner application of the baseline ACO algorithm (option 2 for solving sub-instances)	{0.0, 0.1, ..., 0.8, 0.9}

Table 3.2 Summary of the parameters that arise in each algorithm.

Parameter	Algorithms										
	Aco	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-SAP	Aco-EA	Aco-PAP	Aco ^{2o}
n_a	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ρ	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
d_{rate}	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ρ^{neg}		✓	✓		✓	✓				✓	✓
γ								✓	✓		
p_{exp_a}									✓		
α											✓
t_{sub}		✓	✓	✓	✓	✓	✓				
n_a^{sub}					✓	✓	✓				
ρ^{sub}					✓	✓	✓				
d_{rate}^{sub}					✓	✓	✓				

3.7 EXPERIMENTAL EVALUATION

The experiments were conducted on a cluster of machines with Intel[®] Xeon[®] CPU 5670 CPUs with 12 cores (2.933 GHz) and at least 32 GB RAM. For solving the sub-instances in Aco_{neg}⁺, Aco_{neg} and Aco⁺ we used CPLEX 12.10 in one-threaded mode. For our experiments we used a benchmark set of 90 problem instances with 500 items from the OR-Library (<http://people.brunel.ac.uk/~mastjjb/jeb/info.html>, accessed on 20 January 2021). This set consists of 30 instances with 5, 10, and 30 resources. Moreover, each of these three subsets contains 10 instances with resource tightness 0.25, 0.5, and 0.75. Roughly, the higher the value of the resource tightness, the more items can be placed in the knapsack. These 90 problem instances are generally known to be the most difficult ones available in the literature for heuristic solvers.

3.7.1 Algorithm Tuning

The scientific parameter tuning tool irace [123] was used for the purpose of parameter tuning. In particular we produced for each of the 11 algorithms (resp., algorithm versions) exactly one parameter value set. In the context of tuning the algorithms for the MDKP, we randomly selected one of the 10 problem instances for each combination of “the number of resources” (5, 10, 30) and the instance tightness (0.25, 0.5, 0.75). Consequently, nine problem instances were used for tuning in the case of the MDKP. Remember that the parameter value domains considered for tuning are provided in Table 3.1. The parameter values that were

determined by `irace` for the 11 algorithms are provided in Table 3.3.

Table 3.3 Parameter values for all algorithms for solving the MDKP

Parameter	Algorithms										
	ACO	ACO ⁺ _{neg}	ACO _{neg}	ACO ⁺	ACO-ACO ⁺ _{neg}	ACO-ACO _{neg}	ACO-ACO ⁺	ACO-SAP	ACO-EA	ACO-PAP	ACO ^{2o}
n_a	20	10	20	20	10	10	20	20	20	3	10
ρ	0.3	0.4	0.1	0.4	0.1	0.1	0.3	0.1	0.2	0.1	0.3
d_{rate}	0.7	0.1	0.7	0.4	0.6	0.7	0.8	0.8	0.8	0.8	0.9
ρ^{neg}	--	0.2	0.5	--	0.4	0.5	--	--	--	0.1	0.2
γ	--	--	--	--	--	--	--	0.9	0.7	--	--
p_{exp_a}	--	--	--	--	--	--	--	--	0.3	--	--
α	--	--	--	--	--	--	--	--	--	--	0.95
t^{sub}	--	7	3	5	3	9	3	--	--	--	--
n_a^{sub}	--	--	--	--	5	10	10	--	--	--	--
ρ^{sub}	--	--	--	--	0.3	0.2	0.4	--	--	--	--
d_{rate}^{sub}	--	--	--	--	0.7	0.7	0.7	--	--	--	--

3.7.2 Results

Using the previously determined parameter values, each of the 11 considered algorithms was applied 100 times to each of the 90 MDKP instances. This was done with a time limit of 500 s per run. Note that, in this way, the same computational resources were given to all 11 algorithms. The choice of 100 runs per instance in the case of the MDKP was done in order to produce results that are comparable to the best existing approaches from the literature, which were also applied 100 times to each problem instance. The numerical results of these experiments are presented in Tables 3.5, 3.6, and 3.7 concerning the values of the best solutions found, Tables 3.8, 3.9, and 3.10 concerning the average of 100 runs, and Tables 3.11, 3.12, and 3.13 concerning the average computation times.

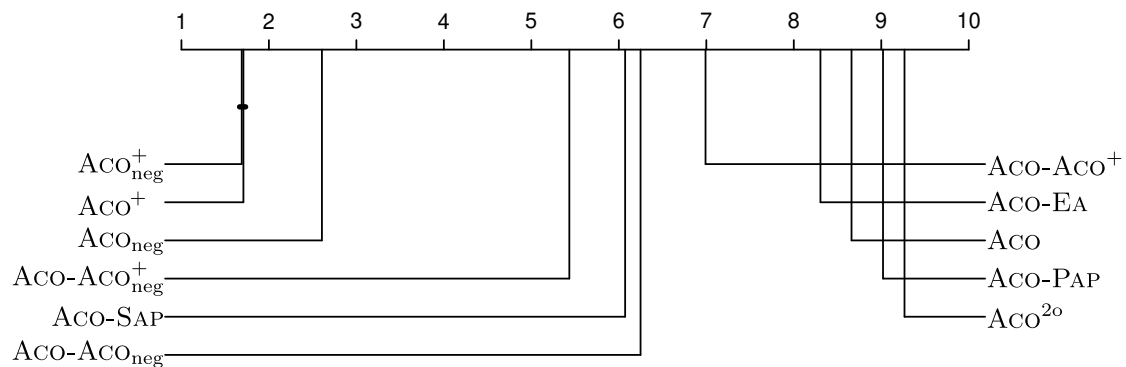
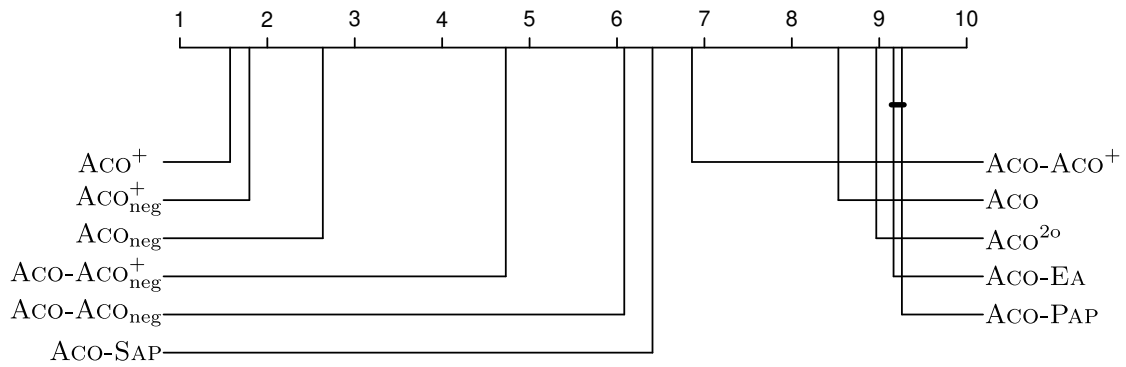


Fig. 3.2 Critical difference plot concerning all MDKP instances

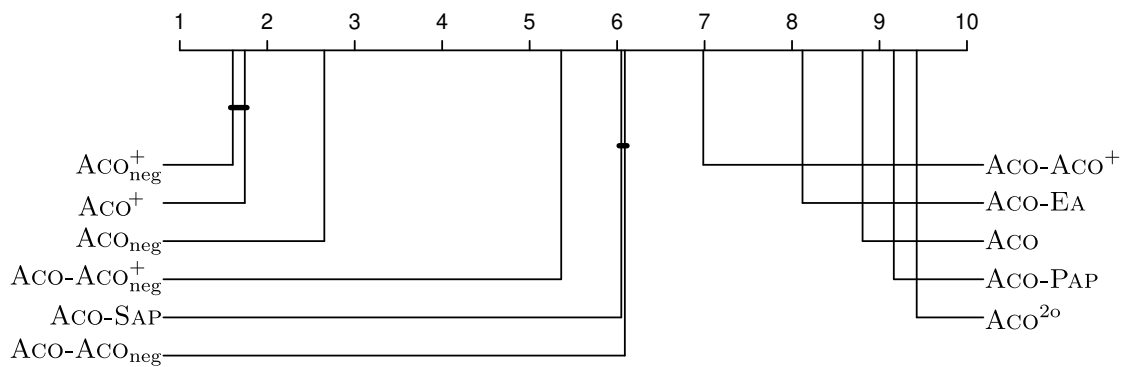
In addition, we present a comparative analysis of the 11 algorithms in terms of *critical difference* (CD) plots [124] and so-called *heatmaps*. In order to produce the average ranks of all algorithms—both for the whole set of problem instances as well as for instance subsets—the Friedman test was applied for the purpose of comparing the 11 approaches simultaneously. In this way we also obtained the rejection of the hypothesis that the 11 techniques perform equally. Subsequently, all pairwise algorithm comparisons were performed using the Nemenyi post-hoc test [125]. The obtained results are shown graphically (CD plots and heatmaps). The CD plots show the average algorithm ranks (horizontal axis) with respect to the considered (sub-)set of instances. In those cases in which the performances of two algorithms are below the critical difference threshold—based on a significance level of 0.05—the two algorithms are considered as statistically equivalent. This is indicated by bold horizontal bars joining the markers of the respective algorithm variants.

Figure 3.2 shows the CD plot for the whole set of 90 MDKP instances, while Fig. 3.3 and Fig. 3.4 present more fine-grained results concerning instances with different numbers of resources and with a varying instance tightness. The heatmaps in Figure 3.5 complement this more fine-grained presentation of the results. The 11 algorithms are distributed into three heatmap graphics. Each heatmap (out of 11 heatmaps in total) has three rows: one for each number of resources (5, 10, 30). Moreover, each heatmap has three columns: one for each considered instance tightness (0.25, 0.5, 0.75). Interestingly, from a global point of view (Fig. 3.2) the relative difference between the algorithm performances shows that our negative learning variants using option 1 perform best. $\text{Aco}_{\text{neg}}^+$ has a slight advantage over Aco^+ , which is not statistically significant.

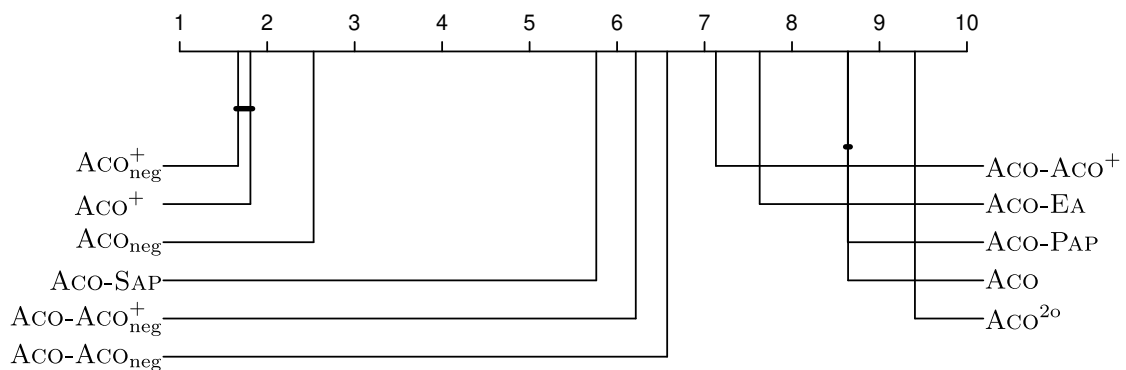
When studying the results in a more fine-grained way, the following observations can be made. The negative learning component of our algorithm proposal seems to gain importance with a growing number of resources. This can especially be observed for algorithm variants $\text{Aco}_{\text{neg}}^+$, $\text{Aco-Aco}_{\text{neg}}^+$ and $\text{Aco-Aco}_{\text{neg}}$. However, there is an interesting difference between $\text{Aco}_{\text{neg}}^+$ and $\text{Aco-Aco}_{\text{neg}}^+$: while $\text{Aco}_{\text{neg}}^+$ improves with an increasing instance tightness, the opposite is the case for $\text{Aco-Aco}_{\text{neg}}^+$. The relative performance of Aco-SAP , the best one of the negative learning variants chosen from the literature, is contrary to the relative performance of $\text{Aco-Aco}_{\text{neg}}^+$. In other words, the relative performance of Aco-SAP improves with a decreasing number of resources and with an increasing instance tightness.



(a) Instances with density 0.25

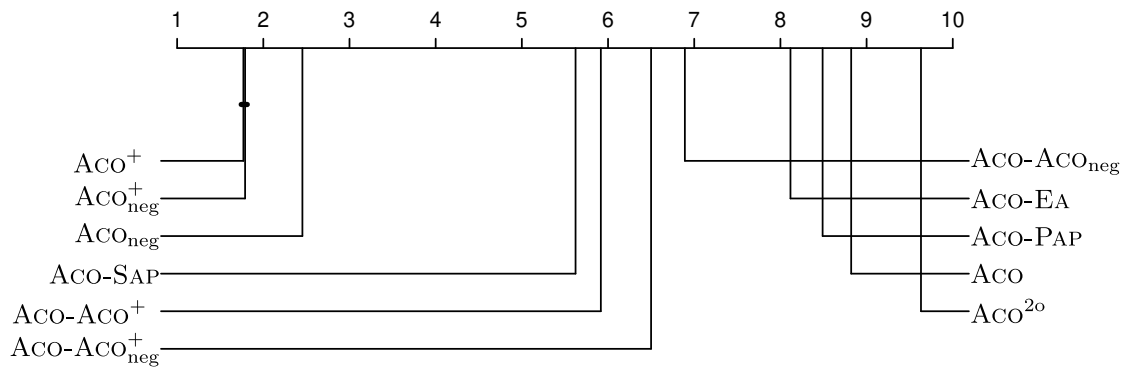


(b) Instances with density 0.5

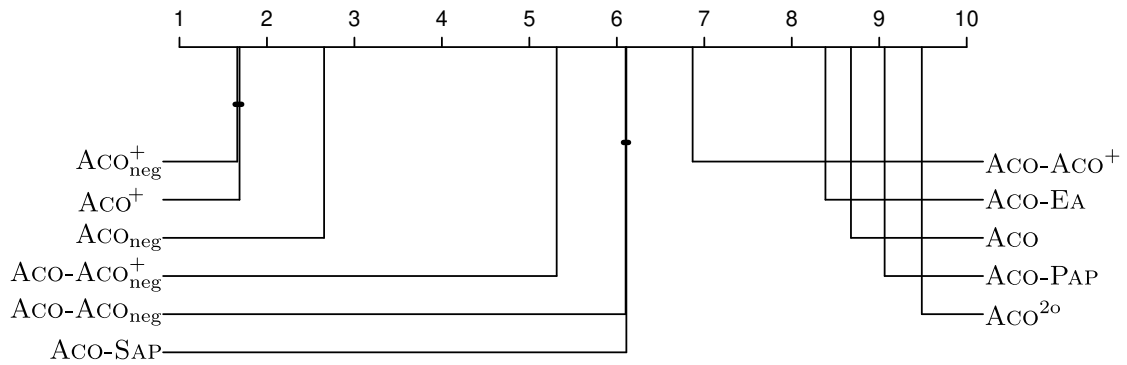


(c) Instances with density 0.75

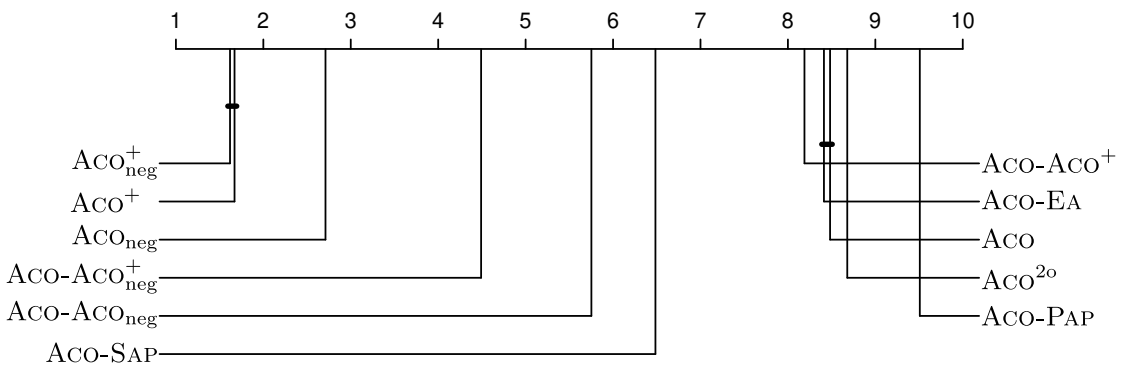
Fig. 3.3 Critical difference plots for MDKP instance groups based on their densities



(a) Instances with 5 resources



(b) Instances with 10 resources



(c) Instances with 30 resources

Fig. 3.4 Critical difference plots for MDKP instance groups based on their number of resources

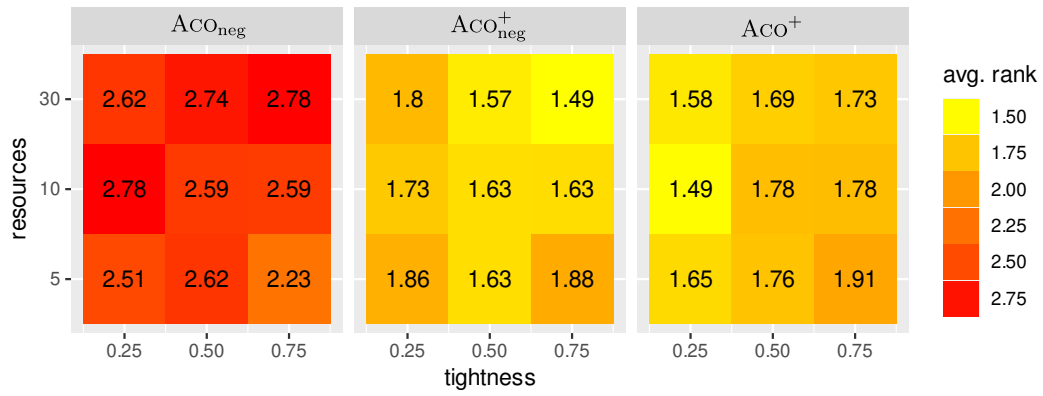
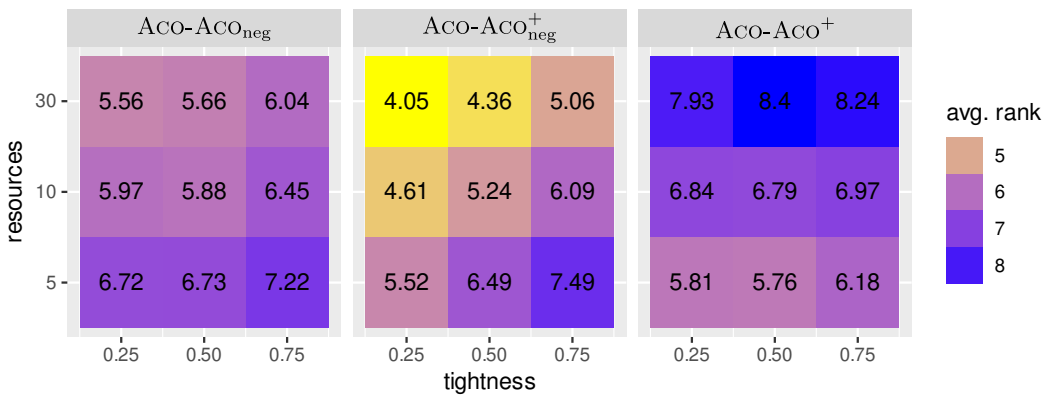
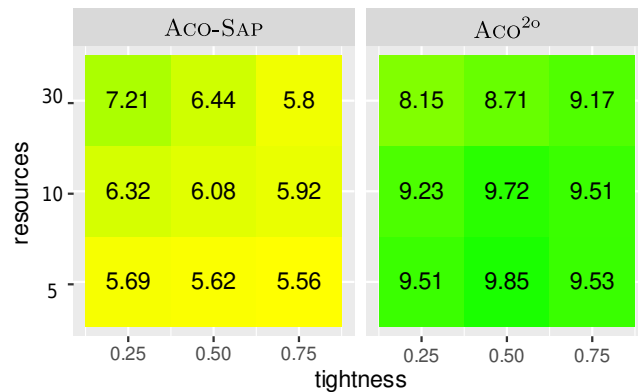
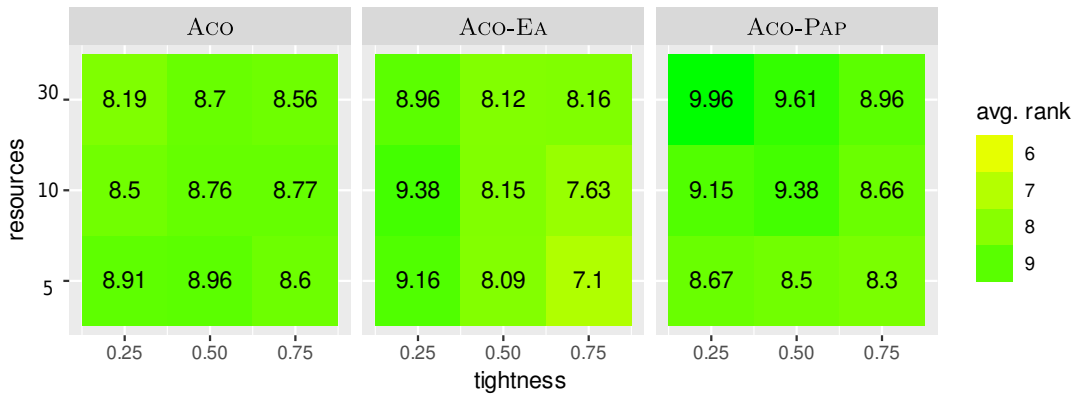
(a) Algorithms ACO_{neg}^+ , ACO_{neg} , and ACO^+ (b) Algorithms $ACO-ACO_{neg}^+$, $ACO-ACO_{neg}$, and $ACO-ACO^+$ (c) Algorithms ACO , $ACO-EA$, $ACO-PAP$, $ACO-SAP$, and ACO^{2o}

Fig. 3.5 Heatmaps concerning the results for the MDKP

3.7.3 Search Trajectory Network Analysis

We provide five plots of *Search Trajectory Networks* (STNs) [126] to compare the detail of algorithms' behavior during their search process. We made these STN plots by using a method proposed in the work of Ochoa et al. [127] and the corresponding *R scripts* provided at <https://github.com/gabro8a/STNs.git>. We obtained the data for these STN plots by running each of the 11 algorithms on problem instance `cb_5_500.0` ten times with the same parameter settings as used for the final experimental evaluation. We applied 80% *search space partitioning* to all the STN plots in this chapter.

An STN is a directed graph $G(N, E)$ with node set N representing *search states* (solutions and objective values) and edge set E representing the transitions between a sequence of search states. Figure 3.6 provides an example of an STN plot for three algorithms. The different color of the edges show the trajectories of the algorithm runs in the search space. For example, blue edges belong to the trajectories of Aco_{neg} , while green ones belong to the trajectories of $\text{Aco}_{\text{neg}}^+$. Each trajectory starts with a node which is a yellow rectangle node and ends with either a red circle (if the nodes corresponds to the best solution found) or a black triangle (otherwise). A grey-colored node represents a location in the

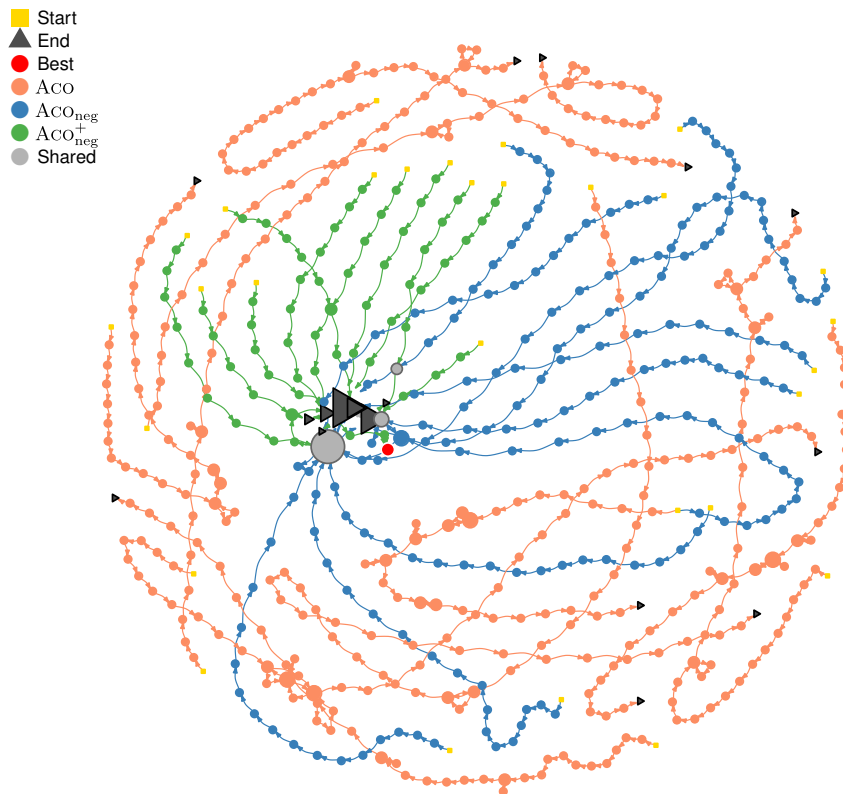


Fig. 3.6 Search trajectory network of Aco , Aco_{neg} , and $\text{Aco}_{\text{neg}}^+$ applied to problem instance `cb_5_500.0`

search space through which more than one algorithm has passed. Moreover, the bigger a node the more algorithm trajectories have passed through this node. STN visualization often helps to understand numerical results.

Figure 3.6 shows the STN plot for the baseline Aco algorithm and the negative learning variants, Aco_{neg} and Aco_{neg}^+ . The STN plot clearly shows the different characteristics between the two negative learning Aco variants and the baseline Aco algorithm. Despite of starting their search in diverse locations, the trajectories of Aco_{neg} and Aco_{neg}^+ converge to a common area where we can see several overlaps in their search trajectories. This behavior is not present in the trajectories of the baseline Aco algorithm. Its trajectories start at diverse locations and end also in diverse locations. This could mean that Aco is not attracted by any specific area of the search space. We can also observe in the STN plot that the trajectories of Aco_{neg} are longer than the ones of Aco_{neg}^+ . Several trajectories of the two negative learning variants find common final results, identified by several large black triangles. However, the best result is found only in one trajectory of Aco_{neg}^+ .

Figure 3.7 shows the STN plot for algorithms Aco^+ , Aco_{neg} , and Aco_{neg}^+ . Unlike the finding in Fig. 3.6, there is no significant characteristic difference between the three variants in terms of the direction of their trajectories. All trajectories in the plot converge to the same area of attraction. However, we can observe a notable contrast in their trajectory lengths. The variant Aco_{neg} has the longest average trajectory length, followed by Aco_{neg}^+ and Aco^+ .

Figure 3.8 presents the comparison of our best performing negative learning Aco variant— Aco_{neg}^+ —and the other two from our proposal— $Aco-Aco_{neg}$ and $Aco-Aco_{neg}^+$ —that use *MMAS* as their additional algorithmic component. The plot shows that trajectories of these two variants do not converge to a common area like the ones of Aco_{neg}^+ . Interestingly, this trend is also present in Fig. 3.9, where we plotted the STN composed of the trajectories of Aco_{neg}^+ , $Aco-EA$, and $Aco-SAP$, and in Fig. 3.10, where we plotted the STN composed of the trajectories of Aco_{neg}^+ , $Aco-PAP$, and Aco^{2o} . None of the trajectories from these competing negative learning Aco variants show the same behavior displayed by the ones of Aco_{neg}^+ . Furthermore, Aco_{neg}^+ finds the best result in this problem instance, not found by any of the remaining variants. These findings suggest that Aco_{neg}^+ has very different characteristics when compared to the baseline Aco and other negative learning Aco variants. Combined with the fact that our approach generated the best result, we believe that our negative learning proposal significantly improves over the baseline Aco, and it is superior to the existing negative learning Aco approaches.

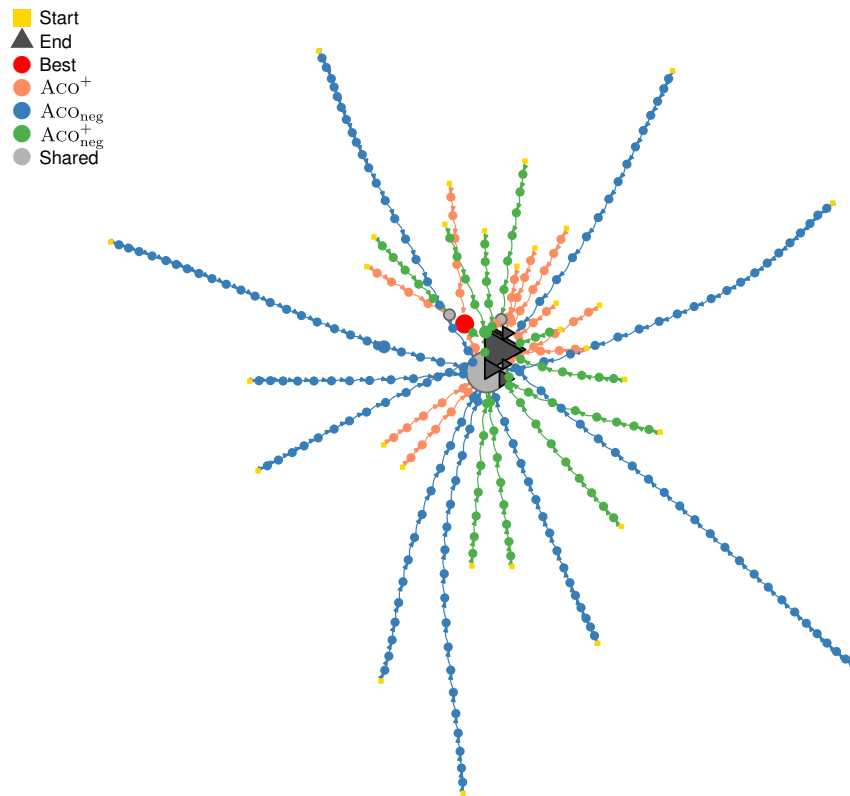


Fig. 3.7 Search trajectory network of Aco^+ , Aco_{neg} , and Aco_{neg}^+ applied to problem instance $cb_5_500.0$

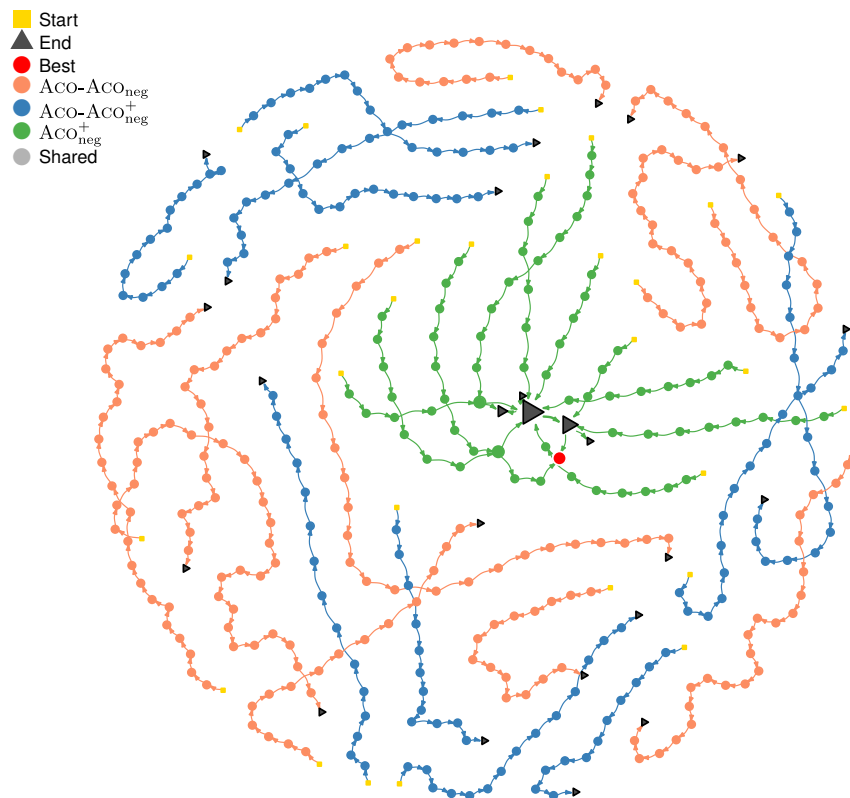


Fig. 3.8 Search trajectory network of $Aco-Aco_{neg}$, $Aco-Aco_{neg}^+$, and Aco_{neg}^+ applied to problem instance $cb_5_500.0$

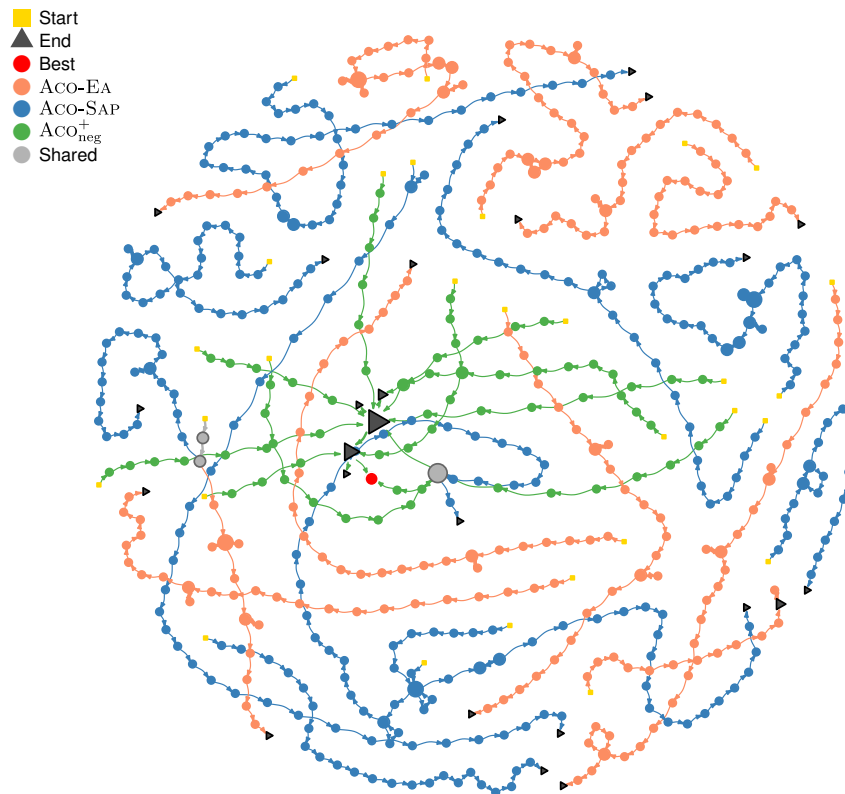


Fig. 3.9 Search trajectory network of ACO-EA, ACO-SAP, and ACO_{neg}⁺ applied to problem instance cb_5_500.0

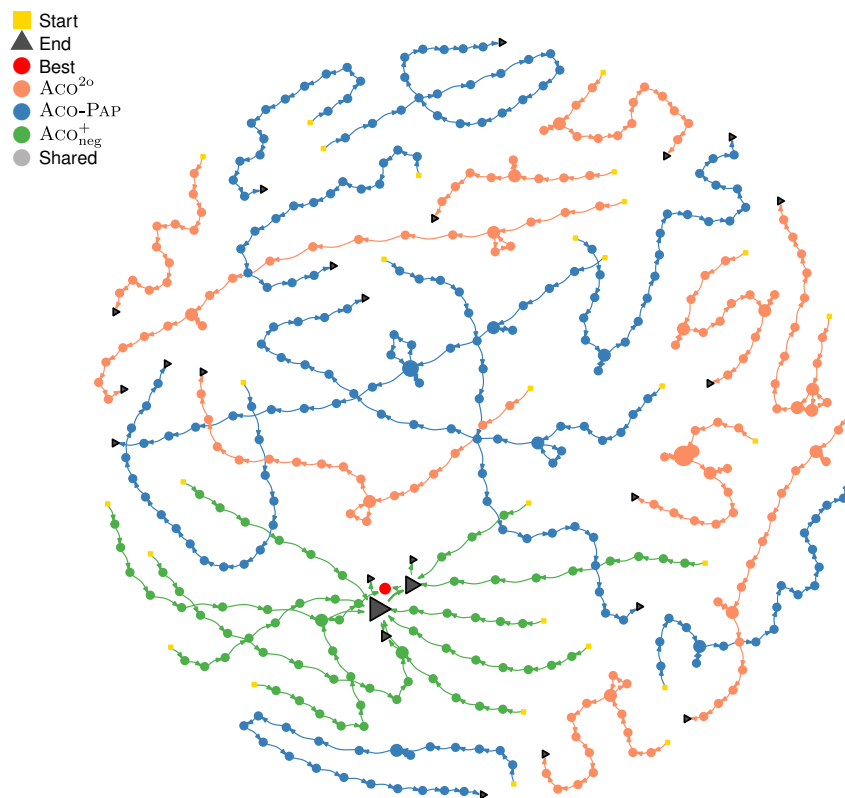


Fig. 3.10 Search trajectory network of ACO^{2o}, ACO-PAP, and ACO_{neg}⁺ applied to problem instance cb_5_500.0

3.7.4 Comparison to the State-of-the-Art

Even though the objective of this study is not necessarily to outperform current state-of-the-art algorithms for the chosen problems, we are certainly interested to know how our globally best algorithm ($\text{Aco}_{\text{neg}}^+$) performs in comparison to the state-of-the-art. In the context of the MDKP, we compare $\text{Aco}_{\text{neg}}^+$ to the current state-of-the-art algorithms: a sophisticated particle swarm optimization algorithm (DQPSO) from [103], published in 2020, and a powerful evolutionary algorithm (TPTEA) from [102], published in 2018. As these two algorithms—in their original papers—were applied to the 90 benchmark problems used in this work, it was not required to conduct additional experiments with $\text{Aco}_{\text{neg}}^+$.

The detailed comparisons between our negative learning Aco algorithms to these state-of-the-art approaches are provided in Tables 3.14, 3.15, and 3.16. Additionally, a summarized comparison of our globally best algorithm— $\text{Aco}_{\text{neg}}^+$ —with the state-of-the-art approaches is provided in Table 3.4. Each row contains average results for the 10 problem instances for each combination of the number of resources (5, 10, 30) and the instance tightness (0.25, 0.5, 0.75). In particular, we show averages concerning the best solutions found (table columns 3–5), the average solution quality obtained (table columns 6–8), and the average computation times required (table columns 9–11). We were surprised to see that $\text{Aco}_{\text{neg}}^+$ can actually compete with current state-of-the-art algorithms. The state-of-the-art results were even improved by $\text{Aco}_{\text{neg}}^+$ in some cases, especially for what concerns medium instance tightness for 5 and 10 resources, and low instance tightness for 30 resources. Moreover, the computation time of $\text{Aco}_{\text{neg}}^+$ is much lower than that of TPTEA, and comparable to the one required by DQPSO.

Table 3.4 Summarized comparison between $\text{Aco}_{\text{neg}}^+$ and the state-of-the-art algorithms for the MDKP

# Resources	Tightness	Best			Average			Average Time		
		TPTEA	DQPSO	$\text{Aco}_{\text{neg}}^+$	TPTEA	DQPSO	$\text{Aco}_{\text{neg}}^+$	TPTEA	DQPSO	$\text{Aco}_{\text{neg}}^+$
5	0.25	120629.2	120627.7	120628.6	120612.70	120619.81	120611.94	3228.31	117.53	208.31
	0.5	219511.6	219511.9	219512.7	219505.29	219505.79	219507.46	2673.01	79.51	161.87
	0.75	302363.4	302362.8	302363.0	302359.76	302358.98	302356.40	2129.25	66.05	141.21
10	0.25	118602.3	118613.2	118613.5	118548.87	118574.88	118574.00	3639.40	125.70	232.00
	0.5	217318.5	217318.5	217321.9	217281.33	217282.37	217283.24	3811.47	141.36	184.32
	0.75	302601.4	302593.1	302590.6	302583.25	302574.51	302568.88	2950.25	93.70	174.51
30	0.25	115571.0	115518.0	115605.3	115494.70	115421.82	115505.89	3943.07	476.50	231.05
	0.5	216266.2	216195.3	216236.2	216200.55	216130.38	216186.44	3542.84	407.97	220.29
	0.75	302445.1	302413.8	302419.8	302414.08	302353.54	302374.68	3451.25	433.77	216.52

3.8 CONCLUSIONS

Most learning-based metaheuristics, such as ant colony optimization, particle swarm optimization, and evolutionary algorithms, are based on learning from positive examples, or *positive learning*. On the other hand, nature demonstrates that learning from negative examples, or *negative learning*, can be quite valuable. In fact, during the last two decades, various attempts have been made to develop a mechanism to include negative learning into Aco. However, only a few of the works demonstrated that the proposed mechanism was indeed useful. This research aimed to develop a new negative learning mechanism for Aco and demonstrate its effectiveness. Our mechanism's primary concept is that negative feedback should not be extracted from the main Aco algorithm. A separate algorithmic component should instead generate it. After developing our new negative learning framework, we investigated two algorithmic options for generating negative information: (1) using the mathematical programming solver CPLEX, and (2) using the baseline Aco algorithm, but with additional, independent runs for solving sub-instances of the original problem instances.

The MDKP was used as a test case for all the algorithm versions. In order to compare our algorithm proposal with current techniques, four negative learning mechanisms from the literature were built on top of the chosen baseline Aco algorithm. The findings have indicated, first and foremost, that the suggested negative learning mechanism is superior to current techniques from the literature, particularly when employing CPLEX for providing negative feedback information. Second, we have demonstrated that, while negative learning is not always useful for all problem instances, it may be quite effective for subsets of problem instances that share specific features. The proposed negative learning mechanism proved notably effective for solving MDKP instances for problem cases with many resources. From a broader perspective, it was also demonstrated that adding negative learning is generally not harmful since the globally best-performing algorithm version employs negative learning. Finally, we demonstrated that our globally best-performing algorithm variation can compete with state-of-the-art heuristic solvers for the MDKP.

Table 3.5 Best results of all algorithms tested on OR-LIB instances with 5 resources

Instance	Tightness	Aco ⁺ _{neg}	Aco _{neg}	Aco ⁺	Aco	Aco-Aco ⁺ _{neg}	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5.500.0	0.25	120148	120148	120148	120074	120110	120081	120124	120033	120101	120064	120080
5.500.1	0.25	117879	117864	117879	117733	117823	117824	117771	117749	117834	117806	117735
5.500.2	0.25	121131	121125	121129	121026	121059	121060	121068	120992	121076	121047	121057
5.500.3	0.25	120804	120804	120804	120697	120752	120738	120755	120672	120747	120685	120645
5.500.4	0.25	122319	122319	122319	122217	122287	122275	122280	122219	122319	122245	122244
5.500.5	0.25	122024	122024	122024	121925	121991	121984	121984	121982	121984	121955	121929
5.500.6	0.25	119127	119117	119127	119056	119076	119080	119094	119020	119072	119052	119016
5.500.7	0.25	120568	120568	120568	120420	120508	120454	120492	120445	120489	120450	120438
5.500.8	0.25	121575	121575	121575	121422	121475	121445	121463	121352	121414	121368	121265
5.500.9	0.25	120711	120699	120717	120565	120623	120631	120627	120581	120634	120573	120517
5.500.10	0.50	218428	218428	218428	218282	218305	218282	218306	218306	218352	218263	218224
5.500.11	0.50	221202	221191	221202	220975	221043	221053	221065	220944	221018	220878	220854
5.500.12	0.50	217542	217534	217542	217441	217411	217444	217446	217430	217454	217425	217355
5.500.13	0.50	223560	223560	223560	223491	223511	223514	223530	223527	223508	223472	223476
5.500.14	0.50	218966	218966	218966	218897	218943	218913	218943	218906	218907	218875	218863
5.500.15	0.50	220530	220527	220530	220425	220418	220432	220476	220456	220484	220464	220451
5.500.16	0.50	219989	219989	219989	219884	219905	219917	219987	219943	219973	219939	219876
5.500.17	0.50	218215	218215	218215	218134	218121	218123	218165	218125	218118	218116	218065
5.500.18	0.50	216976	216976	216976	216878	216909	216909	216946	216900	216939	216938	216896
5.500.19	0.50	219719	219717	219717	219598	219647	219664	219656	219656	219687	219610	219616
5.500.20	0.75	295828	295828	295828	295702	295702	295749	295790	295735	295771	295752	295713
5.500.21	0.75	308086	308086	308086	308026	307998	308011	308027	307991	308039	307970	307979
5.500.22	0.75	299796	299796	299796	299748	299732	299743	299788	299751	299751	299750	299742
5.500.23	0.75	306480	306480	306480	306426	306436	306469	306469	306466	306469	306443	306413
5.500.24	0.75	300342	300342	300342	300280	300288	300265	300288	300247	300289	300284	300310
5.500.25	0.75	302571	302571	302571	302542	302529	302560	302535	302533	302560	302513	302495
5.500.26	0.75	301339	301329	301339	301286	301325	301285	301284	301284	301301	301275	301272
5.500.27	0.75	306454	306454	306454	306353	306367	306415	306415	306415	306417	306415	306333
5.500.28	0.75	302828	302818	302828	302747	302749	302759	302764	302814	302823	302777	302737
5.500.29	0.75	299906	299906	299906	299825	299831	299801	299842	299837	299886	299822	299813

Table 3.6 Best results of all algorithms tested on OR-LIB instances with 10 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10.500.0	0.25	117809	117779	117809	117507	117681	117502	117507	117403	117502	117525	117513
10.500.1	0.25	119188	119182	119229	119000	119058	119081	119075	118964	119066	118974	118924
10.500.2	0.25	119211	119194	119211	118942	118962	118948	118983	118859	118961	118860	118881
10.500.3	0.25	118813	118813	118813	118602	118692	118697	118697	118601	118737	118621	118550
10.500.4	0.25	116509	116471	116509	116209	116265	116224	116185	116057	116176	115995	115988
10.500.5	0.25	119504	119461	119466	119278	119338	119305	119338	119174	119308	119214	119157
10.500.6	0.25	119790	119777	119827	119508	119722	119660	119653	119534	119691	119691	119691
10.500.7	0.25	118320	118277	118312	117892	118117	118011	117918	117863	118006	117844	117873
10.500.8	0.25	117781	117776	117779	117620	117607	117578	117551	117496	117586	117472	117435
10.500.9	0.25	119210	119192	119210	118885	119064	118880	118915	118864	118908	118863	118773
10.500.10	0.50	217377	217318	217377	217108	217177	217201	217128	217155	217212	217059	217009
10.500.11	0.50	219077	219041	219063	218877	218858	218825	218649	218703	218759	218583	218554
10.500.12	0.50	217793	217792	217797	217540	217620	217597	217611	217523	217636	217462	217448
10.500.13	0.50	216868	216868	216868	216635	216663	216589	216719	216601	216685	216557	216556
10.500.14	0.50	213859	213831	213859	213448	213610	213628	213524	213608	213631	213469	213491
10.500.15	0.50	215073	215071	215062	214866	214932	214957	214896	214870	214936	214853	214810
10.500.16	0.50	217931	217899	217931	217644	217723	217727	217783	217683	217779	217674	217600
10.500.17	0.50	219984	219949	219984	219678	219789	219705	219662	219689	219760	219890	219588
10.500.18	0.50	214375	214346	214375	213897	214189	214115	214044	213924	214013	214032	213953
10.500.19	0.50	220882	220846	220886	220619	220706	220664	220734	220606	220739	220563	220552
10.500.20	0.75	304359	304344	304353	304156	304269	304256	304170	304187	304254	304161	304075
10.500.21	0.75	302371	302331	302379	302216	302293	302220	302303	302240	302296	302239	302148
10.500.22	0.75	302408	302408	302416	302227	302228	302226	302237	302247	302323	302177	302202
10.500.23	0.75	300747	300743	300743	300590	300637	300653	300667	300644	300668	300627	300569
10.500.24	0.75	304374	304340	304374	304266	304323	304266	304275	304264	304323	304340	304288
10.500.25	0.75	301796	301751	301781	301448	301575	301508	301361	301260	301448	301260	301127
10.500.26	0.75	304949	304949	304952	304779	304851	304866	304901	304897	304892	304882	304892
10.500.27	0.75	296456	296456	296456	296271	296278	296316	296330	296300	296334	296263	296162
10.500.28	0.75	301357	301313	301353	301149	301215	301236	301304	301179	301287	301167	301103
10.500.29	0.75	307089	307072	307072	306867	306951	306981	306897	306885	306931	306856	306836

Table 3.7 Best results of all algorithms tested on OR-LIB instances with 30 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
30.500.0	0.25	116014	115915	115947	115031	115478	115124	114840	114828	114919	114634	114998
30.500.1	0.25	114734	114718	114749	114084	114298	114253	114032	114004	114086	113952	114118
30.500.2	0.25	116685	116669	116705	115727	116009	115763	115682	115680	115643	115426	115839
30.500.3	0.25	115310	115228	115310	114506	114891	114554	114390	114422	114549	114310	114385
30.500.4	0.25	116516	116448	116525	115242	115811	115300	115154	115139	115154	114974	115344
30.500.5	0.25	115734	115643	115741	115102	115524	115338	114994	115085	115253	114908	115128
30.500.6	0.25	114181	114058	114181	113008	113655	113324	113137	113112	113189	113020	113068
30.500.7	0.25	114344	114344	114344	113096	113399	113274	113012	113062	113129	113083	113214
30.500.8	0.25	115419	115419	115419	114389	114837	114523	114138	114195	114369	114212	114448
30.500.9	0.25	117116	117020	117116	116195	116465	116151	116000	116196	116091	115862	116126
30.500.10	0.50	218081	218033	218068	217319	217688	217546	217191	217308	217426	217159	217330
30.500.11	0.50	214626	214626	214626	213579	214099	213698	213336	213517	213646	213413	213371
30.500.12	0.50	215914	215903	215918	215398	215598	215504	215567	215518	215520	215418	215575
30.500.13	0.50	217863	217827	217862	216839	217325	217123	216729	216797	216976	216795	216839
30.500.14	0.50	215640	215566	215635	214751	215081	214804	214560	214759	214803	214790	214590
30.500.15	0.50	215867	215853	215875	214777	215262	215110	214776	214863	214954	214729	214885
30.500.16	0.50	215883	215798	215883	215326	215610	215555	215245	215308	215414	215385	215232
30.500.17	0.50	216463	216419	216542	215692	216055	215821	215887	215931	215913	215546	215572
30.500.18	0.50	217333	217312	217333	216653	217041	216749	216688	216652	216738	216601	216658
30.500.19	0.50	214692	214657	214691	213796	214096	213934	213830	213909	213935	213719	213897
30.500.20	0.75	301667	301643	301667	301177	301326	301271	301291	301270	301279	301225	301192
30.500.21	0.75	300055	299996	300045	299578	299621	299639	299588	299523	299683	299575	299438
30.500.22	0.75	305080	305018	305069	304753	304754	304797	304665	304673	304810	304666	304631
30.500.23	0.75	302004	302001	302001	301441	301664	301575	301369	301458	301497	301432	301417
30.500.24	0.75	304427	304413	304416	304046	304055	304049	303907	303916	303956	303866	303944
30.500.25	0.75	296960	296909	296959	296394	296512	296442	296353	296462	296544	296427	296468
30.500.26	0.75	303335	303304	303322	302772	302958	302819	302819	302834	302924	302640	302934
30.500.27	0.75	306956	306941	306999	306511	306650	306481	306455	306363	306532	306337	306374
30.500.28	0.75	303178	303151	303178	302543	302676	302551	302250	302290	302521	302255	302341
30.500.29	0.75	300536	300512	300532	300077	300166	300220	300068	300072	300156	300196	300095

Table 3.8 Average results of all algorithms tested on OR-LIB instances with 5 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5.500.0	0.25	120120.0	120107.3	120122.5	119945.6	119998.3	119967.9	120003.1	119902.9	120015.5	119940.3	119933.7
5.500.1	0.25	117854.2	117833.3	117862.6	117614.5	117726.1	117688.4	117691.0	117562.5	117698.3	117591.5	117551.9
5.500.2	0.25	121113.0	121112.0	121114.0	120883.1	120982.2	120954.7	120970.7	120886.6	120987.1	120890.9	120864.1
5.500.3	0.25	120784.2	120782.7	120785.8	120534.8	120662.5	120627.1	120622.6	120490.4	120633.4	120557.2	120491.1
5.500.4	0.25	122319.0	122318.3	122319.0	122090.7	122191.6	122167.6	122185.9	122086.4	122196.1	122112.5	122099.8
5.500.5	0.25	122008.4	122000.8	122012.0	121798.6	121905.7	121876.5	121888.0	121828.7	121898.4	121819.8	121788.0
5.500.6	0.25	119103.3	119086.2	119106.5	118894.1	118980.0	118947.3	118986.3	118902.6	118998.9	118929.4	118902.4
5.500.7	0.25	120567.1	120560.3	120567.3	120278.4	120401.3	120375.6	120427.9	120317.2	120395.4	120309.8	120313.0
5.500.8	0.25	121563.2	121540.4	121566.0	121176.3	121350.8	121293.6	121315.3	121148.4	121309.2	121158.3	121030.0
5.500.9	0.25	120686.8	120673.7	120689.3	120417.8	120533.7	120513.3	120542.7	120373.7	120535.4	120398.2	120362.0
5.500.10	0.50	218425.8	218422.5	218426.4	218112.8	218198.2	218197.1	218209.9	218143.6	218226.8	218141.3	218080.8
5.500.11	0.50	221190.6	221188.3	221191.2	220763.0	220933.2	220901.7	220863.3	220722.7	220888.2	220710.3	220609.0
5.500.12	0.50	217533.3	217520.5	217532.8	217229.4	217331.9	217314.4	217336.7	217267.8	217348.4	217214.5	217167.6
5.500.13	0.50	223559.2	223558.3	223558.5	223354.2	223421.6	223420.6	223447.5	223415.6	223445.7	223384.3	223362.2
5.500.14	0.50	218966.0	218963.4	218966.0	218763.8	218808.4	218817.9	218833.9	218766.3	218828.3	218729.4	218695.8
5.500.15	0.50	220519.9	220501.9	220521.6	220287.0	220345.6	220339.1	220388.1	220322.8	220384.4	220336.7	220279.5
5.500.16	0.50	219988.0	219969.6	219987.3	219764.0	219815.4	219810.9	219847.5	219751.9	219841.4	219764.2	219681.7
5.500.17	0.50	218199.5	218173.6	218190.7	217934.1	218034.6	218020.2	218044.6	217974.9	218046.7	217968.2	217901.2
5.500.18	0.50	216976.0	216976.0	216976.0	216758.1	216817.4	216814.9	216854.7	216810.8	216864.4	216804.9	216780.6
5.500.19	0.50	219716.4	219701.3	219709.9	219429.1	219532.5	219534.9	219556.5	219476.6	219557.4	219446.6	219385.2
5.500.20	0.75	295828.0	295828.0	295828.0	295600.2	295645.2	295665.8	295693.1	295640.4	295685.0	295601.1	295577.6
5.500.21	0.75	308079.2	308076.8	308079.6	307879.9	307913.8	307912.2	307908.5	307890.9	307927.9	307860.3	307804.9
5.500.22	0.75	299795.0	299794.3	299795.8	299640.6	299645.0	299659.2	299704.0	299670.0	299691.4	299631.8	299629.0
5.500.23	0.75	306477.4	306475.0	306476.9	306317.1	306340.3	306351.7	306382.8	306362.8	306395.8	306333.3	306325.1
5.500.24	0.75	300342.0	300342.0	300342.0	300145.4	300189.0	300189.3	300202.9	300180.3	300220.8	300160.0	300146.0
5.500.25	0.75	302553.0	302549.5	302552.4	302395.4	302433.0	302436.0	302450.1	302437.5	302471.7	302410.5	302389.0
5.500.26	0.75	301330.0	301328.9	301329.1	301199.3	301212.3	301214.0	301221.2	301211.7	301232.5	301180.6	301164.0
5.500.27	0.75	306439.8	306435.0	306438.6	306255.0	306301.2	306302.8	306314.5	306303.9	306328.0	306290.0	306230.2
5.500.28	0.75	302814.8	302810.8	302814.7	302637.4	302664.0	302680.4	302705.2	302689.4	302724.0	302675.1	302644.4
5.500.29	0.75	299904.7	299904.6	299904.9	299735.3	299736.6	299733.7	299754.6	299749.0	299784.2	299742.6	299684.3

Table 3.9 Average results of all algorithms tested on OR-LIB instances with 10 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10.500.0	0.25	117750.7	117718.4	117769.3	117234.0	117471.6	117372.4	117281.8	117101.7	117308.8	117105.8	117159.5
10.500.1	0.25	119157.5	119136.4	119160.6	118713.1	118900.1	118864.1	118866.8	118758.9	118874.8	118767.0	118761.8
10.500.2	0.25	119167.7	119097.2	119180.8	118658.7	118873.4	118809.0	118766.5	118581.2	118785.1	118625.6	118596.4
10.500.3	0.25	118796.8	118769.1	118803.8	118368.5	118547.4	118510.8	118449.7	118293.5	118501.0	118345.0	118324.2
10.500.4	0.25	116459.0	116432.6	116461.4	115817.2	116109.3	116039.0	115979.5	115815.6	115998.4	115753.1	115799.3
10.500.5	0.25	119454.4	119438.9	119454.6	119057.8	119210.5	119150.2	119131.0	118946.8	119124.9	118980.0	118979.7
10.500.6	0.25	119762.8	119730.6	119772.1	119329.9	119544.6	119469.6	119451.9	119328.9	119488.9	119366.6	119364.8
10.500.7	0.25	118265.8	118237.0	118269.4	117650.9	117959.0	117834.8	117722.4	117586.5	117772.0	117583.8	117561.6
10.500.8	0.25	117750.9	117714.7	117742.6	117266.9	117478.7	117420.8	117346.0	117128.2	117370.1	117172.2	117189.2
10.500.9	0.25	119174.4	119140.6	119185.8	118623.8	118850.9	118698.1	118703.6	118578.2	118715.6	118522.0	118515.3
10.500.10	0.50	217320.4	217304.7	217321.1	216828.9	216994.8	216988.9	216993.4	216892.6	216975.9	216799.9	216770.6
10.500.11	0.50	219037.7	219018.9	219034.4	218465.8	218704.5	218625.4	218426.5	218335.6	218509.6	218224.4	218272.9
10.500.12	0.50	217766.7	217755.4	217765.9	217213.5	217445.1	217426.4	217418.8	217301.9	217398.9	217219.1	217179.3
10.500.13	0.50	216828.2	216801.2	216822.3	216391.8	216511.2	216429.0	216448.0	216388.9	216502.1	216346.7	216335.1
10.500.14	0.50	213829.0	213794.8	213829.6	213241.0	213454.1	213455.5	213373.5	213346.8	213454.3	213235.2	213210.7
10.500.15	0.50	215035.5	215001.5	215028.0	214592.7	214800.8	214786.1	214755.5	214705.8	214800.0	214631.9	214594.9
10.500.16	0.50	217888.8	217877.4	217886.5	217441.7	217608.4	217588.4	217569.2	217522.9	217597.0	217440.8	217441.1
10.500.17	0.50	219952.8	219931.2	219952.6	219389.0	219556.6	219526.7	219425.0	219361.6	219517.2	219241.0	219240.3
10.500.18	0.50	214326.8	214287.0	214310.9	213656.5	213981.2	213915.5	213843.6	213700.5	213833.2	213573.0	213571.7
10.500.19	0.50	220846.5	220823.4	220846.2	220335.6	220534.6	220521.7	220503.1	220425.4	220513.0	220382.4	220324.0
10.500.20	0.75	304344.2	304344.0	304343.2	303954.3	304089.0	304101.8	304013.2	304005.3	304094.0	303943.3	303853.9
10.500.21	0.75	302333.3	302317.7	302337.7	302012.3	302117.9	302114.5	302096.4	302094.1	302139.5	302054.1	302015.7
10.500.22	0.75	302399.9	302354.4	302391.3	302036.0	302110.2	302112.0	302132.1	302063.5	302122.8	301995.7	301988.0
10.500.23	0.75	300740.4	300705.8	300737.7	300400.1	300486.8	300498.3	300466.8	300458.9	300510.5	300438.1	300370.0
10.500.24	0.75	304345.9	304340.0	304344.6	304023.0	304132.3	304141.4	304120.4	304119.4	304163.2	304074.2	304060.6
10.500.25	0.75	301753.0	301712.9	301749.9	301180.9	301403.3	301290.8	301036.8	300997.3	301164.3	300951.6	300861.0
10.500.26	0.75	304949.0	304949.0	304949.0	304601.9	304722.6	304675.9	304722.3	304724.7	304764.1	304663.0	304679.3
10.500.27	0.75	296451.2	296428.0	296446.4	296021.6	296174.8	296102.2	296164.3	296079.0	296166.7	296067.7	295972.2
10.500.28	0.75	301315.1	301295.3	301307.8	300925.7	301064.0	301064.2	301018.3	301016.1	301081.4	300972.1	300912.4
10.500.29	0.75	307056.7	307005.0	307051.2	306674.8	306785.3	306789.5	306783.9	306724.2	306801.3	306656.2	306632.3

Table 3.10 Average results of all algorithms tested on OR-LIB instances with 30 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
30.500.0	0.25	115881.0	115860.7	115888.3	114568.5	115193.7	114849.3	114505.6	114379.1	114563.7	114247.9	114638.6
30.500.1	0.25	114693.2	114678.2	114696.1	113698.0	114109.9	113890.3	113773.1	113684.4	113799.5	113658.3	113737.1
30.500.2	0.25	116655.1	116624.9	116659.5	115185.4	115786.0	115512.1	115223.9	115132.3	115319.9	114948.3	115260.2
30.500.3	0.25	115201.3	115145.7	115213.2	113974.0	114553.6	114332.0	114027.1	113942.7	114145.4	113859.0	113914.0
30.500.4	0.25	116390.9	116368.2	116402.0	114781.4	115555.3	115053.9	114814.5	114740.5	114899.0	114542.3	114786.0
30.500.5	0.25	115643.2	115624.4	115653.9	114768.4	115168.8	114943.6	114737.8	114616.5	114805.3	114491.8	114709.2
30.500.6	0.25	114004.1	113956.6	114013.5	112646.6	113394.7	112982.1	112876.5	112699.5	112884.6	112555.7	112712.1
30.500.7	0.25	114220.4	114151.9	114214.7	112744.4	113178.9	112946.3	112770.7	112607.4	112815.3	112463.7	112683.5
30.500.8	0.25	115357.9	115328.4	115419.0	114024.2	114510.3	114192.1	113868.6	113834.4	113984.8	113701.5	113983.4
30.500.9	0.25	117011.9	116919.2	117018.7	115599.7	116210.1	115919.4	115687.3	115632.6	115774.4	115476.5	115626.2
30.500.10	0.50	218017.5	217956.2	218005.0	217005.4	217348.7	217156.2	216835.1	216909.6	217079.6	216795.0	216832.9
30.500.11	0.50	214605.0	214554.6	214593.6	213046.9	213675.0	213379.4	213047.5	213086.9	213275.4	212991.3	212997.7
30.500.12	0.50	215853.7	215814.0	215853.7	215042.5	215336.8	215294.4	215092.4	215137.4	215267.5	215031.0	215143.4
30.500.13	0.50	217827.0	217768.2	217809.9	216442.4	217034.8	216805.9	216297.7	216408.9	216650.3	216162.1	216399.6
30.500.14	0.50	215574.3	215535.4	215566.8	214334.8	214735.7	214549.4	214228.8	214333.9	214469.9	214132.4	214253.8
30.500.15	0.50	215804.4	215721.0	215810.9	214373.3	214971.1	214728.7	214385.3	214482.4	214608.0	214309.5	214462.8
30.500.16	0.50	215807.3	215764.3	215818.3	214893.1	215287.1	215175.8	214948.7	215023.9	215145.1	214902.1	214856.3
30.500.17	0.50	216428.6	216397.2	216427.1	215263.4	215770.9	215598.0	215420.3	215371.7	215526.5	215147.2	215275.0
30.500.18	0.50	217302.3	217214.4	217297.8	216168.5	216702.9	216459.9	216348.3	216259.9	216422.7	216073.0	216268.2
30.500.19	0.50	214644.4	214629.0	214637.7	213389.0	213824.2	213677.5	213605.2	213541.1	213634.7	213354.8	213448.1
30.500.20	0.75	301642.0	301609.1	301644.5	300870.9	301100.9	301062.8	300992.0	300968.9	301095.3	300942.0	300920.0
30.500.21	0.75	299990.7	299928.0	299976.2	299174.0	299438.5	299406.9	299324.9	299241.6	299414.9	299251.6	299126.3
30.500.22	0.75	305023.9	304985.9	305014.8	304337.0	304542.6	304520.9	304383.1	304454.0	304579.0	304359.1	304370.0
30.500.23	0.75	301927.4	301884.9	301920.4	301128.9	301418.5	301290.4	301131.5	301189.5	301324.4	301093.9	301112.4
30.500.24	0.75	304409.4	304369.9	304409.5	303657.2	303803.6	303766.9	303672.0	303583.0	303762.1	303548.5	303538.3
30.500.25	0.75	296922.9	296887.3	296912.0	296124.3	296319.0	296239.4	296095.3	296168.2	296287.7	296084.8	296097.0
30.500.26	0.75	303299.4	303252.0	303274.6	302442.7	302775.8	302642.8	302547.6	302542.9	302671.7	302393.3	302490.5
30.500.27	0.75	306916.4	306893.7	306917.9	306058.4	306384.3	306269.0	306142.5	306058.6	306255.4	306011.1	306009.9
30.500.28	0.75	303119.0	303085.4	303120.1	302141.0	302444.1	302253.3	301960.5	302009.7	302199.0	301965.5	301926.8
30.500.29	0.75	300495.8	300477.4	300496.2	299796.9	299992.2	299973.8	299814.9	299852.0	299969.8	299836.5	299752.5

Table 3.11 Average computation time of all algorithms tested on OR-LIB instances with 5 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5.500.0	0.25	241.5	213.8	227.3	286.1	272.1	340.2	289.8	240.3	256.4	267.5	274.1
5.500.1	0.25	233.2	195.9	202.8	261.8	316.3	329.7	278.4	261.7	271.8	231.9	309.7
5.500.2	0.25	182.3	194.6	180.8	252.2	265.2	335.3	292.9	258.4	223.7	274.0	283.6
5.500.3	0.25	244.0	210.0	171.5	305.0	276.7	354.1	272.0	256.0	262.6	260.2	287.3
5.500.4	0.25	99.9	108.5	69.0	279.6	303.1	335.7	250.7	277.7	236.6	279.2	285.0
5.500.5	0.25	240.1	185.7	178.2	277.7	265.5	346.1	280.7	244.7	256.1	235.2	270.5
5.500.6	0.25	222.5	257.4	224.9	292.6	276.8	336.2	281.0	260.0	259.0	242.5	256.8
5.500.7	0.25	159.3	202.2	126.4	296.0	306.3	362.3	278.0	276.6	238.7	264.9	293.0
5.500.8	0.25	237.3	243.7	202.1	302.0	302.7	348.5	293.4	282.9	248.7	275.8	326.0
5.500.9	0.25	223.0	176.4	236.7	303.0	299.7	362.7	287.9	281.4	261.6	268.3	331.5
5.500.10	0.50	212.1	163.3	207.4	306.9	320.9	327.3	300.9	288.3	257.0	267.1	323.8
5.500.11	0.50	147.2	148.3	191.2	306.9	344.9	333.3	306.2	258.7	275.9	276.4	334.4
5.500.12	0.50	244.6	246.1	224.5	311.9	323.3	333.2	324.6	303.5	274.5	283.0	308.1
5.500.13	0.50	128.9	84.0	89.3	278.1	287.3	335.3	294.3	267.6	259.0	254.9	304.1
5.500.14	0.50	40.2	158.9	6.9	259.3	317.9	306.6	265.9	277.5	244.5	281.7	315.1
5.500.15	0.50	241.1	164.2	228.6	308.0	309.5	341.8	299.1	244.8	243.7	243.7	286.3
5.500.16	0.50	183.4	184.7	181.3	297.5	331.1	342.9	289.2	273.9	246.3	307.2	313.5
5.500.17	0.50	237.3	230.3	186.7	332.6	328.9	352.4	290.0	265.1	275.1	271.6	297.8
5.500.18	0.50	26.6	34.9	24.0	323.0	301.6	337.0	279.3	276.9	270.8	255.1	306.0
5.500.19	0.50	157.3	187.4	208.5	314.7	327.3	333.5	294.1	295.5	275.5	318.2	318.1
5.500.20	0.75	7.1	19.3	5.1	310.2	309.2	307.9	271.3	281.8	262.2	270.6	302.3
5.500.21	0.75	155.2	178.0	128.6	288.2	323.4	333.5	277.4	240.3	256.7	253.0	302.3
5.500.22	0.75	147.3	175.2	142.4	278.8	294.0	340.0	280.4	252.6	262.2	261.6	285.1
5.500.23	0.75	159.4	196.0	176.0	283.6	296.0	294.0	273.6	234.6	266.3	292.4	295.9
5.500.24	0.75	77.1	57.5	59.7	318.0	330.9	326.9	313.2	269.3	257.3	292.1	310.5
5.500.25	0.75	172.3	155.0	210.4	293.9	264.6	309.3	301.6	230.8	251.7	253.8	289.7
5.500.26	0.75	151.2	127.3	186.8	278.2	294.2	305.4	286.8	231.4	261.9	254.1	284.7
5.500.27	0.75	248.5	203.1	221.7	297.7	319.9	324.0	293.1	260.0	270.5	266.7	300.7
5.500.28	0.75	203.2	112.4	179.9	298.7	303.7	336.4	294.8	255.3	262.7	264.6	269.2
5.500.29	0.75	90.8	94.4	118.3	279.2	317.1	333.2	294.2	275.0	285.6	263.7	310.0

Table 3.12 Average computation time of all algorithms tested on OR-LIB instances with 10 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10.500.0	0.25	240.3	251.8	217.5	265.6	320.5	328.2	306.2	246.2	228.8	278.9	289.0
10.500.1	0.25	235.3	191.9	232.5	308.8	301.9	326.2	270.0	254.3	253.5	242.5	279.1
10.500.2	0.25	220.1	242.8	181.8	278.0	282.6	348.9	301.0	235.7	248.9	264.6	255.0
10.500.3	0.25	222.4	241.3	220.5	252.8	289.4	321.3	292.5	268.5	254.5	262.2	267.5
10.500.4	0.25	234.4	64.3	216.2	296.1	294.7	325.0	263.8	219.4	242.2	276.1	304.7
10.500.5	0.25	208.5	178.1	182.9	249.8	295.3	349.7	247.6	218.7	228.6	229.6	271.6
10.500.6	0.25	248.8	213.2	215.4	266.5	294.5	336.9	273.1	223.8	230.9	250.8	277.6
10.500.7	0.25	239.4	162.4	233.2	292.1	317.1	335.4	283.2	238.3	235.5	289.2	298.0
10.500.8	0.25	228.4	225.1	236.6	258.5	287.3	360.3	293.3	254.9	249.3	272.5	304.8
10.500.9	0.25	242.4	251.1	231.9	243.2	302.9	311.1	303.3	263.6	236.4	256.5	308.7
10.500.10	0.50	147.6	184.8	146.6	281.3	327.4	325.0	300.9	251.5	267.8	281.6	285.0
10.500.11	0.50	140.0	218.5	70.5	271.0	327.0	328.0	311.4	267.6	268.0	282.1	333.1
10.500.12	0.50	146.9	30.0	128.4	309.5	305.7	334.7	263.4	277.5	265.7	285.1	300.2
10.500.13	0.50	234.5	242.1	206.2	306.0	326.3	342.7	293.2	271.3	261.9	283.7	305.8
10.500.14	0.50	144.2	215.8	57.4	322.5	325.9	312.2	322.1	265.3	264.7	270.9	330.8
10.500.15	0.50	216.5	257.2	235.3	333.7	337.8	323.2	309.4	251.0	239.9	272.3	324.4
10.500.16	0.50	234.9	216.0	211.7	301.0	312.7	315.0	276.0	259.7	268.0	249.0	310.1
10.500.17	0.50	148.2	208.5	155.4	299.9	348.0	334.6	309.4	272.7	267.8	297.7	331.6
10.500.18	0.50	236.0	180.1	238.8	320.4	337.0	328.6	302.7	277.1	258.6	274.5	357.8
10.500.19	0.50	194.2	235.8	181.6	310.4	327.9	342.8	287.4	266.8	238.9	273.0	314.4
10.500.20	0.75	123.0	89.5	103.6	327.4	340.9	291.4	293.3	225.9	249.0	273.2	311.8
10.500.21	0.75	233.4	157.1	201.9	287.7	340.4	294.6	281.5	234.5	256.4	248.6	306.5
10.500.22	0.75	224.1	190.7	244.8	295.4	315.8	319.1	240.7	252.0	235.6	276.8	286.2
10.500.23	0.75	177.9	226.9	218.6	325.3	338.0	321.2	288.4	261.6	256.0	254.6	305.3
10.500.24	0.75	134.3	19.9	112.4	320.5	298.4	348.4	289.2	256.7	247.8	264.3	298.3
10.500.25	0.75	228.1	160.0	244.1	317.3	359.3	341.4	318.4	269.1	274.4	280.2	322.7
10.500.26	0.75	7.1	53.6	9.8	322.0	341.4	317.0	299.0	263.4	278.4	262.7	274.5
10.500.27	0.75	221.0	233.4	216.2	291.5	333.3	313.4	288.2	270.0	283.8	251.5	297.3
10.500.28	0.75	211.1	172.3	210.4	300.7	313.1	331.1	297.8	251.5	262.3	251.9	312.6
10.500.29	0.75	185.1	212.9	181.7	296.7	347.8	326.1	278.2	217.0	251.1	253.3	316.4

Table 3.13 Average computation time of all algorithms tested on OR-LIB instances with 30 resources

Instance	Tightness	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
30.500.0	0.25	224.5	218.1	250.4	309.3	323.0	339.7	301.2	287.6	254.4	272.9	339.3
30.500.1	0.25	197.2	181.4	146.2	301.0	331.6	320.3	264.7	245.0	236.2	254.3	271.8
30.500.2	0.25	185.3	204.8	129.2	307.5	323.5	332.2	301.2	259.7	272.1	258.5	304.1
30.500.3	0.25	242.2	237.3	249.7	320.9	299.6	306.0	304.4	286.1	283.9	269.1	340.7
30.500.4	0.25	243.0	173.8	229.7	328.8	356.2	313.5	295.5	254.0	246.4	287.2	328.4
30.500.5	0.25	234.0	244.6	185.5	261.3	308.3	306.4	282.7	226.3	261.6	267.2	327.5
30.500.6	0.25	238.7	234.6	248.6	321.9	321.9	348.2	294.3	260.2	278.0	275.1	322.8
30.500.7	0.25	265.8	224.3	249.6	306.7	318.2	319.4	268.6	286.6	275.9	282.5	291.7
30.500.8	0.25	232.3	204.2	78.9	310.5	341.6	304.7	322.8	272.2	290.3	264.0	315.2
30.500.9	0.25	247.5	227.1	225.8	309.6	343.4	317.7	307.3	268.2	251.0	292.6	310.6
30.500.10	0.50	246.8	235.7	238.1	328.0	347.3	307.3	332.7	294.9	296.4	265.2	352.7
30.500.11	0.50	215.5	188.1	216.5	331.5	360.8	317.8	335.9	306.8	303.6	257.0	361.1
30.500.12	0.50	194.1	221.3	229.4	314.9	359.6	286.6	327.6	290.8	287.4	339.4	320.5
30.500.13	0.50	211.5	210.3	185.8	356.1	371.5	320.6	347.4	285.7	309.3	302.3	366.0
30.500.14	0.50	232.9	190.8	224.5	331.1	362.7	316.3	323.4	273.6	279.7	272.3	327.5
30.500.15	0.50	246.2	229.3	248.9	339.4	346.9	321.0	316.8	287.7	281.4	288.7	351.5
30.500.16	0.50	226.6	210.3	236.3	329.8	349.5	322.1	339.7	300.0	291.3	278.8	338.6
30.500.17	0.50	184.6	171.3	142.1	350.4	365.6	306.1	312.2	296.4	282.8	296.9	348.0
30.500.18	0.50	202.4	200.0	197.7	354.9	338.9	318.1	298.4	292.2	256.9	311.8	342.4
30.500.19	0.50	242.3	186.7	236.4	332.6	353.9	291.4	282.5	281.1	288.7	330.2	332.3
30.500.20	0.75	193.8	171.0	170.7	341.7	342.9	307.7	296.2	279.6	257.6	291.9	330.0
30.500.21	0.75	231.9	237.2	236.3	348.0	316.9	288.5	331.9	279.5	292.0	291.0	337.9
30.500.22	0.75	217.7	126.8	241.5	325.2	325.9	313.3	327.8	285.1	273.1	276.0	333.7
30.500.23	0.75	249.0	233.0	217.6	372.6	350.4	314.3	335.6	290.1	269.4	332.2	353.3
30.500.24	0.75	218.9	215.5	221.2	311.7	355.8	323.1	325.0	311.4	272.2	298.4	342.9
30.500.25	0.75	218.6	182.8	236.1	320.2	365.3	287.0	321.8	297.8	295.8	287.1	342.7
30.500.26	0.75	209.2	221.7	221.1	350.0	354.8	312.8	330.0	302.7	283.4	315.6	344.1
30.500.27	0.75	187.5	158.2	183.7	335.4	358.7	306.7	313.3	289.6	262.9	279.6	344.6
30.500.28	0.75	223.8	195.3	228.6	297.0	352.9	292.7	343.1	256.9	272.7	269.4	342.1
30.500.29	0.75	214.8	215.2	216.7	346.9	339.5	263.1	334.6	265.2	269.9	246.7	325.1

Table 3.14 Comparison of Aco_{neg}^+ , Aco_{neg} , and Aco with the current state of the art on OR-LIB instances with 5 resources.

Instance	best					average					average time				
	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+
5.500.0	120148	120148	120074	120148	120148	120126.90	120137.80	119945.60	120107.30	120119.97	3753.2	110.6	286.1	213.8	241.5
5.500.1	117879	117864	117733	117864	117879	117850.83	117852.40	117614.53	117833.32	117854.25	3876.2	102.3	261.8	195.9	233.2
5.500.2	121131	121131	121026	121125	121131	121112.23	121125.87	120883.06	121112.00	121112.98	3148.0	99.6	252.2	194.6	182.3
5.500.3	120804	120804	120697	120804	120804	120786.40	120795.59	120534.75	120782.67	120784.25	2917.9	156.6	305.0	210.0	244.0
5.500.4	122319	122319	122217	122319	122319	122319.00	122316.00	122090.68	122318.27	122319.00	1936.3	96.9	279.6	108.5	99.9
5.500.5	122024	122024	121925	122024	122024	122008.83	122014.48	121798.64	122000.75	122008.44	4421.1	108.9	277.7	185.7	240.1
5.500.6	119127	119127	119056	119117	119127	119120.50	119122.50	118894.12	119086.25	119103.31	3419.2	109.7	292.6	257.4	222.5
5.500.7	120568	120568	120420	120568	120568	120548.10	120564.88	120278.40	120560.33	120567.13	2738.8	119.5	296.0	202.3	159.3
5.500.8	121575	121575	121422	121575	121575	121559.17	121560.67	121176.27	121540.42	121563.21	2719.1	136.8	302.0	243.7	237.3
5.500.9	120717	120717	120565	120699	120711	120695.00	120707.86	120417.82	120673.69	120686.81	3353.3	134.4	303.0	176.9	223.0
5.500.10	218428	218426	218282	218428	218428	218411.27	218414.45	218112.77	218422.48	218425.78	4100.0	103.4	306.9	163.3	212.1
5.500.11	221191	221202	220975	221191	221202	221184.90	221177.44	220763.01	221188.29	221190.57	3560.1	99.7	306.9	148.3	147.2
5.500.12	217542	217536	217441	217534	217542	217525.90	217533.89	217229.35	217520.54	217533.29	3836.3	119.0	311.9	246.1	244.6
5.500.13	223560	223560	223491	223560	223560	223558.87	223559.98	223354.22	223558.32	223559.16	2139.3	39.8	278.1	84.0	128.9
5.500.14	218966	218966	218897	218966	218966	218966.00	218965.68	218763.77	218963.44	218966.00	171.5	73.3	259.3	158.9	40.2
5.500.15	220530	220530	220425	220527	220530	220528.07	220526.80	220287.05	220501.85	220519.92	3183.0	68.2	308.0	164.2	241.1
5.500.16	219989	219989	219884	219989	219989	219985.90	219988.26	219763.97	219969.64	219988.03	2798.7	115.0	297.5	184.7	183.4
5.500.17	218215	218215	218134	218215	218215	218200.37	218198.35	217934.09	218173.63	218199.48	3700.4	86.5	332.6	230.3	237.3
5.500.18	216976	216976	216878	216976	216976	216976.00	216976.00	216758.08	216976.00	216976.00	607.9	67.5	323.0	34.9	26.6
5.500.19	219719	219719	219598	219717	219719	219715.60	219717.02	219429.09	219701.27	219716.38	2632.9	22.7	314.7	187.4	157.3
5.500.20	295828	295828	295702	295828	295828	295828.00	295828.00	295600.23	295828.00	295828.00	552.3	33.0	310.2	19.3	7.1
5.500.21	308086	308086	308026	308086	308086	308081.87	308079.50	307879.87	308076.79	308079.20	3225.2	99.4	288.2	178.0	155.2
5.500.22	299796	299796	299748	299796	299796	299796.00	299796.00	299640.61	299794.32	299795.04	637.5	20.7	278.8	175.2	147.3
5.500.23	306480	306480	306426	306480	306480	306478.47	306478.56	306317.09	306474.95	306477.40	2626.0	62.3	283.6	196.0	159.4
5.500.24	300342	300342	300280	300342	300342	300340.67	300342.00	300145.43	300342.00	300342.00	3454.1	23.2	318.0	57.5	77.1
5.500.25	302571	302571	302542	302571	302571	302565.40	302562.25	302395.36	302549.53	302553.01	2156.6	65.9	293.9	155.0	172.3
5.500.26	301339	301339	301286	301329	301339	301330.67	301329.21	301199.31	301328.94	301330.01	589.5	71.2	278.2	127.3	151.2
5.500.27	306454	306454	306353	306454	306454	306454.00	306448.36	306255.01	306434.97	306439.75	1707.3	97.3	297.7	203.1	248.5
5.500.28	302828	302828	302747	302818	302828	302820.70	302823.70	302637.37	302810.85	302814.84	2852.2	58.8	298.7	112.4	203.2
5.500.29	299910	299904	299825	299906	299906	299901.80	299902.19	299735.32	299904.58	299904.74	3491.8	128.7	279.2	94.4	90.8
average	214168.07	214167.47	214069.20	214165.20	214168.10	214159.25	214161.52	213927.83	214151.18	214158.60	2676.9	87.7	294.0	163.6	170.5

Table 3.15 Comparison of Aco_{neg}^+ , Aco_{neg} , and Aco with the current state of the art on OR-LIB instances with 10 resources.

Instance	best					average					average time				
	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+
10.500.0	117801	117779	117507	117779	117809	117736.17	117754.82	117233.97	117718.42	117750.73	3664.7	105.3	265.6	251.8	240.3
10.500.1	119200	119206	119000	119182	119188	119137.47	119179.74	118713.1	119136.45	119157.51	3354.9	152.7	308.8	191.9	235.3
10.500.2	119159	119215	118942	119194	119211	119108.27	119162.61	118658.7	119097.25	119167.66	4469.7	127.8	278.0	242.8	220.1
10.500.3	118829	118813	118602	118813	118813	118793.93	118777.36	118368.5	118769.11	118796.83	3150.1	39.8	252.8	241.4	222.4
10.500.4	116456	116509	116209	116471	116509	116405.17	116460.83	115817.22	116432.6	116458.96	3582.2	146.2	296.1	64.3	234.4
10.500.5	119483	119470	119278	119461	119504	119441.80	119435.57	119057.84	119438.87	119454.37	3646.9	157.4	249.8	178.2	208.5
10.500.6	119775	119827	119508	119777	119790	119739.70	119782.76	119329.87	119730.6	119762.77	3789.5	134.1	266.5	213.2	248.8
10.500.7	118323	118320	117892	118277	118320	118258.27	118265.30	117650.85	118236.99	118265.76	4228.5	146.7	292.1	162.4	239.4
10.500.8	117801	117781	117620	117776	117781	117705.97	117763.24	117266.94	117714.68	117750.95	3168.7	122.3	258.5	225.1	228.4
10.500.9	119196	119212	118885	119192	119210	119161.90	119166.54	118623.82	119140.6	119174.44	3338.8	124.7	243.2	251.1	242.4
10.500.10	217351	217365	217108	217318	217377	217313.67	217326.03	216828.93	217304.72	217320.43	3984.4	76.7	281.3	184.8	147.7
10.500.11	219059	219063	218877	219041	219077	219022.70	219027.54	218465.83	219018.89	219037.71	3598.2	136.0	271.0	218.5	140.0
10.500.12	217847	217847	217540	217792	217793	217786.73	217760.63	217213.53	217755.37	217766.66	4010.4	173.1	309.5	30.0	146.9
10.500.13	216868	216843	216635	216868	216868	216836.33	216824.05	216391.77	216801.19	216828.19	3403.7	125.3	306.0	242.1	234.5
10.500.14	213814	213843	213448	213831	213859	213780.27	213817.08	213241.02	213794.81	213829.04	4095.0	134.4	322.5	215.8	144.2
10.500.15	215086	215062	214866	215071	215073	215049.57	215034.68	214592.73	215001.51	215035.49	3622.3	166.4	333.7	257.2	216.5
10.500.16	217926	217931	217644	217899	217931	217884.80	217884.36	217441.7	217877.42	217888.83	4281.3	181.3	301.0	216.0	234.9
10.500.17	219984	219984	219678	219949	219984	219947.37	219965.07	219389.02	219931.22	219952.76	3908.2	102.0	299.9	208.5	148.2
10.500.18	214363	214382	213897	214346	214375	214327.43	214337.14	213656.45	214286.95	214326.8	3999.5	165.9	320.4	180.1	236.0
10.500.19	220887	220865	220619	220846	220882	220864.43	220847.10	220335.62	220823.42	220846.47	3211.7	152.5	310.4	235.8	194.3
10.500.20	304387	304387	304156	304344	304359	304364.47	304354.01	303954.3	304344	304344.23	3025.1	67.4	327.4	89.5	123.0
10.500.21	302379	302358	302216	302331	302371	302364.47	302346.04	302012.31	302317.73	302333.34	2883.6	98.6	287.7	157.1	233.4
10.500.22	302416	302408	302227	302408	302408	302398.13	302399.10	302035.97	302354.42	302399.89	3497.9	77.4	295.4	190.7	224.1
10.500.23	300784	300784	300590	300743	300747	300758.80	300745.46	300400.11	300705.8	300740.37	967.0	58.6	325.3	226.9	177.9
10.500.24	304374	304374	304266	304340	304374	304361.13	304353.75	304023.03	304340	304345.91	3509.5	112.6	320.5	19.9	134.3
10.500.25	301796	301766	301448	301751	301796	301740.43	301752.48	301180.92	301712.89	301753.02	3808.8	106.4	317.3	160.0	228.1
10.500.26	304952	304949	304779	304949	304949	304952.00	304949.00	304601.92	304949	304949	3132.6	16.2	322.0	53.6	7.1
10.500.27	296478	296459	296271	296456	296456	296455.53	296444.16	296021.63	296428.01	296451.24	3138.7	153.6	291.5	233.4	221.0
10.500.28	301359	301357	301149	301313	301357	301349.27	301332.24	300925.65	301295.35	301315.08	3187.4	163.1	300.7	172.3	211.1
10.500.29	307089	307089	306867	307072	307089	307088.23	307068.83	306674.76	307005.03	307056.72	2351.9	83.1	296.7	212.9	185.1
average	212840.70	212841.60	212590.80	212819.67	212842.00	212804.48	212810.58	212336.93	212782.11	212808.71	3467.04	120.26	295.1	184.2	196.9

Table 3.16 Comparison of Aco_{neg}^+ , Aco_{neg} , and Aco with the current state of the art on OR-LIB instances with 30 resources.

Instance	best					average					average time				
	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+	TPTEA	DQPSO	Aco	Aco_{neg}	Aco_{neg}^+	TPTEA	DQPSO	ACO	Aco_{neg}	Aco_{neg}^+
30.500.0	115968	115952	115031	115915	116014	115897.20	115805.36	114568.47	115860.72	115880.99	3969.2	346.1	309.3	218.1	224.5
30.500.1	114769	114734	114084	114718	114734	114733.00	114633.84	113697.99	114678.19	114693.18	3548.9	525.6	301.0	181.4	197.2
30.500.2	116708	116741	115727	116669	116685	116619.10	116581.00	115185.45	116624.93	116655.06	4413.0	334.7	307.5	204.8	185.3
30.500.3	115313	115236	114506	115228	115310	115251.60	115206.66	113973.97	115145.66	115201.33	3499.0	460.4	320.9	237.3	242.2
30.500.4	116455	116372	115242	116448	116516	116364.80	116296.31	114781.43	116368.18	116390.87	3436.2	484.0	328.8	173.8	243.0
30.500.5	115734	115673	115102	115643	115734	115674.00	115631.11	114768.43	115624.42	115643.17	3847.9	347.6	261.3	244.6	234.0
30.500.6	114085	113961	113008	114058	114181	114037.10	113929.22	112646.63	113956.57	114004.06	4784.7	732.4	321.9	234.6	238.7
30.500.7	114278	114189	113096	114344	114344	114164.40	114043.17	112744.36	114151.89	114220.38	4109.9	445.9	306.7	224.3	265.8
30.500.8	115288	115319	114389	115419	115419	115221.43	115170.67	114024.21	115328.43	115357.94	4043.3	524.3	310.5	204.2	232.3
30.500.9	117112	117003	116195	117020	117116	116984.37	116920.85	115599.74	116919.21	117011.92	3778.6	564.0	309.6	227.1	247.5
30.500.10	218104	218043	217319	218033	218081	218069.60	218008.51	217005.44	217956.25	218017.45	3163.2	328.6	328.0	235.7	246.8
30.500.11	214645	214551	213579	214626	214626	214544.93	214473.44	213046.90	214554.61	214604.97	3796.3	321.6	331.5	188.1	215.5
30.500.12	215946	215883	215398	215903	215914	215898.80	215841.16	215042.45	215814.04	215853.66	4007.6	376.1	314.9	221.3	194.1
30.500.13	217910	217807	216839	217827	217863	217831.33	217774.97	216442.36	217768.17	217826.98	3259.2	423.1	356.1	210.3	211.5
30.500.14	215689	215601	214751	215566	215640	215602.07	215514.71	214334.81	215535.41	215574.33	3945.5	437.2	331.1	190.8	232.9
30.500.15	215840	215774	214777	215853	215867	215766.23	215698.08	214373.28	215721.05	215804.39	3558.5	520.9	339.4	229.3	246.2
30.500.16	215907	215871	215326	215798	215883	215857.23	215773.37	214893.07	215764.32	215807.29	3176.9	316.2	329.8	210.3	226.6
30.500.17	216542	216452	215692	216419	216463	216459.73	216348.59	215263.37	216397.22	216428.56	3642.6	459.7	350.4	171.3	184.6
30.500.18	217340	217290	216653	217312	217333	217304.30	217244.15	216168.53	217214.44	217302.34	3460.7	457.6	354.9	200.0	202.4
30.500.19	214739	214681	213796	214657	214692	214671.30	214626.78	213388.99	214629.05	214644.38	3417.9	438.7	332.6	186.7	242.3
30.500.20	301675	301643	301177	301643	301667	301641.63	301628.38	300870.90	301609.06	301641.95	2849.2	580.0	341.7	171.0	193.8
30.500.21	300055	300013	299578	299996	300055	300035.73	299944.24	299174.03	299927.98	299990.68	3862.8	401.2	348.0	237.2	231.9
30.500.22	305087	305055	304753	305018	305080	305080.47	304993.46	304337.02	304985.88	305023.90	3784.6	482.8	325.2	126.8	217.7
30.500.23	302015	302004	301441	302001	302004	301983.60	301928.64	301128.88	301884.88	301927.37	3200.1	310.3	372.6	233.0	249.0
30.500.24	304462	304404	304046	304413	304427	304427.53	304393.26	303657.20	304369.92	304409.38	3102.0	296.8	311.7	215.5	218.9
30.500.25	296999	296962	296394	296909	296960	296964.97	296864.15	296124.35	296887.26	296922.90	3743.2	459.0	320.2	182.8	218.6
30.500.26	303364	303360	302772	303304	303335	303335.60	303253.82	302442.69	303252.04	303299.44	2828.3	402.5	350.0	221.7	209.2
30.500.27	306999	306999	306511	306941	306956	306972.50	306930.64	306058.39	306893.70	306916.43	3922.4	535.4	335.4	158.2	187.5
30.500.28	303199	303162	302543	303151	303178	303168.53	303099.06	302140.97	303085.41	303118.96	3283.0	486.2	297.0	195.3	223.8
30.500.29	300596	300536	300077	300512	300536	300530.23	300499.76	299796.94	300477.36	300495.77	3936.9	383.5	346.9	215.2	214.8
average	211427.40	211375.70	210660.07	211378.13	211420.43	211369.80	211301.90	210256.04	211312.88	211355.67	3645.7	439.4	326.5	205.0	222.6

CHAPTER 4

APPLICATION TO THE MINIMUM DOMINATING SET PROBLEM

4.1 INTRODUCTION

This Chapter describes our negative learning Aco application to the *Minimum Dominating Set* (MDS) problem [128]. Some parts of this Chapter are also presented in our paper [104] that was published in the Mathematics Journal (<https://doi.org/10.3390/math9040361>). This work is scientifically done in exactly the same way as our negative learning Aco application to the MDKP, which is explained in Chapter 3. As in the context of the MDKP, our negative learning Aco application to the MDS is also compared to existing negative learning approaches from the related literature. The obtained results for the MDS problem also show that our negative learning mechanism significantly improves over standard Aco and over negative learning approaches from the literature. Moreover, our approach obtains results that are comparable to the current state-of-the-art approaches for the MDS, at least in the context of problem instances that are not too large.

4.2 THE MINIMUM DOMINATING SET PROBLEM

The classical MDS problem—which is *NP*-hard—can be stated as follows. Given an undirected graph $G = (V, E)$, with V being the set of vertices and E the set of edges. Given a vertex $v_i \in V$, $N(v_i) \subset V$ denotes the neighborhood of v_i in G . A subset $S \subseteq V$ is called a *dominating set* if and only if for each vertex $v_i \in V$ the following holds: (1) $v_i \in S$ or (2) there is at least one $v_j \in N(v_i)$ with $v_j \in S$. The MDS requires finding a feasible solution of minimum cardinality.

4.2.1 ILP Model for the MDS Problem

A standard integer linear programming (ILP) model for the MDS problem can be stated as follows.

$$\text{minimize } \sum_{v_i \in V} x_i \quad (4.1)$$

subject to:

$$\sum_{v_j \in N(v_i)} x_j \geq 1 - x_i \quad \forall v_i \in V \quad (4.2)$$

$$x_i \in \{0, 1\} \quad \forall v_i \in V \quad (4.3)$$

The model consists of a binary variable x_i for each vertex $v_i \in V$. The objective function counts the selected vertices, and the constraints (4.2) ensure that each vertex either belongs to the solution or has, at least, one neighbor that forms part of the solution. In the literature, there are many variants of the MDS problem. Examples include the minimum connected dominating set problem [129], the minimum total dominating set problem [130] and the minimum vertex weight dominating set problem [131]. The currently best metaheuristic approach for solving the MDS problem is a two-goal local search with inference rules from [109].

4.3 *MMAS* IMPLEMENTATION TO THE MDS

As in the case of the previous chapter, much of the text in this section and the next section already appear in Chapter 2 in the context of the description of the baseline *MMAS* algorithm. It is repeated here for the benefit of the reader. All of our negative learning Aco variants are based again to the same baseline Aco algorithm, which is *MMAS*. Remember that this algorithm keeps three solutions all the time: (1) the iteration-best solution S^{ib} which is the best solution found in an iteration, (2) the restart-best solution S^{rb} which is the best solution generated since the last restart of the algorithm, and (3) the best-so-far solution S^{bsf} which is the best-overall solution found during the execution of the algorithm. Solutions S^{bsf} and S^{rb} are initialized to `NULL` when the algorithm starts, see line 2 of Algorithm 2.1 in Chapter 2. Furthermore, the algorithm makes use of a Boolean control variable `bs_update` $\in \{\text{TRUE}, \text{FALSE}\}$ and the convergence factor $cf \in [0, 1]$ for deciding on the pheromone update mechanism and on the question whether or not to restart the algorithm. The convergence factor is set to zero and `bs_update` is set to `FALSE`

at the start of the algorithm. The pheromone model \mathcal{T} in the context of the MDS problem consists of a pheromone value $\tau_i \geq 0$ for each vertex $v_i \in V$, where V is the complete set of vertices of the input graph. These pheromone values are all initialized to 0.5 by function `InitializePheromoneValues(\mathcal{T})` of Algorithm 2.1. Note that V corresponds to the complete set of solution components of the description in Chapter 2.

At each iteration, n_a solutions are generated by function `Construct_Solution(\mathcal{T})` of Algorithm 2.1, based on pheromone information and on greedy information. The detailed explanation of the solutions construction mechanism is provided in Section 4.3.1. The generated solutions are stored in set $\mathcal{S}^{\text{iter}}$, and the best one from $\mathcal{S}^{\text{iter}}$ is stored as S^{ib} ; see lines 5–10 of Algorithm 2.1. Then, the restart-best and best-so-far solutions— S^{rb} and S^{bsf} —are updated with S^{ib} , if appropriate; see lines 11 and 12 of Algorithm 2.1. Afterward, the pheromone update is conducted in function `ApplyPheromoneUpdate(\mathcal{T} , cf , bs_update, S^{ib}, S^{rb}, S^{bsf})` and the new value for the convergence factor cf is computed in function `ComputeConvergenceFactor(\mathcal{T})`; lines 13 and 14 of Algorithm 2.1. The detailed explanation of the pheromone update mechanism and of the calculation of cf is provided in Section 4.3.2. The algorithm might be restarted depending on the values of cf and `bs_update`. This restart is implemented by re-initializing all pheromone values and by setting the restart-best solution S^{rb} to `NULL` and `bs_update` to `TRUE`.

4.3.1 Solution Construction

In the following we say that, if a vertex v_i is added to a solution S under construction, then v_i covers itself and all its neighbors, that is, all $v_j \in N(v_i)$. Moreover, given a set $S \subset V$ —that is, a solution under construction—we denote by $N(v_i \mid S) \subseteq N(v_i)$ the set of uncovered neighbors of $v_i \in V$. The solution construction mechanism is shown in Algorithm 4.1. It starts with an empty solution $S = \emptyset$. Then, at each step, exactly one of the vertices of those that do not yet form part of S or that—with respect to S —have uncovered neighbors (\bar{V}) is chosen in function `ChooseFrom(\bar{V})` and added to S . The choice of a vertex in `ChooseFrom(\bar{V})` is done as follows. First, a probability $\mathbf{p}(v_i)$ is calculated for each $v_i \in \bar{V}$:

$$\mathbf{p}(v_i) := \frac{\eta_i \cdot \tau_i}{\sum_{v_k \in \bar{V}} \eta_k \cdot \tau_k} \quad (4.4)$$

Hereby, $\eta_i := |N(v_i \mid S)| + 1$ is the greedy information that we used. Then, a random number $r \in [0, 1]$ is drawn. If $r \leq d_{\text{rate}}$, v_j (to be added to S) is selected such that $\mathbf{p}(v_j) \geq \mathbf{p}(v_i)$ for all $v_i \in \bar{V}$. Otherwise, v_j is chosen by

Algorithm 4.1 MDS solution construction

```

1: input: a graph  $G = (V, E)$ 
2:  $S := \emptyset$ 
3:  $\bar{V} := \{v_i \in V \mid v_i \notin S \text{ and } N(v_i \mid S) \neq \emptyset\}$ 
4: while  $\bar{V} \neq \emptyset$  do
5:    $v_j := \text{ChooseFrom}(\bar{V})$ 
6:    $S := S \cup \{v_j\}$ 
7:    $\bar{V} := \{v_i \in V \mid v_i \notin S \text{ or } N(v_i \mid S) \neq \emptyset\}$ 
8: end while
9: output: a valid solution  $S$ 

```

roulette-wheel-selection based on the calculated probabilities.

4.3.2 Pheromone Update and Convergence Factor

Function `ApplyPheromoneUpdate`($\mathcal{T}, cf, bs_update, S^{ib}, S^{rb}, S^{bsf}$) of Algorithm 2.1 is implemented in *MMAS* for MDS problems in the following way. Weights κ_{ib} , κ_{rb} and κ_{bsf} are given to the solution components of S^{ib} , S^{rb} , and S^{bsf} , respectively. The values of these weights depends on the values of cf and bs_update . Moreover, they are regulated by a schedule which is provided in Table 2.1. Note that the sum of κ_{ib} , κ_{rb} , and κ_{bsf} must always equal to 1. Subsequently, each pheromone value τ_i of each vertex v_i is updated by using Eqn. (4.5).

$$\tau_i := \tau_i + \rho \cdot (\xi_i - \tau_i) \quad , \quad (4.5)$$

Hereby, $\rho \in [0, 1]$ in Eqn. (4.5) is the learning rate, which is an important variable of the Aco algorithm. The parameter ξ_i in this equation stores the accumulative update received by each vertex v_i . The value of this parameter is calculated by using Eqn. (4.6).

$$\xi_i := \kappa_{ib} \cdot \Delta(S^{ib}, v_i) + \kappa_{rb} \cdot \Delta(S^{rb}, v_i) + \kappa_{bsf} \cdot \Delta(S^{bsf}, v_i) \quad (4.6)$$

Function $\Delta(S, v_i)$ evaluates to 1 if and only if vertex v_i forms part of solution S . Otherwise, the function evaluates to 0. Accordingly, any vertex that does not appear in either S^{ib} , S^{rb} , or S^{bsf} will have its $\xi_i = 0$. Thus, according to Eqn. (4.5) its new pheromone value is lower than the previous one. After this update, each pheromone that has a value outside of the range of $\tau_{\max} = 0.999$ and $\tau_{\min} = 0.001$ is set back to the appropriate upper or lower limit value. Hence, a complete convergence of the algorithm is avoided.

After the pheromone update, the new value of convergence factor is updated

in function $\text{ComputeConvergenceFactor}(\mathcal{T})$ of Algorithm 2.1 by using Eqn. (4.7).

$$cf := 2 \left(\left(\frac{\sum_{\tau_i \in \mathcal{T}} \max\{\tau_{\max} - \tau_i, \tau_i - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right) \quad (4.7)$$

Hereby, \mathcal{T} in Eqn. (4.7) represents all pheromone values of all $v_i \in V$. When all pheromone values are initialized to 0.5, the value of cf is zero. The algorithm is said to have reached convergence when pheromone values have either value τ_{\min} or τ_{\max} . In this condition, the value of cf evaluates to one. The value of cf is between 0 and 1 in all other conditions.

4.4 ADDING NEGATIVE LEARNING TO \mathcal{MMAS}

The negative learning mechanism is added to the baseline \mathcal{MMAS} according to the same strategy as already explained in Section 2.3. As in the general case of subset selection problems in Chapter 2, an additional pheromone model \mathcal{T}^{neg} that stores the negative learning information is employed in this application. This pheromone model consists of a pheromone value $\tau_i^{\text{neg}} \in \mathcal{T}^{\text{neg}}$ for each vertex $v_i \in V$. All the τ_i^{neg} are initialized to $\tau_{\min} = 0.001$, as described in Section 2.3 in the context of subset selection problems.

Figure 4.1 provides an illustrative example on how the negative learning mechanism is implemented in the context of the MDS problem. The negative learning information is generated in an instruction $S^{\text{sub}} :=$

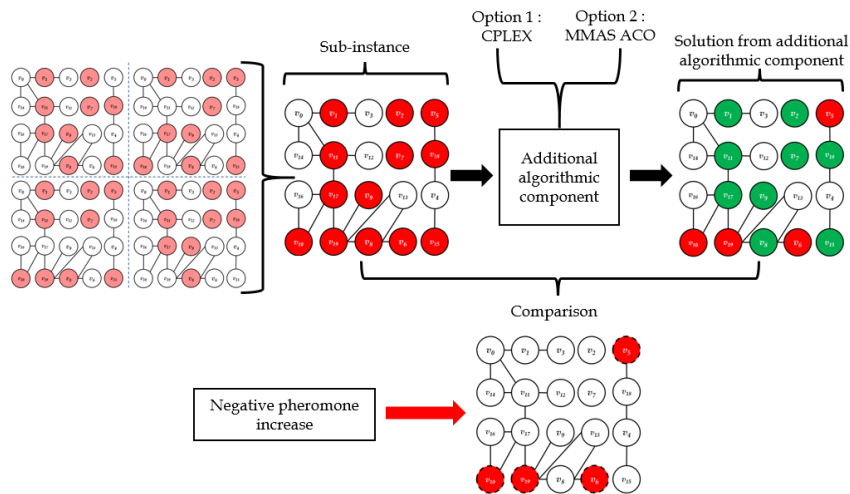


Fig. 4.1 Illustrative example of the negative learning mechanism for the MDS problem

$\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ (Eqn. (2.5)) which is added between lines 9 and 10 of the baseline \mathcal{MMAS} algorithm (Algorithm 2.1). In the example shown in Fig. 4.1, there are four solutions generated in an iteration of the baseline Aco algorithm. Function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ creates a sub-instance $V' \subseteq V$ by merging all four solutions from $\mathcal{S}^{\text{iter}}$. Subsequently, the sub-instance is delivered to an optimization algorithm that finds the best-possible solution that only consists of vertices from sub-instance V' . Similar to our negative learning Aco application to the MDKP (Section 3.4), we have experimented with two options for the *inner optimization algorithm* that solves the sub-instance V' : (1) ILP solver CPLEX 12.10 and (2) the baseline \mathcal{MMAS} algorithm. In the first option, the ILP model from Section 4.2.1 is used by adding the following constraint.

$$x_i = 0 \quad \forall v_i \in V \setminus V' \quad (4.8)$$

These constraints imply that CPLEX can not choose vertices that are not in the sub-instance V' . The same strategy is implemented in the use of the second option. In particular, the main \mathcal{MMAS} algorithm supplies the inner \mathcal{MMAS} with a sub-instance V' that consists of vertices found only in solutions from $\mathcal{S}^{\text{iter}}$. Consequently, in its solution construction, the choice of the inner \mathcal{MMAS} algorithm is limited to vertices from V' . In addition to this configuration, the inner \mathcal{MMAS} is set to operate with its own parameter settings (pheromone model, learning rate, determinism rate, etc.) and its best-so-far solution S^{bsf} is initialized with the iteration-best solution S^{ib} of the main \mathcal{MMAS} .

Given a maximum computation time of t^{sub} CPU seconds, the inner optimization algorithm—CPLEX or \mathcal{MMAS} —has $(1 - cf)t^{\text{sub}} + 0.1cf$ seconds of allotted computation time for solving the sub-instance V' . This way, the inner optimization algorithm receives less computation time allocation for solving the sub-instance V' as the search is approaching convergence. Remember that the reason behind this arrangement is that, as the search is approaching convergence, solutions in $\mathcal{S}^{\text{iter}}$ become more and more similar and the sub-instance V' is having smaller size. Consequently, less time is needed to explore the sub-instance V' .

At the end of its allotted execution time, function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ generates a solution S^{sub} whose components are represented by green circles in Fig. 4.1. Afterward, the function uses the solution for updating the negative pheromone values in the following way. First, it compares the vertices in the sub-instance V' to the ones in the solution S^{sub} . In the example of Fig. 4.1, we can see that nine vertices out of thirteen vertices available in the sub-instance V' are chosen to form part of S^{sub} . The remaining four vertices are marked as low-quality

solution components. Subsequently, they are penalized by having their negative pheromone increased. For each vertex in set V' , its negative pheromone value is updated by using Eqn. (4.9).

$$\tau_i^{\text{neg}} := \tau_i^{\text{neg}} + \rho^{\text{neg}} \cdot (\xi_i^{\text{neg}} - \tau_i^{\text{neg}}) \quad (4.9)$$

Hereby, ρ^{neg} in Eqn. (4.9) is the negative learning rate. The value of parameter ξ_i^{neg} in Eqn. (4.9) is equal to one if $v_i \notin S^{\text{sub}}$, otherwise ξ_i^{neg} is equal to zero. Afterwards, the solution S^{sub} is returned to the main *MMAS* algorithm. Then, a second instruction, $\mathcal{S}^{\text{iter}} := \mathcal{S}^{\text{iter}} \cup \{S^{\text{sub}}\}$ (Eqn. (2.6)) which merges the S^{sub} into the $\mathcal{S}^{\text{iter}}$, is added after $S^{\text{sub}} := \text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ (Eqn. (2.5)).

In the solution construction of the negative learning Aco variants, the negative pheromone values are used by replacing the standard *MMAS* calculation in Eqn. (4.4) with with the one in Eqn. (4.10).

$$\mathbf{p}(c_i) := \frac{\eta_i \cdot \tau_i \cdot (1 - \tau_i^{\text{neg}})}{\sum_{c_k \in \bar{C}} \eta_k \cdot \tau_k \cdot (1 - \tau_k^{\text{neg}})} \quad (4.10)$$

In this way, any vertex that has high negative pheromone value will have lower probability to be chosen as component of solutions in the current and in following iterations.

4.5 PROPOSALS FROM THE LITERATURE

As in our negative learning Aco application to the MDKP (Chapter 3), we re-implemented four negative learning strategies from the works of Montgomery and Randall [84] and Ramos et al. [87] to the MDS problems for the purpose of comparison. The detail description of each strategy is provided in Section 3.5. This Section is provided for reviewing several important aspects of our re-implementations of these strategies.

First of all, remember that we re-introduced those strategies in the baseline *MMAS* algorithm for ensuring a fair comparison. According to their common feature, we implemented these extensions of the baseline *MMAS* to maintain the *iteration-worst* solution (S^{iw}), the *restart-worst* solution (S^{rw}) and the *worst-so-far* solution (S^{wsf}) in addition to the *iteration-best* solution (S^{ib}), the *restart-best* solution (S^{rb}) and the *best-so-far* solution (S^{bsf}). Solutions S^{rw} and S^{wsf} are initialized to `NULL` at the start of the algorithm, as in the case of S^{rb} and S^{bsf} . Subsequently, S^{rw} and S^{wsf} are compared with S^{iw} . In particular, if S^{iw} is worse

than S^{rw} then S^{rw} is updated with S^{iw} . Similarly, if S^{iw} is worse than S^{wsf} then S^{wsf} is updated with S^{iw} .

The negative learning strategies from the works of Montgomery and Randall [84] consist of three variants: (1) ACO-SAP, (2) ACO-EA, and (3) ACO-PAP. The first variant, ACO-SAP, generates the negative learning information simply by choosing the worst solution found in an iteration of standard-MMAS solution-constructions. Furthermore, this variant does not employ a dedicated pheromone model for storing the negative learning information. Hence, the negative learning information is used only for reducing pheromone values of the standard pheromone model by means of an additional update procedure which is implemented after the MMAS standard pheromone update. Before this update is implemented, a set B is generated by joining the vertices in solutions S^{iw} , S^{rw} and S^{wsf} , that is, $B := S^{iw} \cup S^{rw} \cup S^{wsf}$. Then, all those vertices whose pheromone value receives an update from at least one of the solutions S^{ib} , S^{rb} , or S^{bsf} in the current iteration are removed from B . In other words, only the pheromone values of those vertices that appear in low-quality solutions, without forming part of good solutions, are subject to a pheromone value decrease as also shown in Eqn. (3.11).

Similar to ACO-SAP, the second variant from Montgomery and Randall—ACO-EA—does not employ a dedicated pheromone model for storing the negative learning information. This variant however, generates its negative learning information in a different way. In contrast to ACO-SAP which simply chooses the worst solution at each iteration, ACO-EA allocates several of its solution constructions for searching low-quality solutions by making use of the opposite of the pheromone values (see Eqn. (3.13)). Subsequently, this negative learning information is updated with the same mechanism as the one implemented in ACO-SAP.

The third negative learning Aco variant from Montgomery and Randall is labeled ACO-PAP. Different to the previous variants, this variant employs a dedicated pheromone model \mathcal{T}^{neg} —in addition to the standard model \mathcal{T} —for storing negative learning information. The update mechanism for this negative pheromone model follows the update mechanism for the standard pheromone model, as explained in sub-Section 3.5.3. Similar to the ACO-EA variant, ACO-PAP allocates several of its solution constructions for searching low-quality solutions. In ACO-PAP however, the search for low-quality solutions is implemented gradually. This feature is enabled by a parameter λ that regulates the influence of both τ_i and τ_i^{neg} in Eqn. (3.16) for each of the solution constructions during an iteration.

The fourth negative learning Aco variant that we re-implemented in this work is taken from the work of Ramos et al. [87]. This variant is labelled as Aco^{2o} . As in ACO-PAP from the work of Montgomery and Randall, Aco^{2o} maintains a negative pheromone model and a similar update mechanism. Furthermore, Aco^{2o} also regulates the influence of τ_i and τ_i^{neg} in its solution constructions. A parameter which is called $\alpha \in [0, 1]$ in Eqn. (3.18) is used for this purpose. In contrast to λ that divides and changes the portions of τ_i and τ_i^{neg} during ACO-PAP 's iteration, α simply allocates the portions of both pheromone types. This allocation remains consistent during the run of this algorithm variant. All in all, these four strategies differ to each other in various aspects representing most of the negative learning approaches from the existing literature, as explained in Section 1.5. Therefore, it is interesting to see which of these strategies will deliver the best performance and how they perform in comparison to our proposals.

4.6 SUMMARY OF THE TESTED ALGORITHMS

As in our negative learning Aco application to the MDKP in Chapter 3, we tested the same six negative learning variants. A detailed description of these variants is provided in Section 3.6. A summary of these variants is presented in Table 4.1. Each of these variants uses one of two optimization algorithms as inner algorithm. The ILP solver Cplex is used by variants $\text{Aco}_{\text{neg}}^+$, Aco_{neg} , and Aco^+ while MMAS is used by variants $\text{Aco-Aco}_{\text{neg}}^+$, $\text{Aco-Aco}_{\text{neg}}$, and Aco-Aco^+ . The solution S^{sub} , generated by the inner algorithm, is utilized differently. Variants $\text{Aco}_{\text{neg}}^+$ and $\text{Aco-Aco}_{\text{neg}}^+$ take the most benefit from the solution S^{sub} by using it to update their negative pheromone and their best results. In contrast, variants Aco_{neg} and $\text{Aco-Aco}_{\text{neg}}$ use S^{sub} exclusively to update their negative pheromone values. Similarly, variants Aco^+ and Aco-Aco^+ use S^{sub} to update only their best result.

As in the case of the MDKP, we also compare our six algorithm variants ($\text{Aco-Aco}_{\text{neg}}^+$, Aco_{neg} , Aco^+ , $\text{Aco-Aco}_{\text{neg}}^+$, $\text{Aco-Aco}_{\text{neg}}$, and Aco-Aco^+) to the four variants from the literature (ACO-SAP , ACO-EA , ACO-PAP and Aco^{2o}) and to the baseline algorithm MMAS . Each of these 11 algorithms has its own parameters that are not always present in other variants. A summary of these parameters and their description is provided in Table 3.1. Moreover, an overview on the parameters that are involved in each of the 11 algorithms is provided in Table 3.2.

Table 4.1 Features comparison between our negative learning Aco variants

Specifications	Algorithms					
	Aco_{neg}^+	Aco_{neg}	Aco^+	$Aco-Aco_{neg}^+$	$Aco-Aco_{neg}$	$Aco-Aco^+$
Type of the inner algorithm:						
• CPLEX	✓	✓	✓			
• MMAS				✓	✓	✓
The update where S^{sub} of the inner algorithm is used:						
• negative pheromone update	✓	✓		✓	✓	
• best result update	✓		✓	✓		✓

4.7 EXPERIMENTAL EVALUATION

The experiments concerning the MDS problem were performed on a cluster of machines with two Intel® Xeon® Silver 4210 CPUs with 10 cores of 2.20 GHz and 92 Gbytes of RAM. For solving the sub-instances in Aco_{neg}^+ , Aco_{neg} and Aco^+ we used CPLEX 12.10 in one-threaded mode.

Concerning the MDS problem, we generated a benchmark instance set with instances of different sizes (number of vertices $n \in \{5000, 10000\}$), different densities (percentage of all possible edges $d \in \{0.1, 0.5, 1.0, 5.0\}$) and different graph types (random graphs and random geometric graphs). For each combination of n , d and graph type, 10 random instances were generated. This makes a total of 160 problem instances.

4.7.1 Algorithm Tuning

The scientific parameter tuning tool *irace* [123] was used for the purpose of parameter tuning. In particular we produced for each of the 11 algorithms (resp., algorithm versions) exactly one parameter value set. For the purpose of tuning the algorithms for the MDS problem, we additionally generated for each combination of n , d (density), and graph type exactly one random instance. In other words, 16 problem instances were used for tuning, and the tuner was given a maximal budget of 2000 algorithm applications. The parameter values that were determined by *irace* for the 11 algorithms are provided in Tables 4.2.

Table 4.2 Parameter values for all algorithms for solving the MDS problem

Parameter	Algorithms										
	ACO	ACO ⁺ _{neg}	ACO _{neg}	ACO ⁺	ACO-ACO ⁺ _{neg}	ACO-ACO _{neg}	ACO-ACO ⁺	ACO-SAP	ACO-EA	ACO-PAP	ACO ^{2o}
n_a	3	20	20	20	3	10	10	20	10	3	3
ρ	0.4	0.1	0.1	0.5	0.1	0.2	0.5	0.4	0.5	0.1	0.2
d_{rate}	0.9	0.9	0.8	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
ρ^{neg}	--	0.4	0.4	--	0.2	0.5	--	--	--	0.2	0.2
γ	--	--	--	--	--	--	--	0.6	0.6	--	--
p_{expa}	--	--	--	--	--	--	--	--	0.1	--	--
α	--	--	--	--	--	--	--	--	--	--	0.96
t^{sub}	--	8	7	6	6	7	8	--	--	--	--
n_a^{sub}	--	--	--	--	3	3	3	--	--	--	--
ρ^{sub}	--	--	--	--	0.3	0.5	0.4	--	--	--	--
$d_{\text{rate}}^{\text{sub}}$	--	--	--	--	0.7	0.7	0.5	--	--	--	--

4.7.2 Results

Using the previously determined parameter values, each of the 11 considered algorithms was applied 30 times—that is, with 30 different random seeds—to each of the 160 MDS problem instances. Hereby, 500 CPU seconds were chosen as a time limit for the graphs with 5000 nodes, whereas 1000 CPU seconds were chosen as a time limit for each run concerning the graphs with 10000 nodes.

We present a comparative analysis of the 11 algorithms in terms of *critical difference* (CD) plots [124] and so-called *heatmaps*. In order to produce the average ranks of all algorithms—both for the whole set of problem instances as well as for instance subsets—the Friedman test was applied for the purpose of comparing the 11 approaches simultaneously. In this way we also obtained the rejection of the hypothesis that the 11 techniques perform equally. Subsequently, all pairwise algorithm comparisons were performed using the Nemenyi post-hoc test [125]. The obtained results are shown graphically (CD plots and heatmaps). The CD plots show the average algorithm ranks (horizontal axis) with respect to the considered (sub-)set of instances. In those cases in which the performances of two algorithms are below the critical difference threshold—based on a significance level of 0.05—the two algorithms are considered as statistically equivalent. This is indicated by bold horizontal bars joining the markers of the respective algorithm variants.

Figure 4.2 shows the CD plot for the whole set of 160 MDS instances, while

Fig. 4.3a and Fig. 4.3b present more fine-grained results concerning random graphs (RGs) and random geometric graphs (RGGs), respectively. Furthermore, the heatmaps in Fig. 4.4 show the average ranks of the 11 algorithms in an even more fine-grained way. The graphic shows exactly one heatmap for each algorithm. The ones of algorithms $\text{Aco}_{\text{neg}}^+$, Aco_{neg} and Aco^+ are shown in Fig. 4.4a, the ones of algorithms $\text{Aco-Aco}_{\text{neg}}^+$, $\text{Aco-Aco}_{\text{neg}}$ and Aco-Aco^+ in Fig. 4.4b, and the ones of the remaining five algorithms in Fig. 4.4c. The upper part of each heatmap shows the results for RGs, while the lower part concerns the results for RGGs. Each of these parts has two columns: the first one contains the results for the graphs with 5000 nodes, and the second one for the ones with 10,000 nodes. Moreover, each part has four rows, showing the results for the four considered graph densities. In general, the more yellow the cell of a heatmap, the better is the relative performance of the corresponding algorithm for the respective combination of features (graph type, graph size, and density).

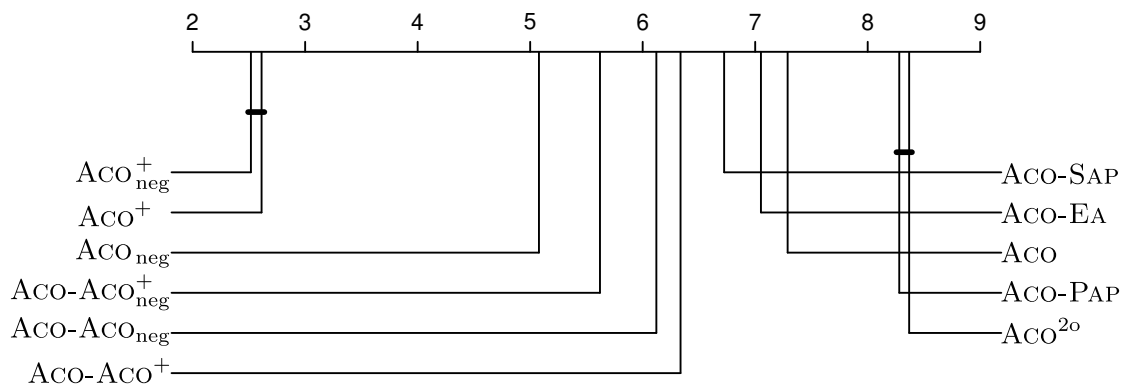


Fig. 4.2 Critical difference plot for all MDS instances

The global CD plot from Fig. 4.2 allows to make the following observations. All the six algorithm variants proposed in this work significantly improve over the remaining five algorithm variants, that is, over the baseline *MMAS* (*Aco*) and over the four considered negative learning variants from the literature. The three algorithm variants that make use of *Cplex* for generating the negative feedback outperform the other three variants with statistical significance. This shows the importance of the way in which the negative feedback is generated. In fact, the more accurate the negative feedback, the better the global performance of the algorithm.

Concerning the four negative learning mechanisms from the literature, it is shown that only *ACO-SAP* and *ACO-EA* are able to outperform the baseline *MMAS* algorithm. In contrast, *ACO-PAP* and *ACO^{2o}* perform significantly worse than the baseline *MMAS* algorithm. When comparing variants $\text{Aco}_{\text{neg}}^+$ and Aco_{neg} with Aco^+ , it can be observed that $\text{Aco}_{\text{neg}}^+$ has only a slight advantage over Aco^+ (which

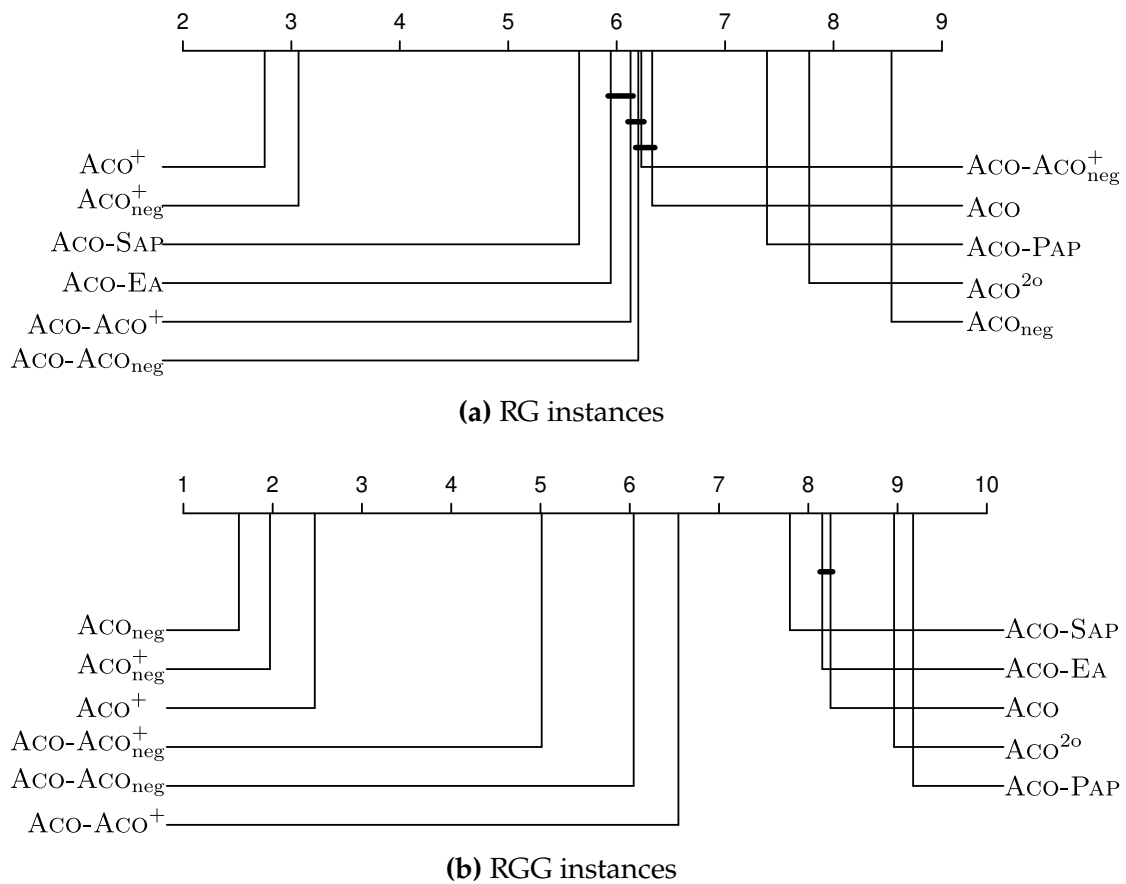
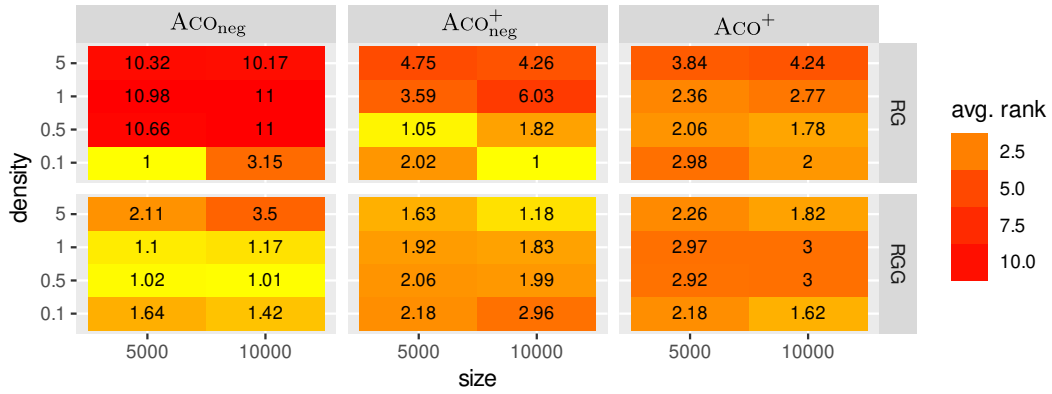


Fig. 4.3 Critical difference plots concerning different graph types

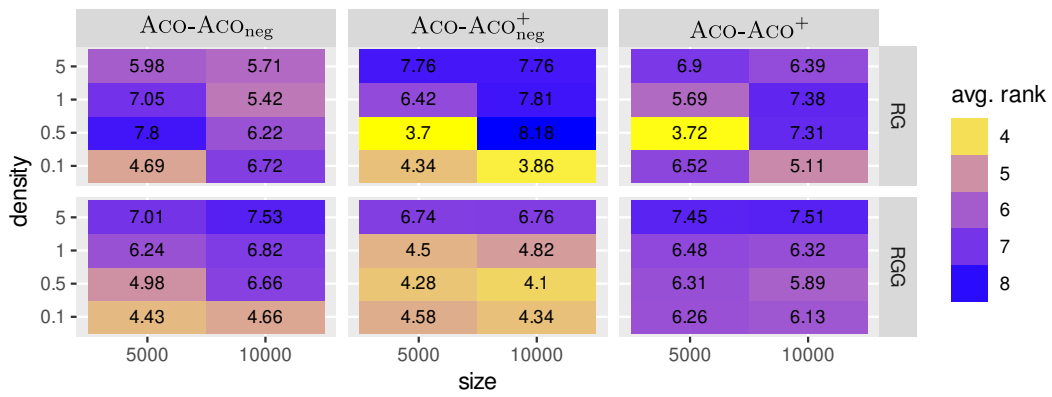
is not statistically significant). This means that, even though negative learning is useful, the additional positive feedback obtained by making use of solution S^{sub} for updating solutions S^{ib} and S^{rb} is very powerful.

The comparison of the three algorithms making use of *MMAS* as inner optimization algorithm ($\text{Aco-Aco}_{\text{neg}}^+$, $\text{Aco-Aco}_{\text{neg}}$ and Aco-Aco^+) shows a significant difference to the comparison concerning the three algorithms using *Cplex*. The two versions that make use of negative learning ($\text{Aco-Aco}_{\text{neg}}^+$ and $\text{Aco-Aco}_{\text{neg}}$) outperform the version without negative learning (Aco-Aco^+) with statistical significance. This can probably be explained by the lower quality of the positive feedback information, as solutions S^{sub} can be expected to be generally worse than solutions S^{sub} of the algorithm version using *Cplex*.

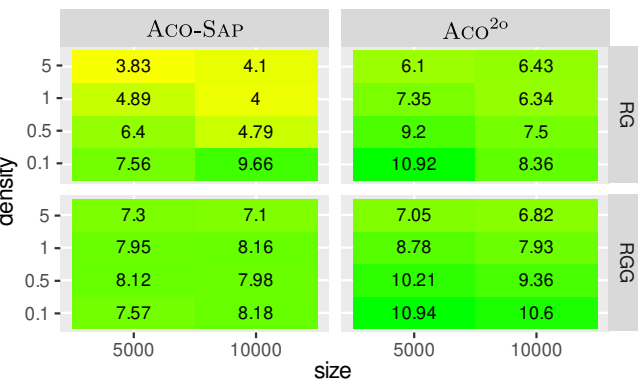
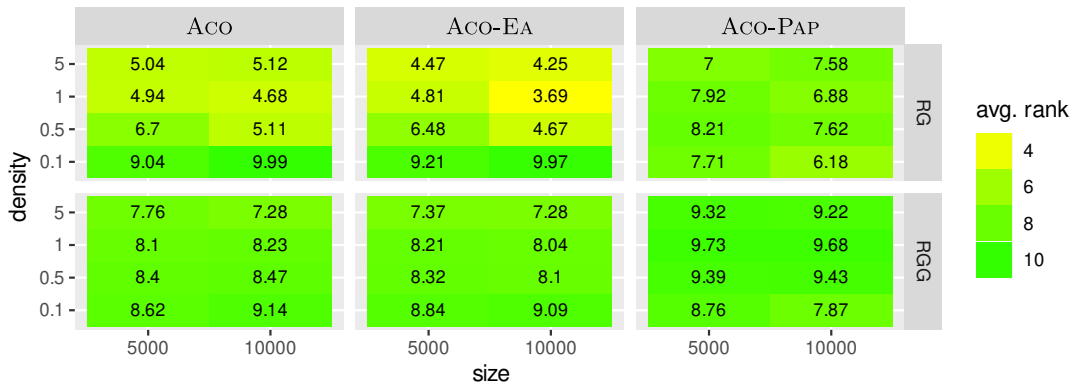
When looking at the results in a more fine-grained way, the following can be observed. Interestingly, the graph type seems to have a big influence on the relative performance of the algorithms. In the case of RGs, for example, Aco^+ is the clear winner of the comparison with $\text{Aco}_{\text{neg}}^+$ in second place. However, the really interesting aspect is that Aco_{neg} finishes last with statistical significance.



(a) Algorithms ACO_{neg}⁺, ACO_{neg}, and ACO⁺



(b) Algorithms ACO-ACO_{neg}⁺, ACO-ACO_{neg}, and ACO-ACO⁺



(c) Algorithms ACO, ACO-EA, ACO-PAP, ACO-PAP, and ACO^{2o}

Fig. 4.4 Heatmaps concerning the results for the MDS problem

This means that negative learning seems even to be harmful in the case of RGs. On the contrary, Aco_{neg} is the clear winner of the competition in the context of RGGs, with $\text{Aco}_{\text{neg}}^+$ finishing in second place (with statistical significance), and Aco^+ only in third place. This means that, in the case of RGGs, negative learning is much more important than the additional positive feedback provided by solution S^{sub} , which even seems harmful. Another interesting aspect is that, in the context of RGs, two negative learning versions from the literature (ACO-SAP and ACO-EA) clearly outperform our proposed negative learning variants using MMAS .

The heatmaps from Fig. 4.4 also indicate some interesting tendencies. Negative learning in the context of our algorithm variants $\text{Aco}_{\text{neg}}^+$, Aco_{neg} , $\text{Aco-Aco}_{\text{neg}}^+$ and $\text{Aco-Aco}_{\text{neg}}$ seems to gain importance with an increasing sparsity of the graphs. On the other side, in the context of RGs, it is clearly shown that the relative quality of ACO-SAP and ACO-EA grows with increasing graph size (number of vertices) and with increasing density.

4.7.3 Search Trajectory Network Analysis

As in Section 3.7.3, we provide STN plots in order to see if additional details of the algorithms' behavior can be identified. There are ten STN plots in this sub-section. Figure 4.5 to Fig. 4.9 present STN plots for problem instance 5000.rgg.0.1.0 and Fig. 4.10 to Fig. 4.14 present the ones for problem instance 5000.rgg.1.0.1. Note that the two problem instances differ in their edge densities.

These STN plots were made according to the procedure proposed in [127] and the corresponding R scripts provided in <https://github.com/gabro8a/STNs.git>. All data for these STN plots was obtained by applying each of the 11 algorithms to problem instances 5000.rgg.0.1.0 and 5000.rgg.1.0.1 ten times with the same parameter value settings used for the final experimental evaluation. We applied 98% *search space partitioning* in order to obtain the STN plots in this chapter. See Section 3.7.3 for the definition of a STN and the description of a STN plot.

Figure 4.5 shows the STN plot with the trajectories of the baseline Aco algorithm and the ones from our negative learning variants, Aco_{neg} and $\text{Aco}_{\text{neg}}^+$. The STN plot shows that there are many locations of best-found solutions of the same quality (red dots). In fact, note that all trajectories of both negative learning variants— Aco_{neg} and $\text{Aco}_{\text{neg}}^+$ —find best-found solutions. Comparing the trajectories from these two variants, we can observe that the trajectories of Aco_{neg} are longer than the ones of $\text{Aco}_{\text{neg}}^+$. Interestingly, the trajectories of the baseline Aco variant are concentrated in three areas, and none of them can find any best-found solution.

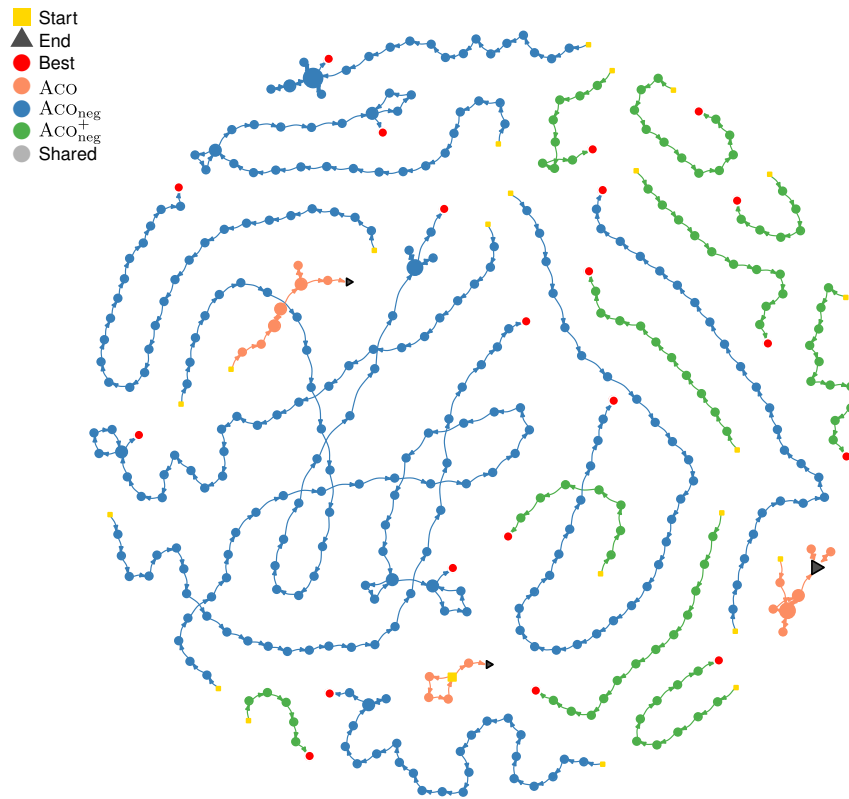


Fig. 4.5 Search trajectory network with the trajectories of Aco , Aco_{neg} , and Aco_{neg}^+ applied to problem instance 5000.rgg.0.1.0

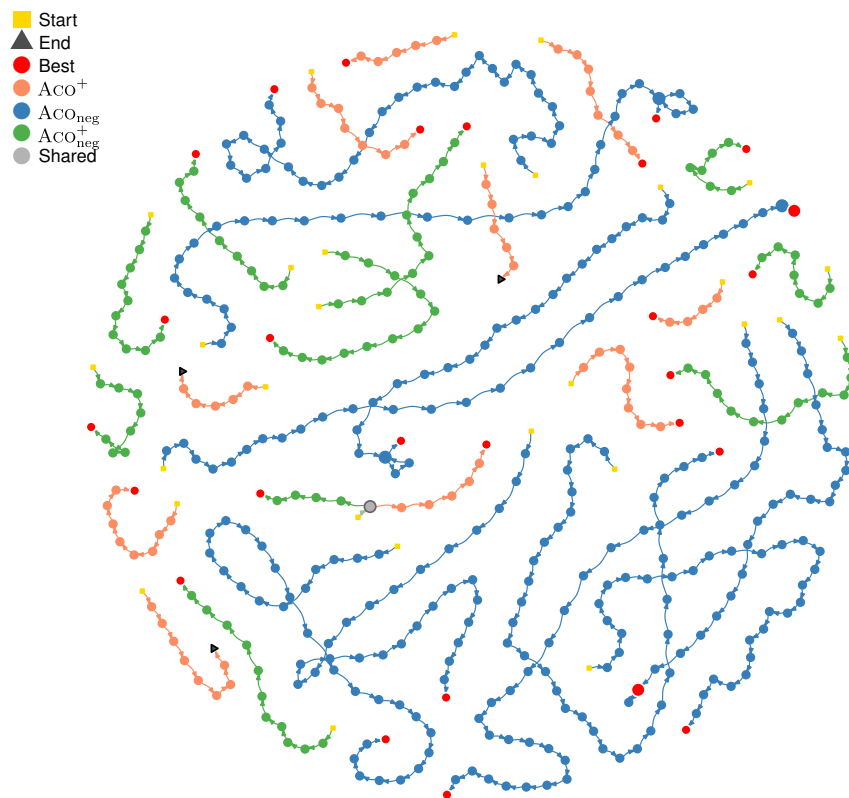


Fig. 4.6 Search trajectory network with the trajectories of Aco^+ , Aco_{neg} , and Aco_{neg}^+ applied to problem instance 5000.rgg.0.1.0

Figure 4.6 shows the STN plot concerning the three negative learning variants— Aco^+ , Aco_{neg} , and $\text{Aco}_{\text{neg}}^+$ —that use CPLEX as their additional algorithmic component. This plot shows that while all trajectories of Aco_{neg} and $\text{Aco}_{\text{neg}}^+$ can find best-found solutions, not all trajectories of Aco^+ can accomplish the same achievement. Comparing the length of the trajectories in Fig. 4.6, we can see that the ones from Aco_{neg} are the longest, followed by the ones from $\text{Aco}_{\text{neg}}^+$ and Aco^+ .

Figure 4.7 presents an STN plot that compares the search characteristics of $\text{Aco}_{\text{neg}}^+$ —our best performing negative learning Aco variant for problem instance 5000.rgg.0.1.0—to the ones of two other variants from our proposal that use *MMAS* as their additional algorithmic component. This plot shows that no trajectories of $\text{Aco-Aco}_{\text{neg}}$ or $\text{Aco-Aco}_{\text{neg}}^+$ can find any best-found solution as achieved by the ones from $\text{Aco}_{\text{neg}}^+$. Furthermore, we can see that several trajectories from these variants show *overlaps* in several locations within their own paths. Note that most of these locations are quite far from the locations of the final solutions that they found, which implies that these locations are areas of *local optima*, respectively *plateaus*. This fact suggests that $\text{Aco-Aco}_{\text{neg}}$ and $\text{Aco-Aco}_{\text{neg}}^+$ can escape these areas and significantly improve their solutions over

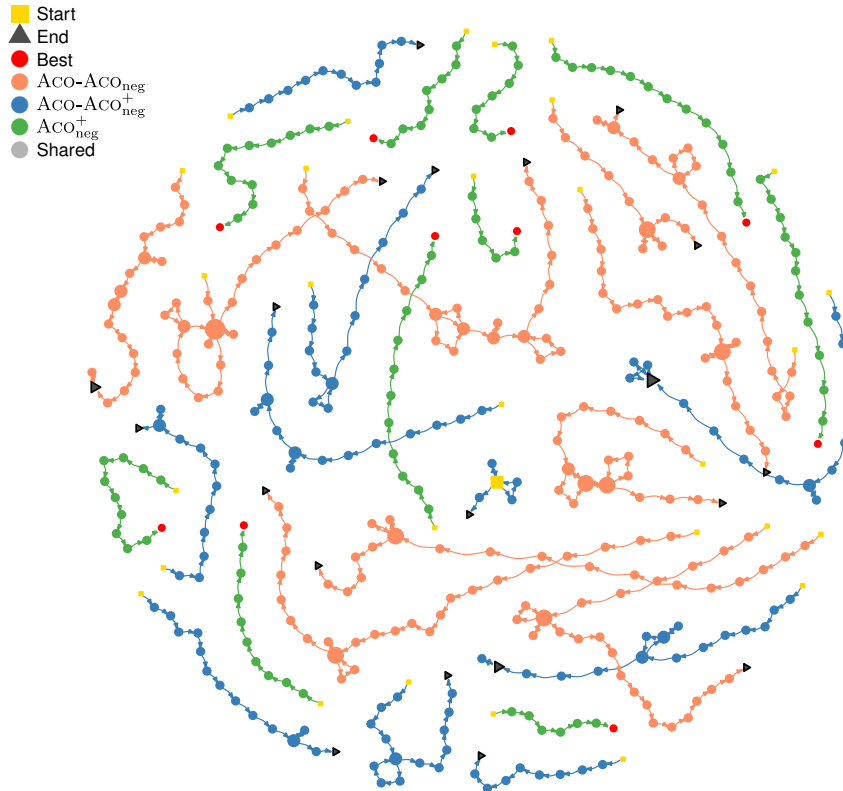


Fig. 4.7 Search trajectory network with the trajectories of $\text{Aco-Aco}_{\text{neg}}$, $\text{Aco-Aco}_{\text{neg}}^+$, and $\text{Aco}_{\text{neg}}^+$ applied to problem instance 5000.rgg.0.1.0

time. We can see that trajectories of the baseline Aco in Fig. 4.5 also show the same behaviour. However, we can notice that these trajectories do not improve significantly (in terms of number of steps) after escaping from the areas of local optima. Combined with the comparison of their numerical results in Table 4.5 and Table 4.9, this STN feature comparison clearly shows that both $\text{Aco-Aco}_{\text{neg}}$ and $\text{Aco-Aco}_{\text{neg}}^+$ perform significantly better than the baseline Aco.

We can also identify similar overlapping behaviour in the trajectories of the competitor negative learning Aco variants in Fig. 4.8 and Fig. 4.9. Notice that none of them can find a best-found solution as accomplished by our best-performing negative learning Aco variant $\text{Aco}_{\text{neg}}^+$. Moreover, we can observe that the trajectories of these algorithms show relatively more overlaps than the ones from $\text{Aco-Aco}_{\text{neg}}$ and $\text{Aco-Aco}_{\text{neg}}^+$ in Fig. 4.7. The comparison of the numerical results in Table 4.5 and Table 4.9 also confirms that they perform worse than our negative learning Aco variants, $\text{Aco-Aco}_{\text{neg}}$ and $\text{Aco-Aco}_{\text{neg}}^+$. The trajectories of our second-best negative learning Aco variant— Aco_{neg} —in Fig. 4.5 or Fig. 4.6 also show a small degree of overlapping traits, yet all of them can find best-found solutions. The issue of overlaps indicates the effectiveness of our negative learning Aco approach, given the fact that no trajectory of $\text{Aco}_{\text{neg}}^+$ exhibits any

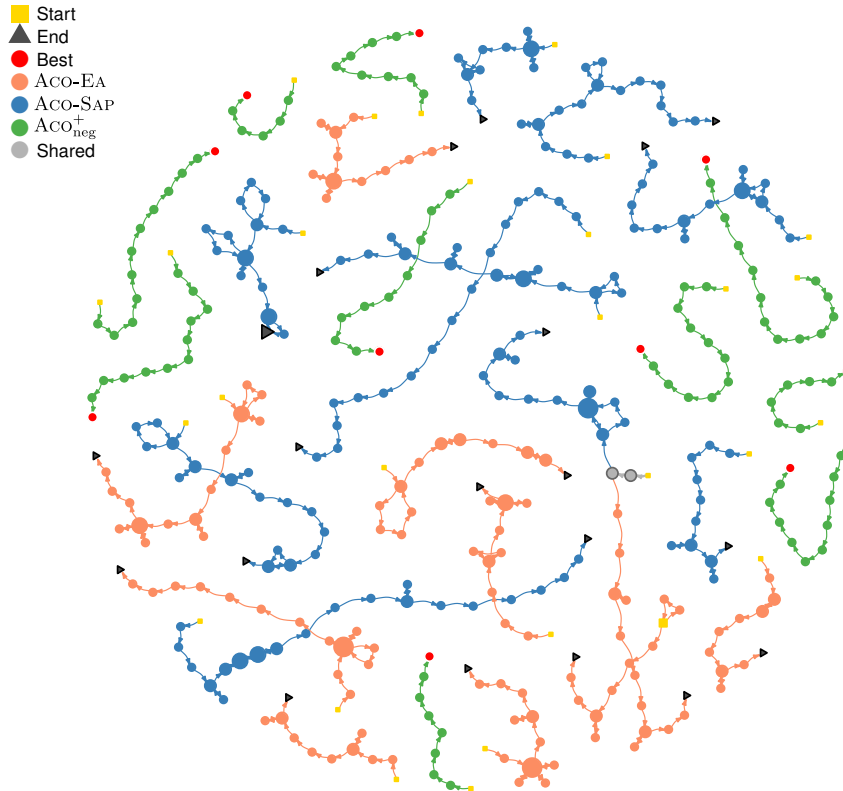


Fig. 4.8 Search trajectory network with the trajectories of ACO-EA, ACO-SAP, and $\text{Aco}_{\text{neg}}^+$ applied to problem instance 5000.rgg.0.1.0

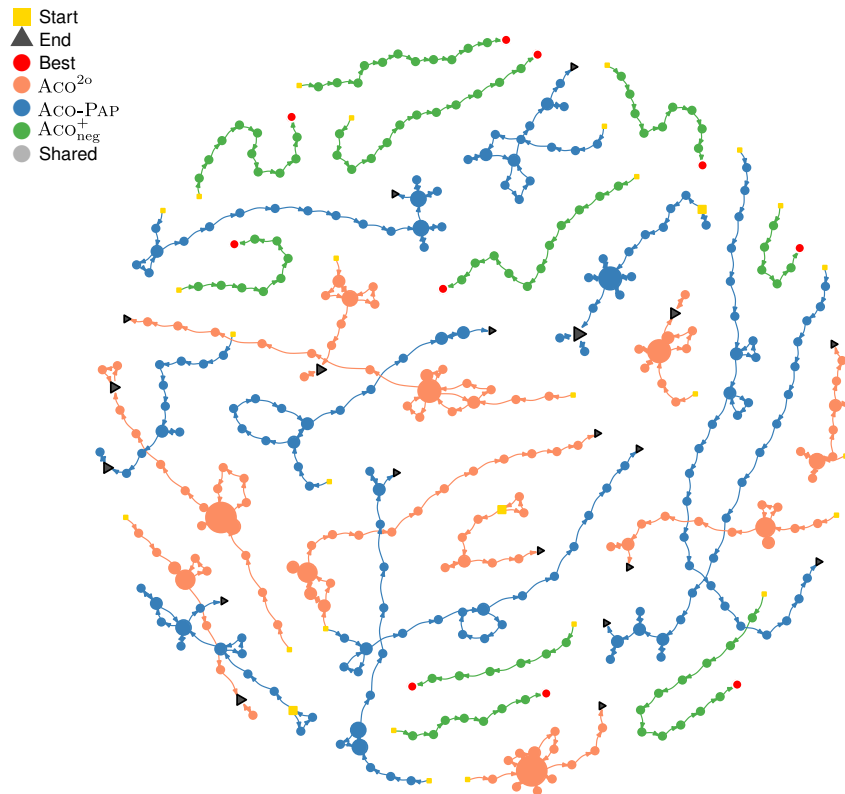


Fig. 4.9 Search trajectory network with the trajectories of ACO^{2o} , ACO-PAP , and $\text{ACO}^+_{\text{neg}}$ applied to problem instance 5000.rgg.0.1.0

overlaps, combined with the fact that all of them find a best-found solution, is strong evidence for the effectiveness of our negative learning mechanism in the context of escaping the areas of local optima. Overall, this problem instance—5000.rgg.0.1.0—has provided a unique example where we can contrast and evaluate the efficiency of the negative learning mechanism for escaping from local minima.

Figure 4.10 to Fig. 4.14 show the STN plots for the algorithm variants applied to problem instance 5000.rgg.1.0.1. We can see that these STN plots have different characteristics than those concerning problem instance 5000.rgg.0.1.0. In this problem instance, ACO_{neg} —instead of $\text{ACO}^+_{\text{neg}}$ —is the best performing negative learning Aco variant. Moreover, we can see that all algorithm variants need fewer steps to obtain their final results for this problem instance than when applied to problem instance 5000.rgg.0.1.0, as indicated by the reduced number of nodes between the start and end of each trajectory. This seems to happen because this problem instance has a higher edge density than problem instance 5000.rgg.1.0.1. Hence, vertices in the input graph are more connected to each other. Due to this condition, solutions are smaller (in terms of number of nodes) resulting in a more narrow range of different objective values. Consequently, the algorithms have

less opportunities to improve their solutions further.

Figure 4.10 shows an STN plot of three algorithm variants: ACO , ACO_{neg}^+ , and ACO_{neg} applied to problem instance 5000.rgg.1.0.1. In contrast to the plot in Fig. 4.5—showing the STN of the same group of algorithm variants applied to problem instance 5000.rgg.0.1.0—in which trajectory overlaps arise only in individual trajectories of ACO and ACO_{neg} , this plot shows several overlaps between trajectories of different algorithm variants in addition to the ones within their individual trajectories. Curiously, the trajectories of ACO_{neg} show fewer tendencies of being attracted to the common areas that attract trajectories of the two other algorithm variants. Figure 4.10 shows two trajectories of ACO_{neg} that have no overlap with any other trajectory. Two more trajectories of ACO_{neg} show overlaps but only with each other. Interestingly, the best-found solution in this problem instance is obtained by one of two trajectories of ACO_{neg} that have no overlap with any other trajectory. The numerical results comparison in Table 4.5 and Table 4.9 combined with this STN feature comparison shows that ACO_{neg} performs better than ACO_{neg}^+ in this problem instance. We believe that this is because ACO_{neg} only uses the negative learning information—and does not take advantage of the additional positive learning—provided by the additional algorithmic component,

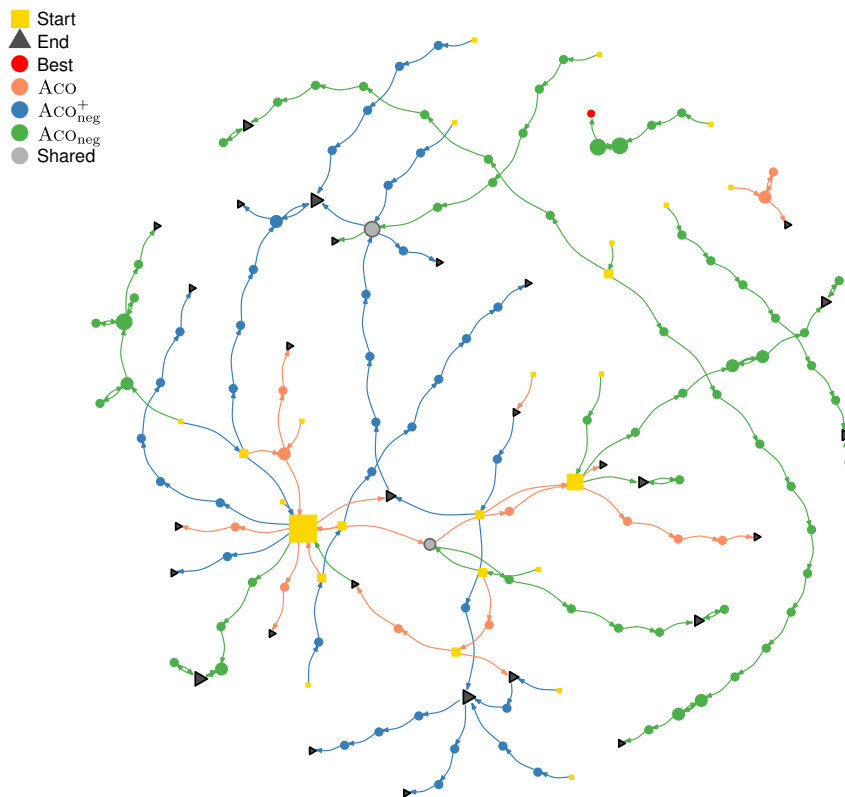


Fig. 4.10 Search trajectory network with the trajectories of ACO , ACO_{neg}^+ , and ACO_{neg} applied to problem instance 5000.rgg.1.0.1

CPLEX, which seems to be misleading. Moreover, Aco_{neg} does, therefore, not get trapped too often in local minima as the $\text{Aco}_{\text{neg}}^+$ does when following the strong positive feedback provided by CPLEX.

Figure 4.11 shows the STN plot concerning the algorithm variants— Aco^+ , $\text{Aco}_{\text{neg}}^+$, and Aco_{neg} —that use CPLEX as their additional algorithmic component. This plot contains an example of an even harmful effect of exclusively using the strong positive feedback mechanism from the additional algorithmic component. The variant Aco^+ , which only uses the positive feedback information provided by CPLEX without making use of the negative feedback information, has most of its trajectories connected and located in a common area. We know from Table 4.5 and Table 4.9 that Aco^+ has the worst numerical results in this group of algorithm variants. Combining this observation with its STN characteristics, we believe that the strong positive feedback mechanism from CPLEX has prevented Aco^+ from exploring other areas of the search space of this problem instance. In contrast, we can see that in Fig. 4.11 the trajectories of Aco_{neg} are less connected to other trajectories. Moreover, they are distributed in different places of the plot, and one of them finds the best-found solution. Hence we believe that in this comparison, Aco_{neg} has excelled in performance because it has a better

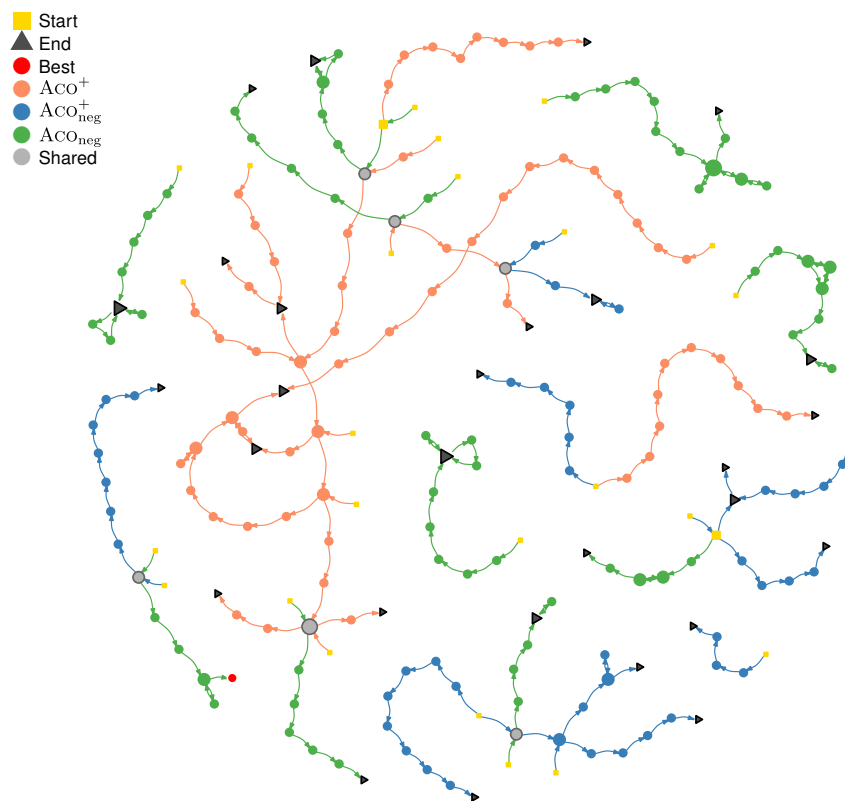


Fig. 4.11 Search trajectory network with the trajectories of Aco^+ , $\text{Aco}_{\text{neg}}^+$, and Aco_{neg} applied to problem instance 5000.rgg.1.0.1

exploration capability—due to giving more influence to the negative feedback mechanism—than the other two algorithm variants.

Figure 4.12 shows an STN plot containing the trajectories of Aco_{neg} —our best-performing negative learning Aco variant for this problem instance—with two other variants from our proposal— $\text{Aco-Aco}_{\text{neg}}$ and $\text{Aco-Aco}_{\text{neg}}^+$ —that use \mathcal{MMAS} as their additional algorithmic component. This plot shows a similar pattern to the one observed in Fig. 4.10. There are many overlaps of trajectories of all three algorithm variants, which indicates the presence of local minima that strongly attract them. In this comparison, we can see that the trajectories of Aco_{neg} also show a higher degree of overlap both among themselves and with the ones of other algorithm variants. In particular, we can see one trajectory of Aco_{neg} finding the best-found solution right after a shared node. Interestingly, we can also observe this behaviour in Fig. 4.13 and Fig. 4.14 where we compare the trajectories of Aco_{neg} to the ones from the competing negative learning Aco variants. Accordingly, we believe that in the context of these comparisons, Aco_{neg} performs significantly better than the other variants due to its better exploitation capability.

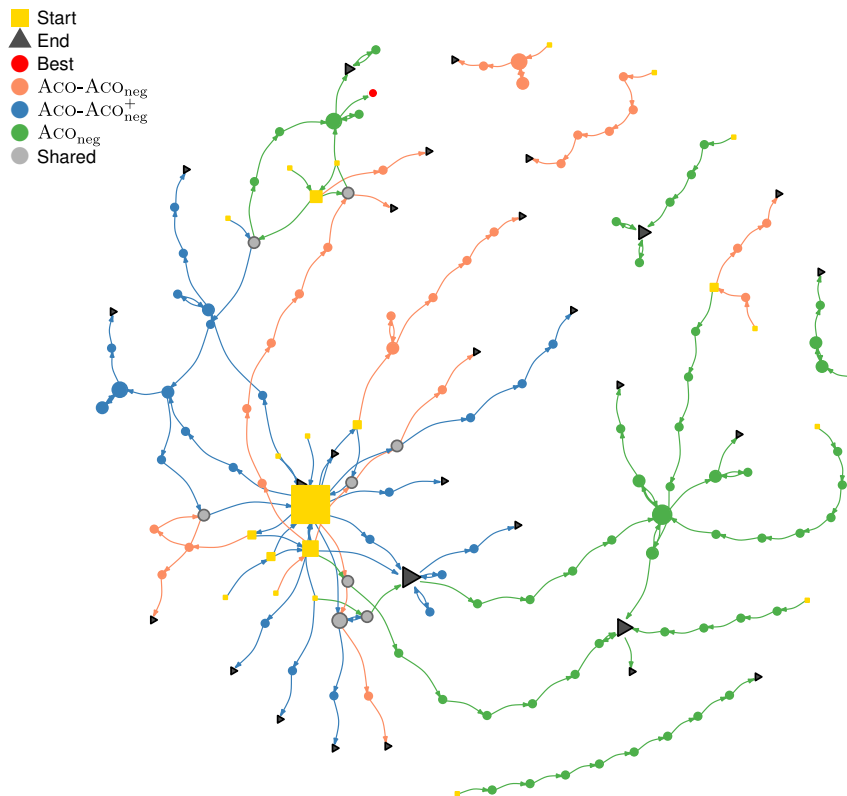


Fig. 4.12 Search trajectory network with the trajectories of $\text{Aco-Aco}_{\text{neg}}$, $\text{Aco-Aco}_{\text{neg}}^+$, and Aco_{neg} applied to problem instance 5000.rgg.1.0.1

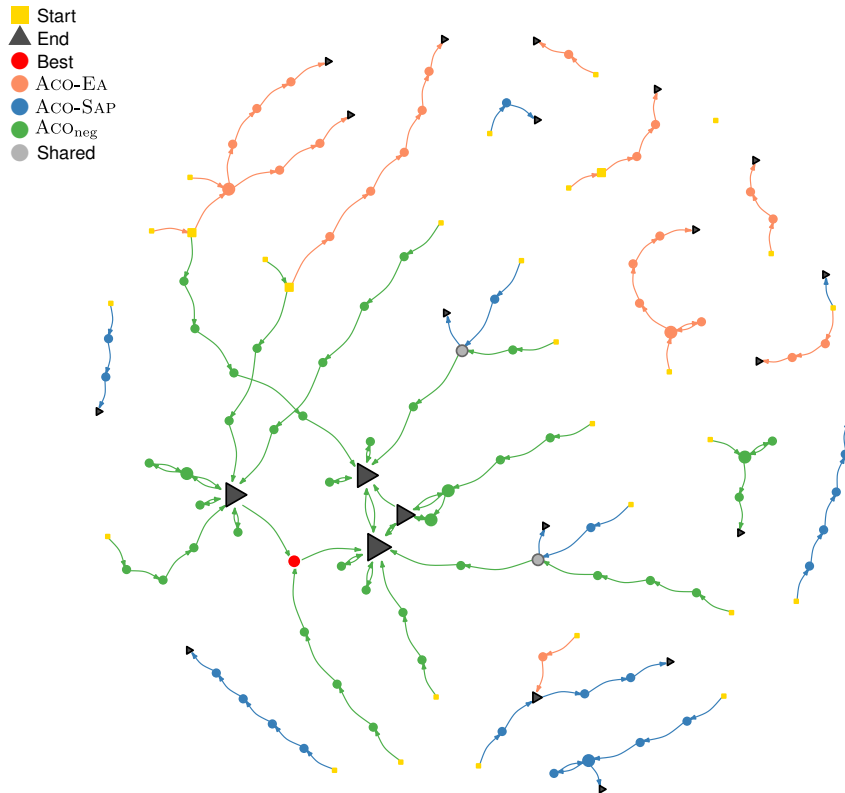


Fig. 4.13 Search trajectory network with the trajectories of ACO-EA, ACO-SAP, and ACO_{neg} applied to problem instance 5000.rgg.1.0.1

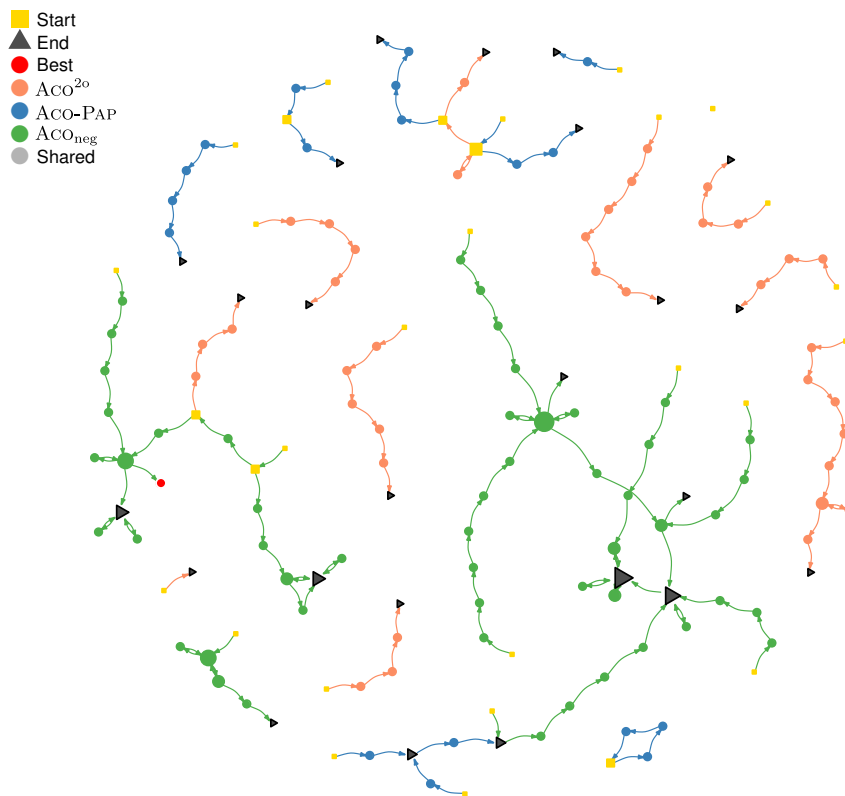


Fig. 4.14 Search trajectory network with the trajectories of ACO^{2o}, ACO-PAP, and ACO_{neg} applied to problem instance 5000.rgg.1.0.1

4.7.4 Comparison to the State of the Art

In the case of the MDS problem we chose for the purpose of a comparison to current state-of-the-art approaches one of the classical benchmark sets, which was also used in one of the latest published works [109]. This benchmark set is labeled UDG and consists of 120 graphs with numbers of vertices between 50 and 1000. For each of the six graph sizes, UDG contains graphs of two different densities. The benchmark set consists of 10 graphs per combination of graph size and graph density. Following the procedure from [109], we applied $\text{Aco}_{\text{neg}}^+$ 10 times with a time limit of 1000 CPU seconds for each application to each of the 120 instances of set UDG. Note that we did not specifically tune the parameters of $\text{Aco}_{\text{neg}}^+$. Instead, the same parameter values as in the previous section were used. The results are shown in a summarized way—as in [109]—in Table 4.3.

Table 4.3 MDS problem: summarized comparison to the state of the art. Competitor names are accompanied by publication year and the reference.

Instance Family	CC ² FS	FastMWDS	RLS _o	ScBppw	FastDS	$\text{Aco}_{\text{neg}}^+$
	2017 [105]	2018 [106]	2018 [107]	2019 [108]	2020 [109]	
V50U150	12.9	12.9	12.9	12.9	12.9	12.9
V50U200	9.4	9.4	9.4	9.4	9.4	9.4
V100U150	17.0	17.0	17.0	17.3	17.0	17.0
V100U200	10.4	10.4	10.4	10.6	10.4	10.4
V250U150	18.0	18.0	18.0	19.0	18.0	18.0
V250U200	10.8	10.8	10.8	11.5	10.8	10.8
V500U150	18.5	18.5	18.6	20.1	18.5	18.5
V500U200	11.2	11.2	11.2	12.4	11.2	11.2
V800U150	19.0	19.0	19.1	20.9	19.0	19.0
V800U200	11.7	11.7	11.9	12.6	11.8	11.7
V1000U150	19.1	19.1	19.2	21.3	19.1	19.1
V1000U200	12.0	12.0	12.0	13.0	12.0	12.0

In particular, each table row presents the results for the 10 instances of the respective instance family. For each of the six compared algorithms, the provided number is the average over the best solutions found for each of the 10 instances within 10 runs per instance. The best result per table row is indicated in bold face. Surprisingly, it can be observed that $\text{Aco}_{\text{neg}}^+$ matches the performance of the best two approaches. It is also worth mentioning that the five competitors of $\text{Aco}_{\text{neg}}^+$ in this table were all published since 2017 and are all based on local search. In particular, algorithm RLS_o [107] was shown to outperform all existing ACO and hyper-heuristic algorithms, which were the state-of-the-art before this recent start of focused research efforts on sophisticated local search algorithms. Concerning computation time, in [109] it is stated that CC²FS requires on average 0.21 s, FastMWDS requires 0.83 s, and FastDS requires 22.19 s to obtain the best

solutions of each run. $\text{Aco}_{\text{neg}}^+$ is slower by requiring on average 36.14 s.

4.8 CONCLUSIONS

Learning-based metaheuristics, including Aco, are generally based on *positive learning*. Nature, however, shows that learning from negative examples can be beneficial. Several works have been made over the last two decades to find a way to incorporate negative learning into Aco. Yet, only a few of them demonstrated that the proposed mechanism was truly useful. The goal of this work was thus to create and demonstrate the performance of a new negative learning mechanism for Aco. The main idea of our mechanism is that negative feedback should not be extracted directly from the main Aco algorithm. Instead, it should be generated by a separate algorithmic component. Indeed, after developing a new negative learning framework, we tested two algorithmic options for producing negative information: (1) using the mathematical programming solver CPLEX, and (2) using the baseline Aco algorithm, but with additional applications for solving sub-instances of the original problem instances.

All algorithm variants considered were applied to the MDS, an NP-hard combinatorial optimization problems from the class of subset selection problems. In addition, four negative learning mechanisms from the literature were implemented on top of the chosen baseline Aco algorithm in order to compare our proposals to existing approaches. The obtained results demonstrated that the proposed negative learning mechanism—particularly when using CPLEX to generate negative feedback information—outperforms the existing approaches from the literature. Furthermore, we demonstrated that, while negative learning is not useful for to all problem instances, it can be very useful for subsets of problem instances with specific characteristics. This is relevant in the context of the MDS problem, for example, for sparse graphs. Globally, it was also demonstrated that adding negative learning is generally not harmful because the globally best-performing algorithm variant employs negative learning. Finally, we were able to demonstrate that our globally best-performing algorithm variant can compete with current state-of-the-art algorithms for the MDS problem.

Table 4.4 Best results of all algorithms tested on MDS random graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rg.0.1.0	0.1	1049	1040	1054	1160	1122	1127	1148	1156	1150	1150	1177
5000.rg.0.1.1	0.1	1042	1035	1051	1151	1119	1124	1143	1151	1139	1149	1168
5000.rg.0.1.2	0.1	1049	1042	1053	1133	1114	1115	1129	1143	1141	1136	1155
5000.rg.0.1.3	0.1	1060	1053	1068	1163	1132	1138	1150	1157	1154	1159	1177
5000.rg.0.1.4	0.1	1052	1044	1062	1156	1122	1123	1144	1153	1146	1144	1173
5000.rg.0.1.5	0.1	1041	1033	1046	1142	1112	1118	1131	1147	1136	1136	1162
5000.rg.0.1.6	0.1	1060	1054	1071	1165	1139	1141	1152	1168	1155	1157	1176
5000.rg.0.1.7	0.1	1051	1044	1058	1148	1120	1124	1133	1149	1141	1146	1164
5000.rg.0.1.8	0.1	1059	1051	1063	1158	1122	1129	1142	1157	1148	1150	1170
5000.rg.0.1.9	0.1	1055	1047	1063	1158	1125	1129	1137	1155	1138	1144	1179
5000.rg.0.5.0	0.5	340	358	346	354	347	354	348	354	352	353	358
5000.rg.0.5.1	0.5	341	359	345	355	349	357	351	355	356	355	356
5000.rg.0.5.2	0.5	339	359	345	355	349	353	350	354	354	355	356
5000.rg.0.5.3	0.5	340	356	344	353	350	356	347	353	352	355	356
5000.rg.0.5.4	0.5	340	360	347	355	351	356	349	355	355	356	356
5000.rg.0.5.5	0.5	338	357	344	353	348	352	350	352	351	354	354
5000.rg.0.5.6	0.5	340	359	347	353	351	355	350	352	353	355	355
5000.rg.0.5.7	0.5	338	357	342	352	349	354	349	354	353	354	355
5000.rg.0.5.8	0.5	338	358	345	349	349	355	350	354	352	355	357
5000.rg.0.5.9	0.5	343	357	349	357	353	357	350	357	356	358	358
5000.rg.1.0.0	1.0	209	216	209	211	210	210	212	209	210	212	209
5000.rg.1.0.1	1.0	208	217	209	211	210	207	210	208	211	211	211
5000.rg.1.0.2	1.0	210	218	208	210	211	211	210	210	210	210	210
5000.rg.1.0.3	1.0	207	218	208	210	210	212	210	210	211	211	210
5000.rg.1.0.4	1.0	209	216	208	209	210	209	208	209	211	211	211
5000.rg.1.0.5	1.0	208	216	209	210	208	212	211	211	210	211	211
5000.rg.1.0.6	1.0	208	213	209	211	210	211	210	210	210	211	212
5000.rg.1.0.7	1.0	208	216	209	209	210	212	210	210	210	211	212
5000.rg.1.0.8	1.0	209	217	210	210	210	210	209	210	210	211	212
5000.rg.1.0.9	1.0	207	216	208	209	210	211	210	210	210	211	210

Continuation of Table 4.4 Best results of all algorithms tested on MDS random graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rg.5.0.0	5.0	59	62	60	60	60	61	60	60	60	61	60
5000.rg.5.0.1	5.0	60	62	60	60	61	61	60	60	60	60	60
5000.rg.5.0.2	5.0	60	61	60	60	60	60	61	60	60	61	60
5000.rg.5.0.3	5.0	60	60	60	60	60	61	60	60	60	60	60
5000.rg.5.0.4	5.0	60	61	59	60	61	60	60	59	60	60	60
5000.rg.5.0.5	5.0	61	62	60	61	60	60	61	60	60	61	61
5000.rg.5.0.6	5.0	60	61	60	60	61	61	60	60	60	60	60
5000.rg.5.0.7	5.0	60	61	61	61	60	61	61	60	61	61	61
5000.rg.5.0.8	5.0	60	62	60	60	61	60	61	60	59	61	60
5000.rg.5.0.9	5.0	60	62	60	60	61	60	61	60	60	60	60

Table 4.5 Best results of all algorithms tested on MDS random geometric graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rgg.0.1.0	0.1	991	991	991	1042	1022	1020	1034	1041	1036	1042	1055
5000.rgg.0.1.1	0.1	1000	1000	1000	1054	1030	1028	1043	1061	1051	1057	1072
5000.rgg.0.1.2	0.1	990	989	990	1043	1023	1023	1036	1040	1042	1043	1059
5000.rgg.0.1.3	0.1	993	993	993	1039	1017	1017	1028	1040	1037	1039	1055
5000.rgg.0.1.4	0.1	996	996	996	1057	1033	1034	1047	1056	1053	1059	1070
5000.rgg.0.1.5	0.1	997	997	997	1054	1028	1029	1042	1053	1046	1051	1066
5000.rgg.0.1.6	0.1	999	999	999	1057	1030	1031	1047	1061	1054	1057	1068
5000.rgg.0.1.7	0.1	996	995	995	1055	1025	1029	1043	1052	1049	1056	1064
5000.rgg.0.1.8	0.1	995	995	995	1038	1020	1017	1028	1038	1035	1035	1051
5000.rgg.0.1.9	0.1	1004	1004	1004	1055	1031	1033	1044	1047	1051	1053	1061

Continuation of Table 4.5 Best results of all algorithms tested on MDS random geometric graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rgg.0.5.0	0.5	229	226	232	268	257	258	265	267	266	268	269
5000.rgg.0.5.1	0.5	228	225	229	267	258	259	268	267	270	273	273
5000.rgg.0.5.2	0.5	228	226	231	267	260	259	266	265	264	269	271
5000.rgg.0.5.3	0.5	228	225	230	264	257	258	264	264	266	271	272
5000.rgg.0.5.4	0.5	228	224	229	268	257	260	262	264	266	270	271
5000.rgg.0.5.5	0.5	227	224	229	266	258	261	263	268	270	269	272
5000.rgg.0.5.6	0.5	227	225	229	267	258	261	264	265	267	270	271
5000.rgg.0.5.7	0.5	228	225	231	266	255	259	263	265	266	268	270
5000.rgg.0.5.8	0.5	230	227	231	269	259	261	263	265	266	271	270
5000.rgg.0.5.9	0.5	230	225	234	271	260	264	267	273	271	275	276
5000.rgg.1.0.0	1.0	121	120	124	142	139	143	141	144	141	144	142
5000.rgg.1.0.1	1.0	121	119	123	147	143	144	146	147	147	150	148
5000.rgg.1.0.2	1.0	123	121	127	146	143	147	145	147	147	151	151
5000.rgg.1.0.3	1.0	121	119	124	145	142	143	143	148	144	148	147
5000.rgg.1.0.4	1.0	120	120	121	143	140	142	142	145	144	145	145
5000.rgg.1.0.5	1.0	121	120	125	143	140	144	142	144	144	147	145
5000.rgg.1.0.6	1.0	121	120	124	146	140	143	143	146	143	147	144
5000.rgg.1.0.7	1.0	122	120	125	144	140	142	143	144	143	146	146
5000.rgg.1.0.8	1.0	122	121	124	146	143	143	146	146	146	148	146
5000.rgg.1.0.9	1.0	121	119	125	144	139	141	146	144	145	147	147
5000.rgg.5.0.0	5.0	25	25	26	30	32	31	32	32	32	33	32
5000.rgg.5.0.1	5.0	25	25	26	32	33	33	33	33	32	33	33
5000.rgg.5.0.2	5.0	25	25	26	32	31	32	31	32	32	31	32
5000.rgg.5.0.3	5.0	25	25	26	32	32	31	31	33	32	33	32
5000.rgg.5.0.4	5.0	25	25	26	31	32	31	30	32	32	32	31
5000.rgg.5.0.5	5.0	25	25	26	32	31	32	32	32	31	32	32
5000.rgg.5.0.6	5.0	26	25	26	32	31	32	32	32	31	33	31
5000.rgg.5.0.7	5.0	25	25	26	32	31	32	32	32	33	32	31
5000.rgg.5.0.8	5.0	26	26	26	32	32	33	33	33	33	34	32
5000.rgg.5.0.9	5.0	26	26	26	31	31	32	31	30	32	32	32

Table 4.6 Best results of all algorithms tested on MDS random graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rg.0.1.0	0.1	1297	1352	1321	1431	1365	1396	1379	1431	1432	1395	1416
10000.rg.0.1.1	0.1	1297	1352	1330	1429	1359	1399	1385	1428	1432	1389	1420
10000.rg.0.1.2	0.1	1300	1352	1337	1438	1369	1407	1393	1437	1438	1404	1429
10000.rg.0.1.3	0.1	1295	1345	1330	1425	1364	1394	1381	1428	1426	1387	1421
10000.rg.0.1.4	0.1	1299	1354	1334	1436	1368	1403	1394	1439	1438	1402	1425
10000.rg.0.1.5	0.1	1285	1335	1322	1411	1346	1384	1370	1419	1417	1377	1409
10000.rg.0.1.6	0.1	1288	1339	1325	1424	1357	1385	1379	1429	1427	1396	1407
10000.rg.0.1.7	0.1	1296	1351	1333	1430	1363	1402	1387	1429	1429	1401	1413
10000.rg.0.1.8	0.1	1290	1349	1325	1426	1359	1396	1380	1424	1425	1386	1416
10000.rg.0.1.9	0.1	1293	1342	1323	1423	1360	1389	1376	1423	1420	1389	1411
10000.rg.0.5.0	0.5	421	440	422	426	427	427	427	425	426	428	428
10000.rg.0.5.1	0.5	422	437	422	425	428	425	429	424	424	426	425
10000.rg.0.5.2	0.5	422	440	424	427	427	425	426	427	421	428	428
10000.rg.0.5.3	0.5	416	438	421	425	427	425	426	425	427	427	428
10000.rg.0.5.4	0.5	423	440	423	427	427	428	428	426	427	430	430
10000.rg.0.5.5	0.5	421	438	421	425	426	426	423	425	423	427	426
10000.rg.0.5.6	0.5	421	441	421	425	428	428	428	425	425	430	429
10000.rg.0.5.7	0.5	422	441	424	429	430	429	430	428	426	430	431
10000.rg.0.5.8	0.5	418	437	422	426	427	425	422	423	424	426	425
10000.rg.0.5.9	0.5	422	441	422	426	424	426	427	426	424	428	427
10000.rg.1.0.0	1.0	249	259	249	251	252	250	250	250	250	250	251
10000.rg.1.0.1	1.0	252	262	250	252	253	252	253	250	250	253	252
10000.rg.1.0.2	1.0	251	261	250	251	251	251	250	251	250	250	250
10000.rg.1.0.3	1.0	252	262	249	250	252	251	251	251	251	251	253
10000.rg.1.0.4	1.0	250	260	251	251	253	251	253	250	251	252	252
10000.rg.1.0.5	1.0	252	261	250	253	255	252	253	250	251	253	252
10000.rg.1.0.6	1.0	250	261	250	251	252	251	252	250	251	250	252
10000.rg.1.0.7	1.0	251	261	250	252	253	252	252	249	250	253	252
10000.rg.1.0.8	1.0	250	260	250	252	250	250	252	251	250	253	251
10000.rg.1.0.9	1.0	251	261	249	250	251	249	252	250	249	252	251

Continuation of Table 4.6 Best results of all algorithms tested on MDS random graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rg.5.0.0	5.0	70	71	70	70	70	71	70	70	70	71	70
10000.rg.5.0.1	5.0	70	71	69	70	71	70	70	70	70	70	70
10000.rg.5.0.2	5.0	70	72	70	70	70	70	70	70	70	70	71
10000.rg.5.0.3	5.0	70	71	70	70	70	70	70	70	70	70	70
10000.rg.5.0.4	5.0	70	72	70	70	70	69	70	70	70	70	70
10000.rg.5.0.5	5.0	69	71	69	69	70	70	70	70	70	70	69
10000.rg.5.0.6	5.0	70	71	70	69	70	70	70	70	70	71	70
10000.rg.5.0.7	5.0	70	71	70	70	70	70	70	70	70	70	70
10000.rg.5.0.8	5.0	70	71	70	70	70	70	70	70	70	70	70
10000.rg.5.0.9	5.0	70	72	70	69	71	70	70	70	70	70	70

Table 4.7 Best results of all algorithms tested on MDS random geometric graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rgg.0.1.0	0.1	1055	1049	1052	1225	1170	1168	1211	1225	1219	1217	1234
10000.rgg.0.1.1	0.1	1055	1048	1048	1229	1166	1179	1210	1232	1225	1219	1233
10000.rgg.0.1.2	0.1	1051	1047	1049	1220	1167	1164	1197	1216	1215	1209	1230
10000.rgg.0.1.3	0.1	1054	1048	1050	1229	1170	1174	1206	1223	1217	1209	1242
10000.rgg.0.1.4	0.1	1052	1047	1047	1223	1166	1162	1202	1220	1212	1213	1224
10000.rgg.0.1.5	0.1	1051	1046	1045	1233	1166	1173	1204	1219	1218	1215	1238
10000.rgg.0.1.6	0.1	1047	1042	1041	1220	1161	1159	1191	1214	1206	1217	1227
10000.rgg.0.1.7	0.1	1056	1049	1052	1221	1161	1167	1195	1217	1215	1207	1234
10000.rgg.0.1.8	0.1	1058	1052	1053	1233	1170	1172	1205	1225	1227	1227	1241
10000.rgg.0.1.9	0.1	1058	1053	1051	1232	1171	1174	1212	1224	1226	1224	1228

Continuation of Table 4.7 Best results of all algorithms tested on MDS random geometric graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	ACO-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
10000.rgg.0.5.0	0.5	248	244	264	302	294	301	299	304	302	304	306
10000.rgg.0.5.1	0.5	247	242	261	302	290	295	297	302	303	303	303
10000.rgg.0.5.2	0.5	246	240	263	302	293	298	301	301	301	303	304
10000.rgg.0.5.3	0.5	247	241	260	297	290	295	293	298	301	297	301
10000.rgg.0.5.4	0.5	247	243	263	300	291	298	294	300	299	302	301
10000.rgg.0.5.5	0.5	247	242	260	300	291	295	294	297	296	299	301
10000.rgg.0.5.6	0.5	248	243	263	302	289	295	296	303	301	304	302
10000.rgg.0.5.7	0.5	246	240	260	303	289	298	297	299	298	303	306
10000.rgg.0.5.8	0.5	247	244	262	301	289	296	290	300	299	301	299
10000.rgg.0.5.9	0.5	246	241	259	301	288	296	295	299	299	301	302
10000.rgg.1.0.0	1.0	129	125	138	159	155	159	159	159	159	159	160
10000.rgg.1.0.1	1.0	127	125	138	159	154	158	159	159	158	159	157
10000.rgg.1.0.2	1.0	127	126	138	158	154	157	156	157	158	158	157
10000.rgg.1.0.3	1.0	128	127	138	157	155	155	155	157	156	160	158
10000.rgg.1.0.4	1.0	127	126	139	158	152	153	156	158	157	160	157
10000.rgg.1.0.5	1.0	128	126	139	159	154	157	155	157	160	162	159
10000.rgg.1.0.6	1.0	127	126	139	160	156	158	159	159	157	163	160
10000.rgg.1.0.7	1.0	128	125	139	159	154	156	157	158	159	160	158
10000.rgg.1.0.8	1.0	127	125	138	158	156	155	158	158	157	162	158
10000.rgg.1.0.9	1.0	127	127	139	159	153	156	155	157	158	156	156
10000.rgg.5.0.0	5.0	26	29	27	34	34	33	34	32	33	34	33
10000.rgg.5.0.1	5.0	26	29	27	34	33	34	34	35	34	35	34
10000.rgg.5.0.2	5.0	26	30	27	36	35	34	35	35	36	36	34
10000.rgg.5.0.3	5.0	26	30	27	34	33	34	34	34	34	36	33
10000.rgg.5.0.4	5.0	26	31	27	34	34	33	35	34	34	34	33
10000.rgg.5.0.5	5.0	26	29	27	35	34	34	35	35	34	35	34
10000.rgg.5.0.6	5.0	26	30	27	35	33	35	34	34	33	37	34
10000.rgg.5.0.7	5.0	26	30	27	35	32	35	34	35	35	35	35
10000.rgg.5.0.8	5.0	26	29	27	34	34	34	35	34	34	35	34
10000.rgg.5.0.9	5.0	26	29	27	34	34	33	34	35	35	34	33

Table 4.8 Average results of all algorithms tested on MDS random graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	ACO-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
5000.rg.0.1.0	0.1	1052.3	1043.3	1058.6	1169.7	1132.8	1135.6	1155.9	1170.4	1160.8	1163.1	1186.9
5000.rg.0.1.1	0.1	1046.3	1036.5	1057.7	1161.1	1129.0	1131.3	1147.8	1161.5	1153.2	1156.4	1177.2
5000.rg.0.1.2	0.1	1051.7	1044.4	1056.7	1153.1	1122.8	1124.3	1139.8	1154.5	1147.3	1147.1	1172.4
5000.rg.0.1.3	0.1	1063.0	1055.4	1075.1	1174.3	1142.1	1145.5	1163.3	1174.1	1166.7	1167.7	1189.1
5000.rg.0.1.4	0.1	1056.2	1046.8	1069.7	1168.9	1134.6	1135.4	1154.1	1168.3	1161.9	1162.5	1184.2
5000.rg.0.1.5	0.1	1045.5	1036.5	1056.9	1158.8	1121.3	1125.3	1143.1	1159.0	1149.8	1149.0	1177.8
5000.rg.0.1.6	0.1	1065.6	1057.0	1076.6	1175.7	1145.2	1149.7	1162.7	1178.3	1168.7	1168.1	1190.3
5000.rg.0.1.7	0.1	1054.2	1046.1	1062.2	1159.7	1128.8	1130.6	1144.7	1160.9	1153.7	1154.5	1174.7
5000.rg.0.1.8	0.1	1062.1	1054.2	1067.7	1166.2	1133.0	1136.4	1153.8	1171.3	1159.6	1160.2	1182.9
5000.rg.0.1.9	0.1	1058.9	1050.4	1070.3	1168.3	1133.3	1137.1	1152.7	1170.5	1159.4	1160.6	1187.0
5000.rg.0.5.0	0.5	344.2	363.5	348.9	357.4	353.2	357.8	352.9	357.3	356.5	358.8	359.9
5000.rg.0.5.1	0.5	345.1	365.0	350.1	358.0	353.6	359.9	353.8	358.3	358.4	359.8	360.7
5000.rg.0.5.2	0.5	343.7	364.9	349.4	357.9	353.2	358.3	353.6	357.4	357.5	359.5	360.6
5000.rg.0.5.3	0.5	344.4	363.3	349.3	357.4	353.5	359.0	353.1	357.2	357.5	359.4	360.8
5000.rg.0.5.4	0.5	345.7	365.1	349.8	358.1	353.9	359.8	354.0	358.0	357.9	359.9	361.4
5000.rg.0.5.5	0.5	343.1	362.9	348.0	355.7	351.6	357.4	352.0	355.8	355.8	357.9	359.2
5000.rg.0.5.6	0.5	344.8	364.1	350.0	358.4	354.4	359.3	353.9	357.7	358.0	359.9	360.6
5000.rg.0.5.7	0.5	343.6	363.2	348.1	356.3	352.6	357.6	352.6	356.8	356.0	358.5	359.9
5000.rg.0.5.8	0.5	343.2	363.6	348.6	356.7	352.5	358.2	352.8	356.3	356.5	358.2	360.0
5000.rg.0.5.9	0.5	347.2	366.9	352.4	360.6	357.1	361.6	356.4	360.2	359.8	362.2	363.9
5000.rg.1.0.0	1.0	211.8	220.0	211.4	213.2	213.1	213.9	213.4	212.8	212.5	214.7	214.2
5000.rg.1.0.1	1.0	211.4	220.1	210.8	212.5	213.4	213.4	212.7	212.1	212.3	214.0	213.7
5000.rg.1.0.2	1.0	212.0	220.6	211.2	212.7	213.7	213.7	213.4	212.8	212.5	214.4	213.8
5000.rg.1.0.3	1.0	212.5	221.4	211.7	213.0	213.6	214.3	213.3	212.9	213.0	214.4	214.0
5000.rg.1.0.4	1.0	211.2	220.0	210.2	211.7	213.1	212.7	212.0	212.0	212.4	213.2	212.9
5000.rg.1.0.5	1.0	211.4	220.3	211.0	212.4	213.0	214.0	213.3	212.3	212.5	213.8	214.0
5000.rg.1.0.6	1.0	212.5	219.5	211.4	213.0	213.8	213.7	213.1	212.8	212.7	214.2	214.1
5000.rg.1.0.7	1.0	211.4	220.6	210.8	212.3	213.1	213.4	212.8	212.2	212.5	214.3	214.1
5000.rg.1.0.8	1.0	211.7	220.8	211.5	213.1	213.8	214.2	213.3	212.8	212.8	214.4	214.4
5000.rg.1.0.9	1.0	211.2	220.4	210.4	211.8	213.2	213.1	212.3	212.2	212.1	213.9	213.2

Continuation of Table 4.8 Average results of all algorithms tested on MDS random graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rg.5.0.0	5.0	61.0	63.0	60.7	61.2	61.9	61.5	61.6	61.1	60.9	62.0	61.5
5000.rg.5.0.1	5.0	61.1	63.2	61.1	61.2	62.2	61.6	61.8	61.1	61.0	61.7	61.5
5000.rg.5.0.2	5.0	61.3	63.0	61.0	61.3	61.8	61.3	61.8	61.0	61.0	61.8	61.6
5000.rg.5.0.3	5.0	60.9	62.9	60.7	61.2	61.9	61.5	61.7	61.1	60.8	61.3	61.2
5000.rg.5.0.4	5.0	61.2	63.1	60.8	61.2	62.3	61.5	61.8	61.0	61.0	61.7	61.6
5000.rg.5.0.5	5.0	61.3	63.1	61.0	61.3	62.1	61.5	61.8	61.3	61.0	61.9	61.7
5000.rg.5.0.6	5.0	61.4	63.4	61.3	61.3	62.3	61.6	61.8	61.3	60.9	61.9	61.6
5000.rg.5.0.7	5.0	61.2	63.0	61.2	61.6	61.9	61.6	61.9	61.0	61.1	61.8	61.5
5000.rg.5.0.8	5.0	61.0	62.8	60.8	61.1	62.0	61.3	61.5	61.2	60.9	61.7	61.4
5000.rg.5.0.9	5.0	61.2	63.1	61.0	61.2	62.0	61.5	61.8	61.1	61.1	61.9	61.5

Table 4.9 Average results of all algorithms tested on MDS random geometric graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rgg.0.1.0	0.1	991.1	991.0	991.2	1049.2	1026.0	1025.4	1041.5	1051.3	1045.1	1051.7	1063.4
5000.rgg.0.1.1	0.1	1001.2	1000.3	1001.3	1064.6	1038.2	1037.5	1055.5	1067.0	1060.5	1066.0	1082.0
5000.rgg.0.1.2	0.1	991.8	990.1	991.4	1052.3	1026.4	1028.5	1040.9	1051.9	1047.2	1052.2	1068.2
5000.rgg.0.1.3	0.1	993.0	993.0	993.1	1046.9	1023.6	1022.0	1037.0	1049.2	1044.9	1048.1	1063.5
5000.rgg.0.1.4	0.1	996.4	996.0	996.4	1063.4	1039.1	1038.4	1054.2	1064.2	1061.2	1066.2	1081.4
5000.rgg.0.1.5	0.1	997.7	997.0	998.3	1060.9	1034.3	1034.7	1049.7	1061.2	1054.7	1058.9	1075.7
5000.rgg.0.1.6	0.1	999.1	999.0	999.0	1067.8	1040.0	1038.3	1055.6	1068.9	1062.8	1067.9	1082.0
5000.rgg.0.1.7	0.1	996.1	995.7	996.0	1064.4	1033.5	1033.6	1051.4	1064.7	1061.3	1063.6	1080.0
5000.rgg.0.1.8	0.1	995.2	995.0	995.2	1049.6	1026.2	1022.9	1038.3	1049.3	1044.5	1048.4	1063.7
5000.rgg.0.1.9	0.1	1004.6	1004.1	1004.4	1061.5	1037.9	1037.0	1050.0	1062.1	1057.5	1062.8	1075.2

Continuation of Table 4.9 Average results of all algorithms tested on MDS random geometric graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	ACO-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
5000.rgg.0.5.0	0.5	231.5	228.0	235.1	271.9	261.0	264.1	268.2	271.3	271.3	274.0	275.6
5000.rgg.0.5.1	0.5	230.3	227.4	231.8	275.8	263.8	264.3	271.1	275.1	274.7	277.2	278.8
5000.rgg.0.5.2	0.5	231.1	228.1	233.7	273.8	263.4	265.8	270.0	273.7	273.1	275.9	277.6
5000.rgg.0.5.3	0.5	230.4	227.4	233.4	272.4	261.9	264.0	268.1	271.5	271.8	274.6	276.7
5000.rgg.0.5.4	0.5	229.8	226.4	232.1	274.4	263.2	264.8	269.1	272.9	272.1	275.0	277.4
5000.rgg.0.5.5	0.5	229.1	225.9	231.9	272.5	262.4	264.5	269.3	273.8	273.2	275.7	277.7
5000.rgg.0.5.6	0.5	229.5	226.3	232.5	273.6	262.7	266.9	269.1	273.2	272.9	274.8	276.6
5000.rgg.0.5.7	0.5	230.6	227.7	233.6	271.7	261.3	263.7	267.1	272.3	271.4	272.9	275.8
5000.rgg.0.5.8	0.5	231.6	228.4	234.2	273.3	263.1	265.9	269.9	273.6	273.3	275.7	278.5
5000.rgg.0.5.9	0.5	231.9	227.9	237.7	277.6	265.8	268.9	273.0	277.3	277.5	279.6	280.9
5000.rgg.1.0.0	1.0	123.2	121.7	127.6	147.1	142.8	146.3	145.5	147.8	147.8	149.0	148.7
5000.rgg.1.0.1	1.0	122.9	120.7	126.6	151.2	145.9	147.4	149.1	150.7	150.6	152.9	151.2
5000.rgg.1.0.2	1.0	125.0	122.8	130.2	151.8	147.7	150.3	150.5	152.1	152.0	153.8	153.1
5000.rgg.1.0.3	1.0	123.1	121.0	127.7	149.6	144.9	147.7	147.8	150.3	149.3	152.2	150.8
5000.rgg.1.0.4	1.0	122.5	121.0	124.5	147.3	143.4	145.3	145.7	147.7	147.4	149.1	148.5
5000.rgg.1.0.5	1.0	123.4	121.5	127.7	148.9	144.8	147.3	146.8	149.0	147.7	150.7	149.1
5000.rgg.1.0.6	1.0	123.4	122.0	127.7	150.3	143.9	147.1	147.7	149.1	148.2	150.5	149.7
5000.rgg.1.0.7	1.0	123.5	121.2	127.9	148.1	144.3	146.0	147.0	148.2	148.5	150.0	149.4
5000.rgg.1.0.8	1.0	124.3	122.5	128.0	150.6	145.4	148.3	149.1	150.5	150.6	152.6	150.9
5000.rgg.1.0.9	1.0	123.0	121.3	128.1	149.6	145.0	147.7	148.7	149.5	150.0	151.3	150.3
5000.rgg.5.0.0	5.0	26.1	26.8	26.7	34.4	33.9	33.9	33.7	34.2	34.2	35.3	33.8
5000.rgg.5.0.1	5.0	26.1	26.6	26.9	34.9	34.5	34.7	34.8	34.7	34.4	36.1	34.4
5000.rgg.5.0.2	5.0	25.9	26.3	26.4	33.6	32.9	33.4	33.5	33.5	33.6	34.1	33.5
5000.rgg.5.0.3	5.0	26.1	26.8	26.6	33.9	33.7	33.6	34.2	34.2	34.1	35.1	33.7
5000.rgg.5.0.4	5.0	26.0	26.5	26.4	33.3	33.1	33.0	33.2	33.0	33.0	34.0	32.9
5000.rgg.5.0.5	5.0	26.3	26.7	26.9	33.6	33.5	33.7	33.6	33.4	33.4	34.6	33.4
5000.rgg.5.0.6	5.0	26.3	26.6	26.9	34.1	33.6	33.5	33.9	33.7	33.8	34.9	33.8
5000.rgg.5.0.7	5.0	26.3	26.9	26.6	34.0	33.1	33.4	33.6	33.8	34.0	34.4	33.6
5000.rgg.5.0.8	5.0	26.2	26.8	26.9	34.5	34.1	34.5	34.8	34.6	34.4	35.5	34.3
5000.rgg.5.0.9	5.0	26.4	26.5	26.4	33.7	32.7	33.2	33.2	33.1	33.1	33.9	33.5

Table 4.10 Average results of all algorithms tested on MDS random graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rg.0.1.0	0.1	1303.2	1364.8	1338.3	1440.7	1372.8	1409.3	1395.2	1440.2	1439.6	1403.9	1429.9
10000.rg.0.1.1	0.1	1304.3	1366.4	1339.2	1438.4	1372.9	1408.4	1394.2	1438.9	1437.5	1401.7	1430.7
10000.rg.0.1.2	0.1	1309.4	1371.0	1343.8	1448.4	1380.5	1417.9	1402.1	1446.6	1446.5	1410.3	1437.8
10000.rg.0.1.3	0.1	1302.3	1361.3	1337.5	1435.1	1372.2	1405.7	1391.7	1437.0	1435.7	1400.4	1428.2
10000.rg.0.1.4	0.1	1306.3	1368.9	1341.7	1446.4	1380.6	1414.3	1400.0	1446.6	1445.6	1411.6	1436.5
10000.rg.0.1.5	0.1	1293.7	1349.7	1328.5	1423.4	1360.6	1396.2	1380.8	1424.6	1424.2	1390.7	1419.4
10000.rg.0.1.6	0.1	1297.4	1359.0	1333.6	1434.7	1368.1	1402.3	1390.7	1435.9	1434.0	1401.2	1423.3
10000.rg.0.1.7	0.1	1303.0	1364.4	1338.1	1441.6	1376.7	1411.5	1396.7	1440.4	1437.9	1407.4	1430.6
10000.rg.0.1.8	0.1	1299.5	1360.1	1333.7	1434.7	1368.3	1405.9	1389.2	1433.2	1433.5	1396.5	1424.6
10000.rg.0.1.9	0.1	1298.6	1355.5	1332.0	1432.8	1368.9	1403.1	1389.2	1432.3	1430.3	1399.0	1422.3
10000.rg.0.5.0	0.5	424.7	444.3	425.6	429.5	432.9	431.0	431.5	429.4	429.3	431.3	431.8
10000.rg.0.5.1	0.5	425.0	443.9	425.5	428.5	431.9	429.9	431.5	428.4	429.0	431.6	430.5
10000.rg.0.5.2	0.5	425.7	444.8	426.8	430.3	433.4	431.2	432.4	430.1	429.7	432.6	433.1
10000.rg.0.5.3	0.5	424.7	444.7	425.0	429.3	431.9	429.6	431.3	428.5	428.7	431.2	431.1
10000.rg.0.5.4	0.5	425.8	445.3	426.7	430.3	433.8	431.8	432.5	429.8	429.9	432.5	432.4
10000.rg.0.5.5	0.5	424.5	443.1	424.8	428.6	430.9	429.3	430.0	428.3	427.7	430.7	430.2
10000.rg.0.5.6	0.5	426.0	446.2	425.9	430.1	432.7	430.9	432.2	430.0	430.3	432.8	432.7
10000.rg.0.5.7	0.5	427.7	446.6	427.9	432.0	434.8	432.5	433.5	431.9	431.4	433.8	434.1
10000.rg.0.5.8	0.5	425.1	443.6	424.8	428.6	431.3	429.4	430.3	428.1	428.4	431.0	430.7
10000.rg.0.5.9	0.5	426.7	445.1	425.9	429.8	432.5	430.7	432.0	429.6	429.9	431.9	432.0
10000.rg.1.0.0	1.0	253.2	264.2	252.0	253.3	255.0	253.6	254.0	252.5	252.7	254.2	253.6
10000.rg.1.0.1	1.0	255.2	265.3	252.8	253.8	256.4	254.4	255.5	253.5	253.4	255.6	255.2
10000.rg.1.0.2	1.0	253.9	265.1	252.3	253.1	254.6	253.6	254.9	253.1	253.0	254.0	253.9
10000.rg.1.0.3	1.0	254.6	264.2	252.7	253.3	255.5	254.0	254.5	253.2	253.2	254.7	254.5
10000.rg.1.0.4	1.0	254.1	264.5	252.8	253.9	255.7	254.1	255.6	253.0	253.3	254.8	254.8
10000.rg.1.0.5	1.0	255.1	264.9	253.0	254.3	256.4	254.2	255.6	253.1	253.5	255.7	255.5
10000.rg.1.0.6	1.0	254.0	264.4	252.0	253.1	255.0	253.8	255.1	253.1	253.2	254.3	253.8
10000.rg.1.0.7	1.0	254.3	265.0	252.7	253.8	255.4	254.3	255.1	253.2	253.5	255.2	254.9
10000.rg.1.0.8	1.0	253.6	263.4	251.6	253.2	254.8	253.1	254.5	252.6	252.5	254.4	253.7
10000.rg.1.0.9	1.0	253.5	263.8	251.8	252.6	254.2	252.9	254.5	251.9	252.4	254.0	253.5

Continuation of Table 4.10 Average results of all algorithms tested on MDS random graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rg.5.0.0	5.0	70.9	72.7	71.1	71.1	71.9	71.3	71.2	70.8	71.0	71.9	71.4
10000.rg.5.0.1	5.0	71.0	72.9	70.8	71.3	72.2	71.3	71.5	70.9	70.7	71.9	71.5
10000.rg.5.0.2	5.0	70.9	73.0	70.8	71.2	71.8	71.3	71.3	71.0	70.9	71.7	71.7
10000.rg.5.0.3	5.0	70.9	72.6	70.8	71.1	71.8	71.0	71.6	70.9	70.7	71.8	71.5
10000.rg.5.0.4	5.0	70.8	73.0	71.0	71.2	71.9	71.2	71.4	70.8	70.9	71.9	71.4
10000.rg.5.0.5	5.0	70.8	72.8	70.6	70.9	71.7	71.1	71.5	70.9	70.8	72.1	71.2
10000.rg.5.0.6	5.0	70.9	72.5	70.9	71.0	71.5	71.2	71.4	70.9	70.8	71.8	71.4
10000.rg.5.0.7	5.0	70.9	73.0	71.0	71.1	72.0	71.2	71.2	70.9	70.9	71.7	71.4
10000.rg.5.0.8	5.0	70.9	73.0	70.9	71.0	71.9	71.4	71.7	71.0	70.9	71.7	71.4
10000.rg.5.0.9	5.0	70.8	72.9	70.8	71.1	72.0	71.4	71.6	70.8	70.7	71.7	71.3

Table 4.11 Average results of all algorithms tested on MDS random geometric graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rgg.0.1.0	0.1	1059.3	1052.8	1054.3	1243.7	1182.9	1187.5	1221.3	1244.5	1237.3	1237.3	1252.4
10000.rgg.0.1.1	0.1	1057.3	1051.7	1052.5	1242.0	1185.3	1189.6	1221.7	1244.9	1237.8	1237.3	1255.2
10000.rgg.0.1.2	0.1	1055.5	1051.0	1051.9	1236.3	1174.6	1175.4	1212.6	1233.7	1229.8	1225.4	1244.8
10000.rgg.0.1.3	0.1	1057.1	1050.7	1052.8	1242.4	1182.6	1186.8	1219.4	1239.8	1235.2	1232.4	1251.6
10000.rgg.0.1.4	0.1	1055.3	1049.5	1049.8	1235.1	1178.8	1183.3	1215.0	1238.6	1228.7	1227.0	1245.4
10000.rgg.0.1.5	0.1	1055.9	1049.7	1049.9	1244.1	1181.0	1184.1	1217.2	1244.5	1236.2	1236.2	1253.2
10000.rgg.0.1.6	0.1	1049.9	1045.6	1044.6	1230.7	1171.3	1173.7	1207.5	1232.3	1223.8	1226.2	1244.2
10000.rgg.0.1.7	0.1	1060.8	1053.0	1055.6	1232.9	1174.7	1178.4	1210.7	1231.6	1227.5	1224.5	1245.6
10000.rgg.0.1.8	0.1	1060.7	1055.1	1056.2	1247.6	1184.1	1189.6	1222.2	1246.1	1240.1	1238.4	1257.1
10000.rgg.0.1.9	0.1	1061.0	1056.4	1055.0	1243.8	1186.1	1190.3	1221.5	1243.1	1241.3	1236.8	1252.2

Continuation of Table 4.11 Average results of all algorithms tested on MDS random geometric graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rgg.0.5.0	0.5	251.1	246.3	267.8	309.5	297.3	306.4	303.8	308.1	307.9	310.5	310.5
10000.rgg.0.5.1	0.5	251.5	245.0	268.4	307.8	297.6	305.8	302.8	307.1	307.6	309.8	309.3
10000.rgg.0.5.2	0.5	249.5	244.4	267.0	306.5	296.7	304.5	303.8	306.9	306.3	310.2	308.6
10000.rgg.0.5.3	0.5	250.6	245.3	265.2	303.6	293.1	300.8	299.2	304.4	304.9	305.2	306.6
10000.rgg.0.5.4	0.5	249.9	245.1	266.2	307.4	295.2	302.4	302.1	305.1	304.8	307.7	307.2
10000.rgg.0.5.5	0.5	250.6	245.3	267.5	304.6	295.2	302.0	299.4	303.4	303.9	306.0	306.4
10000.rgg.0.5.6	0.5	251.4	245.5	268.6	309.3	296.9	304.4	304.0	308.3	306.6	310.3	309.5
10000.rgg.0.5.7	0.5	249.9	244.1	265.9	308.0	295.3	303.5	303.0	307.3	306.1	309.0	309.1
10000.rgg.0.5.8	0.5	251.0	245.8	268.1	304.9	295.0	302.2	300.9	305.3	305.0	306.5	307.0
10000.rgg.0.5.9	0.5	249.0	244.5	267.0	304.9	294.9	302.2	300.9	304.9	305.1	306.5	307.1
10000.rgg.1.0.0	1.0	130.3	128.0	142.0	163.2	159.3	162.5	161.4	162.8	162.9	165.1	162.8
10000.rgg.1.0.1	1.0	129.5	127.4	141.3	162.6	158.3	161.0	161.2	163.2	162.9	164.4	162.0
10000.rgg.1.0.2	1.0	129.7	128.6	141.6	162.9	159.1	161.8	160.8	162.1	162.6	164.2	163.1
10000.rgg.1.0.3	1.0	130.1	128.4	141.3	162.2	157.9	161.1	160.5	162.3	161.5	164.0	162.1
10000.rgg.1.0.4	1.0	129.0	127.7	141.2	162.5	157.3	159.8	160.5	161.6	162.1	164.0	162.0
10000.rgg.1.0.5	1.0	130.4	128.7	141.7	162.1	158.8	161.2	159.7	161.6	162.6	163.9	162.0
10000.rgg.1.0.6	1.0	130.7	128.2	142.0	164.2	160.6	162.3	161.6	164.0	163.7	166.4	163.8
10000.rgg.1.0.7	1.0	129.8	128.1	142.1	162.1	158.1	161.0	160.6	161.9	162.2	163.3	161.9
10000.rgg.1.0.8	1.0	129.8	127.6	141.7	163.5	159.1	161.2	161.4	163.4	163.4	165.1	162.9
10000.rgg.1.0.9	1.0	130.2	128.5	141.7	162.1	157.6	159.9	159.5	162.2	161.7	163.4	161.4
10000.rgg.5.0.0	5.0	27.1	32.7	27.6	35.7	35.3	35.6	35.7	35.4	35.2	36.5	35.5
10000.rgg.5.0.1	5.0	26.9	33.3	27.8	36.4	36.1	36.9	36.6	36.3	36.3	37.3	36.3
10000.rgg.5.0.2	5.0	27.0	33.2	27.6	37.6	36.3	37.0	37.5	37.5	37.3	38.2	37.0
10000.rgg.5.0.3	5.0	27.1	33.1	27.8	36.0	35.9	36.3	36.5	36.3	36.2	37.3	35.9
10000.rgg.5.0.4	5.0	27.0	32.9	27.9	35.9	35.7	36.2	36.0	35.8	36.0	36.6	35.6
10000.rgg.5.0.5	5.0	27.1	33.2	27.8	36.1	36.3	36.9	36.8	36.8	36.6	37.5	36.5
10000.rgg.5.0.6	5.0	26.9	32.8	28.1	37.0	36.6	37.1	36.6	36.9	36.9	38.1	36.2
10000.rgg.5.0.7	5.0	27.0	33.1	27.9	37.2	36.7	37.1	37.0	36.5	36.8	38.7	36.7
10000.rgg.5.0.8	5.0	26.9	32.8	27.5	36.6	36.1	36.4	36.5	36.1	36.2	37.1	36.3
10000.rgg.5.0.9	5.0	26.9	32.1	27.3	35.8	35.7	35.7	36.0	36.2	35.7	36.6	35.3

Table 4.12 Average computation time of all algorithms tested on MDS random graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
5000.rg.0.1.0	0.1	225.9	171.0	418.0	454.2	443.5	420.9	428.2	450.4	472.7	417.7	414.4
5000.rg.0.1.1	0.1	193.0	273.3	434.6	455.6	432.6	441.4	412.3	454.8	454.0	421.3	398.3
5000.rg.0.1.2	0.1	226.6	261.2	409.6	454.9	423.8	404.8	417.9	450.0	465.0	425.4	432.8
5000.rg.0.1.3	0.1	205.1	182.9	394.2	447.2	435.3	408.4	441.0	455.4	468.9	421.9	402.4
5000.rg.0.1.4	0.1	200.5	196.4	448.4	459.6	426.7	428.1	437.9	463.5	456.5	400.6	393.9
5000.rg.0.1.5	0.1	182.6	234.9	434.3	444.4	416.7	423.1	416.3	459.5	457.5	440.3	390.4
5000.rg.0.1.6	0.1	175.3	188.7	439.1	454.5	435.9	407.8	429.7	441.5	439.3	409.1	409.5
5000.rg.0.1.7	0.1	240.7	187.3	426.1	450.7	436.1	414.2	416.6	460.5	452.1	433.6	409.1
5000.rg.0.1.8	0.1	145.3	161.3	437.4	448.8	442.2	416.2	404.3	449.5	444.8	419.3	407.2
5000.rg.0.1.9	0.1	226.9	252.4	446.9	453.0	434.0	421.6	435.8	456.8	447.1	412.5	404.2
5000.rg.0.5.0	0.5	232.7	328.1	306.2	250.0	280.3	267.6	251.8	278.5	327.7	284.5	209.1
5000.rg.0.5.1	0.5	246.2	327.0	284.6	312.9	246.0	242.0	277.9	297.8	292.2	288.4	324.3
5000.rg.0.5.2	0.5	231.7	280.9	289.2	318.1	263.2	256.1	292.5	272.5	306.6	307.8	275.1
5000.rg.0.5.3	0.5	285.0	325.3	255.9	346.3	227.6	219.5	314.6	274.9	261.9	240.2	276.1
5000.rg.0.5.4	0.5	261.7	333.9	307.4	266.9	248.7	231.3	247.5	315.0	328.2	293.6	262.7
5000.rg.0.5.5	0.5	262.2	295.9	276.4	305.1	285.0	216.0	283.7	319.0	313.7	249.0	262.5
5000.rg.0.5.6	0.5	223.6	291.9	297.3	283.2	291.5	270.7	309.4	314.2	312.0	230.4	258.4
5000.rg.0.5.7	0.5	240.0	364.9	267.8	367.8	254.7	236.8	276.2	316.8	271.5	243.2	316.0
5000.rg.0.5.8	0.5	265.8	338.4	273.5	282.0	268.7	255.6	307.3	316.2	299.3	277.7	217.5
5000.rg.0.5.9	0.5	271.3	347.3	292.9	322.8	210.1	241.4	321.6	364.6	346.4	278.2	293.1
5000.rg.1.0.0	1.0	226.1	338.2	222.0	200.7	295.7	180.2	261.9	215.9	229.9	208.3	181.8
5000.rg.1.0.1	1.0	246.8	348.7	238.0	215.5	234.9	189.4	241.9	220.0	172.8	217.7	223.5
5000.rg.1.0.2	1.0	256.2	334.8	216.9	211.8	250.4	254.2	193.7	203.3	191.5	209.4	188.6
5000.rg.1.0.3	1.0	261.7	322.4	217.4	188.7	225.5	185.2	294.7	235.7	190.8	181.8	183.0
5000.rg.1.0.4	1.0	244.8	336.2	264.5	254.3	243.2	228.5	266.1	234.5	213.3	195.1	220.0
5000.rg.1.0.5	1.0	249.6	333.6	280.5	234.7	220.7	181.1	201.5	201.3	149.1	233.8	209.7
5000.rg.1.0.6	1.0	251.7	329.3	242.2	227.0	232.5	209.6	236.9	252.7	196.1	242.6	169.6
5000.rg.1.0.7	1.0	257.8	317.8	249.5	273.6	259.5	216.8	201.5	235.1	218.1	195.7	185.6
5000.rg.1.0.8	1.0	242.5	339.2	222.2	203.6	229.6	243.0	290.3	222.2	254.6	173.5	220.2
5000.rg.1.0.9	1.0	243.2	321.9	251.7	231.1	228.4	217.7	281.0	170.2	207.0	170.2	189.0

Continuation of Table 4.12 Average computation time of all algorithms tested on MDS random graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rg.5.0.0	5.0	27.9	14.8	70.6	145.2	76.2	20.1	38.1	89.0	107.0	107.1	80.8
5000.rg.5.0.1	5.0	23.5	12.2	112.4	134.4	36.3	34.7	16.9	135.3	122.7	127.8	125.0
5000.rg.5.0.2	5.0	14.0	14.5	46.7	181.7	31.6	21.2	34.8	140.3	106.6	133.8	92.3
5000.rg.5.0.3	5.0	34.5	12.2	133.7	134.0	85.1	5.2	26.3	129.1	132.6	113.8	124.1
5000.rg.5.0.4	5.0	31.7	12.9	106.8	156.8	14.6	35.6	35.7	156.6	166.9	113.0	141.8
5000.rg.5.0.5	5.0	22.4	17.3	150.0	195.7	33.7	34.7	45.5	81.6	145.9	83.1	105.3
5000.rg.5.0.6	5.0	24.4	12.9	35.5	129.7	34.0	28.5	52.5	103.0	128.9	89.3	114.0
5000.rg.5.0.7	5.0	31.6	19.8	109.1	117.1	57.1	5.1	54.1	67.8	121.6	140.3	139.6
5000.rg.5.0.8	5.0	15.3	13.8	92.4	159.0	19.0	24.9	55.5	73.0	142.8	67.9	108.7
5000.rg.5.0.9	5.0	13.5	12.3	72.8	180.5	80.2	11.2	35.8	140.0	110.1	120.5	140.3

Table 4.13 Average computation time of all algorithms tested on MDS random geometric graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
5000.rgg.0.1.0	0.1	247.0	116.4	220.8	433.5	411.4	433.2	461.6	452.3	455.8	440.4	424.7
5000.rgg.0.1.1	0.1	152.6	239.0	218.7	451.2	426.4	429.9	435.3	452.8	452.2	452.4	405.1
5000.rgg.0.1.2	0.1	360.5	270.2	333.7	450.6	411.4	452.2	446.8	455.6	457.9	442.9	434.8
5000.rgg.0.1.3	0.1	103.8	41.6	142.5	443.9	430.8	442.4	452.5	444.4	439.0	452.2	416.2
5000.rgg.0.1.4	0.1	320.8	130.6	236.0	448.4	428.4	436.5	428.9	437.3	455.1	459.2	420.2
5000.rgg.0.1.5	0.1	258.9	140.1	225.7	450.3	412.1	441.7	441.9	428.7	456.3	426.3	431.3
5000.rgg.0.1.6	0.1	215.3	60.6	217.1	447.6	417.4	441.8	445.3	448.6	460.5	440.2	441.3
5000.rgg.0.1.7	0.1	190.6	127.7	181.0	457.4	440.6	447.8	433.4	451.2	452.6	454.9	430.2
5000.rgg.0.1.8	0.1	261.5	75.4	264.3	455.2	435.0	434.4	452.8	438.6	442.3	426.3	417.1
5000.rgg.0.1.9	0.1	243.1	106.0	238.3	462.2	441.7	436.9	434.8	435.1	458.1	450.7	444.0

Continuation of Table 4.13 Average computation time of all algorithms tested on MDS random geometric graphs with 5000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	ACO-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
5000.rgg.0.5.0	0.5	287.3	215.9	396.8	398.4	346.0	353.0	370.6	379.4	432.3	419.6	401.8
5000.rgg.0.5.1	0.5	254.0	216.7	413.3	422.7	384.0	385.0	418.9	418.5	425.9	388.3	388.4
5000.rgg.0.5.2	0.5	266.9	214.0	362.5	428.2	385.5	358.6	406.1	404.8	415.7	358.4	394.4
5000.rgg.0.5.3	0.5	236.1	199.9	366.3	415.5	404.5	394.9	405.4	412.9	402.3	404.7	381.9
5000.rgg.0.5.4	0.5	281.4	241.9	375.0	400.6	343.6	373.7	397.5	397.8	422.2	380.8	384.5
5000.rgg.0.5.5	0.5	312.7	224.1	391.4	411.6	372.3	372.6	403.4	415.4	403.5	398.5	389.0
5000.rgg.0.5.6	0.5	246.1	172.6	361.2	413.4	401.8	367.6	403.6	415.7	415.8	402.8	399.8
5000.rgg.0.5.7	0.5	230.5	210.2	414.3	412.6	399.0	356.2	402.3	401.0	434.3	380.7	363.0
5000.rgg.0.5.8	0.5	259.5	152.6	378.2	422.1	401.7	369.9	393.0	418.3	424.6	392.8	404.3
5000.rgg.0.5.9	0.5	223.3	205.7	404.5	391.4	412.7	373.3	371.8	442.9	411.4	391.2	397.1
5000.rgg.1.0.0	1.0	184.6	170.4	368.9	411.3	362.9	300.0	375.7	353.9	356.0	377.2	337.2
5000.rgg.1.0.1	1.0	195.7	174.7	366.6	351.6	345.2	345.3	407.5	367.6	373.3	358.4	386.7
5000.rgg.1.0.2	1.0	204.4	187.1	368.1	388.4	284.1	300.9	389.8	355.9	376.7	361.8	340.9
5000.rgg.1.0.3	1.0	233.4	166.2	365.9	324.6	352.2	259.9	372.2	357.5	355.4	349.2	323.2
5000.rgg.1.0.4	1.0	178.9	190.7	358.8	381.4	336.2	310.8	392.9	347.0	370.1	377.0	338.4
5000.rgg.1.0.5	1.0	258.1	205.1	347.0	382.7	376.3	316.1	353.0	337.8	385.3	325.0	354.0
5000.rgg.1.0.6	1.0	198.1	182.6	347.8	386.0	343.3	297.3	356.5	367.1	364.4	348.9	361.4
5000.rgg.1.0.7	1.0	234.3	206.0	354.0	391.6	323.7	312.8	366.5	357.5	347.2	343.9	363.6
5000.rgg.1.0.8	1.0	192.5	195.4	382.7	391.8	333.0	268.2	341.2	348.3	362.5	304.0	356.8
5000.rgg.1.0.9	1.0	215.8	187.0	389.1	342.5	307.5	264.6	360.6	385.4	351.8	333.8	389.5
5000.rgg.5.0.0	5.0	109.3	359.6	82.1	230.8	214.4	225.6	281.5	281.3	268.2	283.3	253.1
5000.rgg.5.0.1	5.0	61.8	364.0	73.9	221.6	208.5	232.3	220.8	235.5	230.3	247.9	225.4
5000.rgg.5.0.2	5.0	70.1	378.7	138.1	261.5	218.8	252.3	220.5	205.7	207.9	239.2	250.9
5000.rgg.5.0.3	5.0	91.5	376.5	110.2	255.6	235.0	237.5	254.1	242.3	226.3	242.7	268.7
5000.rgg.5.0.4	5.0	110.2	377.8	68.7	230.2	249.6	265.3	272.5	235.4	226.9	251.3	228.3
5000.rgg.5.0.5	5.0	77.5	359.2	117.2	229.0	228.9	253.3	241.7	273.3	252.2	220.6	281.9
5000.rgg.5.0.6	5.0	79.3	347.1	89.5	206.2	245.9	294.5	278.2	245.4	240.0	257.3	226.3
5000.rgg.5.0.7	5.0	86.7	366.1	96.3	265.3	225.8	247.9	290.8	242.2	256.1	225.1	286.2
5000.rgg.5.0.8	5.0	104.2	362.5	102.4	263.7	192.0	230.4	226.8	226.0	234.5	236.5	226.8
5000.rgg.5.0.9	5.0	75.4	363.4	89.5	214.7	227.2	224.3	244.0	255.2	286.8	256.2	228.1

Table 4.14 Average computation time of all algorithms tested on MDS random graphs with 10000 vertices

Instance	Density	Aco ⁺ _{neg}	Aco _{neg}	Aco ⁺	Aco	Aco-Aco ⁺ _{neg}	Aco-Aco _{neg}	Aco-Aco ⁺	ACO-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
10000.rg.0.1.0	0.1	520.0	542.8	870.3	848.7	729.4	387.3	753.5	887.6	840.9	620.8	729.9
10000.rg.0.1.1	0.1	467.1	540.7	901.9	835.7	716.4	476.2	805.0	852.4	883.2	650.6	749.0
10000.rg.0.1.2	0.1	397.8	556.3	874.7	805.1	643.9	469.1	780.6	838.0	881.1	679.3	697.6
10000.rg.0.1.3	0.1	527.7	542.5	874.2	856.5	651.9	498.7	794.2	823.4	868.6	701.9	745.9
10000.rg.0.1.4	0.1	465.5	555.3	899.8	820.3	751.5	436.1	751.0	869.6	880.4	751.6	750.2
10000.rg.0.1.5	0.1	429.3	540.9	888.6	851.3	763.6	466.8	777.5	859.6	863.7	640.0	663.7
10000.rg.0.1.6	0.1	466.6	536.8	885.5	845.7	642.6	496.6	771.3	846.2	849.9	744.2	698.8
10000.rg.0.1.7	0.1	510.9	559.7	863.4	814.2	714.4	587.3	769.4	892.5	912.9	714.0	725.3
10000.rg.0.1.8	0.1	596.3	537.3	917.3	873.1	738.1	473.0	709.3	871.6	844.0	644.2	618.2
10000.rg.0.1.9	0.1	512.9	546.3	910.0	856.3	756.3	483.0	714.7	895.1	834.1	763.9	739.8
10000.rg.0.5.0	0.5	559.2	828.9	576.3	429.9	466.2	475.4	461.8	517.1	401.5	449.0	567.3
10000.rg.0.5.1	0.5	576.9	844.5	497.0	426.5	487.2	398.0	539.7	350.8	361.3	401.9	520.3
10000.rg.0.5.2	0.5	557.4	809.4	613.3	449.6	379.3	344.5	543.0	423.6	494.8	390.5	394.3
10000.rg.0.5.3	0.5	588.4	831.6	597.5	414.8	489.5	406.8	467.8	456.4	450.2	492.7	520.0
10000.rg.0.5.4	0.5	580.5	824.7	595.5	550.4	426.0	393.8	522.7	403.3	554.2	442.6	475.6
10000.rg.0.5.5	0.5	591.1	861.6	706.0	369.9	445.6	442.3	534.9	413.7	554.9	579.7	473.4
10000.rg.0.5.6	0.5	558.7	830.2	579.3	456.3	481.9	491.6	523.2	362.6	482.4	483.1	424.3
10000.rg.0.5.7	0.5	597.8	853.2	624.1	518.6	522.9	406.5	525.9	336.0	434.0	434.9	559.7
10000.rg.0.5.8	0.5	578.4	828.7	566.1	435.4	545.8	411.1	505.4	492.6	430.0	511.3	533.7
10000.rg.0.5.9	0.5	594.1	846.4	572.8	523.4	538.0	445.6	624.6	341.4	462.9	446.3	480.7
10000.rg.1.0.0	1.0	327.8	11.3	387.3	365.0	417.8	454.2	440.7	311.1	271.3	383.7	478.2
10000.rg.1.0.1	1.0	136.0	11.9	393.0	461.2	360.0	406.2	376.3	455.5	318.2	404.5	455.2
10000.rg.1.0.2	1.0	332.2	11.7	387.5	361.0	468.6	475.3	432.4	399.1	272.4	524.2	392.6
10000.rg.1.0.3	1.0	140.0	14.1	478.7	430.8	350.0	360.6	542.7	411.7	542.8	480.2	427.6
10000.rg.1.0.4	1.0	301.2	9.9	486.6	421.2	282.9	472.7	463.8	481.8	489.7	355.4	433.8
10000.rg.1.0.5	1.0	104.8	11.8	497.8	351.2	407.4	386.6	383.0	474.5	363.5	478.4	362.7
10000.rg.1.0.6	1.0	233.2	10.9	517.2	437.8	438.0	373.4	426.7	432.9	331.5	372.9	505.3
10000.rg.1.0.7	1.0	269.8	11.4	487.1	423.5	446.5	473.9	404.0	484.9	418.1	394.4	320.5
10000.rg.1.0.8	1.0	236.4	11.2	474.7	370.1	394.5	372.8	449.9	428.6	443.2	414.4	387.4
10000.rg.1.0.9	1.0	199.2	12.3	447.9	424.6	350.1	384.9	433.1	455.5	333.0	402.8	430.1

Continuation of Table 4.14 Average computation time of all algorithms tested on MDS random graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	ACO-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
10000.rg.5.0.0	5.0	62.6	71.3	52.1	292.1	14.4	30.0	86.7	371.2	50.4	15.6	267.8
10000.rg.5.0.1	5.0	60.3	64.3	61.8	264.7	21.0	35.8	54.1	298.5	69.6	13.7	307.4
10000.rg.5.0.2	5.0	57.6	75.4	101.8	211.8	17.2	30.5	95.3	136.2	50.6	36.0	236.4
10000.rg.5.0.3	5.0	67.5	77.7	68.5	327.5	12.2	40.7	47.6	227.9	53.7	35.1	220.6
10000.rg.5.0.4	5.0	57.5	63.4	46.0	260.5	17.7	28.6	57.9	287.6	61.8	33.6	192.5
10000.rg.5.0.5	5.0	44.9	62.8	82.4	252.6	17.4	26.8	73.0	305.1	64.0	79.5	335.8
10000.rg.5.0.6	5.0	46.5	76.5	58.3	314.7	19.9	38.2	19.8	233.5	58.1	14.1	217.5
10000.rg.5.0.7	5.0	41.9	63.5	53.7	275.4	12.5	26.8	81.0	414.4	54.7	13.8	157.2
10000.rg.5.0.8	5.0	68.7	56.8	58.1	233.8	14.3	31.9	96.5	147.0	66.3	14.2	146.9
10000.rg.5.0.9	5.0	79.6	48.2	66.5	391.5	15.0	29.7	70.1	416.4	92.6	46.5	284.3

Table 4.15 Average computation time of all algorithms tested on MDS random geometric graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	ACO-EA	ACO-SAP	ACO-PAP	ACO ^{2o}
10000.rgg.0.1.0	0.1	743.5	647.1	809.1	906.1	882.6	805.8	890.8	907.4	902.6	858.8	869.8
10000.rgg.0.1.1	0.1	673.6	383.3	841.7	914.9	889.2	784.1	879.9	899.0	917.7	903.0	835.1
10000.rgg.0.1.2	0.1	632.5	448.5	779.5	903.7	866.7	846.8	858.5	922.1	921.5	856.0	836.7
10000.rgg.0.1.3	0.1	681.9	479.5	868.3	891.0	869.3	843.9	884.2	925.9	897.1	885.4	824.9
10000.rgg.0.1.4	0.1	655.6	492.4	812.8	918.4	876.0	832.1	867.3	919.3	892.2	862.7	847.6
10000.rgg.0.1.5	0.1	676.5	596.3	838.1	924.8	877.2	805.7	906.9	950.6	921.6	868.4	873.7
10000.rgg.0.1.6	0.1	649.7	526.0	802.2	923.4	815.0	855.1	885.9	903.5	925.2	847.2	812.6
10000.rgg.0.1.7	0.1	712.6	551.6	874.0	931.6	846.2	815.8	915.1	912.9	915.9	889.7	798.2
10000.rgg.0.1.8	0.1	537.2	479.9	786.6	942.4	892.7	811.6	893.4	923.1	930.8	873.1	813.4
10000.rgg.0.1.9	0.1	653.9	413.4	841.3	915.9	844.2	822.1	901.6	929.1	904.4	916.9	834.6

Continuation of Table 4.15 Average computation time of all algorithms tested on MDS random geometric graphs with 10000 vertices

Instance	Density	Aco _{neg} ⁺	Aco _{neg}	Aco ⁺	Aco	Aco-Aco _{neg} ⁺	Aco-Aco _{neg}	Aco-Aco ⁺	Aco-EA	Aco-SAP	Aco-PAP	Aco ^{2o}
10000.rgg.0.5.0	0.5	490.5	388.2	816.0	792.9	734.0	405.7	824.8	796.3	810.1	732.3	704.3
10000.rgg.0.5.1	0.5	461.1	483.2	835.2	820.0	830.1	396.6	860.6	818.5	770.2	790.5	715.7
10000.rgg.0.5.2	0.5	386.1	451.3	828.9	809.7	677.7	428.7	825.3	881.4	851.3	820.6	752.3
10000.rgg.0.5.3	0.5	398.7	449.3	869.2	830.8	735.1	551.9	800.5	760.4	777.8	779.3	726.4
10000.rgg.0.5.4	0.5	416.6	553.0	836.1	751.4	771.7	597.1	809.5	739.4	813.3	742.2	732.7
10000.rgg.0.5.5	0.5	505.5	425.2	803.4	691.8	790.5	521.8	812.6	839.5	793.9	669.5	663.0
10000.rgg.0.5.6	0.5	483.2	439.3	815.6	826.7	783.0	470.8	809.1	803.4	846.8	711.8	692.6
10000.rgg.0.5.7	0.5	352.1	456.6	835.9	734.5	745.6	530.3	814.8	788.4	786.3	695.0	736.0
10000.rgg.0.5.8	0.5	396.1	506.5	813.2	767.7	758.4	466.4	835.6	803.6	831.8	745.7	689.9
10000.rgg.0.5.9	0.5	444.4	472.7	798.7	804.8	747.9	613.0	819.7	834.4	785.0	772.7	764.1
10000.rgg.1.0.0	1.0	539.6	535.1	666.4	675.5	632.5	517.2	708.1	729.1	687.2	715.2	697.6
10000.rgg.1.0.1	1.0	409.6	529.5	654.2	704.3	673.3	497.5	661.8	677.8	683.0	688.1	692.1
10000.rgg.1.0.2	1.0	597.3	533.7	564.1	662.8	701.0	518.0	693.7	709.2	747.9	709.0	671.2
10000.rgg.1.0.3	1.0	470.1	533.3	614.5	678.0	685.3	597.9	767.0	665.7	638.6	630.8	650.6
10000.rgg.1.0.4	1.0	495.1	528.1	627.5	804.9	633.7	626.3	760.5	780.9	752.7	615.0	670.6
10000.rgg.1.0.5	1.0	450.0	542.1	601.5	715.5	653.3	500.8	680.6	725.9	621.0	707.8	617.6
10000.rgg.1.0.6	1.0	485.9	542.1	596.2	748.9	658.5	461.0	729.3	698.3	749.9	711.4	614.7
10000.rgg.1.0.7	1.0	540.7	532.1	659.9	745.6	653.1	491.2	703.9	754.0	736.9	624.6	669.9
10000.rgg.1.0.8	1.0	531.8	535.0	533.0	686.8	709.0	626.5	635.9	736.5	725.7	672.6	643.3
10000.rgg.1.0.9	1.0	524.3	532.2	544.5	732.5	710.2	486.3	661.1	730.4	705.6	667.1	676.3
10000.rgg.5.0.0	5.0	252.1	890.4	325.9	478.3	476.9	404.5	444.9	481.8	642.7	411.0	454.9
10000.rgg.5.0.1	5.0	411.0	840.6	200.8	535.7	589.3	435.3	567.7	505.1	560.9	537.3	585.6
10000.rgg.5.0.2	5.0	276.4	872.5	321.8	558.7	568.1	588.3	631.7	546.3	721.0	579.2	533.5
10000.rgg.5.0.3	5.0	385.6	845.2	191.0	611.0	509.8	482.4	643.0	550.4	547.0	484.7	391.6
10000.rgg.5.0.4	5.0	296.4	866.7	202.3	471.2	528.4	505.1	537.9	436.8	521.9	440.2	381.7
10000.rgg.5.0.5	5.0	308.1	834.5	228.5	514.6	521.1	490.6	595.9	656.4	563.6	405.8	499.2
10000.rgg.5.0.6	5.0	267.6	837.9	358.9	534.4	527.8	504.4	544.1	556.9	540.2	349.4	624.0
10000.rgg.5.0.7	5.0	304.2	873.8	237.1	455.9	607.2	517.7	710.7	518.6	703.6	517.6	498.8
10000.rgg.5.0.8	5.0	203.2	857.0	301.1	431.5	599.7	479.7	572.1	451.1	545.3	472.1	534.1
10000.rgg.5.0.9	5.0	255.4	869.4	299.1	507.9	548.8	491.6	542.7	509.1	615.0	519.6	593.7

CHAPTER 5

APPLICATION TO THE MAXIMUM SATISFIABILITY PROBLEM

5.1 INTRODUCTION

This chapter describes the application of our negative learning Aco approach to the Maximum Satisfiability problem (MaxSAT) [113, 114], which differs substantially from the problems considered in Chapter 3 and Chapter 4. Given a multiset of clauses, where each clause is a disjunction of Boolean literals, the goal of MaxSAT is to find a truth assignment that maximizes the number of satisfied clauses or, equivalently, that minimizes the number of unsatisfied clauses.

This work aims to prove the general applicability of our negative learning Aco framework by (1) implementing the approach to an optimization problem that has different characteristics than the problems considered to date, and (2) exploring other options for the additional algorithmic component that provides negative feedback to the main Aco algorithm. There are only a few Aco approaches in the literature for MaxSAT. Therefore, this work not only makes significant contributions to our negative learning Aco framework but also to the use of Aco for MaxSAT solving.

In earlier works we proved that our negative learning Aco approach works very well with CPLEX and *MMAS* as additional algorithmic components that provide negative feedback to Aco [101, 104]. In this work, we take a step further and consider two state-of-the-art MaxSAT solvers, *SATLike-c(w)* [115] and *SLSMcs* [115], as additional algorithmic components. The empirical results show that our approach also performs very well with these new algorithmic components. Moreover, the experimental investigation shows that all our negative learning Aco variants significantly outperform the baseline Aco, CPLEX, and MaxSAT solvers. Therefore, the obtained results provide additional evidence of the general applicability and the effectiveness of our negative learning Aco approach. In particular, it might be very interesting to see for the MaxSAT community that our approach can be seen as a general framework for the

improvement of existing MaxSAT solvers. Moreover, considering our findings, we believe that this algorithmic framework might be very useful also for other combinatorial optimization approaches.

5.2 THE MAXIMUM SATISFIABILITY PROBLEM

MaxSAT is an NP-hard optimization problem which can be stated as follows. Given is a set of n Boolean variables $X = \{x_1, x_2, \dots, x_n\}$. A *clause* is a *disjunction of literals* and each literal is either a variable x_i (that is a *positive* literal) or its negation \bar{x}_i (that is a *negative* literal). The variable x_i can take the truth value 0 for FALSE or 1 for TRUE. A Conjunctive Normal Form (CNF) formula ϕ is a conjunction of a set of m clauses $C = \{c_1, c_2, \dots, c_m\}$. A valid solution S to a MaxSAT problem is in the form of a complete truth assignment to all variables in X . The optimization objective of MaxSAT is to satisfy as many clauses in ϕ as possible.

Weighted MaxSAT is a variant of MaxSAT in which each clause has an associated positive weight and its optimization objective is to maximize the sum of weights of the satisfied clauses.

5.2.1 ILP Model for the MaxSAT

A standard integer linear programming model for Weighted MaxSAT can be stated as follows [114]:

$$\max \sum_{z_j \in Z} w_j \cdot z_j \quad (5.1)$$

subject to the constraints:

$$\sum_{i \in I_j^+} x_i + \sum_{i \in I_j^-} (1 - x_i) \geq z_j \quad \forall z_j \in Z \quad (5.2)$$

$$z_j \in \{0, 1\} \quad \forall z_j \in Z \quad (5.3)$$

$$x_i \in \{0, 1\} \quad \forall x_i \in X \quad (5.4)$$

The model consists of a set Z of binary variables z_1, z_2, \dots, z_m for each corresponding clause in C . Variable z_j takes value 1 if clause c_j is satisfied; otherwise it takes value 0. The sets I_j^+ and I_j^- contain the positive and negative literals indexes in clause c_j , respectively. Parameter w_j represents the weight of clause c_j . We implemented all the approaches in this work for unweighted MaxSAT. Therefore, all the clauses have weight 1. The objective function (5.1)

counts the number of satisfied clauses and the restriction (5.2) ensures that each satisfied clause has at least one satisfied literal.

5.2.2 Existing Approaches to MaxSAT

There exist only a few Aco approaches (e.g. [132–135]) for solving MaxSAT, despite its practical relevance. The community working on satisfiability testing, for example, has solved challenging optimization problems by first encoding them as MaxSAT instances and then solving the resulting encodings with a MaxSAT solver. Nowadays, MaxSAT offers a competitive generic problem solving formalism for combinatorial optimization. For example, MaxSAT has been applied to solve optimization problems in domains as diverse as bioinformatics [136, 137], combinatorial testing [138], community detection in complex networks [139], diagnosis [140], planning [141], scheduling [142] and team formation [143]. Moreover, the MaxSAT community holds an annual MaxSAT Evaluation (MSE) since 2006 [115, 144]. This event has promoted the implementation of highly optimized MaxSAT solvers and the creation of a wide collection of MaxSAT instances from different domains. Thus, MaxSAT is a suitable test problem to validate the general applicability of the negative learning Aco approach in an extremely competitive scenario.

Negative learning Aco has not been used so far to solve MaxSAT. Despite being a state-of-the-art metaheuristic, Aco itself has been applied just a few times to MaxSAT. The first Aco application was due to Drias and Ibri [133], who used a variant of Aco, known as Ant Colony System (ACS) [12], to solve *weighted* MaxSAT. This algorithm works by generating an initial solution to which a number of successive variable *flips* are applied. Drias and Ibri also added *parallelization* to their sequential ACS technique by using *synchronous* and *asynchronous* methods, but the empirical results showed that their algorithm did not outperform the existing approaches.

Another Aco implementation for MaxSAT was due to Pinto et al. [134]. They used an ACS variant to solve two unweighted and three weighted types of *static* and *dynamic* MaxSAT instances. Their implementation works by constructing solutions in two phases: (1) variable selection, which is done randomly, and (2) value selection, which is based on a *heuristic* and on *pheromone* values. This Aco variant outperforms the baseline local search algorithm [145, 146]. The authors, however, admitted that *WalkSAT* [147, 148] and other native MaxSAT solvers were yet a substantial challenge for their proposal.

Villagra and Baran [135] developed Max-Min-SAT, a version of Aco specifically designed to solve MaxSAT. This algorithm borrowed the *adaptive fitness function*

from genetic algorithms and is available in three variants: (1) ACO_{SAW} , which uses the *step-wise adaptation of weights*, (2) ACO_{REF} , which implements *refining functions*, and (3) ACO_{REFSAW} , which employs both the step-wise adaptation of weights and refining functions. An empirical comparison on the basis of 50 random Max-3SAT instances showed that Villagra and Baran's approach did not outperform the *WalkSAT* MaxSAT solver.

The satisfiability testing community has been very active in the development of MaxSAT solvers. As a result, their performance has improved dramatically in the last years, as witnessed by the results of the different editions of the MaxSAT Evaluation. The efforts have mainly focused on developing local search and exact MaxSAT solvers.

Local search MaxSAT solvers start from an initial complete assignment and, at each step, they flip the Boolean value of a selected variable to find a better solution using a heuristic. The most critical point of such solvers is that they can be trapped in local optima, and so they must incorporate suitable strategies to escape from local optima. Among the best performing solvers, we find Dist [149], CCEHC [150], SATLike [151] and SATLike3.0 [152].

There are two main groups of exact MaxSAT solvers: branch-and-bound (BnB) and SAT-based solvers. BnB MaxSAT solvers implement the branch-and-bound scheme and are competitive on random and some types of crafted instances. At each node of the search tree, they apply some inference rules and compute a lower bound by detecting disjoint inconsistent subsets of soft clauses with unit propagation [153, 154]. Representative BnB solvers are MaxSatz [155, 156] and Ahmaxsat [157]. BnB MaxSAT solvers can become competitive on industrial instances by incorporating the recently defined clause learning mechanism defined in [158].

SAT-based MaxSAT solvers proceed by reformulating the MaxSAT optimization problem into a sequence of SAT decision problems [113] and are particularly competitive on industrial instances. These solvers could still be divided into three subgroups: model-guided, core-guided and Minimum Hitting Sets (MHS-)guided solvers. Model-guided approaches reduce the problem of deciding whether there exists an assignment for the MaxSAT instance with a cost less than or equal to a given k to SAT, and successively decrease k until an unsatisfiable SAT instance is found. Examples of such solvers are SAT4J-Maxsat [159] and Pacose [160]. Core-guided and MHS-guided approaches consider a MaxSAT instance as a SAT instance and call a CDCL SAT solver to identify an unsatisfiable subset of soft clauses, called a *core*. Then, they relax this core and solve the relaxed instance with a CDCL SAT

solver to identify another core, repeating this process until deriving a satisfiable instance. The difference between them is that core-guided solvers relax a core using cardinality constraints, while MHS-guided solvers remove one clause from each detected core so that the number of different clauses removed from the cores is minimized by solving a minimum hitting set instance with an integer programming solver. The solvers Open-WBO [161], WPM3 [162] and RC2 [163] are representative core-guided solvers, and the solvers MHS [164] and MaxHS [165] are representative MHS-guided solvers.

5.3 NEGATIVE LEARNING ACO FOR MAXSAT

Our negative learning Aco for MaxSAT is—as in the previous chapters—based on a *MMAS* variant implemented in the *hypercube* framework [14] as the baseline algorithm. Depending on the type of additional algorithmic component that is used for providing negative feedback to the main Aco algorithm, we constructed four variants: (1) $\text{ACO-SAT}_{\text{neg}}^+$, which uses the MaxSAT solver *SATLike-c(w)*; (2) $\text{ACO-SLS}_{\text{neg}}^+$, which uses the MaxSAT solver *SLSMcs*; and (3) $\text{ACO}_{\text{neg}}^+$, respectively ACO_{neg} , which apply the integer linear programming (ILP) solver *Cplex*. $\text{ACO-SAT}_{\text{neg}}^+$, $\text{ACO-SLS}_{\text{neg}}^+$ and $\text{ACO}_{\text{neg}}^+$ take benefit from both the positive and negative feedback information obtained by the solvers, whereas ACO_{neg} only uses *Cplex* as negative feedback provider. Algorithm 5.1 displays the pseudo-code of the general algorithmic framework of all these variants.

Again, for the benefit of the reader we repeat much of the general description of the baseline algorithm here. Following the framework of *MMAS*, we keep three solutions at any time: (1) the best solution constructed at the current iteration (S^{ib}), (2) the best solution found since the last restart of the algorithm (S^{rb}), and (3) the best overall solution (S^{bsf}). A *convergence factor* cf and a Boolean control variable `bs_update` are used to manage the pheromone update. Both S^{rb} and S^{bsf} are initialized as empty sets (line 3 in Algorithm 5.1). Moreover, cf and `bs_update` are initialized to 0 and `FALSE`, respectively.

For the application to MaxSAT, the algorithm applies a standard pheromone model \mathcal{T} that consists of pheromone $\tau_{\langle(x_i,j)\rangle} \geq 0$ for each Boolean value $j \in \{0, 1\}$ to be assigned to each Boolean variable x_i . In addition to the standard pheromone model, the algorithm also employs a negative pheromone model \mathcal{T}^{neg} that consists of negative pheromone $\tau_{\langle(x_i,j)\rangle}^{\text{neg}}$ for each Boolean value j to be assigned to each Boolean variable x_i . The pheromones in \mathcal{T} are initialized to 0.5 while the pheromones in \mathcal{T}^{neg} are initialized to $\tau_{\min} = 0.001$ at the start of the algorithm by

Algorithm 5.1 Negative Learning Aco for unweighted MaxSAT

```

1: input: a MaxSAT problem instance consisting of a set  $C$  and a set  $X$ 
2: parameters:  $n_a, \rho, d_{rate}, \rho^{neg}, t^{sub}$ 
3:  $S^{bsf} := \text{NULL}, S^{rb} := \text{NULL}, cf := 0, \text{bs\_update} := \text{FALSE}$ 
4: InitializePheromoneValues( $\mathcal{T}, \mathcal{T}^{neg}$ )
5: while termination condition is not met do
6:    $\mathcal{S}^{iter} := \emptyset$ 
7:   for  $k = 1, \dots, n_a$  do
8:      $S^k := \text{Construct\_Solution}(\mathcal{T}, \mathcal{T}^{neg})$ 
9:      $\mathcal{S}^{iter} := \mathcal{S}^{iter} \cup \{S^k\}$ 
10:  end for
11:   $S^{sub} := \text{SolveSubinstance}(\mathcal{S}^{iter})$ 
12:   $S^{ib} := \text{argmax}\{f(S) \mid S \in \mathcal{S}^{iter} \cup \{S^{sub}\}\}$ 
13:  if  $S^{ib}$  better than  $S^{rb}$  then  $S^{rb} := S^{ib}$ 
14:  if  $S^{ib}$  better than  $S^{bsf}$  then  $S^{bsf} := S^{ib}$ 
15:  ApplyPheromoneUpdate( $\mathcal{T}, \mathcal{T}^{neg}, cf, \text{bs\_update}, S^{ib}, S^{rb}, S^{bsf}, S^{sub}$ )
16:   $cf := \text{ComputeConvergenceFactor}(\mathcal{T})$ 
17:  if  $cf > 0.999$  then
18:    if  $\text{bs\_update} = \text{TRUE}$  then
19:       $S^{rb} := \text{NULL}$ , and  $\text{bs\_update} := \text{FALSE}$ 
20:      InitializePheromoneValues( $\mathcal{T}, \mathcal{T}^{neg}$ )
21:    else
22:       $\text{bs\_update} := \text{TRUE}$ 
23:    end if
24:  end if
25: end while
26: output:  $S^{bsf}$ , the best solution found by the algorithm

```

function InitializePheromoneValues($\mathcal{T}, \mathcal{T}^{neg}$) (line 4 of Algorithm 5.1). Based on greedy and pheromone information, then n_a solutions are generated at each iteration according to function Construct_Solution($\mathcal{T}, \mathcal{T}^{neg}$) (lines 6 – 10 of Algorithm 5.1). Further explanations on how this function works are given after this general description.

Figure 5.1 shows an illustrative example on how the negative learning is added to the baseline Aco in the context of the MaxSAT problem. The example shows that five solutions generated in the current baseline Aco iteration are added to set \mathcal{S}^{iter} . Subsequently, function SolveSubinstance(\mathcal{S}^{iter}) (line 11 of Algorithm 5.1) builds a sub-instance I^{sub} in the form of a MaxSAT *partial* solution. The *pre-assigned* variables in this partial solution are stored in set $X' \subseteq X$, which contains the variables that have been assigned the same truth value in each $S^k \in \mathcal{S}^{iter}$. In the illustrative example in Fig. 5.1, variables x_2, x_5 , and x_6 in the \mathcal{S}^{iter} are assigned values 1, 0, and 1 values, respectively, in each of the five solutions. Consequently, from seven variables in the sub-instance I^{sub} , variables

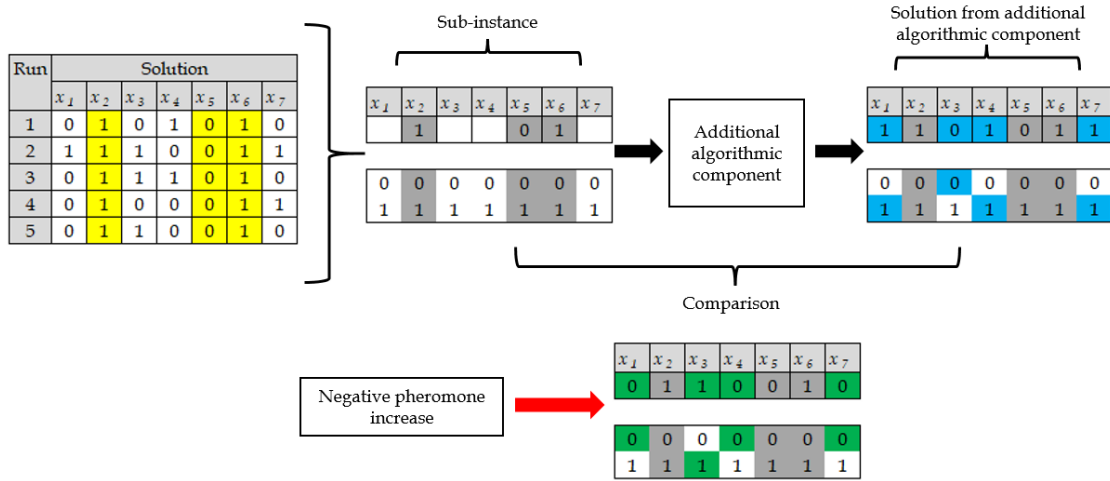


Fig. 5.1 Illustrative example of the negative learning Aco approach for the MaxSAT problem

x_2 , x_5 , and x_6 are pre-assigned with values 1, 0, and 1, respectively. With this, the additional algorithmic component can only work on the remaining variables whose values are still unassigned.

Depending on the specific variant of the negative learning Aco to be applied, the function then chooses either CPLEX or one of the two MaxSAT solvers for solving sub-instance I^{sub} . After trying to solve the sub-instance for a maximum time of t^{sub} CPU seconds, the function returns a solution S^{sub} . Next, S^{sub} is compared with the solutions in S^{iter} . The solution with the best objective function value becomes the *iteration-best* solution S^{ib} (line 12 of Algorithm 5.1). Note that, in the case of algorithm variant Aco_{neg} , S^{sub} is excluded from the set of solutions from which S^{ib} is selected. Hence, in this variant, S^{sub} is not used for updating the three solutions maintained by the algorithm. Afterwards, the *restart-best* solution S^{rb} and the *best-so-far* solution are updated with S^{ib} (lines 13–14 of Algorithm 5.1). Finally, the pheromone update and the calculation of the convergence factor are implemented by functions $\text{ApplyPheromoneUpdate}(\mathcal{T}, \mathcal{T}^{\text{neg}}, cf, \text{bs_update}, S^{\text{ib}}, S^{\text{rb}}, S^{\text{bsf}}, S^{\text{sub}})$ and $\text{ComputeConvergenceFactor}(\mathcal{T})$ (lines 15–16 of Algorithm 5.1), respectively. If $cf > 0.999$ and $\text{bs_update} = \text{TRUE}$, the algorithm is restarted (lines 17–24 of Algorithm 5.1). In the following, the functions in the algorithm are described in more detail.

5.3.1 Solution Construction

Function $\text{Construct_Solution}(\mathcal{T}, \mathcal{T}^{\text{neg}})$ generates a new solution S^k in two phases: (1) variable selection and (2) value selection. In the first phase, a variable x_i is taken from the set $\hat{X} \subseteq X$ that contains the variables that have not been

assigned a value in solution S^k . The probability $\mathbf{p}(x_i)$ of selecting variable x_i is calculated according to Eqn. 5.5.

$$\mathbf{p}(x_i)^{\text{phase-1}} := \frac{\eta_i}{\sum_{x_j \in \hat{X}} \eta_j} \quad (5.5)$$

where η_i is the greedy information for variable selection. More specifically, η_i is the number of occurrences of the variable in the current instance. Afterwards, a random number $r \in [0, 1]$ is generated. The variable that has the highest value of $\mathbf{p}(x_i)$ in Eqn. 5.5 is directly selected if $r \leq d_{\text{rate}}$. Otherwise, the variable is randomly selected using *roulette wheel selection*. Hereby, d_{rate} is the so-called determinism rate.

In the second phase of the solution construction, a truth value is assigned to the selected variable x_i , in a way similar to the one of the first phase. The probability of assigning truth value j to variable x_i is calculated with the Eqn. 5.6.

$$\mathbf{p}(j)^{\text{phase-2}} := \frac{\eta_{\langle x_i, j \rangle} \cdot \tau_{\langle x_i, j \rangle} \cdot (1 - \tau_{\langle x_i, j \rangle}^{\text{neg}})}{\sum_{s \in \{0,1\}} \eta_{\langle x_i, k \rangle} \cdot \tau_{\langle x_i, k \rangle} \cdot (1 - \tau_{\langle x_i, k \rangle}^{\text{neg}})} \quad , \quad (5.6)$$

where

$$\eta_{\langle x_i, j \rangle} := \frac{1}{1 + \text{cost}(S^k, \{\langle x_i, j \rangle\}) - \text{cost}(S^k)} \quad . \quad (5.7)$$

The greedy information $\eta_{\langle x_i, j \rangle}$ for the truth value selection in Eqn. 5.7 is inversely proportional to the number of new *constraint* violations in the partial solution S^k . Hereby $\text{cost}(S^k)$ represents the number of constraint violations in the partial solution S^k , while $\text{cost}(S^k, \{\langle x_i, j \rangle\})$ represents the number of constraint violations in the partial solution S^k if truth value j is assigned to variable x_i . These two phases of the solution construction are repeated until all $x_i \in X$ are assigned a truth value.

5.3.2 Pheromone Update and Convergence Factor

Function `ApplyPheromoneUpdate`(\mathcal{T} , \mathcal{T}^{neg} , cf , `bs_update`, S^{ib} , S^{rb} , S^{bsf} , S^{sub}) updates the standard pheromone model \mathcal{T} and the negative pheromone model \mathcal{T}^{neg} at each iteration. The standard pheromone model \mathcal{T} is updated in the same way as in all *MMAS* algorithms implemented in the hypercube framework. The

value of each standard pheromone $\tau_{\langle(x_i,j)\rangle}$ is updated with the Eqn. 5.8.

$$\tau_{\langle(x_i,j)\rangle} := \tau_{\langle(x_i,j)\rangle} + \rho \cdot (\xi_{\langle(x_i,j)\rangle} - \tau_{\langle(x_i,j)\rangle}) \quad (5.8)$$

where:

$$\xi_{\langle(x_i,j)\rangle} := \kappa_{ib} \cdot \Delta(S^{ib}, x_i, j) + \kappa_{rb} \cdot \Delta(S^{rb}, x_i, j) + \kappa_{bs} \cdot \Delta(S^{bsf}, x_i, j) \quad (5.9)$$

The weights κ_{ib} , κ_{rb} , and κ_{bs} in Eqn. 5.9 represent the influence of solutions S^{ib} , S^{rb} , and S^{bsf} , respectively, on the amount of pheromone deposit, and ρ is the learning rate. The values of these weights are determined based on the states of *cf* and *bs_update* as shown in Table 2.1. Note that in each state, the sum of κ_{ib} , κ_{rb} , and κ_{bs} is equal to 1. Furthermore, $\Delta(S, x_i, j)$ evaluates to 1 if, and only if, the truth value j is assigned to variable x_i in the corresponding solution; otherwise, $\Delta(S, x_i, j)$ evaluates to 0. For preventing the algorithm to reach complete *convergence*, the pheromone values are limited in the range of $\tau_{\min} = 0.001$ to $\tau_{\max} = 0.999$. Any pheromone that falls below τ_{\min} is set back to τ_{\min} and any pheromone that exceeds τ_{\max} is set back to τ_{\max} .

Function `ApplyPheromoneUpdate`(\mathcal{T} , \mathcal{T}^{neg} , *cf*, *bs_update*, S^{ib} , S^{rb} , S^{bsf} , S^{sub}) also updates negative pheromones with a similar mechanism as the one used for the standard pheromone update. However, in the case of the negative pheromone values, Eqn. 5.8 is only used to update the negative pheromone values corresponding to the truth values of variables in $X \setminus X'$, that is, variables that did not have already a pre-assigned value in the sub-instance I^{sub} . In the illustrative example in Fig. 5.1, negative pheromone update is only applied to the truth values of variables x_1, x_3, x_4 , and x_7 since their values are not pre-assigned in the sub-instance I^{sub} . In this example, the truth values 1, 0, 1, and 1 are assigned to variables x_1, x_3, x_4 , and x_7 , respectively. As a consequence of this assignment, negative pheromone increase are given to the truth values 0, 1, 0, and 0 which are not assigned to variables x_1, x_3, x_4 , and x_7 , respectively. In general, the update formula for the negative pheromone values is presented in Eqn. 5.10.

$$\tau_{\langle(x_i,j)\rangle}^{\text{neg}} := \tau_{\langle(x_i,j)\rangle}^{\text{neg}} + \rho^{\text{neg}} \cdot (\xi_{\langle(x_i,j)\rangle}^{\text{neg}} - \tau_{\langle(x_i,j)\rangle}^{\text{neg}}) \quad (5.10)$$

Hereby, ρ^{neg} is the negative learning rate. Furthermore, for all $x_i \in X \setminus X'$, $\xi_{\langle(x_i,0)\rangle}^{\text{neg}}$ is set to 1 if x_i has value 1 in solution S^{sub} , to 0 otherwise. Moreover, $\xi_{\langle(x_i,1)\rangle}^{\text{neg}}$ is set to 1 if x_i has value 0 in solution S^{sub} , to 0 otherwise. Hence, our algorithm gives penalty in the form of a negative pheromone increase to each Boolean value that is not assigned in S^{sub} to a variable $x_i \in X \setminus X'$.

Function `ComputeConvergenceFactor(\mathcal{T})` calculates the value of cf needed to regulate the update of the standard pheromone model \mathcal{T} by using Eqn. 5.11.

$$cf := 2 \left(\left(\frac{\sum_{\tau \in \mathcal{T}} \max\{\tau_{\max} - \tau, \tau - \tau_{\min}\}}{|\mathcal{T}| \cdot (\tau_{\max} - \tau_{\min})} \right) - 0.5 \right) \quad (5.11)$$

With this equation, the value of cf is equal to zero when all pheromone values are initialized to 0.5. On the contrary, the value of cf is equal to one when all pheromone values are either τ_{\min} or τ_{\max} . In the rest of the conditions, the value of cf is between 0 and 1.

5.4 EXPERIMENTAL EVALUATION

We performed the experimental evaluation of our negative learning Aco variants, the baseline Aco algorithm without negative learning, the ILP solver CPLEX, and the two chosen MaxSAT solvers on a cluster of machines with two Intel® Xeon® Silver 4210 CPUs with 10 cores of 2.20 GHz and 92 GB of RAM. The version of CPLEX used by Aco variants $\text{Aco}_{\text{neg}}^+$ and Aco_{neg} , as well as in standalone model, was 12.10, in one-threaded mode. The MaxSAT solvers *SATLike-c(w)* and *SLSMcs* used by variants $\text{Aco-SAT}_{\text{neg}}^+$ and $\text{Aco-SLS}_{\text{neg}}^+$ are taken from <https://maxsat-evaluations.github.io/2020/descriptions.html> (accessed on the 3rd of June 2021).

5.4.1 Problem instances

First, we decided to compare our negative learning Aco approaches with the Aco approaches for MaxSAT by Pinto et al. [134] and Villagra and Baran [135]. Next, we also want to compare our approaches with the state-of-the-art MaxSAT solvers *SATLike-c(w)* and *SLSMcs* from the MaxSAT Evaluation 2020 (MSE 2020). For this purpose, we tested our negative learning Aco variants on the problem instances from [134, 135] as well as on a wide range of problem instances from MSE 2016 (<http://maxsat.ia.udl.cat/benchmarks/>, accessed on the 21st of April 2021) and MSE 2020 (<https://maxsat-evaluations.github.io/2020/benchmarks.html>, accessed on the 3rd of June 2021).

The specifications of these MaxSAT instances are given in Table 5.9 (Pinto et al.), Table 5.11 (Villagra and Baran), and Table 5.12 (MSE 2016 and MSE 2020), where n_l , n_x , and n_c denote the number of literals, variables and clauses, respectively. From the work of Pinto et al., we took two unweighted instances used

to test their Aco approach for the static MaxSAT problem. Each of these instances has three literals per clause, 250 variables, and 1065 clauses. The MaxSAT instances from Villagra and Baran consist of *phase-transition* instances (instances 1–25 in Table 5.11) and *over-constrained* instances (instances 26–50 in Table 5.11). Each of these 50 instances has three literals per clause and 50 variables. Each phase-transition instance has 215 clauses while each over-constraint instance has 323 clauses. The chosen instance set from MSE 2020 and MSE 2016 consists of four groups: (1) *maxcut*, (2) *highgirth*, (3) *ramsey*, and (4) *set-covering*. In Table 5.12, we sort these 113 instances according to the number of literals, the number of variables, and the number of clauses. Overall, these instances vary considerably in terms of size and structure.

5.4.2 Algorithm tuning and test settings

The baseline Aco as well as the negative learning variants require well-working configurations of their parameter values. We used the scientific tuning software *irace* [123] for parameter tuning purposes. In particular, we carried out separate tuning runs for each of the considered MaxSAT instance groups. Concerning the instances by Pinto et al., we chose instance number 1 from Table 5.9 for parameter tuning. The parameter values obtained for this instance group are presented in Table 5.1. Pinto et al. employed an Aco approach using a single ant that was evaluated for 100 runs and each run consisted of 100 iterations. Consequently, we limited our algorithms to match the number of their Aco algorithm’s solution constructions. In particular, we limited the execution of $\text{ACO}_{\text{neg}}^+$, ACO_{neg} , and Aco to 6, 14, and 50 iterations, respectively.

Table 5.1 Parameter values obtained for all Aco algorithms concerning the Pinto et al. instances

	Algorithm	n_a	ρ	d_{rate}	ρ^{neg}	t^{sub}
1	$\text{ACO}_{\text{neg}}^+$	16	0.1	0.6	0.5	11
2	ACO_{neg}	7	0.1	0.4	0.5	3
3	Aco	2	0.4	0.2	n/a	n/a

Concerning the instances of Villagra and Baran, we chose the first five instances from each of the two instance types (phase-transition and over-constrained) for tuning. The parameter values obtained for this instance group are presented in Table 5.2. Villagra and Baran employed 10 ants in their Aco variants and limited the executions to 10000 iterations for each of the 10 test runs for every instance. Adjusting to their test setting, we limited the execution of $\text{ACO}_{\text{neg}}^+$, ACO_{neg} , and Aco to 20000, 16666, and 8333 iterations, respectively.

Table 5.2 Parameter values obtained for all Aco algorithms concerning the Villagra and Baran instances

	Algorithm	n_a	ρ	d_{rate}	ρ^{neg}	t^{sub}
1	ACO_{neg}^+	5	0.1	0.7	0.4	18
2	ACO_{neg}	6	0.1	0.7	0.2	17
3	Aco	12	0.1	0.2	n/a	n/a

As shown in Table 5.12, the instance set selected from the MSE 2016 and MSE 2020 Evaluations is very diverse in its specifications. For tuning purposes, we divided these instances into 8 sub-groups based on their type and size: (1) $maxcut_1$, (2) $maxcut_2$, (3) $highgirth$, (4) $ramsey_1$, (5) $ramsey_2$, (6) $setcov_1$, (7) $setcov_2$, and (8) $setcov_3$. We took the first two instances from each of these sub-groups for the tuning process. As an exception, we took the first two instances from each configuration of n_l , n_x , and n_c for the sub-group $highgirth$. Therefore, for this sub-group we used a total of eight instances for tuning. The obtained parameter values are presented in Table 5.3 ($ACO-SAT_{neg}^+$), Table 5.4 ($ACO-SLS_{neg}^+$), Table 5.5 (ACO_{neg}^+), Table 5.6 (ACO_{neg}), and Table 5.7 (Aco). We limited the execution time of all the algorithms tested on this instance group to 300 seconds, corresponding to one of the time limits used for the MSE 2020 (<https://maxsat-evaluations.github.io/2020/rankings.html>, accessed on the 3rd of June 2021).

5.4.3 Results

The empirical results of all algorithms applied to the instances of Pinto et al. are presented in Table 5.9. Note that the results are provided in terms of the average number of satisfied clauses obtained within 100 runs, hence, a higher value represents a better result. Moreover, the results under the header Aco_{Pinto} are the results of the Aco version from Pinto et al. [134]. Additionally, results marked in bold correspond to the best result of the comparison for each table row. In summary, the results show that ACO_{neg}^+ is the best algorithm for these instances. They also show that even though ACO_{neg} is outperformed by CPLEX, it still performs significantly better than Aco_{Pinto} . Compared to Aco, each of our negative learning approaches produces a remarkable improvement over the baseline algorithm.

Table 5.10 shows the empirical results of all the algorithms applied to the instances of Villagra and Baran in a summarized way. In particular, results are averaged over the 25 instances of each of the two instance sub-groups. In addition, the number of instances solved to optimality for each sub-group are given in brackets after the corresponding average results. In the context of these

Table 5.3 Parameter values obtained for ACO-SAT_{neg}⁺, MSE 2016 and 2020 instances

	Instance Group	n_a	ρ	d_{rate}	ρ^{neg}	t^{sub}
1	<i>maxcut</i> ₁	16	0.5	0.7	0.3	14
2	<i>maxcut</i> ₂	17	0.3	0.7	0.3	71
3	<i>highgirth</i>	15	0.1	0.6	0.5	85
4	<i>ramsey</i> ₁	6	0.3	0.8	0.2	132
5	<i>ramsey</i> ₂	20	0.4	0.1	0.2	137
6	<i>setcov</i> ₁	20	0.2	0.8	0.4	64
7	<i>setcov</i> ₂	19	0.4	0.9	0.4	139
8	<i>setcov</i> ₃	13	0.4	0.7	0.4	119

Table 5.4 Parameter values obtained for ACO-SLS_{neg}⁺, MSE 2016 and 2020 instances

	Instance Group	n_a	ρ	d_{rate}	ρ^{neg}	t^{sub}
1	<i>maxcut</i> ₁	16	0.5	0.7	0.3	14
2	<i>maxcut</i> ₂	17	0.3	0.7	0.3	71
3	<i>highgirth</i>	11	0.3	0.3	0.2	2
4	<i>ramsey</i> ₁	6	0.3	0.8	0.2	132
5	<i>ramsey</i> ₂	20	0.4	0.1	0.2	137
6	<i>setcov</i> ₁	20	0.2	0.8	0.4	64
7	<i>setcov</i> ₂	19	0.4	0.9	0.4	139
8	<i>setcov</i> ₃	13	0.4	0.7	0.4	119

Table 5.5 Parameter values obtained for Aco_{neg}⁺, MSE 2016 and 2020 instances

	Instance Group	n_a	ρ	d_{rate}	ρ^{neg}	t^{sub}
1	<i>maxcut</i> ₁	15	0.4	0.0	0.3	20
2	<i>maxcut</i> ₂	10	0.4	0.0	0.3	25
3	<i>highgirth</i>	7	0.1	0.4	0.4	3
4	<i>ramsey</i> ₁	3	0.2	0.8	0.3	3
5	<i>ramsey</i> ₂	4	0.1	0.2	0.3	2
6	<i>setcov</i> ₁	20	0.1	0.9	0.3	10
7	<i>setcov</i> ₂	6	0.3	0.6	0.5	27
8	<i>setcov</i> ₃	19	0.2	0.9	0.1	29

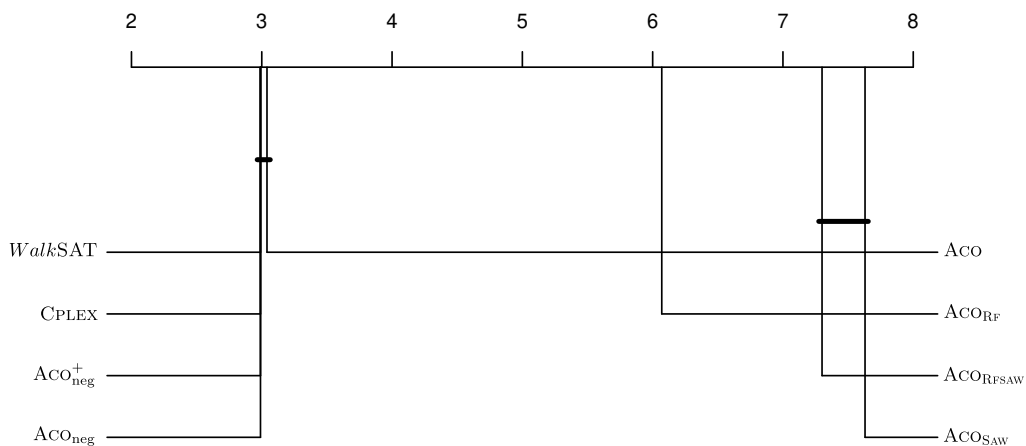
Table 5.6 Parameter values obtained for Aco_{neg}, MSE 2016 and 2020 instances

	Instance Group	n_a	ρ	d_{rate}	ρ^{neg}	t^{sub}
1	<i>maxcut</i> ₁	2	0.4	0.9	0.5	10
2	<i>maxcut</i> ₂	2	0.1	0.9	0.3	91
3	<i>highgirth</i>	7	0.1	0.4	0.5	18
4	<i>ramsey</i> ₁	16	0.3	0.1	0.3	1
5	<i>ramsey</i> ₂	16	0.3	0.0	0.4	1
6	<i>setcov</i> ₁	3	0.2	0.9	0.4	67
7	<i>setcov</i> ₂	18	0.4	0.9	0.4	56
8	<i>setcov</i> ₃	3	0.2	0.9	0.1	71

Table 5.7 Parameter values obtained for Aco, MSE 2016 and 2020 instances

	Instance Group	n_a	ρ	d_{rate}	ρ^{neg}	t^{sub}
1	<i>maxcut</i> ₁	19	0.2	0.9	n/a	n/a
2	<i>maxcut</i> ₂	5	0.1	0.9	n/a	n/a
3	<i>highgirth</i>	13	0.1	0.3	n/a	n/a
4	<i>ramsey</i> ₁	10	0.3	0.1	n/a	n/a
5	<i>ramsey</i> ₂	12	0.5	0.0	n/a	n/a
6	<i>setcov</i> ₁	15	0.2	0.9	n/a	n/a
7	<i>setcov</i> ₂	19	0.4	0.9	n/a	n/a
8	<i>setcov</i> ₃	20	0.4	0.9	n/a	n/a

instances, our negative learning Aco variants are compared to the MaxSAT solver *WalkSAT* as well as the Aco variants from Villagra and Baran: ACO_{SAW} , ACO_{RF} , and ACO_{RFSAW} . Each result in the table indicates the average number of satisfied clauses obtained within 10 algorithm runs. Additionally, we made use of the R package *scmp* [124] to facilitate the interpretation of the results in Table 5.11. This statistical tool works as follows. First, the results from all algorithms are compared simultaneously using the Friedman test for obtaining the rejection to the hypothesis that all the algorithms perform equally. Next, a set of pairwise comparisons are performed using the Nemenyi post-hoc test [125] and, eventually, the output of this statistical analysis is presented as a *critical difference* (CD) plot in Fig. 5.2. The horizontal axis of the CD plot represents the range of algorithm ranks, while each of the vertical lines represents the average rank of the corresponding algorithm. Bold horizontal lines connecting algorithm markers means that the corresponding algorithms performed statistically equivalent i.e. the critical difference is not greater than the significance level of 0.05. Fig. 5.2

**Fig. 5.2** Critical difference plot concerning the results of the test on Villagra and Baran's instances

shows that all of our negative learning approaches, as well as the baseline Aco, perform statistically better than each of the Aco versions from Villagra and Baran. Furthermore, all our Aco versions perform statistically equivalent to the MaxSAT solver *WalkSAT* and the ILP solver *Cplex* for this instance group.

Table 5.12 presents the results of all the algorithms applied to the selected MSE 2016 and MSE 2020 instances. Note that this table provides the best result of each (stochastic) algorithm, while the average results are presented in Table 5.13. Also note that the results in Table 5.12 are given in terms of the number of *violated* clauses. Thus, a lower value represents a better result. For facilitating the interpretation of the results obtained for this instance group, we additionally present the data from Table 5.12 in a summarized way in Table 5.8. In addition, we conducted the same statistical analysis with *scmamp* (as explained above) to the data from Table 5.12 and present the result as a CD plot in Fig. 5.3.

In particular, Table 5.8 shows the number of instances for which each one of the negative learning Aco variants performs better, worse, or equally with its individual algorithmic components. These summarized results indicate that, in general, each of our negative learning Aco variants improves both over the baseline Aco and over each of the solvers that are used internally for solving sub-instances. Among all the negative learning Aco variants, $\text{ACO-SAT}_{\text{neg}}^+$ achieved the highest number of improvements over the baseline Aco. It improves in 108 of 113 problem instances. Compared with the internally used MaxSAT solver *SATLike-c(w)*, however, it improves over the result of *SATLike-c(w)* only in 11.5% of all the problem instances. Nevertheless, $\text{ACO-SAT}_{\text{neg}}^+$ can be called the best algorithm for this instance group according to the CD plot in Fig. 5.3, even though no statistical difference can be detected with respect to *SATLike-c(w)* and $\text{ACO}_{\text{neg}}^+$. Furthermore, all remaining negative learning Aco variants also significantly improve over both the baseline Aco approach and their internally

Table 5.8 Comparative performance of our negative learning Aco variants with their individual algorithmic components

Aco variant - component	Comparison		
	better	worse	equal
1 $\text{ACO-SAT}_{\text{neg}}^+ - \text{SATLike-c(w)}$	13	9	91
2 $\text{ACO-SAT}_{\text{neg}}^+ - \text{ACO}$	108	0	5
3 $\text{ACO-SLS}_{\text{neg}}^+ - \text{SLSMcs}$	87	2	24
4 $\text{ACO-SLS}_{\text{neg}}^+ - \text{ACO}$	105	1	7
5 $\text{ACO}_{\text{neg}}^+ - \text{Cplex}$	103	1	9
6 $\text{ACO}_{\text{neg}}^+ - \text{ACO}$	107	0	6
7 $\text{ACO}_{\text{neg}} - \text{Cplex}$	87	20	6
8 $\text{ACO}_{\text{neg}} - \text{ACO}$	88	14	11

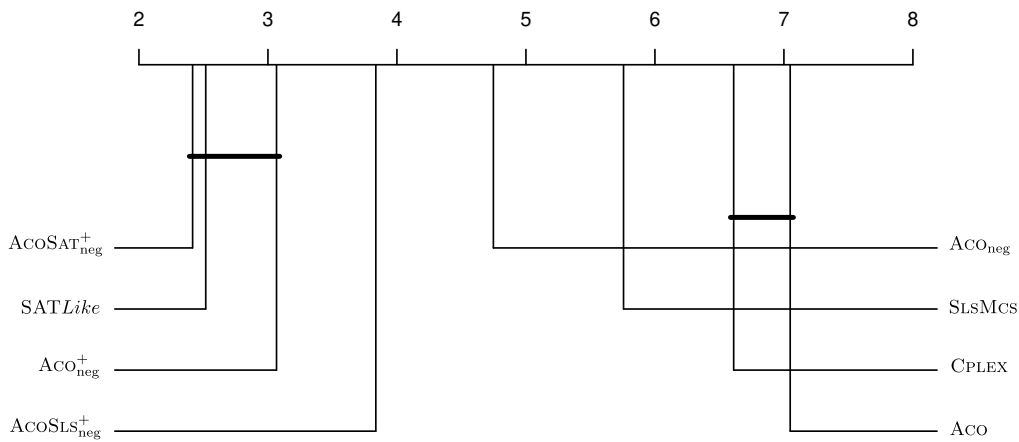


Fig. 5.3 Critical difference plot concerning the results of the tests for the MSE 2020 and 2016 instances

used solvers. Even ACO_{neg} , the variant that does not take advantage of the internally derived CPLEX result for updating its own best result, improves over both the baseline Aco approach and its constituent solver CPLEX in the context of most of the problem instances. Furthermore, the statistical analysis graphically presented in Fig. 5.3 also shows that ACO_{neg} outperforms the MaxSAT solver SLSMcs . Hence, this proves the effectiveness of our negative learning strategy. Moreover, these results indicate an interesting aspect: our negative learning Aco framework can potentially be used for improving the results of MaxSAT solvers that are already very successful in standalone-mode.

5.4.4 Search Trajectory Network Analysis

As in the cases of the MDKP and MDS problems, we used STN plots in this section to learn something about the algorithms' behavior. In particular, we provide four plots that compare STNs composed of the trajectories of five algorithm variants: the baseline Aco variant and four negative leaning Aco variants. Figures 5.4 and Fig. 5.5 show STN plots for problem instance HG-4SAT-V150-C1350-100 from the MSE 2016 while Fig. 5.6 and Fig. 5.7 show the ones for problem instance `scpc12_maxsat` from the MSE 2020.

These STN plots were created by using the technique described in [127] and the R scripts available at <https://github.com/gabro8a/STNs.git>. All the data for these STN plots was collected by applying each of the five algorithm variants 10 times to problem instances HG-4SAT-V150-C1350-100 and `scpc12_maxsat` using the the same parameter value settings used for their performance test. All STN plots in this chapter were subjected to 90% search space partitioning. See

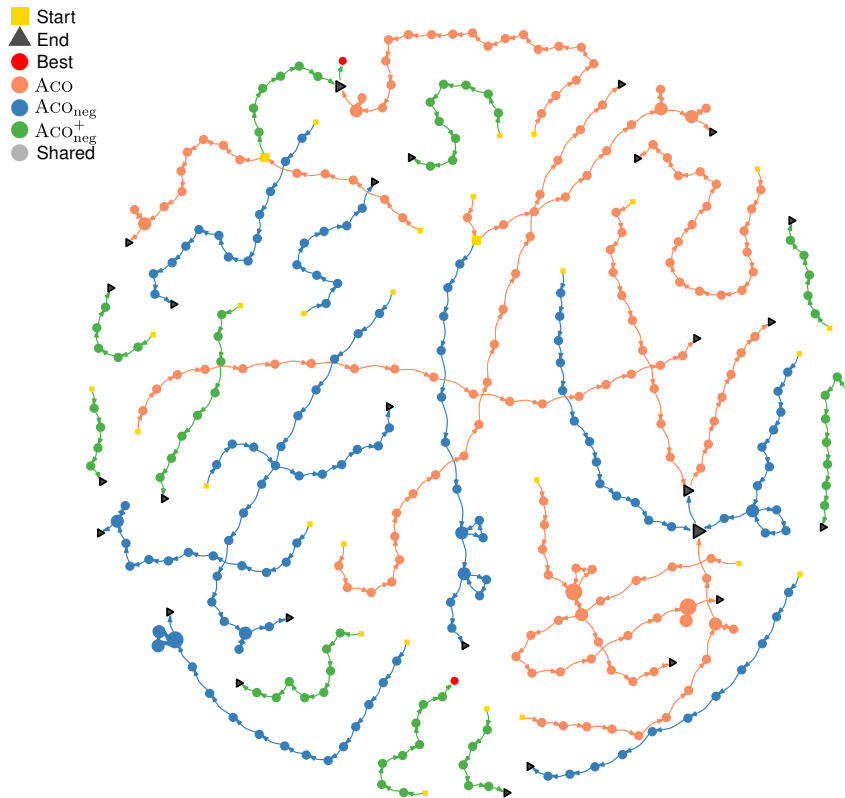


Fig. 5.4 Search trajectory networks concerning Aco , Aco_{neg} , and Aco_{neg}^+ applied to problem instance HG-4SAT-V150-C1350-100

Section 3.7.3 for the definition of a STN and the description of a STN plot.

Figure 5.4 shows a comparison between the baseline Aco algorithm and the negative learning Aco variants Aco_{neg} and Aco_{neg}^+ that use $Cplex$ as their additional algorithmic component. This plot shows only a small number of overlaps between trajectories of different algorithm variants. However, we can see some overlaps (or loops) in the individual trajectories, especially those from the baseline Aco and Aco_{neg} . In contrast to them, the trajectories of Aco_{neg}^+ have almost no overlap with those from other algorithm variants or among themselves. Moreover, the ones from Aco_{neg}^+ are much shorter than the trajectories of the other two algorithm variants. Finally, we can see that two trajectories of Aco_{neg}^+ find two different optimal solutions.

Figure 5.5 shows an STN plot concerning our three negative learning Aco variants— Aco_{neg}^+ , $Aco-SLS_{neg}^+$, and $Aco-SAT_{neg}^+$ —that use three different optimization approaches— $Cplex$, $SLSMcs$, and $SATLike$ —as their additional algorithmic components. We can see that the trajectories of $Aco-SLS_{neg}^+$ and $Aco-SAT_{neg}^+$ have different characteristics than the ones of Aco_{neg}^+ . All trajectories of $Aco-SAT_{neg}^+$ obtain some optimal solutions. Most of the trajectories of $Aco-SLS_{neg}^+$ converge to a common location in the plot; one of these trajectories then finds an

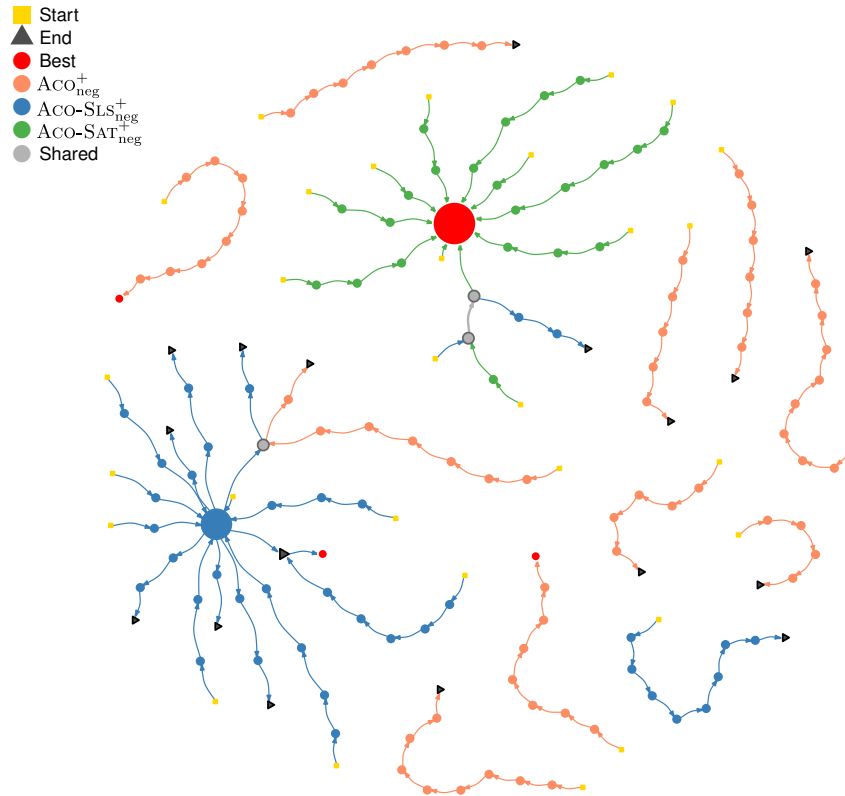


Fig. 5.5 Search trajectory networks concerning $\text{ACO}_{\text{neg}}^+$, $\text{ACO-SLS}_{\text{neg}}^+$, and $\text{ACO-SAT}_{\text{neg}}^+$ applied to problem instance HG-4SAT-V150-C1350-100

optimal solution, different to the one found by $\text{ACO-SAT}_{\text{neg}}^+$. This finding shows that both $\text{ACO-SAT}_{\text{neg}}^+$ and $\text{ACO-SLS}_{\text{neg}}^+$ are strongly attracted by two different locations in the search space. In contrast, the trajectories of $\text{ACO}_{\text{neg}}^+$ are distributed in different locations. Nevertheless, two of its trajectories can find optimal solutions, which causes that $\text{ACO}_{\text{neg}}^+$ performs better than $\text{ACO-SLS}_{\text{neg}}^+$. This analysis is in accordance with the numerical result comparison in Table 5.12 and Table 5.13, which shows that $\text{ACO-SAT}_{\text{neg}}^+$ exhibits the best performance for this problem instance, followed by $\text{ACO}_{\text{neg}}^+$ and then $\text{ACO-SLS}_{\text{neg}}^+$.

Figure 5.6 shows a comparison among the trajectories of ACO , ACO_{neg} , and $\text{ACO}_{\text{neg}}^+$ applied to problem instance `scplr12_maxsat`. The plot shows many overlaps between trajectories of the three algorithm variants. Interestingly, the best solution in this comparison is found by a trajectory of $\text{ACO}_{\text{neg}}^+$, located relatively far from the point of attraction of other trajectories. A similar observation can also be made in Fig. 5.7. The location of the best solution found in this comparison is found by a trajectory of $\text{ACO-SLS}_{\text{neg}}^+$ in an area which is apparently not related to areas of local minima in the plot. Accordingly, we believe that the exploration capability of the algorithm variant plays a more important role in finding good solutions for this problem instance.

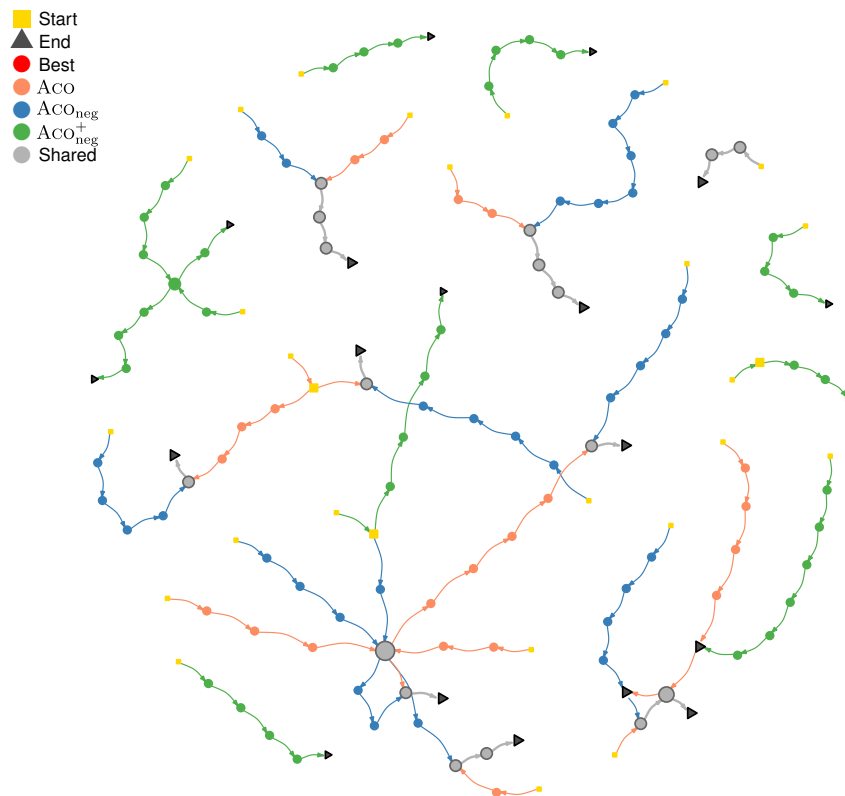


Fig. 5.6 Search trajectory networks concerning ACO, ACO_{neg}, and ACO_{neg}⁺ applied to problem instance `scplr12_maxsat`

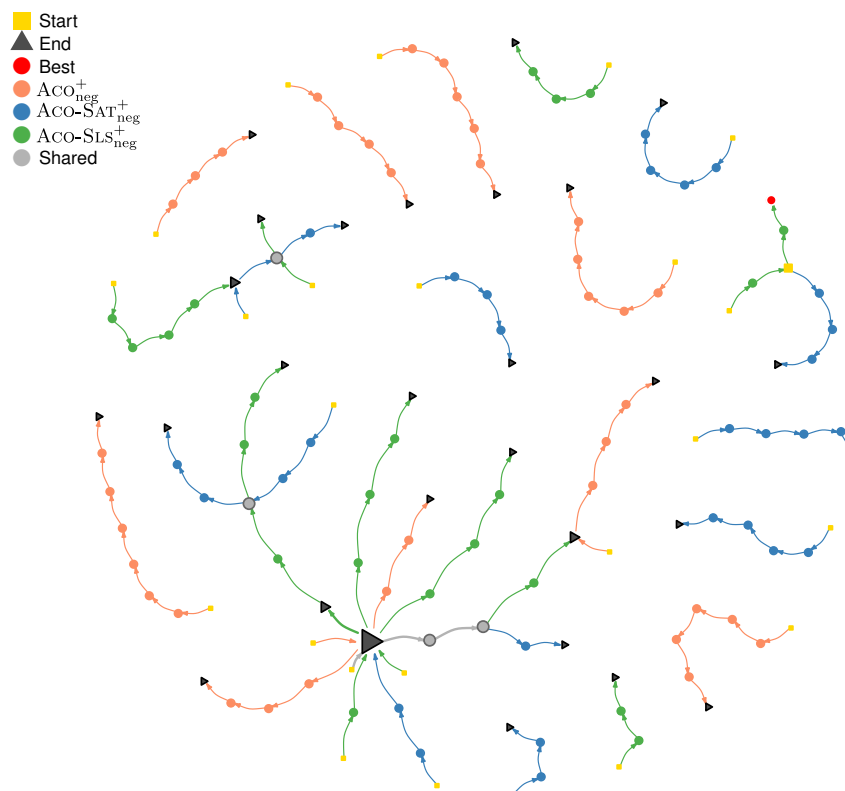


Fig. 5.7 Search trajectory networks concerning ACO_{neg}⁺, ACO-SAT_{neg}⁺, and ACO-SLS_{neg}⁺ applied to problem instance `scplr12_maxsat`

5.5 CONCLUSIONS

Ant colony optimization (Aco) was subject to several major improvements and extensions in its history. Most of these extensions, however, deal exclusively with the improvement of the positive learning mechanisms. Observing that negative learning works in synergy with positive learning in nature, several works were presented in the literature to integrate negative learning into Aco in the past decades. Most of these works, however, produced limited successes. In previous work, we introduced a novel strategy for the implementation and use of negative learning in Aco. In contrast with other negative learning proposals, we made use of an additional algorithmic component to provide negative feedback to the main Aco algorithm. Further, we also implemented an effective cooperation mechanism between the main Aco approach and the additional algorithmic component through the use of a sub-instance that is not only reduced in size but that also contains high quality solutions. Our strategy was proven to be useful for the improvement of the performance of the baseline Aco algorithm in the context of a range of sub-set selection problems.

In this work, we applied the negative learning Aco strategy to the MaxSAT problem, an optimization problem which is substantially different from the problems considered to date. Moreover, this problem is an extremely well studied optimization problem for which a wide range of high performance solvers are available for comparison. Also, Aco approaches were rarely implemented for this optimization problem and most of the existing implementations are far from being able to compete with state-of-the-art approaches. Hence, testing our negative learning proposal on MaxSAT provides a good opportunity to demonstrate the general applicability as well as the effectiveness of our approach. In addition to the ILP solver CPLEX that we already employed in previous work, we made use of two high-performance MaxSAT solvers, *SATLike-c(w)* and *SLSMcs*, as new options for the additional algorithmic component to be internally used with negative learning Aco. In this study, we evaluated the resulting negative learning Aco variants on three instance groups. In the context of the first two instance groups, the results show that our negative learning Aco variants perform significantly better than the baseline Aco as well as existing Aco variants from the literature. In the third instance group, consisting of instances used for recent MaxSAT evaluations, the obtained results showed that all our negative learning Aco variants were able to improve over the baseline Aco approach and over each

of the internally used solvers. This is, in our opinion, a very interesting result, as it shows that high-performance MaxSAT solvers can even be improved by using them for solving sub-instances within our framework.

Table 5.9 Average results for Pinto's instances

	Instance	n_l	n_x	n_c	ACO_{Pinto}	Cplex	ACO_{neg}^+	ACO_{neg}	Aco
1	uuf250-01	3	250	1065	1047.20	1062.00	1063.28	1055.88	1040.64
2	uuf250-02	3	250	1065	1043.90	1060.00	1061.92	1053.02	1037.64

Table 5.10 Summary of results for Villagra and Baran's instances

	Instance Type	n_l	n_x	n_c	ACO_{SAW}	ACO_{RF}	ACO_{RFSAW}	WalkSAT	Cplex	ACO_{neg}^+	ACO_{neg}	Aco
1	Phase transition (25 instances)	3	50	215	210.656(0)	212.244(0)	211.184(0)	214.72(25)	214.72(25)	214.72(25)	214.72(25)	214.716(24)
2	Over constrained (25 instances)	3	50	323	306.436(0)	314.096(0)	306.964(0)	317.44(25)	317.44(25)	317.44(25)	317.38(22)	317.224(16)

Table 5.11 Average results of all algorithms tested on Villagra and Baran's instances

	Instance	n_l	n_x	n_c	ACO_{SAW}	ACO_{RF}	ACO_{RFSAW}	WalkSAT	Cplex	ACO_{neg}^+	ACO_{neg}	Aco
1	VB-3SAT-V50-C215-1	3	50	215	211.00	212.40	210.80	215.00	215.00	215.00	215.00	215.00
2	VB-3SAT-V50-C215-2	3	50	215	212.40	212.40	212.10	215.00	215.00	215.00	215.00	215.00
3	VB-3SAT-V50-C215-3	3	50	215	212.50	213.20	214.00	215.00	215.00	215.00	215.00	215.00
4	VB-3SAT-V50-C215-4	3	50	215	211.80	212.60	212.50	215.00	215.00	215.00	215.00	214.90
5	VB-3SAT-V50-C215-5	3	50	215	209.60	211.90	208.60	214.00	214.00	214.00	214.00	214.00
6	VB-3SAT-V50-C215-6	3	50	215	207.20	210.10	208.90	214.00	214.00	214.00	214.00	214.00
7	VB-3SAT-V50-C215-7	3	50	215	209.80	211.20	210.30	215.00	215.00	215.00	215.00	215.00
8	VB-3SAT-V50-C215-8	3	50	215	209.20	212.60	211.00	215.00	215.00	215.00	215.00	215.00
9	VB-3SAT-V50-C215-9	3	50	215	207.20	211.70	210.60	215.00	215.00	215.00	215.00	215.00
10	VB-3SAT-V50-C215-10	3	50	215	210.80	211.70	212.10	215.00	215.00	215.00	215.00	215.00

Continuation of Table 5.11 Average results of all algorithms tested on Villagra and Baran's instances

Instance	n_l	n_x	n_c	ACO _{SAW}	ACO _{Rf}	ACO _{RfSAW}	WalkSAT	CPLEX	ACO _{neg} ⁺	ACO _{neg}	ACO	
11	VB-3SAT-V50-C215-11	3	50	215	212.20	213.50	212.10	215.00	215.00	215.00	215.00	215.00
12	VB-3SAT-V50-C215-12	3	50	215	209.20	211.30	208.90	214.00	214.00	214.00	214.00	214.00
13	VB-3SAT-V50-C215-13	3	50	215	211.00	211.80	211.10	215.00	215.00	215.00	215.00	215.00
14	VB-3SAT-V50-C215-14	3	50	215	211.30	212.10	211.00	215.00	215.00	215.00	215.00	215.00
15	VB-3SAT-V50-C215-15	3	50	215	212.50	213.20	212.90	215.00	215.00	215.00	215.00	215.00
16	VB-3SAT-V50-C215-16	3	50	215	210.30	212.10	209.80	214.00	214.00	214.00	214.00	214.00
17	VB-3SAT-V50-C215-17	3	50	215	211.00	211.50	211.30	215.00	215.00	215.00	215.00	215.00
18	VB-3SAT-V50-C215-18	3	50	215	210.60	212.70	211.30	215.00	215.00	215.00	215.00	215.00
19	VB-3SAT-V50-C215-19	3	50	215	213.20	213.40	212.30	215.00	215.00	215.00	215.00	215.00
20	VB-3SAT-V50-C215-20	3	50	215	210.70	212.70	211.90	215.00	215.00	215.00	215.00	215.00
21	VB-3SAT-V50-C215-21	3	50	215	208.00	213.00	211.60	215.00	215.00	215.00	215.00	215.00
22	VB-3SAT-V50-C215-22	3	50	215	210.40	211.00	208.90	215.00	215.00	215.00	215.00	215.00
23	VB-3SAT-V50-C215-23	3	50	215	212.30	213.20	212.40	214.00	214.00	214.00	214.00	214.00
24	VB-3SAT-V50-C215-24	3	50	215	211.30	212.90	212.10	214.00	214.00	214.00	214.00	214.00
25	VB-3SAT-V50-C215-25	3	50	215	210.90	211.90	211.10	214.00	214.00	214.00	214.00	214.00
26	VB-3SAT-V50-C323-1	3	50	323	304.40	313.80	304.20	317.00	317.00	317.00	317.00	317.00
27	VB-3SAT-V50-C323-2	3	50	323	307.10	314.30	309.90	318.00	318.00	318.00	317.90	317.30
28	VB-3SAT-V50-C323-3	3	50	323	308.40	314.60	305.60	317.00	317.00	317.00	317.00	317.00
29	VB-3SAT-V50-C323-4	3	50	323	312.00	315.70	313.20	318.00	318.00	318.00	318.00	317.90
30	VB-3SAT-V50-C323-5	3	50	323	303.00	312.80	301.10	316.00	316.00	316.00	316.00	316.00
31	VB-3SAT-V50-C323-6	3	50	323	307.40	315.20	308.00	318.00	318.00	318.00	318.00	317.30
32	VB-3SAT-V50-C323-7	3	50	323	305.20	314.60	306.00	318.00	318.00	318.00	318.00	318.00
33	VB-3SAT-V50-C323-8	3	50	323	305.30	314.50	305.50	317.00	317.00	317.00	317.00	317.00
34	VB-3SAT-V50-C323-9	3	50	323	304.20	312.40	304.80	316.00	316.00	316.00	316.00	316.00
35	VB-3SAT-V50-C323-10	3	50	323	307.00	313.80	307.80	318.00	318.00	318.00	318.00	317.10
36	VB-3SAT-V50-C323-11	3	50	323	308.00	314.90	305.40	318.00	318.00	318.00	318.00	318.00
37	VB-3SAT-V50-C323-12	3	50	323	304.50	313.80	305.60	317.00	317.00	317.00	317.00	317.00
38	VB-3SAT-V50-C323-13	3	50	323	305.30	315.60	310.70	318.00	318.00	318.00	318.00	317.80
39	VB-3SAT-V50-C323-14	3	50	323	312.60	316.00	313.30	320.00	320.00	320.00	320.00	320.00
40	VB-3SAT-V50-C323-15	3	50	323	307.90	313.90	307.30	317.00	317.00	317.00	317.00	317.00

Continuation of Table 5.11 Average results of all algorithms tested on Villagra and Baran's instances

Instance	n_l	n_x	n_c	ACO _{SAW}	ACO _{Rf}	ACO _{RfSAW}	WalkSAT	CPLEX	ACO _{neg} ⁺	ACO _{neg}	ACO	
41	VB-3SAT-V50-C323-16	3	50	323	311.30	314.80	311.10	319.00	319.00	319.00	319.00	319.00
42	VB-3SAT-V50-C323-17	3	50	323	311.40	318.20	313.40	320.00	320.00	320.00	320.00	320.00
43	VB-3SAT-V50-C323-18	3	50	323	301.10	311.90	300.80	316.00	316.00	316.00	316.00	315.90
44	VB-3SAT-V50-C323-19	3	50	323	303.50	312.20	305.90	316.00	316.00	316.00	316.00	316.00
45	VB-3SAT-V50-C323-20	3	50	323	305.60	313.70	305.30	317.00	317.00	317.00	317.00	317.00
46	VB-3SAT-V50-C323-21	3	50	323	303.20	312.40	303.70	317.00	317.00	317.00	316.80	316.50
47	VB-3SAT-V50-C323-22	3	50	323	304.60	313.10	304.60	316.00	316.00	316.00	316.00	316.00
48	VB-3SAT-V50-C323-23	3	50	323	302.60	311.50	303.60	316.00	316.00	316.00	314.80	314.60
49	VB-3SAT-V50-C323-24	3	50	323	305.50	313.90	307.20	318.00	318.00	318.00	318.00	317.20
50	VB-3SAT-V50-C323-25	3	50	323	309.80	314.80	310.10	318.00	318.00	318.00	318.00	318.00
42	VB-3SAT-V50-C323-17	3	50	323	311.40	318.20	313.40	320.00	320.00	320.00	320.00	320.00
43	VB-3SAT-V50-C323-18	3	50	323	301.10	311.90	300.80	316.00	316.00	316.00	316.00	315.90
44	VB-3SAT-V50-C323-19	3	50	323	303.50	312.20	305.90	316.00	316.00	316.00	316.00	316.00
45	VB-3SAT-V50-C323-20	3	50	323	305.60	313.70	305.30	317.00	317.00	317.00	317.00	317.00
46	VB-3SAT-V50-C323-21	3	50	323	303.20	312.40	303.70	317.00	317.00	317.00	316.80	316.50
47	VB-3SAT-V50-C323-22	3	50	323	304.60	313.10	304.60	316.00	316.00	316.00	316.00	316.00
48	VB-3SAT-V50-C323-23	3	50	323	302.60	311.50	303.60	316.00	316.00	316.00	314.80	314.60
49	VB-3SAT-V50-C323-24	3	50	323	305.50	313.90	307.20	318.00	318.00	318.00	318.00	317.20
50	VB-3SAT-V50-C323-25	3	50	323	309.80	314.80	310.10	318.00	318.00	318.00	318.00	318.00

Table 5.12 Best results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	CPLEX	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
1	san400_0.7_3.clq	<i>maxcut</i> ₁	2	40	1094	230	230	232	230	230	230	245	238
2	san200_0.9_1.clq	<i>maxcut</i> ₁	2	40	1392	313	313	317	313	313	315	325	318
3	p_hat500-1.clq	<i>maxcut</i> ₁	2	42	474	75	75	75	75	75	75	84	79
4	p_hat500-3.clq	<i>maxcut</i> ₁	2	42	1310	284	284	289	284	284	284	297	293
5	maxcut-140-630-0.8-3	<i>maxcut</i> ₂	2	140	1258	165	165	168	165	165	165	234	236

Continuation of Table 5.12 Best results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	Cplex	ACO-SAT ⁺ _{neg}	ACO-SLS ⁺ _{neg}	ACO ⁺ _{neg}	ACO _{neg}	ACO	
6	maxcut-140-630-0.8-4	<i>maxcut</i> ₂	2	140	1258	165	165	170	165	165	165	233	235
7	maxcut-140-630-0.8-20	<i>maxcut</i> ₂	2	140	1258	165	165	165	165	165	165	230	226
8	maxcut-140-630-0.8-44	<i>maxcut</i> ₂	2	140	1258	160	160	162	160	160	160	231	228
9	maxcut-140-630-0.7-3	<i>maxcut</i> ₂	2	140	1260	168	169	173	168	168	169	237	232
10	maxcut-140-630-0.7-33	<i>maxcut</i> ₂	2	140	1260	165	165	169	165	165	165	233	233
11	maxcut-140-630-0.7-49	<i>maxcut</i> ₂	2	140	1260	164	169	170	164	164	164	232	225
12	HG-3SAT-V250-C1000-1	<i>highgirth</i>	3	250	1000	5	13	12	5	7	6	6	10
13	HG-3SAT-V250-C1000-2	<i>highgirth</i>	3	250	1000	5	11	13	5	6	6	7	14
14	HG-3SAT-V250-C1000-3	<i>highgirth</i>	3	250	1000	5	12	13	5	7	6	8	15
15	HG-3SAT-V250-C1000-4	<i>highgirth</i>	3	250	1000	6	11	13	6	8	7	8	15
16	HG-3SAT-V250-C1000-5	<i>highgirth</i>	3	250	1000	6	13	13	6	9	7	6	17
17	HG-3SAT-V250-C1000-6	<i>highgirth</i>	3	250	1000	6	14	10	6	7	6	7	14
18	HG-3SAT-V250-C1000-7	<i>highgirth</i>	3	250	1000	6	8	11	6	8	6	6	16
19	HG-3SAT-V250-C1000-8	<i>highgirth</i>	3	250	1000	5	11	14	5	7	7	6	13
20	HG-3SAT-V250-C1000-9	<i>highgirth</i>	3	250	1000	6	13	14	6	7	7	8	12
21	HG-3SAT-V250-C1000-10	<i>highgirth</i>	3	250	1000	6	15	10	6	6	6	7	14
22	HG-3SAT-V250-C1000-11	<i>highgirth</i>	3	250	1000	6	9	13	6	8	6	8	12
23	HG-3SAT-V250-C1000-12	<i>highgirth</i>	3	250	1000	6	10	16	6	8	6	6	13
24	HG-3SAT-V250-C1000-13	<i>highgirth</i>	3	250	1000	5	9	12	6	8	5	6	14
25	HG-3SAT-V250-C1000-14	<i>highgirth</i>	3	250	1000	6	10	14	5	6	6	6	14
26	HG-3SAT-V250-C1000-15	<i>highgirth</i>	3	250	1000	5	10	12	5	5	5	7	14
27	HG-3SAT-V250-C1000-16	<i>highgirth</i>	3	250	1000	5	14	12	5	6	5	7	12
28	HG-3SAT-V250-C1000-17	<i>highgirth</i>	3	250	1000	6	13	14	6	8	7	8	14
29	HG-3SAT-V250-C1000-18	<i>highgirth</i>	3	250	1000	6	10	11	6	7	7	8	13
30	HG-3SAT-V250-C1000-19	<i>highgirth</i>	3	250	1000	5	10	10	6	8	7	7	14
31	HG-3SAT-V250-C1000-20	<i>highgirth</i>	3	250	1000	7	10	13	7	8	7	7	14
32	HG-3SAT-V250-C1000-21	<i>highgirth</i>	3	250	1000	4	10	10	5	7	6	6	11
33	HG-3SAT-V250-C1000-22	<i>highgirth</i>	3	250	1000	5	16	11	5	7	6	6	14
34	HG-3SAT-V250-C1000-23	<i>highgirth</i>	3	250	1000	6	13	12	6	7	7	7	15
35	HG-3SAT-V250-C1000-24	<i>highgirth</i>	3	250	1000	5	14	9	5	7	5	7	12

Continuation of Table 5.12 Best results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	Cplex	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
36	HG-3SAT-V250-C1000-100	<i>highgirth</i>	3	250	1000	7	12	14	7	7	7	8	17
37	HG-3SAT-V300-C1200-1	<i>highgirth</i>	3	300	1200	7	16	17	7	9	8	9	20
38	HG-3SAT-V300-C1200-2	<i>highgirth</i>	3	300	1200	6	12	13	5	10	8	7	23
39	HG-3SAT-V300-C1200-3	<i>highgirth</i>	3	300	1200	6	14	9	8	10	8	7	22
40	HG-3SAT-V300-C1200-4	<i>highgirth</i>	3	300	1200	7	12	13	7	10	8	10	22
41	HG-3SAT-V300-C1200-5	<i>highgirth</i>	3	300	1200	6	13	16	6	11	8	8	24
42	HG-3SAT-V300-C1200-6	<i>highgirth</i>	3	300	1200	6	15	14	6	11	6	10	16
43	HG-3SAT-V300-C1200-7	<i>highgirth</i>	3	300	1200	5	16	15	5	9	6	10	21
44	HG-3SAT-V300-C1200-8	<i>highgirth</i>	3	300	1200	8	16	14	8	10	8	8	21
45	HG-3SAT-V300-C1200-9	<i>highgirth</i>	3	300	1200	7	13	19	8	10	7	13	22
46	HG-3SAT-V300-C1200-10	<i>highgirth</i>	3	300	1200	7	12	13	7	9	8	9	26
47	HG-3SAT-V300-C1200-11	<i>highgirth</i>	3	300	1200	6	17	14	6	9	8	10	22
48	HG-3SAT-V300-C1200-12	<i>highgirth</i>	3	300	1200	7	13	14	7	10	8	8	22
49	HG-3SAT-V300-C1200-13	<i>highgirth</i>	3	300	1200	7	10	15	7	10	8	9	22
50	HG-3SAT-V300-C1200-14	<i>highgirth</i>	3	300	1200	6	12	14	6	7	8	7	21
51	HG-3SAT-V300-C1200-15	<i>highgirth</i>	3	300	1200	7	16	12	7	9	8	10	25
52	HG-3SAT-V300-C1200-16	<i>highgirth</i>	3	300	1200	6	17	14	6	9	8	10	18
53	HG-3SAT-V300-C1200-17	<i>highgirth</i>	3	300	1200	7	15	17	7	7	7	10	22
54	HG-3SAT-V300-C1200-18	<i>highgirth</i>	3	300	1200	7	11	13	7	8	8	9	22
55	HG-3SAT-V300-C1200-19	<i>highgirth</i>	3	300	1200	6	16	15	7	11	8	11	22
56	HG-3SAT-V300-C1200-20	<i>highgirth</i>	3	300	1200	6	14	15	6	9	9	9	17
57	HG-3SAT-V300-C1200-21	<i>highgirth</i>	3	300	1200	6	9	16	5	8	8	9	21
58	HG-3SAT-V300-C1200-22	<i>highgirth</i>	3	300	1200	6	17	18	6	9	6	11	15
59	HG-3SAT-V300-C1200-23	<i>highgirth</i>	3	300	1200	7	12	15	7	8	7	8	18
60	HG-3SAT-V300-C1200-24	<i>highgirth</i>	3	300	1200	7	12	18	7	8	7	10	25
61	HG-3SAT-V300-C1200-100	<i>highgirth</i>	3	300	1200	7	14	14	7	10	8	8	25
62	HG-4SAT-V100-C900-2	<i>highgirth</i>	4	100	900	2	4	4	2	2	2	2	3
63	HG-4SAT-V100-C900-4	<i>highgirth</i>	4	100	900	2	4	3	2	2	2	2	4
64	HG-4SAT-V100-C900-7	<i>highgirth</i>	4	100	900	2	6	3	2	3	2	3	3
65	HG-4SAT-V100-C900-14	<i>highgirth</i>	4	100	900	2	6	4	2	2	2	2	3

Continuation of Table 5.12 Best results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	CPLEX	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
66	HG-4SAT-V100-C900-19	<i>highgirth</i>	4	100	900	2	4	4	2	3	2	3	3
67	HG-4SAT-V100-C900-20	<i>highgirth</i>	4	100	900	2	5	2	2	2	2	2	3
68	HG-4SAT-V100-C900-23	<i>highgirth</i>	4	100	900	2	3	4	2	2	2	2	3
69	HG-4SAT-V150-C1350-1	<i>highgirth</i>	4	150	1350	1	5	8	1	3	2	2	7
70	HG-4SAT-V150-C1350-2	<i>highgirth</i>	4	150	1350	2	7	8	2	3	2	1	6
71	HG-4SAT-V150-C1350-3	<i>highgirth</i>	4	150	1350	2	6	6	1	1	1	1	2
72	HG-4SAT-V150-C1350-4	<i>highgirth</i>	4	150	1350	2	5	7	1	3	1	1	5
73	HG-4SAT-V150-C1350-5	<i>highgirth</i>	4	150	1350	2	7	7	2	2	2	2	5
74	HG-4SAT-V150-C1350-6	<i>highgirth</i>	4	150	1350	1	9	6	2	2	1	3	6
75	HG-4SAT-V150-C1350-7	<i>highgirth</i>	4	150	1350	2	5	7	2	2	2	3	5
76	HG-4SAT-V150-C1350-8	<i>highgirth</i>	4	150	1350	2	8	10	2	3	2	4	8
77	HG-4SAT-V150-C1350-9	<i>highgirth</i>	4	150	1350	2	7	8	2	2	2	3	7
78	HG-4SAT-V150-C1350-10	<i>highgirth</i>	4	150	1350	2	8	7	1	2	2	2	7
79	HG-4SAT-V150-C1350-11	<i>highgirth</i>	4	150	1350	2	4	5	2	3	2	2	5
80	HG-4SAT-V150-C1350-12	<i>highgirth</i>	4	150	1350	2	6	10	2	3	3	3	5
81	HG-4SAT-V150-C1350-13	<i>highgirth</i>	4	150	1350	2	7	8	1	1	3	3	4
82	HG-4SAT-V150-C1350-14	<i>highgirth</i>	4	150	1350	2	6	7	2	1	2	1	4
83	HG-4SAT-V150-C1350-15	<i>highgirth</i>	4	150	1350	2	5	6	2	1	1	2	5
84	HG-4SAT-V150-C1350-16	<i>highgirth</i>	4	150	1350	2	6	2	2	4	2	2	6
85	HG-4SAT-V150-C1350-17	<i>highgirth</i>	4	150	1350	2	7	4	2	3	2	3	6
86	HG-4SAT-V150-C1350-18	<i>highgirth</i>	4	150	1350	2	6	7	1	3	2	2	5
87	HG-4SAT-V150-C1350-19	<i>highgirth</i>	4	150	1350	2	9	4	1	2	1	2	4
88	HG-4SAT-V150-C1350-20	<i>highgirth</i>	4	150	1350	2	8	10	2	3	2	3	4
89	HG-4SAT-V150-C1350-21	<i>highgirth</i>	4	150	1350	2	6	5	2	3	2	3	8
90	HG-4SAT-V150-C1350-22	<i>highgirth</i>	4	150	1350	2	8	7	2	3	2	2	6
91	HG-4SAT-V150-C1350-23	<i>highgirth</i>	4	150	1350	2	6	7	2	3	3	3	6
92	HG-4SAT-V150-C1350-24	<i>highgirth</i>	4	150	1350	2	6	5	2	3	2	3	4
93	HG-4SAT-V150-C1350-100	<i>highgirth</i>	4	150	1350	2	5	8	2	3	2	3	5
94	scpcyc06_maxsat	<i>setcov</i> ₁	4	192	432	60	60	60	60	60	60	92	93
95	scpcyc07_maxsat	<i>setcov</i> ₁	4	448	1120	158	148	152	158	144	148	268	267

Continuation of Table 5.12 Best results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	Cplex	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
96	scpcyc08_maxsat	<i>setcov</i> ₁	4	1024	2816	392	361	370	390	359	352	718	720
97	scpcyc09_maxsat	<i>setcov</i> ₂	4	2304	6912	836	815	924	803	813	956	1868	1858
98	scpcyc10_maxsat	<i>setcov</i> ₂	4	5120	16640	1922	1917	16640	1923	1908	2244	4516	4553
99	scpcyc11_maxsat	<i>setcov</i> ₂	4	11264	39424	4339	4295	39424	10821	10944	5288	10867	10887
100	ram_k3_n9.ra0	<i>ramsey</i> ₁	6	36	210	1	1	1	1	1	1	1	1
101	ram_k3_n11.ra0	<i>ramsey</i> ₁	6	55	495	7	7	7	7	7	7	7	7
102	ram_k3_n12.ra0	<i>ramsey</i> ₁	6	66	715	10	10	11	10	10	10	10	10
103	ram_k3_n14.ra0	<i>ramsey</i> ₁	6	91	1365	21	21	21	21	21	21	22	22
104	ram_k3_n15.ra0	<i>ramsey</i> ₂	6	105	1820	30	30	31	30	30	30	30	30
105	ram_k3_n16.ra0	<i>ramsey</i> ₂	6	120	2380	39	39	42	39	39	39	40	40
106	ram_k3_n17.ra0	<i>ramsey</i> ₂	6	136	3060	50	50	54	50	50	50	50	50
107	ram_k3_n18.ra0	<i>ramsey</i> ₂	6	153	3876	60	60	70	60	60	62	64	63
108	ram_k4_n18.ra0	<i>ramsey</i> ₂	6	153	6120	9	9	14	9	9	10	12	16
109	ram_k3_n19.ra0	<i>ramsey</i> ₂	6	171	4845	75	75	87	75	75	76	87	76
110	ram_k4_n19.ra0	<i>ramsey</i> ₂	6	171	7752	15	15	29	15	15	15	30	30
111	scpcdr10_maxsat	<i>setcov</i> ₃	126	210	721	25	25	25	25	25	25	44	40
112	scpcdr12_maxsat	<i>setcov</i> ₃	330	495	2542	28	23	24	24	26	23	68	73
113	scpcdr13_maxsat	<i>setcov</i> ₃	495	715	4810	30	28	31	29	26	27	101	86

Table 5.13 Average results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	Cplex	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
1	san400_0.7_3.clq	<i>maxcut</i> ₁	2	40	1094	230.0	230.0	232.0	230.0	230.0	230.3	247.7	240.7
2	san200_0.9_1.clq	<i>maxcut</i> ₁	2	40	1392	313.0	313.0	317.0	313.0	313.0	315.6	328.5	321.3
3	p_hat500-1.clq	<i>maxcut</i> ₁	2	42	474	75.0	75.0	75.0	75.0	75.0	86.4	82.3	82.3
4	p_hat500-3.clq	<i>maxcut</i> ₁	2	42	1310	284.0	284.0	289.0	284.0	284.0	285.4	301.5	295.2
5	maxcut-140-630-0.8-3	<i>maxcut</i> ₂	2	140	1258	165.0	165.0	168.0	165.0	165.0	165.6	240.4	240.1

Continuation of Table 5.13 Average results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	Cplex	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
6	maxcut-140-630-0.8-4	<i>maxcut</i> ₂	2	140	1258	165.0	165.0	170.0	165.0	165.0	165.6	238.7	238.5
7	maxcut-140-630-0.8-20	<i>maxcut</i> ₂	2	140	1258	165.0	165.0	165.0	165.0	165.0	166.3	240.1	236.4
8	maxcut-140-630-0.8-44	<i>maxcut</i> ₂	2	140	1258	160.0	160.0	162.0	160.0	160.0	160.0	238.8	234.7
9	maxcut-140-630-0.7-3	<i>maxcut</i> ₂	2	140	1260	168.0	169.0	173.0	168.0	168.0	170.5	240.9	236.9
10	maxcut-140-630-0.7-33	<i>maxcut</i> ₂	2	140	1260	165.0	165.0	169.0	165.0	165.0	166.6	237.7	236.5
11	maxcut-140-630-0.7-49	<i>maxcut</i> ₂	2	140	1260	164.0	169.0	170.0	164.0	164.0	165.7	238.8	237.1
12	HG-3SAT-V250-C1000-1	<i>highgirth</i>	3	250	1000	5.0	13.0	12.0	5.0	8.0	6.7	7.9	16.1
13	HG-3SAT-V250-C1000-2	<i>highgirth</i>	3	250	1000	5.0	11.0	13.0	5.0	8.6	6.7	8.7	19.4
14	HG-3SAT-V250-C1000-3	<i>highgirth</i>	3	250	1000	5.0	12.0	13.0	5.0	8.6	7.8	10.0	18.9
15	HG-3SAT-V250-C1000-4	<i>highgirth</i>	3	250	1000	6.0	11.0	13.0	6.0	9.8	8.5	10.4	17.9
16	HG-3SAT-V250-C1000-5	<i>highgirth</i>	3	250	1000	6.0	13.0	13.0	6.0	9.4	7.7	9.0	20.1
17	HG-3SAT-V250-C1000-6	<i>highgirth</i>	3	250	1000	6.0	14.0	10.0	6.0	8.8	7.4	9.0	16.9
18	HG-3SAT-V250-C1000-7	<i>highgirth</i>	3	250	1000	6.0	8.0	11.0	6.0	9.3	7.3	8.9	18.1
19	HG-3SAT-V250-C1000-8	<i>highgirth</i>	3	250	1000	5.0	11.0	14.0	5.9	8.7	7.5	9.0	18.7
20	HG-3SAT-V250-C1000-9	<i>highgirth</i>	3	250	1000	6.0	13.0	14.0	6.0	8.9	8.0	9.5	16.7
21	HG-3SAT-V250-C1000-10	<i>highgirth</i>	3	250	1000	6.0	15.0	10.0	6.0	8.1	7.3	8.8	17.5
22	HG-3SAT-V250-C1000-11	<i>highgirth</i>	3	250	1000	6.0	9.0	13.0	6.0	9.4	7.7	9.1	18.6
23	HG-3SAT-V250-C1000-12	<i>highgirth</i>	3	250	1000	6.0	10.0	16.0	6.0	9.3	7.1	9.6	17.6
24	HG-3SAT-V250-C1000-13	<i>highgirth</i>	3	250	1000	5.0	9.0	12.0	6.0	9.4	6.5	8.9	18.4
25	HG-3SAT-V250-C1000-14	<i>highgirth</i>	3	250	1000	6.0	10.0	14.0	5.9	7.5	6.8	7.6	18.2
26	HG-3SAT-V250-C1000-15	<i>highgirth</i>	3	250	1000	5.0	10.0	12.0	5.0	7.9	6.6	8.7	19.7
27	HG-3SAT-V250-C1000-16	<i>highgirth</i>	3	250	1000	5.0	14.0	12.0	5.0	8.1	7.4	9.1	18.7
28	HG-3SAT-V250-C1000-17	<i>highgirth</i>	3	250	1000	6.0	13.0	14.0	6.9	9.7	7.9	9.7	18.8
29	HG-3SAT-V250-C1000-18	<i>highgirth</i>	3	250	1000	6.0	10.0	11.0	6.0	9.1	7.4	9.3	17.7
30	HG-3SAT-V250-C1000-19	<i>highgirth</i>	3	250	1000	5.0	10.0	10.0	6.9	9.7	8.0	8.7	21.5
31	HG-3SAT-V250-C1000-20	<i>highgirth</i>	3	250	1000	7.0	10.0	13.0	7.0	9.3	8.0	9.3	19.0
32	HG-3SAT-V250-C1000-21	<i>highgirth</i>	3	250	1000	4.0	10.0	10.0	5.0	8.6	7.8	8.7	19.0
33	HG-3SAT-V250-C1000-22	<i>highgirth</i>	3	250	1000	5.0	16.0	11.0	5.0	9.2	7.5	7.9	18.0
34	HG-3SAT-V250-C1000-23	<i>highgirth</i>	3	250	1000	6.0	13.0	12.0	6.0	9.7	7.3	9.7	18.3
35	HG-3SAT-V250-C1000-24	<i>highgirth</i>	3	250	1000	5.0	14.0	9.0	5.8	8.3	7.5	8.4	19.8

Continuation of Table 5.13 Average results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	CPLEX	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
36	HG-3SAT-V250-C1000-100	<i>highgirth</i>	3	250	1000	7.0	12.0	14.0	7.0	9.6	7.9	9.5	19.3
37	HG-3SAT-V300-C1200-1	<i>highgirth</i>	3	300	1200	7.0	16.0	17.0	7.0	11.0	9.2	11.5	26.2
38	HG-3SAT-V300-C1200-2	<i>highgirth</i>	3	300	1200	6.0	12.0	13.0	5.6	11.9	9.4	12.2	30.1
39	HG-3SAT-V300-C1200-3	<i>highgirth</i>	3	300	1200	6.0	14.0	9.0	8.0	11.4	9.5	12.7	30.4
40	HG-3SAT-V300-C1200-4	<i>highgirth</i>	3	300	1200	7.0	12.0	13.0	7.0	10.7	9.5	11.8	28.3
41	HG-3SAT-V300-C1200-5	<i>highgirth</i>	3	300	1200	6.0	13.0	16.0	6.8	11.9	9.6	12.3	28.1
42	HG-3SAT-V300-C1200-6	<i>highgirth</i>	3	300	1200	6.0	15.0	14.0	6.0	11.5	8.4	12.8	25.6
43	HG-3SAT-V300-C1200-7	<i>highgirth</i>	3	300	1200	5.0	16.0	15.0	5.0	10.7	8.0	11.0	24.3
44	HG-3SAT-V300-C1200-8	<i>highgirth</i>	3	300	1200	8.0	16.0	14.0	8.0	12.1	10.0	12.8	28.7
45	HG-3SAT-V300-C1200-9	<i>highgirth</i>	3	300	1200	7.0	13.0	19.0	8.0	11.5	9.1	14.5	28.8
46	HG-3SAT-V300-C1200-10	<i>highgirth</i>	3	300	1200	7.0	12.0	13.0	7.0	11.0	9.5	12.4	29.4
47	HG-3SAT-V300-C1200-11	<i>highgirth</i>	3	300	1200	6.0	17.0	14.0	6.0	10.6	9.0	13.0	29.1
48	HG-3SAT-V300-C1200-12	<i>highgirth</i>	3	300	1200	7.0	13.0	14.0	7.0	12.0	9.7	11.7	28.1
49	HG-3SAT-V300-C1200-13	<i>highgirth</i>	3	300	1200	7.0	10.0	15.0	7.0	11.0	9.7	11.2	26.8
50	HG-3SAT-V300-C1200-14	<i>highgirth</i>	3	300	1200	6.0	12.0	14.0	6.1	9.5	8.9	10.7	25.1
51	HG-3SAT-V300-C1200-15	<i>highgirth</i>	3	300	1200	7.0	16.0	12.0	7.4	11.3	10.0	11.9	32.2
52	HG-3SAT-V300-C1200-16	<i>highgirth</i>	3	300	1200	6.0	17.0	14.0	6.0	11.4	9.4	11.8	26.3
53	HG-3SAT-V300-C1200-17	<i>highgirth</i>	3	300	1200	7.0	15.0	17.0	7.1	11.5	9.4	14.1	29.2
54	HG-3SAT-V300-C1200-18	<i>highgirth</i>	3	300	1200	7.0	11.0	13.0	7.4	10.4	9.0	11.3	27.0
55	HG-3SAT-V300-C1200-19	<i>highgirth</i>	3	300	1200	6.0	16.0	15.0	7.0	12.5	9.5	14.0	30.6
56	HG-3SAT-V300-C1200-20	<i>highgirth</i>	3	300	1200	6.0	14.0	15.0	6.0	11.1	9.7	12.5	27.8
57	HG-3SAT-V300-C1200-21	<i>highgirth</i>	3	300	1200	6.0	9.0	16.0	5.0	9.7	9.3	11.2	27.3
58	HG-3SAT-V300-C1200-22	<i>highgirth</i>	3	300	1200	6.0	17.0	18.0	6.0	11.1	9.4	12.4	29.1
59	HG-3SAT-V300-C1200-23	<i>highgirth</i>	3	300	1200	7.0	12.0	15.0	7.0	10.7	8.7	11.1	27.0
60	HG-3SAT-V300-C1200-24	<i>highgirth</i>	3	300	1200	7.0	12.0	18.0	7.0	10.9	9.1	12.8	28.1
61	HG-3SAT-V300-C1200-100	<i>highgirth</i>	3	300	1200	7.0	14.0	14.0	7.0	11.3	9.1	11.4	29.8
62	HG-4SAT-V100-C900-2	<i>highgirth</i>	4	100	900	2.0	4.0	4.0	2.0	3.6	2.0	2.9	3.9
63	HG-4SAT-V100-C900-4	<i>highgirth</i>	4	100	900	2.0	4.0	3.0	2.0	3.2	2.7	3.6	5.1
64	HG-4SAT-V100-C900-7	<i>highgirth</i>	4	100	900	2.0	6.0	3.0	2.0	3.6	2.9	3.6	4.5
65	HG-4SAT-V100-C900-14	<i>highgirth</i>	4	100	900	2.0	6.0	4.0	2.0	3.0	2.3	3.9	4.3

Continuation of Table 5.13 Average results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	Cplex	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
66	HG-4SAT-V100-C900-19	<i>highgirth</i>	4	100	900	2.0	4.0	4.0	2.6	3.1	2.6	3.2	4.5
67	HG-4SAT-V100-C900-20	<i>highgirth</i>	4	100	900	2.0	5.0	2.0	2.0	2.7	2.3	2.8	4.2
68	HG-4SAT-V100-C900-23	<i>highgirth</i>	4	100	900	2.0	3.0	4.0	2.0	2.8	2.0	2.6	4.1
69	HG-4SAT-V150-C1350-1	<i>highgirth</i>	4	150	1350	1.0	5.0	8.0	1.0	4.7	2.9	6.1	8.1
70	HG-4SAT-V150-C1350-2	<i>highgirth</i>	4	150	1350	2.0	7.0	8.0	2.0	4.9	3.3	6.8	9.9
71	HG-4SAT-V150-C1350-3	<i>highgirth</i>	4	150	1350	2.0	6.0	6.0	1.9	2.8	1.3	1.8	6.3
72	HG-4SAT-V150-C1350-4	<i>highgirth</i>	4	150	1350	2.0	5.0	7.0	2.0	3.6	3.0	3.6	8.4
73	HG-4SAT-V150-C1350-5	<i>highgirth</i>	4	150	1350	2.0	7.0	7.0	2.0	4.2	3.2	4.3	9.0
74	HG-4SAT-V150-C1350-6	<i>highgirth</i>	4	150	1350	1.0	9.0	6.0	2.0	4.3	2.4	5.8	9.0
75	HG-4SAT-V150-C1350-7	<i>highgirth</i>	4	150	1350	2.0	5.0	7.0	2.0	4.2	3.7	3.9	8.5
76	HG-4SAT-V150-C1350-8	<i>highgirth</i>	4	150	1350	2.0	8.0	10.0	2.0	4.5	3.7	5.2	10.4
77	HG-4SAT-V150-C1350-9	<i>highgirth</i>	4	150	1350	2.0	7.0	8.0	2.0	4.6	3.3	4.7	10.2
78	HG-4SAT-V150-C1350-10	<i>highgirth</i>	4	150	1350	2.0	8.0	7.0	2.4	4.6	3.5	7.0	8.9
79	HG-4SAT-V150-C1350-11	<i>highgirth</i>	4	150	1350	2.0	4.0	5.0	2.7	4.1	3.1	4.1	8.2
80	HG-4SAT-V150-C1350-12	<i>highgirth</i>	4	150	1350	2.0	6.0	10.0	2.0	3.7	3.1	4.3	7.6
81	HG-4SAT-V150-C1350-13	<i>highgirth</i>	4	150	1350	2.0	7.0	8.0	2.4	4.6	4.0	6.5	8.1
82	HG-4SAT-V150-C1350-14	<i>highgirth</i>	4	150	1350	2.0	6.0	7.0	2.0	4.4	2.8	3.9	8.4
83	HG-4SAT-V150-C1350-15	<i>highgirth</i>	4	150	1350	2.0	5.0	6.0	2.0	3.7	2.6	4.5	9.4
84	HG-4SAT-V150-C1350-16	<i>highgirth</i>	4	150	1350	2.0	6.0	2.0	2.4	5.0	3.4	7.5	9.8
85	HG-4SAT-V150-C1350-17	<i>highgirth</i>	4	150	1350	2.0	7.0	4.0	2.0	4.5	3.6	4.2	9.1
86	HG-4SAT-V150-C1350-18	<i>highgirth</i>	4	150	1350	2.0	6.0	7.0	2.7	4.6	3.0	3.7	8.9
87	HG-4SAT-V150-C1350-19	<i>highgirth</i>	4	150	1350	2.0	9.0	4.0	1.9	2.6	2.4	2.8	9.2
88	HG-4SAT-V150-C1350-20	<i>highgirth</i>	4	150	1350	2.0	8.0	10.0	2.5	4.9	2.8	8.0	8.0
89	HG-4SAT-V150-C1350-21	<i>highgirth</i>	4	150	1350	2.0	6.0	5.0	2.0	4.4	3.4	5.8	9.4
90	HG-4SAT-V150-C1350-22	<i>highgirth</i>	4	150	1350	2.0	8.0	7.0	2.6	4.5	2.7	3.6	8.4
91	HG-4SAT-V150-C1350-23	<i>highgirth</i>	4	150	1350	2.0	6.0	7.0	2.0	5.0	4.1	8.6	9.6
92	HG-4SAT-V150-C1350-24	<i>highgirth</i>	4	150	1350	2.0	6.0	5.0	2.0	4.9	3.2	7.1	8.8
93	HG-4SAT-V150-C1350-100	<i>highgirth</i>	4	150	1350	2.0	5.0	8.0	2.0	5.0	3.6	9.0	8.5
94	scpcyc06_maxsat	<i>setcov₁</i>	4	192	432	60.0	60.0	60.0	60.0	60.3	60.0	95.6	94.5
95	scpcyc07_maxsat	<i>setcov₁</i>	4	448	1120	158.0	148.0	152.0	158.3	149.7	151.0	274.9	274.0

Continuation of Table 5.13 Average results of all algorithms tested on MSE 2020 and 2016 instances

Instance	Group	n_l	n_x	n_c	SATLike	SLSMcs	CPLEX	ACO-SAT _{neg} ⁺	ACO-SLS _{neg} ⁺	ACO _{neg} ⁺	ACO _{neg}	ACO	
96	scpcyc08_maxsat	<i>setcov</i> ₁	4	1024	2816	392.0	361.0	370.0	391.4	361.5	361.7	735.3	735.2
97	scpcyc09_maxsat	<i>setcov</i> ₂	4	2304	6912	836.0	815.0	924.0	824.5	830.7	963.9	1895.9	1872.6
98	scpcyc10_maxsat	<i>setcov</i> ₂	4	5120	16640	1922.0	1917.0	16640.0	1932.4	1916.3	2264.6	4630.2	4615.9
99	scpcyc11_maxsat	<i>setcov</i> ₂	4	11264	39424	4339.0	4295.0	39424.0	11042.9	11003.6	5317.4	11015.8	11040.9
100	ram_k3_n9.ra0	<i>ramsey</i> ₁	6	36	210	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
101	ram_k3_n11.ra0	<i>ramsey</i> ₁	6	55	495	7.0	7.0	7.0	7.0	7.0	7.0	7.0	7.0
102	ram_k3_n12.ra0	<i>ramsey</i> ₁	6	66	715	10.0	10.0	11.0	10.0	10.0	10.0	10.3	10.0
103	ram_k3_n14.ra0	<i>ramsey</i> ₁	6	91	1365	21.0	21.0	21.0	21.0	21.0	21.7	22.6	22.7
104	ram_k3_n15.ra0	<i>ramsey</i> ₂	6	105	1820	30.0	30.0	31.0	30.0	30.0	30.6	30.0	30.0
105	ram_k3_n16.ra0	<i>ramsey</i> ₂	6	120	2380	39.0	39.0	42.0	39.0	39.0	40.0	40.9	40.1
106	ram_k3_n17.ra0	<i>ramsey</i> ₂	6	136	3060	50.0	50.0	54.0	50.0	50.0	51.5	51.4	51.7
107	ram_k3_n18.ra0	<i>ramsey</i> ₂	6	153	3876	60.0	60.0	70.0	60.0	60.0	64.3	69.3	66.6
108	ram_k4_n18.ra0	<i>ramsey</i> ₂	6	153	6120	9.0	9.0	14.0	9.0	9.0	10.9	17.3	19.0
109	ram_k3_n19.ra0	<i>ramsey</i> ₂	6	171	4845	75.0	75.0	87.0	75.0	75.0	79.8	89.9	84.3
110	ram_k4_n19.ra0	<i>ramsey</i> ₂	6	171	7752	15.0	15.0	29.0	15.0	15.0	18.6	33.5	33.3
111	scpclr10_maxsat	<i>setcov</i> ₃	126	210	721	25.0	25.0	25.0	25.0	25.3	25.0	46.6	45.9
112	scpclr12_maxsat	<i>setcov</i> ₃	330	495	2542	28.0	23.0	24.0	27.0	26.7	23.6	80.6	80.9
113	scpclr13_maxsat	<i>setcov</i> ₃	495	715	4810	30.0	28.0	31.0	29.5	26.8	28.5	123.8	122.8

CHAPTER 6

APPLICATION TO THE MINIMUM CAPACITATED DOMINATING SET PROBLEM

6.1 INTRODUCTION

This Chapter provides a detailed description of our negative learning Aco approach for the *Capacitated Minimum Dominating Set* (CapMDS) problem. Some of the material provided in this chapter was presented in ANTS 2020 Twelfth International Conference on Swarm Intelligence October 26-28, 2020 in Barcelona (<https://iridia.ulb.ac.be/ants2020/>) and published in the ANTS 2020 proceedings, Lecture Notes in Computer Science, vol 12421 (https://doi.org/10.1007/978-3-030-60376-2_2). In fact, this was the first problem to which we applied our approach. The performance of the standard Aco algorithm is significantly improved by the negative learning mechanism for most of the considered problem instance types. Moreover, the current state-of-the-art algorithm [100] is improved in the context of 10 out of 36 cases.

6.2 THE CAPMDS PROBLEM

Before introducing the CapMDS problem and the developed algorithms, let us briefly recall some necessary definitions and notions from graph theory. Henceforth, $G = (V, E)$ denotes an undirected graph with a set $V = \{v_1, v_2, \dots, v_n\}$ of n vertices, and a set E of edges. We assume that the given graph neither contains self-loops nor multi-edges. Two vertices $u, v \in V$ are called neighbors—that is, they are adjacent—if and only if $(u, v) = (v, u) \in E$. Furthermore, $N(v) := \{u \in V \mid (v, u) \in E\}$ is called the *(open) neighborhood* of v and denotes the set of neighbors of $v \in V$. In contrast, the *closed neighborhood* $N[v]$ of a vertex $v \in V$ is $N[v] := N(v) \cup \{v\}$. The *degree* $\deg(v)$ of v is defined as the cardinality of the set of neighbors of v , that is, $\deg(v) = |N(v)|$. Any subset $S \subseteq V$ is called a *dominating set* of G if each vertex $v \in V \setminus S$ is adjacent to

at least one vertex from S . A vertex from S is called a *dominator*. As already seen in Chapter 4, given an undirected graph $G = (V, E)$, the classical minimum dominating set (MDS) problem asks to find a smallest-size dominating set $S \subseteq V$.

A problem instance of the CAPMDS problem [166] is given by a tuple (G, Cap) that consists of an undirected (simple) graph $G = (V, E)$ and a capacity function $Cap : V \rightarrow \mathbb{N}$. This capacity function assigns a positive integer $Cap(v) > 0$ to each vertex $v \in V$, indicating the maximum number of adjacent vertices this vertex is allowed to dominate in a valid solution.

A solution S to an instance (G, Cap) is a tuple $(D^S, \{C^S(v) \mid v \in D^S\})$, where $D^S \subseteq V$ is the set of selected dominators, and $\{C^S(v) \mid v \in D^S\}$ is a set that contains for each dominator $v \in D^S$ the (sub-)set $C^S(v) \subseteq N(v)$ of those of its neighbors that are (chosen to be) dominated by v . The following conditions have to be fulfilled in order for S to be a valid solution:

1. $D^S \cup (\bigcup_{v \in D^S} C^S(v)) = V$, that is, all vertices from V are either chosen to be a dominator, or are dominated by at least one dominator.
2. $|C^S(v)| \leq Cap(v)$ for all $v \in D^S$, that is, all chosen dominators dominate at most $Cap(v)$ of their neighbors.

Finally, the objective function value to be minimized is defined as $f(S) := |D^S|$.

6.2.1 ILP Model for the CAPMDS Problem

The following ILP is reproduced from [100]. The model is presented because sub-instances in our negative learning mechanism are again solved by CPLEX. The model works on the following sets of binary variables. First, a binary variable x_v is associated to each vertex $v \in V$ indicating whether or not v is selected as a dominator. Second, the model contains for each edge $(v, v') \in E$ two binary variables $y_{v,v'}$ and $y_{v',v}$. Variable $y_{v,v'}$ takes value one if vertex v is chosen to dominate vertex v' ; similarly for $y_{v',v}$. The CapMDS problem can then be stated as follows:

$$\text{minimize } \sum_{v \in V} x_v \quad (6.1)$$

subject to:

$$\sum_{v' \in N(v)} y_{v',v} \geq 1 - x_v \quad \forall v \in V \quad (6.2)$$

$$\sum_{v' \in N(v)} y_{v,v'} \leq Cap(v) \quad \forall v \in V \quad (6.3)$$

$$y_{v,v'} \leq x_v \quad \forall v \in V, v' \in N(v) \quad (6.4)$$

$$x_v, y_{v,v'} \in \{0, 1\} \quad (6.5)$$

Hereby, the constraint in Eqn. (6.2) ensures that all non-chosen vertices must be dominated by at least one dominator, whereas the constraint in Eqn. (6.3) limits the total number of vertices dominated by a particular vertex v to $Cap(v)$. Consequently, a dominator v can dominate at most $Cap(v)$ vertices from its (open) neighborhood.

6.3 *MMAS* IMPLEMENTATION TO THE CAPMDS

In general terms, the *MMAS* in the hypercube framework works as described in Section 2.2. This Section presents several specific configurations regarding its application to the CapMDS problem. In the context of *MMAS*, the construction of a solution S is done in a step-by-step manner. At each construction step, first, exactly one new dominator $v \in V \setminus D^S$ is chosen. In the second part of the construction step, it is decided which ones of the so-far non-dominated neighbors of v will be dominated by v . Therefore, the pheromone model \mathcal{T} used by our algorithm consists of the following values:

1. A value τ_v for each $v \in V$. These values are used to choose dominators.
2. Values $\tau_{v,v'}$ and $\tau_{v',v}$ for each edge $(v, v') \in E$. These values are used in the second part of each construction step for deciding which ones of its neighbors a newly chosen dominator will dominate.

The function `Construct_Solution(\mathcal{T})` in Algorithm 2.1 is implemented for the CapMDS problem with the following mechanism. It starts with an empty solution $S = (D^S = \emptyset, \emptyset)$. Moreover, the set of non-dominated neighbors of each vertex $v \in V$, denoted by ND_v , is initialized to $N(v)$. At each construction step, first, one vertex v^* is chosen from a set O (options) that includes all those vertices v that still have non-dominated neighbors and that do not already form part of D^S , as described in Eqn. (6.6). Note that the solution construction process stops once $O = \emptyset$.

$$O := \{v \in V \mid ND_v \neq \emptyset, v \notin D^S\} \quad (6.6)$$

The greedy function value $\eta(v)$ of a vertex $v \in O$ is defined as $\eta(v) := \min\{Cap(v), |ND_v|\} + 1$. Based on this greedy function, the probability for a vertex $v \in O$ to be selected is determined by using Eqn. (6.7).

$$\mathbf{p}^{\text{step1}}(v) := \frac{\eta(v) \cdot \tau_v}{\sum_{v' \in O} \eta(v') \cdot \tau_{v'}} \quad (6.7)$$

Given the probabilities from Eqn. (6.7), a vertex $v^* \in O$ is chosen with the same mechanism as described in Section 2.2.1. Note that after choosing v^* , the sets of non-dominated neighbors of the neighbors of v^* are updated by removing v^* .

In the second part of each construction step, a set of $\min\{Cap(v^*), |ND_{v^*}|\}$ non-dominated neighbors of v^* is chosen and placed into $C^S(v^*)$ as follows. In case $|ND_{v^*}| \leq Cap(v^*)$, we set $C^S(v^*) := ND_{v^*}$. Otherwise, vertices are sequentially selected from ND_{v^*} in the following way. First, the probability for each vertex $v \in ND_{v^*}$ to be selected is determined by using Eqn. (6.8)

$$p^{\text{step2}}(v) := \frac{(|ND_v| + 1) \cdot \tau_{v^*,v}}{\sum_{v' \in ND_{v^*}} (|ND_{v'}| + 1) \cdot \tau_{v^*,v'}} \quad (6.8)$$

Then, given the probabilities from Eqn. (6.8), a vertex $\hat{v} \in ND_{v^*}$ is chosen in the same way as outlined in the context of the first part of the construction step. Vertex \hat{v} is then added to an initially empty set $C^S(v^*)$, the respective ND-sets are updated, the probabilities from Eqn. (6.8) are recalculated, and the next vertex from ND_{v^*} is chosen. This process stops once $\min\{Cap(v^*), |ND_{v^*}|\}$ are selected. Finally, $C^S(v^*)$ is added to solution S , and the solution construction process proceeds with the next construction step.

Pheromone update in the context of the CapMDS application is implemented by function `ApplyPheromoneUpdate`(\mathcal{T} , cf , `bs_update`, S^{ib}, S^{rb}, S^{bsf}) in Algorithm 2.1 with the same mechanism as described in Section 2.2.2. For CapMDS however, the variables τ_i and $\xi(c_i)$ in Eqn. (2.2) are replaced with τ_v and ξ_v , respectively. Also, function $\Delta(S, c_i)$ for each of the three solutions S^{ib} , S^{rb} , and S^{bsf} in Eqn. (2.3) is replaced with $\Delta(S, v)$. Moreover, $\Delta(S, v)$ evaluates to 1 if and only if $v \in D^S$ (that is, v is chosen as a dominator). Otherwise, the function evaluates to 0. In the case of pheromone values $\tau_{v,v'}$, the pheromone update is the same, just that functions $\Delta(S, v)$ are replaced by functions $\Delta(S, v, v')$. Hereby, function $\Delta(S, v, v')$ evaluates to 1 if and only if $v \in D^S$ and $v' \in C^S(v)$ (that is, dominator v is chosen to dominate its neighbor v' in solution S). The value of the convergence factor cf is computed in a standard way on the basis of the pheromone values by function `ComputeConvergenceFactor`(\mathcal{T}) in Algorithm 2.1, as described in Sub-Section 2.2.3. Hereby, \mathcal{T} in Eqn. (2.4) stands for the set of all τ_v -values and all $\tau_{v,v'}$ -values.

6.4 ADDING NEGATIVE LEARNING TO \mathcal{MMAS}

In principle, the incorporation of negative learning to the baseline \mathcal{MMAS} is as described in Section 2.3. This Section describes the specific arrangements for its application to the CapMDS problem. For maintaining negative learning information, a negative pheromone value τ_v^{neg} is introduced for all pheromone values τ_v ($v \in V$). Moreover, the negative version $\tau_{v,v'}^{\text{neg}}$ is also introduced for all pheromone values $\tau_{v,v'}$. As described in Section 2.3, these negative pheromone values are initialized to τ_{\min} at the start of the algorithm, and whenever the algorithm is restarted (which still depends exclusively on the standard pheromone values).

In the context of the CapMDS problem we therefore propose the mechanism shown in Fig. 6.1 for identifying the negative learning information. At each iteration of our \mathcal{MMAS} algorithm, the set of solutions generated at the incumbent iteration (S^{iter}) is used for generating a subinstance of the tackled problem instance. Such a subinstance I^{sub} is a tuple $(D^{\text{sub}}, \{C^{\text{sub}}(v) \mid v \in D^{\text{sub}}\})$ where D^{sub} is defined by Eqn. (6.9) and C^{sub} is defined by Eqn. (6.10).

$$D^{\text{sub}} := \bigcup_{S \in S^{\text{iter}}} D^S \tag{6.9}$$

$$C^{\text{sub}}(v) := \bigcup_{\substack{S \in S^{\text{iter}} \\ \text{s.t. } v \in D^S}} C^S(v) \tag{6.10}$$

After generating the n_a solutions per iteration, the ILP solver CPLEX is used (with a time limit of t^{sub} CPU seconds) to solve the corresponding subinstance (if possible) to optimality. Otherwise, the best solution found within the allotted

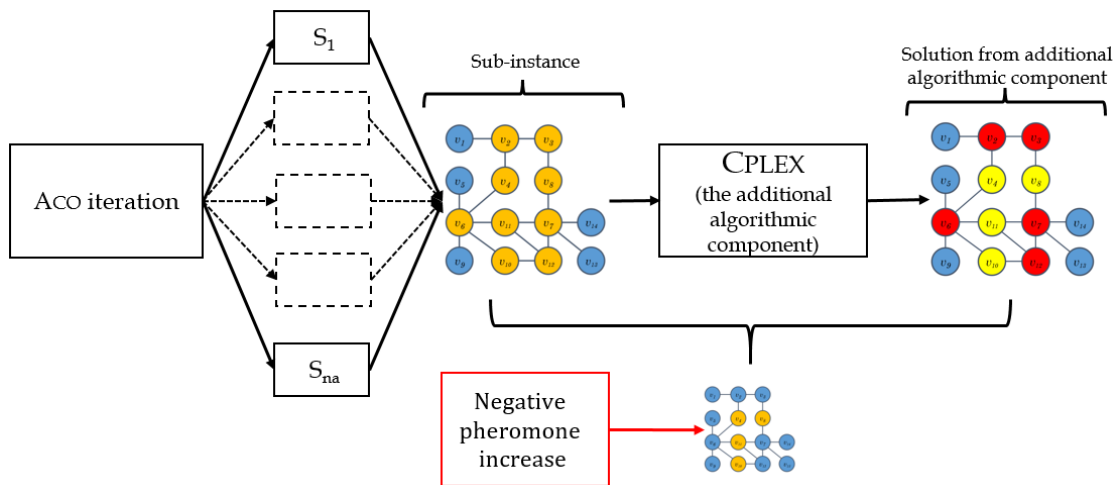


Fig. 6.1 Negative learning Aco for CapMDS

computation time is returned. In any case, the returned solution is denoted by S^{sub} . In order to solve the subinstance, the ILP model from Section 6.2.1 is used with the following additional restrictions. All variables x_v such that $v \notin D^{\text{sub}}$ are set to zero. Moreover, all variables $x_{v,v'}$ such that either $v \notin D$ or $v' \notin C^{\text{sub}}(v)$ are set to zero too.

After obtaining solution S^{sub} both the standard pheromone value update and the update of the negative pheromone values is performed. The update of the negative pheromone values is done with the same formula as in the case of the standard pheromone update (see the description of function `ApplyPheromoneUpdate(cf, bs_update, Sib, Srb, Sbsf)` in Section 6.3). Only the learning rate ρ is replaced by a *negative learning rate* ρ^{neg} , and the definition of the ξ_v (respectively $\xi_{v,v'}$) values changes. In particular, ξ_v is set to 1 for all $v \in D^{\text{sub}}$ with $v \notin D^{\text{SLLP}}$. In all other cases ξ_v is set to 0. Moreover, $\xi_{v,v'}$ is set to 1, for all $v' \in C^{\text{sub}}(v)$ with $v' \notin C^{\text{Ssub}}(v)$. In all other cases $\xi_{v,v'}$ is set to 0. In other words, those *solution components* that form part of the subinstance (and, therefore, form part of at least one of the solutions generated by *MMAS*) but that do not form part of the (possibly optimal) solution S^{sub} to the subinstance, are penalized.

Note that—in contrast to the standard *MMAS* algorithm, which is again denoted by *Aco* in the following section—the algorithm making use of negative learning is henceforth denoted by *Aco_{neg}*. Finally, not taking profit from solution S^{sub} in an additional, more direct, way may result in wasting valuable information. Therefore, we also test an extended version of *Aco_{neg}*, henceforth denoted by *Aco_{neg}⁺*, that updates solutions S^{rb} and S^{bsf} at each iteration with solution S^{sub} if appropriate.

The negative pheromone values are used in the following way to change the probabilities in both phases of each step for the construction of a solution S . Remember that the first phase concerns the choice of the next dominator v^* , and the second phase concerns the choice of a set $C^S(v^*)$ of so-far uncovered neighbors of v^* that v^* will dominate. The updated formula for calculating the probabilities in the first phase is as follows (compare to Eq. 6.7):

$$\mathbf{p}^{\text{step1}}(v) := \frac{\eta(v) \cdot \tau_v \cdot (1 - \tau_v^{\text{neg}})}{\sum_{v' \in O} \eta(v') \cdot \tau_{v'} \cdot (1 - \tau_{v'}^{\text{neg}})} \quad (6.11)$$

In the second phase of each construction step $C^S(v^*)$ is sequentially filled with vertices taken from ND_{v^*} (the set of currently uncovered neighbors of v^*) in the following way. First, the probability for each vertex $v \in \text{ND}_{v^*}$ to be selected is

determined as follows (compare to Eq. 6.8):

$$\mathbf{p}^{\text{step2}}(v) := \frac{(|\text{ND}_v| + 1) \cdot \tau_{v^*,v} \cdot (1 - \tau_{v^*,v}^{\text{neg}})}{\sum_{v' \in \text{ND}_{v^*}} (|\text{ND}_{v'}| + 1) \cdot \tau_{v^*,v'} \cdot (1 - \tau_{v^*,v'}^{\text{neg}})} \quad (6.12)$$

Another change in comparison to the standard way of generating solutions is that, during this second phase, only vertices whose probability $\mathbf{p}^{\text{step2}}(v)$ is greater or equal to 0.001 can be selected. This makes it possible to generate solutions in which a vertex selected as a dominator might not be chosen to dominate as many of its uncovered neighbors as possible in that moment.

6.5 EXPERIMENTAL EVALUATION

All experiments concerning Aco , Aco_{neg} and $\text{Aco}_{\text{neg}}^+$ were performed on a cluster of machines with Intel[®] Xeon[®] CPU 5670 CPUs with 12 cores of 2.933 GHz and a minimum of 32 GB RAM. Moreover, for solving the subinstances in Aco_{neg} and $\text{Aco}_{\text{neg}}^+$ we used CPLEX 12.8 in one-threaded mode.

The proposed algorithms were evaluated on the largest ones of the general graphs benchmark set for the CapMDS problem from [166]. These graphs are characterized by a number of vertices (n), a number of edges (m), a vertex capacity type (uniform vs. variable), and a capacity. In the case of uniform capacities, graphs with three different capacities (2, 5 and α) exist. Hereby, α refers to the average degree of the corresponding graph. In the case of variable capacities, the vertex capacities are—for each vertex—randomly chosen from the following three intervals: $(2, 5)$, $(\alpha/5, \alpha/2)$ and $[1, \alpha]$. For each combination of these graph characteristics, the benchmark set consists of 10 randomly generated graphs.

6.5.1 Algorithm Tuning

All three algorithm variants require parameter values to be set to well-working options. In particular, all three algorithm versions need parameter values for n_a (the number of solutions per iteration), d_{rate} (the determinism rate for solution construction), and ρ (the learning rate). Additionally, Aco_{neg} and $\text{Aco}_{\text{neg}}^+$ require values for parameters ρ^{neg} (the negative learning rate) and t^{sub} (the time limit, in CPU seconds, for CPLEX at each iteration). For the purpose of parameter tuning we made use of irace [123], as in all our experimental work in this thesis. This tool was used for generating one single parameter setting for each algorithm. As tuning instances we chose the first (out of 10) instances for each combination of the four input graph characteristics. Moreover, a budget of 2000 applications

was given to `irace`. The parameter value domains were fixed as follows: $n_a \in \{3, 5, 10, 20\}$, $d_{\text{rate}} \in \{0.1, 0.2, \dots, 0.8, 0.9\}$, $\rho, \rho^{\text{neg}} \in \{0.1, 0.2, 0.3\}$, and $t^{\text{sub}} \in \{2.0, 3.0, 5.0, 10.0\}$ (in seconds). The parameter value settings determined by `irace` are shown in Table 6.1.

Table 6.1 Parameter values obtained for solving CapMDS instances

	Algorithm	n_a	d_{rate}	ρ	ρ^{neg}	t^{sub}
1	Aco	5	0.9	0.1	n.a.	n.a.
2	Aco _{neg}	20	0.7	0.1	0.3	10.0
3	Aco _{neg} ⁺	20	0.6	0.1	0.2	5.0

6.5.2 Numerical Results

Each algorithm was applied exactly once (with a time limit of 1000 CPU seconds) to each problem instance. The results, averaged over 10 instances per table row, are shown in Table 6.2 (uniform capacity graphs) and in Table 6.3 (variable capacity graphs). While the two tables separate the instances with respect to the vertex capacity type (uniform vs. variable), the first three columns of each table provide information about the remaining three input graph characteristics (n , m , and vertex capacity). The fourth table column provides information about the best result known from the literature, while the fifth and sixth table columns present the results of CMSA, which is the current state-of-the-art algorithm from [100]. Both the results of CMSA and of the three Aco versions are shown by means of the average solution quality and the average computation time needed for producing these results.

In order to facilitate an interpretation of these results we provide the corresponding *critical difference* (CD) plots which were made by using R package `scmamp` [124]. As a reminder, this method works as follows. First, the Friedman test was used to compare the three approaches simultaneously. As a consequence of the rejection of the hypothesis that the techniques perform equally, the corresponding pairwise comparisons were performed using the Nemenyi post-hoc test [125]. The obtained results are graphically shown by means of the above-mentioned CD plots in Figure 6.2. In these plots, each considered algorithm variant is placed on the horizontal axis according to its average ranking for the considered subset of problem instances. The performances of those algorithm variants that are below the critical difference threshold (computed with a significance level of 0.05) are considered as statistically equivalent; see the horizontal bars joining the markers of the respective algorithm variants.

Table 6.2 Results for CapMDS graphs with uniform capacity.

n	m	Cap.	Best Known	CMSA		Aco		Aco _{neg}		Aco _{neg} ⁺	
				avg.	time	avg.	time	avg.	time	avg.	time
800	1000	2	267.0	267.0	3.6	285.3	136.2	267.0	8.4	267.0	2.5
800	2000	2	267.0	267.0	3.9	269.4	80.3	269.3	129.3	267.0	67.8
800	5000	2	267.0	267.0	3.2	267.0	59.0	271.1	192.1	267.0	119.4
1000	1000	2	334.0	334.0	7.9	364.0	157.1	334.0	7.2	334.0	0.6
1000	5000	2	334.0	334.0	6.5	334.2	88.0	384.6	50.3	334.0	126.9
1000	10000	2	334.0	334.0	5.8	334.0	32.4	379.5	176.7	337.2	136.7
800	1000	5	242.5	243.1	205.6	262.8	113.7	245.5	89.2	244.4	76.1
800	2000	5	162.8	162.8	574.7	177.0	116.1	163.2	61.0	161.9*	79.1
800	5000	5	134.0	134.0	4.7	135.3	72.4	158.7	6.3	134.0	160.2
1000	1000	5	333.7	333.7	10.5	362.8	141.2	333.7	8.8	333.7	0.6
1000	5000	5	167.0	167.0	40.8	172.2	101.1	206.3	61.4	167.0	173.6
1000	10000	5	167.0	167.0	3.7	167.8	67.3	188.4	8.6	167.0	102.7
800	1000	α	267.0	267.0	4.6	284.0	153.8	267.0	10.1	267.0	2.8
800	2000	α	162.8	162.8	537.3	178.8	93.0	163.4	73.8	162.0*	69.7
800	5000	α	91.1	93.0	717.9	92.9	62.8	90.9	74.0	89.2*	104.3
1000	1000	α	334.0	334.0	13.7	365.1	175.2	334.0	6.9	334.0	0.6
1000	5000	α	132.5	135.0	782.9	137.3	82.0	131.6	65.4	127.3*	116.3
1000	10000	α	81.3	86.8	518.7	82.6	67.9	87.9	98.4	80.7*	133.1

Table 6.3 Results for CapMDS graphs with variable capacity.

n	m	Cap.	Best Known	CMSA		Aco		Aco _{neg}		Aco _{neg} ⁺	
				avg.	time	avg.	time	avg.	time	avg.	time
800	1000	(2, 5)	248.1	248.2	79.2	269.2	131.7	251.8	47.4	249.9	68.6
800	2000	(2, 5)	181.2	181.5	341.7	195.0	98.7	180.8	73.1	179.8*	79.7
800	5000	(2, 5)	134.1	134.1	28.1	139.1	99.6	138.4	127.3	134.1	94.3
1000	1000	(2, 5)	333.8	333.8	3.9	365.6	146.9	333.8	8.8	333.8	1.9
1000	5000	(2, 5)	169.0	169.0	85.1	182.8	86.3	171.2	109.7	169.6	105.0
1000	10000	(2, 5)	167.0	167.0	27.5	168.4	92.3	198.3	7.7	167.0	170.1
800	1000	($\alpha/5, \alpha/2$)	400.0	400.0	2.6	409.3	112.5	400.2	66.7	400.0	0.8
800	2000	($\alpha/5, \alpha/2$)	273.4	273.4	6.5	283.2	87.5	274.6	101.6	273.4	7.6
800	5000	($\alpha/5, \alpha/2$)	115.0	115.1	178.6	123.0	83.7	116.5	77.1	115.0	85.7
1000	1000	($\alpha/5, \alpha/2$)	500.0	500.0	8.6	517.7	122.2	500.0	11.2	500.0	1.0
1000	5000	($\alpha/5, \alpha/2$)	168.1	168.1	77.4	181.3	128.7	170.9	105.4	168.8	92.2
1000	10000	($\alpha/5, \alpha/2$)	104.7	107.1	247.9	104.8	97.1	108.6	131.0	95.6*	121.3
800	1000	[1, α]	300.2	300.2	4.0	316.0	144.9	300.2	6.8	300.2	0.5
800	2000	[1, α]	186.2	186.2	442.6	204.9	105.3	187.3	61.5	185.8*	63.9
800	5000	[1, α]	98.1	98.1	683.7	101.7	63.3	96.8	84.4	95.6*	80.2
1000	1000	[1, α]	400.8	400.8	6.4	409.6	141.5	400.8	7.3	400.8	0.5
1000	5000	[1, α]	143.8	143.8	866.9	151.9	95.4	141.4	101.1	140.6*	98.9
1000	10000	[1, α]	90.1	90.1	541.8	88.2	66.8	87.8	132.1	85.6*	108.0

The graphic in Figure 6.2a shows the CD plot for the uniform capacity instances, and the one in Figure 6.2 for the variable capacity instances. In both graphics it can be seen that both algorithm variants with negative learning (Aco_{neg} and Aco_{neg}⁺) significantly improve over the standard ACO approach. Moreover, Aco_{neg}⁺ improves over Aco_{neg} with statistical significance. This is also the general

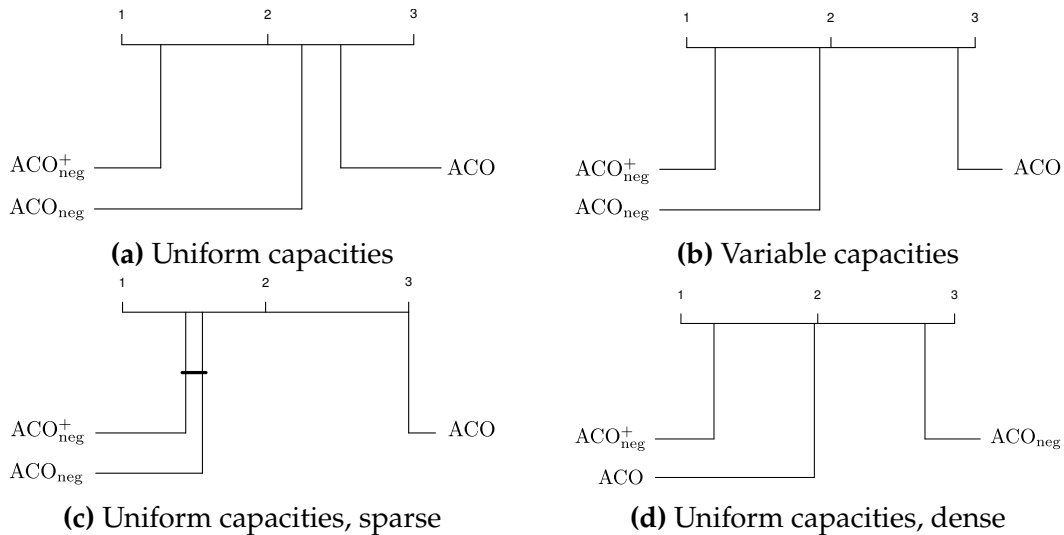


Fig. 6.2 Critical difference plots

picture given by the numerical results in Tables 6.2 and 6.3.

Interestingly, when separating the instances according to different graph densities, it can be noticed that negative learning is especially useful in the context of sparse graphs. In contrast, when moving towards dense graphs the efficacy of negative learning is reduced. In the context of graphs with uniform capacities, it is even the case that standard ACO outperforms ACO_{neg} for dense graphs. This is shown in the context of uniform capacity graphs in Figures 6.2c and 6.2d.

6.6 CONCLUSIONS

In this work we introduced a new approach for making use of negative learning in ant colony optimization. This approach builds, at each iteration, a subinstance of the original problem instance by merging the solution components found in the solutions generated by the ant colony optimization algorithm in that iteration. Then it uses a different optimization technique— $CPLEX$ was used here—for finding the best solution in this subinstance. The solution components from the subinstance that do not form part of this solution are penalized by means of increasing their negative pheromone values. The proposed approach is shown to be very beneficial for the capacitated minimum dominating set problem.

CHAPTER 7

APPLICATION TO THE MINIMUM POSITIVE INFLUENCE DOMINATING SET PROBLEM

7.1 INTRODUCTION

This chapter describes the application of our negative learning Aco for the minimum positive influence dominating set (MPIDS) problem. Some parts of this chapter were also presented in our paper [110] that was published in the Proceedings of the Genetic and Evolutionary Computation Conference Companion, July 2021, Pages 1974–1977, (<https://doi.org/10.1145/3449726.3463130>).

The MPIDS problem [167, 168] is an NP-hard combinatorial optimization problem with applications in social networks. Hereby, each vertex represents an individual and edges indicate relationships, respectively interactions, between those individuals. The problems' background is that ideas and information propagated in social networks can have a significant impact. Social norms theory has shown that the behavior of individuals can be affected by perceptions of others' thoughts and behaviors [169]. Thus, exploiting the relationships among people in social networks can provide great benefits to both economy and society. The aim of the MPIDS problem is to identify a small subset of key influential individuals to speed up the spread of positive influence [170, 171]. Other applications of the MPIDS problem can be found in e-learning software [172], online business [173], drinking, smoking, and drug related problems [167].

In this work we apply our negative learning Aco to the MPIDS problem. In Chapter 3 and 4, this negative learning ACO approach was shown to outperform earlier negative learning approaches such as, for example, [84]. Our results show that our negative learning variant of Aco outperforms both the standard Aco variant and the metaheuristic approaches from the literature for the MPIDS problem.

The rest of this chapter is organized as follows. In Section 7.2 we provide the standard integer linear programming (ILP) model for the MPIDS problem

and highlight several existing approaches to this CO problem. In Section 7.3, we outline our algorithmic proposal. Finally, in Section 7.4 we present and discuss the experimental results.

7.2 THE MINIMUM POSITIVE INFLUENCE DOMINATING SET PROBLEM

In technical terms, the MPIDS problem can be described as follows. Given a simple, connected undirected graph $G = (V, E)$, the problem requires to find a dominating set of minimum cardinality such that at least half of the neighbors of each vertex form part of the dominating set.

7.2.1 ILP Model for the MPIDS

The MPIDS problem can easily be stated in terms of an ILP as follows. The model is based on a binary variable x_i associated to each vertex $v_i \in V$.

$$\text{minimize } \sum_{i=1}^n x_i \quad (7.1)$$

subject to:

$$\sum_{v_j \in N(v_i)} x_j \geq \left\lceil \frac{\text{deg}(v_i)}{2} \right\rceil \quad \forall v_i \in V \quad (7.2)$$

$$x_i \in \{0, 1\} \quad (7.3)$$

Hereby, $N(v_i)$ is the neighborhood of v_i in the input graph G , and $\text{deg}(v_i)$ is the degree of vertex v_i , that is, the number of its neighbors. Equation (7.2) ensures that a feasible solution contains at least half of the neighbors of each vertex $v_i \in V$.

Most of the recent research efforts concerning the MPIDS problem were focused on greedy heuristics [168, 174–177] and on two evolutionary approaches [111, 112].

7.3 NEGATIVE LEARNING ACO FOR MPIDS PROBLEM

As in the previous works in Chapters 3 to 6, we used *MMAS* in the hypercube framework [14] as the baseline Aco algorithm for our negative learning application to the MPIDS Problem. This baseline algorithm works as described in Chapter 2. The standard pheromone model \mathcal{T} for this CO problem contains a value τ_i for each vertex $v_i \in V$. Similarly, the model of the negative pheromone

values (\mathcal{T}^{neg}) contains a value τ_i^{neg} for each vertex $v_i \in V$. The initialization of pheromone values and other parameters is exactly as described in Section 2.2, except for S^{bsf} and S^{rb} which are initialized to V , the worst solution possible. Overall, the algorithm works as outlined in Algorithm 2.1 with the detailed description of the pheromone update function and convergence factor calculation as provided in Sections 2.2.2 and 2.2.3, respectively.

Function $\text{ConstructSolution}(\mathcal{T})$ of Algorithm 2.1 is applied to this CO problem by adopting the solution construction mechanism from the newest available greedy algorithm for the MPIDS problem [177]. First, a pre-processing procedure adds those vertices to a set S_{init} that must form part of an optimal solution. Note that this pre-processing procedure is, of course, only executed once by the algorithm. Each solution construction process starts by initializing the partial solution under construction to S_{init} , that is, $S := S_{\text{init}}$. Given any partial solution $S \subset V$, the number of neighbors of v that must be added to S in order to cover v is computed as $h_S(v) := \lceil \frac{\text{deg}(v)}{2} \rceil - |N_S(v)|$, where $N_S(v) := N(v) \cap S$ denotes the set of neighbors of $v \in V$ belonging to S . We say that v is covered if $h_S(v) \leq 0$, and not covered otherwise. Moreover, let $C_S \subset V$ denote the set of uncovered vertices with respect to S . At each step of the solution construction procedure, first a vertex $v_i \in C_S$ is chosen such that $\text{deg}(v_i) \leq \text{deg}(v_j)$ for all $v_j \in C_S$. Then, $h_S(v_i)$ vertices from $N(v_i) \setminus N_S(v_i)$ are chosen, based on greedy information and on pheromone information. The probability to choose a vertex $v_k \in N(v_i) \setminus N_S(v_i)$ is defined as follows:

$$\mathbf{p}(v_k | S) := \frac{\eta_k \cdot \tau_k \cdot (1 - \tau_k^{\text{neg}})}{\sum_{v_l \in N(v_i) \setminus N_S(v_i)} \eta_l \cdot \tau_l \cdot (1 - \tau_l^{\text{neg}})} \quad (7.4)$$

where $\eta_k := |\{v \in N(v_k) : h_S(v) > 0\}| + 1$. Based on these probabilities, the selection of v^* —that is, the vertex to be added to S —is done as follows. First, a random number $r \in [0, 1]$ is chosen uniformly at random. In case $r \leq d_{\text{rate}}$, $v^* := \text{argmax}\{\mathbf{p}(v | S) \mid v \in N(v_i) \setminus N_S(v_i)\}$. Otherwise, v^* is chosen by *roulette wheel selection*. Remember that $d_{\text{rate}} \in [0, 1]$ —the so-called determinism rate—is an important parameter of the algorithm.

The generation of the information for negative learning is done by adding two new instructions—Eqn. (2.5) and Eqn. (2.6)—which are introduced between lines 9 and 10 of Algorithm 2.1, as outlined in Section 2.3. The modification of the negative pheromone values from \mathcal{T}^{neg} in this CO problem is applied by function $\text{SolveSubinstance}(\mathcal{S}^{\text{iter}}, cf)$ (Eqn. (2.5)). In particular, this update only concerns the negative pheromone values of those vertices that form part of at least one solution of $\mathcal{S}^{\text{iter}}$. The update formula is as follows: $\tau_i^{\text{neg}} := \tau_i^{\text{neg}} + \rho^{\text{neg}} \cdot (\xi_i^{\text{neg}} - \tau_i^{\text{neg}})$,

where ρ^{neg} is the negative learning rate and $\xi_i^{\text{neg}} = 1$ if $c_i \notin S^{\text{sub}}$, resp. $\xi_i^{\text{neg}} = 0$ otherwise. In other words, the negative pheromone values of those components that do not form part of S^{sub} are increased.

7.4 EXPERIMENTAL EVALUATION

Three versions of the proposed algorithm are evaluated. In addition to the full version ($\text{Aco}_{\text{neg}}^+$), we also evaluated the following versions: (1) the standard MMAS version (henceforth simply called Aco). This version is obtained by not executing the update of the negative pheromone values and by setting $S^{\text{sub}} = \emptyset$ at each iteration (that is, Eqn.(2.5) in Section 2.3.2 is not executed); (2) Aco_{neg} , which is obtained by not using S^{sub} for the standard pheromone update, only for the update of the negative pheromone values. All experiments concerning the three algorithm versions were performed on a cluster of machines with Intel[®] Xeon[®] CPU 5670 CPUs with 12 cores of 2.933 GHz and a minimum of 32 GB RAM. Moreover, for solving the sub-instances in Aco_{neg} and $\text{Aco}_{\text{neg}}^+$ we used CPLEX 12.10 in one-threaded mode.

7.4.1 Problem instances

The three algorithms were evaluated on 17 social networks that are usually used in the literature on the MPIDS problem. These networks are of small and medium size. In addition, we evaluated the algorithms on 10 larger social networks from the SNAP library (<https://snap.stanford.edu/data/>).

7.4.2 Algorithm tuning and test settings

Our algorithms require well-working values for n_a (number of solution constructions per iteration), d_{rate} (determinism rate), and ρ (learning rate). Aco_{neg} and $\text{Aco}_{\text{neg}}^+$ require additionally a value for ρ^{neg} (negative learning rate) and a value for t^{sub} (time limit for CPLEX per iteration). We made use of the scientific tuning software irace [123] for the purpose of parameter tuning. This tool was used for generating two parameter settings for each variant: one of the 17 small/medium sized instances, and another one for the 10 large networks. The obtained parameter value settings are shown in Tables 7.1 and 7.2.

Networks CA-AstroPh, Email-Enron amd socfb-Brandeis99 were used for the tuning regarding the small/medium sized instances, and networks Amazon0312 and com-youtube were used for the large ones. Finally, for each

Table 7.1 Tuning results obtained by *irace* for small/medium size networks

Algorithm	n_a	d_{rate}	ρ	ρ^{neg}	t^{sub}
Aco	10	0.4	0.1	n.a.	n.a.
Aco _{neg}	20	0.1	0.1	0.5	30
Aco _{neg} ⁺	20	0.0	0.1	0.2	17

tuning experiment the budget was fixed to 2000 runs, each one with a time limit of 600 CPU seconds. The considered parameter value domains were as follows: $n_a \in \{2, \dots, 20\}$, $d_{\text{rate}} \in \{0.0, 0.1, 0.2, \dots, 0.8, 0.9\}$, $\rho, \rho^{\text{neg}} \in \{0.1, \dots, 0.5\}$, and $t^{\text{sub}} \in \{1, \dots, 30\}$ (in seconds).

Table 7.2 Tuning results obtained by *irace* for large size networks

Algorithm	n_a	d_{rate}	ρ	ρ^{neg}	t^{sub}
Aco	2	0.6	0.1	n.a.	n.a.
Aco _{neg}	10	0.7	0.3	0.5	20
Aco _{neg} ⁺	14	0.6	0.2	0.5	13

The tuning results are used in the execution of the three Aco versions. Furthermore, these Aco variants were applied 10 times, with a CPU time limit of 600 CPU seconds, to each of the 27 problem instances.

7.4.3 Results

The results, in comparison to those of HSIA [112], ILPMA [111], and CPLEX (with a time limit of 2 hours per instance) are shown in Table 7.3. The first 17 of these instances are generally used in the related literature and are of small, resp. medium, size. CPLEX was able to solve 11 of these instances to optimality, as indicated by a value of 0.00 in the column labeled 'Gap (%)'. Note that both HSIA and ILPMA were only applied to 9, resp. 12, of these first 17 problem instances. The remaining 10 instances are larger and were taken from the SNAP database. The results of our three ACO versions are marked by a light gray column background. They are separated into the best result obtained in 10 runs, the average results over 10 runs, and the average computation time.

7.5 DISCUSSION AND CONCLUSIONS

The following observations can be made. First, the standard Aco version obtains results similar to those of HSIA and ILPMA. The two Aco versions with negative learning (Aco_{neg} and Aco_{neg}⁺) clearly outperform the other three competitors.

Concerning the comparison between Aco_{neg} and Aco_{neg}^+ it can be stated that, in the context of the 17 small/medium size instances, Aco_{neg}^+ is only slightly better than Aco_{neg} . However, this difference in quality grows significantly in the context of the 10 larger problem instances. Moreover, CPLEX clearly fails to provide solutions of reasonable quality in the case of five large problem instances, that is, the *deezer_HR* instance, and the four *Amazon* instances. All three Aco versions are clearly superior to CPLEX in these cases.

Summarizing we can state that Aco_{neg}^+ is the new state-of-the-art metaheuristic for solving the MPIDS problem. Moreover, this shows again that making use of negative learning in addition to positive learning can be very beneficial.

Table 7.3 Numerical results of all algorithm tested on MPIDS instances

Network	CPLEX		best					average					average time			
	Result	Gap (%)	HSIA	ILPMA	Aco	Aco _{neg}	Aco _{neg} ⁺	HSIA	ILPMA	Aco	Aco _{neg}	Aco _{neg} ⁺	ILPMA	Aco	Aco _{neg}	Aco _{neg} ⁺
Karate	15	0.00	n.a.	15	15	15	15	n.a.	15.0	15.0	15.0	15.0	0.03	0.00	0.002	0.002
Dolphins	30	0.00	n.a.	30	30	30	30	n.a.	30.0	30.0	30.0	30.0	0.13	0.004	0.008	0.004
Football	63	0.00	n.a.	65	64	63	63	n.a.	65.65	64.6	63.0	63.0	0.54	74.08	40.71	19.73
Jazz	79	0.00	n.a.	n.a.	79	79	79	n.a.	n.a.	79.9	79.0	79.0	n.a.	5.92	0.39	0.12
CA-AstroPh	6740	0.30	6905	6857	6886	6742	6742	6906.6	6865.45	6897.1	6744.1	6743.8	300.41	502.58	282.73	438.62
CA-GrQc	2587	0.00	2597	2594	2588	2587	2587	2598.4	2596.05	2589.1	2587.0	2587.0	45.07	144.90	2.96	0.65
CA-HepPh	4718	0.01	4791	4770	4769	4720	4720	4792.4	4773.85	4772.9	4721.4	4720.3	157.43	526.81	341.16	347.01
CA-HepTh	4471	0.00	4515	4502	4494	4471	4471	4516.2	4506.25	4496.7	4471.0	4471.0	107.93	468.96	13.44	14.62
CA-CondMat	9584	0.06	9729	9683	9692	9587	9588	9734.0	9689.6	9696.3	9588.8	9588.4	506.37	432.98	394.02	471.59
Email-Enron	11682	0.00	11865	11814	11826	11685	11684	11873.4	11818.95	11832.9	11685.2	11684.1	760.08	440.34	221.48	164.53
ncstrlwg2	2994	0.00	3004	3001	2998	2994	2994	3005.4	3002.85	2998.9	2994.9	2994.1	65.69	327.34	18.70	259.49
actors-data	3092	0.24	3143	3130	3145	3093	3093	3144.5	3134.5	3149.0	3093.7	3093.7	137.74	419.42	273.89	260.08
ego-facebook	1973	0.00	1726 ^a	1737 ^a	1974	1973	1973	1726.6 ^a	1741.55 ^a	1974.9	1973.6	1973.1	56.91	16.44	33.27	65.28
socfb-Brandeis- -99	1400	1.41	n.a.	n.a.	1456	1398	1397	n.a.	n.a.	1462.7	1399.0	1397.7	n.a.	398.99	347.46	480.68
socfb-nips-ego	1398	0.00	n.a.	n.a.	1398	1398	1398	n.a.	n.a.	1398.0	1398.0	1398.0	n.a.	2.86	2.71	1.28
socfb-Mich67	1329	1.56	n.a.	n.a.	1384	1329	1327	n.a.	n.a.	1387.9	1329.9	1328.5	n.a.	420.84	335.62	366.89
soc-gplus	8244	0.00	n.a.	n.a.	8294	8244	8244	n.a.	n.a.	8298.3	8244.1	8244.0	n.a.	446.28	169.60	21.20
musae_git	9752	0.00	n.a.	n.a.	10383	10006	9872	n.a.	n.a.	10409.3	10031.3	9828.8	n.a.	589.06	414.45	357.81
loc-gowalla- -edges	67617	0.07	n.a.	n.a.	68815	67946	67943	n.a.	n.a.	68836.9	67972.0	67964.0	n.a.	547.53	550.78	503.48
gemsec_face- -book_artist	15194	1.20	n.a.	n.a.	16010	15537	15480	n.a.	n.a.	16029.0	15593.9	15505.2	n.a.	554.54	488.09	511.42
deezer_HR	54573	95.68	n.a.	n.a.	23413	22906	22840	n.a.	n.a.	23434.7	23152.1	22904.9	n.a.	518.96	426.34	426.67
com-youtube	351281	0.00	n.a.	n.a.	353715	352110	351556	n.a.	n.a.	353975.2	352243.7	351567.1	n.a.	601.46	592.07	599.96
com-dblp	120492	0.08	n.a.	n.a.	121854	120998	120853	n.a.	n.a.	121874.6	121056.7	120932.0	n.a.	465.34	564.34	513.59
Amazon0302	262111	97.50	n.a.	n.a.	134146	132797	131836	n.a.	n.a.	134241.0	132832.6	131901.3	n.a.	370.61	395.71	576.98
Amazon0312	400727	95.41	n.a.	n.a.	180443	180613	180049	n.a.	n.a.	180546.3	180690.8	180284.1	n.a.	597.82	597.19	611.33
Amazon0505	410236	95.19	n.a.	n.a.	182851	182839	182152	n.a.	n.a.	182955.3	182928.9	182464.5	n.a.	596.00	601.65	617.66
Amazon0601	403394	96.94	n.a.	n.a.	179768	179726	179112	n.a.	n.a.	179847.5	179799.0	179662.2	n.a.	598.81	598.46	612.19
average					49351.48	4906.88	48966.59			49381.25	49137.72	49017.73	n.a.	372.92	285.45	305.29

CHAPTER 8

ADDITIONAL WORK: *MMAS* APPLICATION TO THE MULTI-HEAD WEIGHER MACHINES PROBLEM

8.1 INTRODUCTION

This chapter describes the application of *MMAS* to the Multi-head Weigher Machines (MWMs) problem. Some parts of this chapter were also presented in the International Symposium on Advances and Innovation in Mechanical Engineering: *Sustainable Innovation in Disruptive Era*, October 12-14, 2021 in Yogyakarta Indonesia (<https://bkstm.umsida.ac.id/isaime/>). The paper is currently processed by the Conference Committee to be published in the AIP Conference Proceedings. We chose this CO problem as the starting point for the application of hybrid metaheuristics, particularly the negative learning Aco, to actual problems in mechanical engineering, which is the scientific background of the author of this thesis.

MWMs [178] are automatic packing machines used by industries for weighing and packing their products accurately and rapidly. The first of these machines was introduced in 1972 by Ishida [179] and currently more than 31,000 MWM units are installed worldwide [180]. These machines typically consist of several important parts, as shown in Fig. 8.1, and they work as follows. The products are supplied on top of a dispersion feeder and then distributed by a radial feeder to a layer of pool hoppers. A layer of weighing hoppers, located underneath the layer of pool hoppers, will receive these products and will then measure their weights. The measurement data is sent to a computer that will select a number of k weighing hoppers among n available ones in a way such that the total weight of the chosen hopper's content will exceed the target weight of the products package. Finally, the chosen weighing hoppers will then release their content through a discharge chute after which they will be packed. In order to minimize the loss during the production process, however, the total weight must surely be

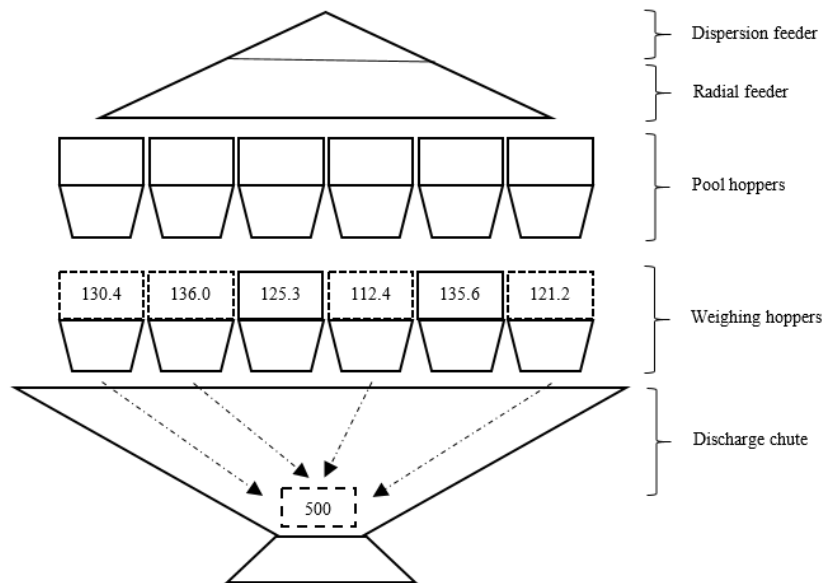


Fig. 8.1 A schema of a single layer MWM

as close as possible to the target weight.

Current trends show that MWM hardware designs as well as packing scenarios are evolving to be more complex and demanding. Today, a multihead weigher machine may employ double layers of weighing hoppers [181] instead of only one layer as in the original design. Handling perishable products that cannot withstand long resident time during the packaging process could also add an extra difficulty in MWM design and operation [182]. Moreover, it is also possible that a packaging scenario may require the MWM to pack multiple types of products instead of just one [183, 184]. Consequently, a well-designed optimization algorithm is needed for anticipating all of these developments.

With the exception of the works of Karuno and Nakahama [185], most of the works in MWM optimizations dealt with only one type of product and used a complete enumeration strategy. Current trends in the MWM packaging scenario, however, may require them to pack additional types of products or to include additional optimization objectives and/or constraints. In any of these cases, the complete enumeration strategy may not always be able to provide the optimum result in a reasonable time frame. Naturally, ant colony optimization is a potential candidate for this purpose.

In this work, we developed and tested an Aco algorithm for MWM optimization problems concerning packing scenarios with one, two, and three types of products. Results from our numerical experiments provide a comprehensive picture on the characteristics of MWM optimization problems as well as the performance of our proposed optimization algorithm in comparison

with the one delivered by CPLEX which served as benchmark. This work is an initial study from which a more sophisticated Aco algorithm such as the one presented in [104] could be designed for more sophisticated MWM packaging scenarios.

8.2 THE MULTI-HEAD WEIGHER MACHINE PROBLEM

The MWM problem can be defined as follows. Given is (1) n_t , the number of product types to be packed, (2) n_i , the number of weighing hoppers allocated for each product type, (3) w_{ij} represents the weight of each product type i inside the j -th weighing hopper allocated for product type i , (4) P represents the target weight of the package, and (5) p_i represents the target weight of each type of product. A valid solution to this optimization problem is a set of hoppers such that the sum of their weights (w_{ij} values) is not less than the target weight P of the package. Moreover, in this solution, the sum of the w_{ij} -values for each product type i must also not be less than the corresponding target weight p_i .

8.2.1 ILP Model for the MWM

The MWM problem can in the following way be expressed by means of an ILP model [184] presented in equations (8.1) to (8.4).

$$\text{minimize } \sum_{i=1}^{n_t} \sum_{j=1}^{n_i} w_{ij} \cdot x_{ij} \quad (8.1)$$

subject to:

$$\sum_{i=1}^{n_t} \sum_{j=1}^{n_i} w_{ij} \cdot x_{ij} \geq P \quad (8.2)$$

$$\sum_{j=1}^{n_i} w_{ij} \cdot x_{ij} \geq p_i, \quad i = 1, 2, \dots, n_t \quad (8.3)$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, 2, \dots, n_t, \quad j = 1, 2, \dots, n_i \quad (8.4)$$

Equation (8.1) of the ILP model represents the MWM optimization objective while inequalities (8.2) and (8.3) each represent MWM optimization constraints. This ILP model is based on binary variables x_{ij} for representing the selection state of the weighing hoppers, that is, the value of x_{ij} is one if the corresponding hopper is selected, and zero otherwise. Note that this model was solved in standalone mode by CPLEX for comparison purposes.

8.2.2 Existing Approaches to the MWM problem

Apart from their extensive use and their vital role in industries, there are few academic works dedicated to study the characteristics of the MWM problem [186]. Narkhede et al. developed a microcontroller based MWM model which operates by choosing three of five available weighing hoppers for every instance of the product weighing process [179]. The optimization algorithm in this MWM model works by monitoring the sum of the weights of each possible combination continuously until there is one that matches the target weight. In the Internet of Things (IoT) framework, Ma'ayan and Dabran made use of MWM operation data from a frozen vegetable factory to create an emulation model that is comprised of 14 weighing hoppers from which four hoppers are chosen in every weighing process [187]. The hopper selection is initiated by sorting the total weights from each possible hopper combination and excluding the ones that have values below the target weight. From the remaining combinations, the one with closest value to the target weight will finally be selected. Garcia-Diaz et al. [188] and Pulido-Rojano et al. [189] also implemented a similar strategy where the optimization software calculates the weight of each possible hopper combination and then selects the one that has the best value.

Karuno and Nakahama took a different approach than the explicit enumerative strategy [185] which was used in all of the previously mentioned works. They implemented a dynamic programming method for an MWM packing scenario in which the optimization condition requires both the total weight and the number of selected product to exceed certain values [183]. In other publications, Karuno and Nakahama also provided greedy heuristic solutions for a mixture packaging of two types of products [184] and for a bi-criteria mixture packaging problem [185]. In these last two works, however, Karuno and Nakahama did not provide numerical experiments that would allow the heuristic performance to be compared to other existing approaches.

8.3 *MMAS* FOR MWM

The *MMAS* in the hypercube framework [14] that we applied to the MWM problem works as described in Chapter 2. The pheromone model \mathcal{T} for this application consists of a pheromone value τ_{ij} for each hopper j of each product type i . The initialization of pheromone values and other parameters is exactly as described in Section 2.2. Overall, the algorithm works as outlined in Algorithm 2.1 with the detailed description of the pheromone update function and convergence

factor calculation as provided in Sections 2.2.2 and 2.2.3, respectively.

Function `Construct_Solution(\mathcal{T})` applied to the MWM problem works as follows. Solution S for this optimization problem is defined as a binary structure such that $s_{ij} \in \{0, 1\}$ indicates the product in j -th hopper allocated for type i -th products. The solution S to be constructed is initialized to $s_{ij} = 0$ for all $i = 1, \dots, n_t$ and $j = 1, \dots, n_i$. Then, at each iteration, a product type i from n_t available product types is selected randomly. Subsequently, a weighing hopper j is selected from hopper set $H_i := \{1, \dots, n_i\}$ allocated to products of type i . The probability for any hopper j to be chosen in the current construction step is defined as follows:

$$\mathbf{p}(j|S) := \frac{\eta_{ij} \cdot \tau_{ij}}{\sum_{j=1}^{n_i} \eta_{ij'} \cdot \tau_{ij'}} \quad \forall j \in H_i \quad (8.5)$$

Hereby, η_{ij} is the so-called greedy function that evaluates the goodness of selecting the j -th hopper allocated for product type i , defined as follows:

$$\eta_{ij} := \left| \frac{1}{w_{ij} - \overline{w_{ij}}} \right| \quad (8.6)$$

Note that $\overline{w_{ij}}$ in Eqn. (8.6) is the average of product's weights in all hoppers.

Based on the probabilities in Eqn. (8.5), the selection of j —that is, the hopper to be added to S —is done as follows. First, a random number $r \in [0, 1]$ is chosen uniformly at random. In case $r \leq d_{\text{rate}}$, $j := \operatorname{argmax}\{\mathbf{p}(j|S) \mid j \in H_i\}$. Otherwise, j is chosen by roulette wheel selection. Note that $d_{\text{rate}} \in [0, 1]$ —the so-called determinism rate—is an important parameter of the algorithm. Once a hopper j is selected, it is removed from hopper set H_i such that $H_i := H_i \setminus \{j\}$. This selection process is repeated until the target weights p_i of each product type as well as the target weight P of the package is reached or exceeded.

8.4 EXPERIMENTAL EVALUATION

In addition to the Aco algorithm we also evaluated the implementation of CPLEX 12.10 in one-threaded mode as the performance benchmark for the MWM optimization problem. All experiments concerning the two algorithms were performed on a cluster of machines with Intel® Xeon® CPU 5670 CPUs with 12 cores of 2.933 GHz and a minimum of 32 GB RAM.

8.4.1 Problem instances

In this work, we generated MWM test cases with configurations as shown in Table 8.1. The target weight for these test cases is 500 grams. The number of product types n_t varies from 1 to 3. For the variation of the number of weighing hoppers n ; we chose 16 and 32. For each of these number of hoppers we decided to set the number of chosen hoppers k to be at least twice the number of product types. With this arrangement, the corresponding average weight in each hopper μ was calculated by dividing the target weight P by the number of chosen hoppers k .

Variable σ in Table 8.1 represents the ratio between the maximum weight deviations with respect to the average weight in each hopper. For simulating the effect of product weight variability, we conduct three different values for σ : 0.1, 0.2, and 0.3 for each available type of product. The values of the weights in each hopper were generated randomly by using the C++ `normal_distribution` function that works by taking mean and standard deviation values as input data.

Table 8.1 MWM test case configurations with target a weight of 500 grams

No.	n_t	n	k	μ (g)	σ_1	σ_2	σ_3
1	1	[16,32]	[2, 3, 4]	[250, 166.67, 125]	[0.1, 0.2, 0.3]	NA	NA
2	2	[16,32]	[4, 5, 6]	[125, 100, 83.33]	[0.1, 0.2, 0.3]	[0.1, 0.2, 0.3]	NA
3	3	[16,32]	[6, 7, 8]	[83.33, 71.43, 62.5]	[0.1, 0.2, 0.3]	[0.1, 0.2, 0.3]	[0.1, 0.2, 0.3]

Table 8.2 shows the number of hoppers allocated to each product type (n_1 , n_2 , and n_3) in relation to the number of chosen hoppers k , number of all hoppers n , and number of product types n_t .

8.4.2 Algorithm tuning and test settings

The Aco algorithm requires well-working values for n_a (number of solution constructions per iteration), d_{rate} (determinism rate), and ρ (learning rate). Again, we made use of the scientific tuning software `irace` [123] for the purpose of parameter tuning. This tool was used for generating a single parameter setting for each number of product types.

As test cases for tuning we chose the first out of the 10 instances for each combination of "number of product type", "number of hoppers", and "number of chosen hoppers". Finally, the budget of `irace` was fixed to 2000 runs, each one with a time limit of 1 CPU second, which is more than enough to represent the required computational time in the actual MWM operation.

The considered parameter value domains were as follows: $n_a \in \{3, 5, 10, 20\}$,

Table 8.2 MWM operational configurations

No.	n_t	n	k	n_1	n_2	n_3
1	1	16	2	16	NA	NA
2	1	16	3	16	NA	NA
3	1	16	4	16	NA	NA
4	1	32	2	32	NA	NA
5	1	32	3	32	NA	NA
6	1	32	4	32	NA	NA
7	2	16	4	8	8	NA
8	2	16	5	8	8	NA
9	2	16	6	8	8	NA
10	2	32	4	16	16	NA
11	2	32	5	16	16	NA
12	2	32	6	16	16	NA
13	3	16	6	5	5	6
14	3	16	7	5	5	6
15	3	16	8	5	5	6
16	3	32	6	10	11	11
17	3	32	7	10	11	11
18	3	32	8	10	11	11

$d_{\text{rate}} \in \{0.0, 0.1, 0.2, \dots, 0.8, 0.9\}$, and $\rho \in \{0.1, \dots, 0.5\}$. The obtained parameter value settings are shown in Table 8.3.

Table 8.3 ACO parameter values obtained by irace

No.	MWM configuration		ACO parameter		
	n_t	n	n_a	d_{rate}	ρ
1	1	16	10	0.5	0.0
2	1	32	17	0.4	0.1
3	2	16	5	0.5	0.0
4	2	32	10	0.4	0.0
5	3	16	15	0.2	0.5
6	3	32	19	0.1	0.1

8.4.3 Results

We set a time limit of 1 second for every test in this research and in each of these test cases CPLEX found the optimum solution. In total, the results of Aco match the ones of CPLEX in 2276 out of 2340 test cases. In the group of test cases with one or two product types, we can see in Tables 8.4 and 8.5 that the results of Aco are found mostly in much shorter time than the ones of CPLEX. In the group of test cases with three product types we can see in Table 8.6 that there are several test configurations in which Aco's performance falls slightly below the one of CPLEX. Even though the average difference of total weights and execution times are only 0.024 grams and 42.42 milliseconds, respectively, this fact shows us the

Table 8.4 CPLEX and ACO results and execution times for the MWM packing problem with one product type

No.	n	k	σ_1	Avg. opt. weights (g)		Avg. time (ms)	
				CPLEX	ACO	CPLEX	ACO
1	16	2	0.1	500.65	500.65	12.96	0.14
2	16	2	0.2	500.59	500.59	11.70	0.08
3	16	2	0.3	501.00	501.00	11.25	0.12
4	16	3	0.1	500.14	500.14	19.45	0.31
5	16	3	0.2	500.21	500.21	14.01	0.47
6	16	3	0.3	500.11	500.11	16.27	0.58
7	16	4	0.1	500.01	500.01	16.82	0.80
8	16	4	0.2	500.02	500.02	19.32	1.50
9	16	4	0.3	500.05	500.05	23.73	1.81
10	32	2	0.1	500.08	500.08	20.93	0.49
11	32	2	0.2	500.34	500.34	23.73	0.41
12	32	2	0.3	500.18	500.18	25.27	1.07
13	32	3	0.1	500.00	500.00	17.75	1.23
14	32	3	0.2	500.00	500.00	30.23	1.52
15	32	3	0.3	500.00	500.00	26.75	3.05
16	32	4	0.1	500.00	500.00	17.39	1.72
17	32	4	0.2	500.00	500.00	31.92	1.93
18	32	4	0.3	500.00	500.00	28.30	0.94

relevance of developing more powerful optimization algorithms in response to the growing complexity of MWM optimization problems.

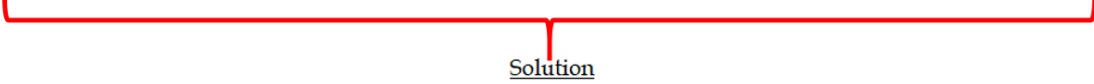
The results of our numerical experiments show the importance of MWM initial setup optimization as already studied by Beretta et al. [186], Del Castillo

Table 8.5 CPLEX and ACO results and execution times for the MWM packing problem with two product types

No.	n	k	σ_1	σ_2	Avg. opt. weights (g)		Avg. time (ms)	
					CPLEX	ACO	CPLEX	ACO
1	16	4	0.1	[0.1, 0.2, 0.3]	506.05	506.05	11.50	0.62
2	16	4	0.2	[0.1, 0.2, 0.3]	507.91	507.91	11.24	0.58
3	16	4	0.3	[0.1, 0.2, 0.3]	505.99	505.99	10.79	0.98
4	16	5	0.1	[0.1, 0.2, 0.3]	536.00	536.00	9.57	1.39
5	16	5	0.2	[0.1, 0.2, 0.3]	513.38	513.38	10.1	1.35
6	16	5	0.3	[0.1, 0.2, 0.3]	512.40	512.40	11.70	5.68
7	16	6	0.1	[0.1, 0.2, 0.3]	501.57	501.57	12.29	3.45
8	16	6	0.2	[0.1, 0.2, 0.3]	503.28	503.28	11.76	2.35
9	16	6	0.3	[0.1, 0.2, 0.3]	504.98	504.98	12.55	3.05
10	32	4	0.1	[0.1, 0.2, 0.3]	500.78	500.78	26.59	10.63
11	32	4	0.2	[0.1, 0.2, 0.3]	501.03	501.03	26.01	11.55
12	32	4	0.3	[0.1, 0.2, 0.3]	501.08	501.08	28.30	14.54
13	32	5	0.1	[0.1, 0.2, 0.3]	517.18	517.18	18.47	16.91
14	32	5	0.2	[0.1, 0.2, 0.3]	504.96	504.96	29.66	38.29
15	32	5	0.3	[0.1, 0.2, 0.3]	504.55	504.55	41.35	47.99
16	32	6	0.1	[0.1, 0.2, 0.3]	500.06	500.06	78.59	113.36
17	32	6	0.2	[0.1, 0.2, 0.3]	500.16	500.16	81.77	233.57
18	32	6	0.3	[0.1, 0.2, 0.3]	500.12	500.12	66.10	233.90

Problem instance

Product 1								Product 2							
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{29}	w_{210}	w_{211}	w_{212}	w_{213}	w_{214}	w_{215}	w_{216}
124.5	134	138.2	130.2	137.8	90	134.3	120.8	106.8	116.1	115	132.7	127.4	124	116.6	125.1



Solution

Product 1								Product 2											
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{29}	w_{210}	w_{211}	w_{212}	w_{213}	w_{214}	w_{215}	w_{216}				
124.5			130.2									127.4			125.1				
Total weights of product 1 (grams)=								254.7		Total weights of product 2 (grams)=								252.5	
Total weights (grams)=																507.2			


Fig. 8.2 Illustrative example of problem instance and solution of configuration number 1 in Table 8.5

et al. [180, 190, 191], as well as Garcia-Diaz et al. [188] and Pulido-Rojano et al. [189]. The quality of the MWM optimization process is influenced not only by the optimization software used during its operation but also by the initial configuration of the MWM. A clear example of this can be seen by comparing configuration number 1 and number 4 in Table 8.5 where the first configuration shows a better average of the optimum weights than the latter.

Figures 8.2 and 8.3 show illustrative examples of product weight distributions in problem instances of configuration number 1 and 4 in Table 8.5. The product weight data used in these configurations is characterized by different average weight values for the same σ_1 and σ_2 variations and combinations. In these examples, both configurations—number 1 and 4—have $\sigma_1 = 0.1$ and $\sigma_2 = 0.1$.

Problem instance

Product 1								Product 2							
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{29}	w_{210}	w_{211}	w_{212}	w_{213}	w_{214}	w_{215}	w_{216}
100.8	111.4	112.6	107.6	113.1	95.9	113.1	109.8	88.2	98.7	108.9	84.1	90	93.3	100.6	123.2



Solution

Product 1								Product 2											
w_{11}	w_{12}	w_{13}	w_{14}	w_{15}	w_{16}	w_{17}	w_{18}	w_{29}	w_{210}	w_{211}	w_{212}	w_{213}	w_{214}	w_{215}	w_{216}				
100.8			107.6		95.9			88.2			84.1	90							
Total weights of product 1 (grams)=								304.3		Total weights of product 2 (grams)=								262.3	
Total weights (grams)=																566.6			

Fig. 8.3 Illustrative example of problem instance and solution of configuration number 4 in Table 8.5

However, the average weight for configuration number 1 was calculated based on the assumption that four weighing hoppers are selected every time. As shown in Fig. 8.2, the products are distributed in each hopper such that their average weight is close to 125 grams, that is 500 grams of target weights divided by four hoppers to be selected at each weighing process. On the other hand, the average weight for configuration number 4 was calculated based on the assumption that five weighing hoppers are selected every time. Figure 8.3 shows that the products are distributed in each hopper such that their average weight is close to 100 grams, that is 500 grams of target weight divided by five hoppers to be selected at each weighing process. With this arrangement, there is a high probability that one or more product types in configuration number 4 will need to select an additional hopper in order to pass the minimum weight requirement, as shown in Fig. 8.3. This is not only the case in configuration number 4 but also in configuration numbers 5, 6, 13, 14, and 15. For these configurations, however, we can see that the average optimum weights are better than the ones for configuration number 4.

Average optimum weight improvements in configurations 5 and 6 over configuration number 4 in Table 8.5 occur as the weights' variability between the hoppers is getting larger. In this way, there are more options for combining pairs of hoppers that carry a larger and a smaller weight, respectively, combining to a total weight that is closer to the target weight of the package. This finding is in accordance to the one found in the numerical experiments of Del Castillo et al. [191] where they suggested that there should be a certain number of hoppers containing considerably less product in order to be able to construct better optimum weight combinations. The average optimum weights in configurations 13, 14, and 15 in Table 8.5 improve over the ones in configurations 4, 5, 6 as the total number of hoppers in the first group of configurations double over the one in the latter. A similar pattern can also be observed in Table 8.6 between configurations 37 to 45 and configurations 10 to 18. This finding indicates that the adverse influence of an unfavourable weight distribution is becoming less dominant as the number of hoppers is increased.

All in all, we can observe in Tables 8.4 to 8.6 that even though optimum solutions for all test cases can be found, their quality is getting worse as the number of product types to be packed is getting larger. This observation shows that an optimization method is needed not only for selecting hoppers during MWM operation but also for determining the initial setup prior to its operation.

Table 8.6 CPLEX and ACO results and execution times for the MWM packing problem with three product types

No.	n	k	σ_1	σ_2	σ_3	Avg. opt. weights (g)		Avg. time (ms)	
						CPLEX	ACO	CPLEX	ACO
1	16	6	0.1	0.1	[0.1, 0.2, 0.3]	524.68	524.68	8.79	2.84
2	16	6	0.1	0.2	[0.1, 0.2, 0.3]	515.31	515.31	7.84	5.83
3	16	6	0.1	0.3	[0.1, 0.2, 0.3]	518.47	518.47	7.94	7.93
4	16	6	0.2	0.1	[0.1, 0.2, 0.3]	514.49	514.49	8.11	6.85
5	16	6	0.2	0.2	[0.1, 0.2, 0.3]	517.77	517.77	8.59	11.57
6	16	6	0.2	0.3	[0.1, 0.2, 0.3]	522.21	522.21	8.49	6.61
7	16	6	0.3	0.1	[0.1, 0.2, 0.3]	519.70	519.70	8.49	4.84
8	16	6	0.3	0.2	[0.1, 0.2, 0.3]	517.94	517.94	8.52	3.38
9	16	6	0.3	0.3	[0.1, 0.2, 0.3]	519.56	519.56	8.78	5.03
10	16	7	0.1	0.1	[0.1, 0.2, 0.3]	579.69	579.69	8.91	0.47
11	16	7	0.1	0.2	[0.1, 0.2, 0.3]	558.42	558.42	9.05	14.93
12	16	7	0.1	0.3	[0.1, 0.2, 0.3]	557.61	557.61	8.74	2.07
13	16	7	0.2	0.1	[0.1, 0.2, 0.3]	556.35	556.35	8.68	17.47
14	16	7	0.2	0.2	[0.1, 0.2, 0.3]	538.22	538.22	9.28	5.89
15	16	7	0.2	0.3	[0.1, 0.2, 0.3]	535.86	535.86	9.61	23.98
16	16	7	0.3	0.1	[0.1, 0.2, 0.3]	557.98	557.98	8.86	1.27
17	16	7	0.3	0.2	[0.1, 0.2, 0.3]	531.73	531.73	8.66	12.69
18	16	7	0.3	0.3	[0.1, 0.2, 0.3]	527.84	527.84	9.49	18.38
19	16	8	0.1	0.1	[0.1, 0.2, 0.3]	525.98	525.98	8.50	1.22
20	16	8	0.1	0.2	[0.1, 0.2, 0.3]	530.88	530.88	9.43	1.00
21	16	8	0.1	0.3	[0.1, 0.2, 0.3]	527.72	527.72	9.37	5.47
22	16	8	0.2	0.1	[0.1, 0.2, 0.3]	525.25	525.25	8.63	3.03
23	16	8	0.2	0.2	[0.1, 0.2, 0.3]	521.25	521.25	9.02	2.67
24	16	8	0.2	0.3	[0.1, 0.2, 0.3]	525.23	525.23	9.45	27.31
25	16	8	0.3	0.1	[0.1, 0.2, 0.3]	525.28	525.28	8.89	2.00
26	16	8	0.3	0.2	[0.1, 0.2, 0.3]	520.85	520.85	8.47	1.84
27	16	8	0.3	0.3	[0.1, 0.2, 0.3]	522.60	522.60	8.76	4.65
28	32	6	0.1	0.1	[0.1, 0.2, 0.3]	502.39	502.40	19.69	85.92
29	32	6	0.1	0.2	[0.1, 0.2, 0.3]	502.93	502.94	20.44	95.88
30	32	6	0.1	0.3	[0.1, 0.2, 0.3]	502.48	502.48	21.48	108.91
31	32	6	0.2	0.1	[0.1, 0.2, 0.3]	502.97	502.97	20.11	63.30
32	32	6	0.2	0.2	[0.1, 0.2, 0.3]	503.63	503.64	18.59	62.94
33	32	6	0.2	0.3	[0.1, 0.2, 0.3]	505.97	506.03	22.21	56.32
34	32	6	0.3	0.1	[0.1, 0.2, 0.3]	503.19	503.24	19.86	92.91
35	32	6	0.3	0.2	[0.1, 0.2, 0.3]	504.56	504.73	19.97	152.07
36	32	6	0.3	0.3	[0.1, 0.2, 0.3]	503.86	503.94	23.21	155.20
37	32	7	0.1	0.1	[0.1, 0.2, 0.3]	546.63	546.64	14.56	20.98
38	32	7	0.1	0.2	[0.1, 0.2, 0.3]	530.32	530.32	19.35	62.11
39	32	7	0.1	0.3	[0.1, 0.2, 0.3]	523.99	524.03	19.28	74.66
40	32	7	0.2	0.1	[0.1, 0.2, 0.3]	528.02	528.03	18.03	50.57
41	32	7	0.2	0.2	[0.1, 0.2, 0.3]	510.68	510.70	18.99	141.93
42	32	7	0.2	0.3	[0.1, 0.2, 0.3]	507.79	507.90	23.83	187.12
43	32	7	0.3	0.1	[0.1, 0.2, 0.3]	529.77	529.77	19.37	83.80
44	32	7	0.3	0.2	[0.1, 0.2, 0.3]	506.53	506.66	21.50	118.84
45	32	7	0.3	0.3	[0.1, 0.2, 0.3]	509.44	509.47	25.15	143.92
46	32	8	0.1	0.1	[0.1, 0.2, 0.3]	509.95	509.95	17.02	77.03
47	32	8	0.1	0.2	[0.1, 0.2, 0.3]	505.20	505.21	21.67	119.06
48	32	8	0.1	0.3	[0.1, 0.2, 0.3]	504.92	504.95	25.49	156.77
49	32	8	0.2	0.1	[0.1, 0.2, 0.3]	505.80	505.81	18.50	93.59
50	32	8	0.2	0.2	[0.1, 0.2, 0.3]	504.17	504.25	21.02	119.61
51	32	8	0.2	0.3	[0.1, 0.2, 0.3]	503.19	503.29	30.81	183.06
52	32	8	0.3	0.1	[0.1, 0.2, 0.3]	505.93	505.96	21.14	108.12
53	32	8	0.3	0.2	[0.1, 0.2, 0.3]	503.45	503.55	32.61	155.28
54	32	8	0.3	0.3	[0.1, 0.2, 0.3]	503.53	503.72	27.39	138.05

8.5 CONCLUSIONS

This work shows that ACO performs very well in tackling the MWM optimization problem with one, two, and three product types. Furthermore, it produces optimal solutions in 97.3% of total test cases, all within excellent execution times. There are many studies on the initial setup optimization for MWM packing problem with one product type. In this work we can see that average optimum weights found in test cases with two and three product types are relatively large when compared to the ones from test cases with only one product type. This shows further the importance of MWM initial setup optimization that should be considered in these types of problems.

CHAPTER 9

CONCLUSIONS AND OUTLOOK

9.1 CONCLUSIONS

Ant Colony Optimization (Aco) is a metaheuristic optimization technique inspired by the foraging behavior of ant colonies in nature. This algorithm consists of two main algorithmic components: solution construction and pheromone update. At each iteration, Aco constructs several solutions, in a probabilistic way, using greedy information as well as pheromone values. Good solutions found in the current iteration and possibly in earlier iterations are used to update the pheromone values. These two algorithmic components are repeatedly executed so that new solutions with similar components to the ones of the previous best solution have a higher probability of being considered in subsequent iterations. In this way, Aco uses the best results in the past to improve the solutions in forthcoming iterations. Hence, it can be said that Aco makes use of positive learning, that is, learning from positive examples.

Since its first introduction in 1991, Aco has evolved into several improved variants, for example, by improving the exploitation of the positive learning mechanism. Nevertheless, nature provides numerous examples which show that negative learning mechanisms are also an integral part of the communication and coordination systems of a range of social insects. Moreover, examples of negative learning in species and inter-species evolution, animal swarm behavior, and human history have already been adopted in metaheuristic techniques such as evolutionary algorithms, extremal optimization, particle swarm optimization [28, 29, 31, 32], and opposition based learning algorithm [33, 34]. The Aco research community has also identified this potential and developed some relevant works. Most of them, however, with limited successes.

The existing variants of negative learning Aco offer numerous features to identify, store, use, and update the negative learning information. Based on how the negative learning information is stored and updated, these variants can be divided into two groups. The first group uses a dedicated pheromone

model to deposit the negative learning information, in addition to the standard pheromone model that keeps the positive learning information. The other group of variants uses the negative learning information to enhance the reduction of the corresponding pheromone values in the standard pheromone model. Comparing these two approaches, we considered that a dedicated pheromone model with its own update mechanism is advantageous for the negative learning Aco implementation. This way, the update of each pheromone type can be managed independently to adapt to the characteristics of different problem instances. Hence, we implemented this feature in our negative learning Aco proposal.

The existing variants of negative learning Aco identify the negative learning information either by using passive or active methods. Passive methods [82–84, 86, 87, 93, 96, 98] are implemented simply by choosing the worst solution in an Aco iteration, without using any dedicated function to search for it. An active identification method [84, 88–90], on the contrary, allocates one or more functions to search for low-quality solutions. Generally, negative learning information is used to reduce the probability of selecting seemingly low-quality solution components in subsequent iterations. Several variants [88–90], however, use the information as a base for generating a neighborhood of solutions. Therefore, instead of using the negative learning information to limit their search area as in other variants, these variants use it to enhance their exploration scope.

Our negative learning Aco employs two novel mechanisms for identifying the negative learning information. The first mechanism creates a sub-instance by merging components of solutions found in an iteration of the baseline *MMAS* algorithm. The second mechanism uses an optimization tool (*Cplex*, *MMAS*, *SATLike*, or *SlsMcs*) as additional algorithmic component in order to find the possibly best solution in the sub-instance. Unlike the existing identification methods, our negative learning Aco identifies presumably bad solution components by comparing solution components in the sub-instance to the ones in the solution found by the additional algorithmic component. Any solution component in the sub-instance that is not present in the additional algorithmic component's solution is considered as a bad solution component. Subsequently, the negative pheromone value of this component is increased. Hence, it has a lower probability of being selected in solutions of the next iteration. This way, our negative learning Aco employs a dedicated mechanism for identifying the negative learning information. However, this mechanism does not directly search for low-quality solutions as in the existing active identification method. Hence, our strategy does not waste computational resources for

finding bad solutions that otherwise can be avoided by the baseline *MMAS* algorithm itself. Moreover, by having this arrangement, the results of the additional algorithmic component can also be used to enhance the positive learning mechanism.

We tested our negative learning Aco proposal on a range of CO problems in order to evaluate its applicability and effectiveness. The summary of these tests is as follows:

- In Chapter 3 we described the application of our negative learning Aco to the MDKP. In this work, we tested ten variants of negative learning Aco. Six of them are from our proposal, and the rest are our re-implementation of the negative learning approaches from the literature [84, 87]. We experimented with ILP solver CPLEX and *MMAS* as the options to provide negative feedback to the baseline *MMAS* algorithm. According to this arrangement, our algorithm variants are divided into two groups based on the type of algorithm used as the additional algorithmic component. In the group where the *MMAS* is used as the additional algorithmic component, we have two instances of the Aco algorithm working independently in the same algorithmic framework. The first *MMAS* serves as the baseline algorithm, and the other one—or the *inner-MMAS*—serves as the additional algorithmic component that provides negative feedback to the baseline Aco algorithm. Each of the two algorithm groups consists of three variants according to the way in which the result of CPLEX or the *inner-MMAS* is used in the baseline *MMAS*. The first variant uses the result of the inner algorithm to enhance both the positive and the negative learning mechanisms. The second and the third variants, on the other hand, each uses the result for reinforcing only the negative or only the positive learning mechanism. The four negative learning approaches from the literature were re-introduced in the context of *MMAS* as the baseline algorithm. Subsequently, each negative learning feature was added to the baseline algorithm. The results show that the negative learning Aco variants—particularly those that use CPLEX for producing the negative feedback information—perform significantly better than the existing approaches from the literature. The results also show that, even though negative learning is not always beneficial for all MDKP problem instances considered in this work, it is remarkably advantageous for several groups of problem instances with rather many resources. In general, it was shown that it is not harmful to add negative learning because the globally best-performing algorithm variant— $\text{Aco}_{\text{neg}}^+$ —uses this

mechanism. Moreover, the results show that the globally best-performing algorithm variant can compete with current state-of-the-art algorithms for the MDKP [102, 103].

- In Chapter 4 we presented the application of our negative learning Aco to the MDS problem. In this chapter, we followed exactly the same procedure as in the case of the MDKP. The results show the same tendency found already in the results of the MDKP experiment. In general, all our negative learning Aco variants outperform the proposals from the literature. Comparing the group of algorithm variants that use CPLEX to the other group that use *MMAS* as the additional algorithmic component, we found that the former ones perform significantly better than the latter ones. This finding shows that the quality of the additional algorithmic component holds an essential role in determining the overall performance of the algorithmic framework. The more accurate the negative feedback, the better the global performance of the algorithm. We have several important findings concerning the way in which the result of the additional algorithmic component is utilized. Generally, variants that use the result to enhance both the negative and positive learning perform better than those that use it to enhance only the negative or only the positive learning mechanism. Interestingly, Aco_{neg} —a variant that uses the result of the additional algorithmic component to enhance only its negative learning mechanism—is the best algorithm for MDS instances of the random geometric graph type. Moreover, $\text{Aco}_{\text{neg}}^+$ —our best-performing algorithm variant—performs competitively to state-of-the-art algorithms for the MDS problems [105–109].
- In Chapter 5 we described the application of our negative learning Aco to the MaxSAT problem, which has different characteristics to the CO problems considered in the previous two chapters. Consequently, it is a perfect test case for evaluating the general applicability of our negative learning approach, especially in the context of how the sub-instance is created and how the negative learning information is identified. So far, there are no applications of negative learning Aco to this CO problem. Despite being a well-known metaheuristic method, Aco has only been applied a few times to solve MaxSAT problems. In contrast, numerous powerful solvers from the field of satisfiability testing are already available for solving this CO problem. Therefore, the MaxSAT problem provides a perfect opportunity to compare the effectiveness of our negative learning Aco approach to the one of existing MaxSAT solvers. We tested our negative learning Aco variants

that use CPLEX against the results of the Aco approaches from the literature on two instance groups. The results show that all of the negative learning Aco variants deliver outstanding performance when compared to the Aco approaches from the literature. In addition to the variants that use ILP solver CPLEX, we developed two new variants in this application, each of which uses a MaxSAT solver—SATLike-c(w) or SLSMcs—as the additional algorithmic component. Testing these new variants and the existing ones on an instance group consisting of problem instances from recent MaxSAT evaluation events, we show that all our negative learning Aco variants improved over the baseline Aco approach and each one of the internally used solvers. This result indicates the effectiveness of our negative learning proposal. Moreover, it also shows that using high-performance MaxSAT solvers to solve sub-instances within our algorithmic framework can further improve their performance.

- In Chapter 6 we provided a detailed description of the application of our negative learning Aco to the CapMDS problem. This CO problem was in fact the first one to which we applied our negative learning Aco at the start of the Ph.D. We tested two negative learning Aco variants in this work, each of which uses ILP solver CPLEX as the additional algorithmic component for providing negative feedback to the baseline Aco algorithm. We used two groups of CapMDS instances—uniform and variable capacity graphs—to test our algorithm variants against the standard MMAS algorithm. In both instance groups, it can be seen that both algorithm variants with negative learning significantly improve over the baseline algorithm MMAS. Comparing these two variants, we observed that Aco_{neg}^+ improves over Aco_{neg} with statistical significance. Interestingly, when separating the instances based on graph densities, we found that negative learning is instrumental in the case of sparse graphs. Furthermore, our best-performing negative learning Aco variant— Aco_{neg}^+ —performs competitively to the state-of-the-art approach for CapMDS problems [100].
- In Chapter 7 we described our negative learning Aco application to the MPIDS problem, a CO problem that has actual application in social networks. As in our application to the CapMDS problem, we developed and tested two negative learning Aco variants in this work: (1) Aco_{neg}^+ and (2) Aco_{neg} . These variants were tested on 17 MPIDS problems that are usually used in the literature and on ten larger social networks from the SNAP

library (<https://snap.stanford.edu/data/>). We compared the obtained results to the ones of *MMAS*, *Cplex*, and the state-of-the-art approaches for MPIDS [111, 112]. The comparison showed that the two negative learning Aco variants outperform all the competitors. Even the baseline Aco was already able to compete with the state of the art. Comparing the performance of the two negative learning Aco variants, we found that they performed competitively on the group of 17 small/medium size problem instances. However, on the ten larger problem instances, $\text{Aco}_{\text{neg}}^+$ performs significantly better than Aco_{neg} . With this finding, we can state that $\text{Aco}_{\text{neg}}^+$ is the new state-of-the-art metaheuristic for solving the MPIDS problem. Moreover, this shows again that using negative learning in addition to positive learning can be very advantageous.

In summary we can say that our negative learning Aco variants improve consistently over the standard Aco algorithm on all considered CO problems. The results also show that our negative learning Aco variants outperform all negative learning approaches from the literature for most of the MDKP and the MDS problem instances used in this work. Moreover, $\text{Aco}_{\text{neg}}^+$ —the best-performing negative learning Aco variant—always performs competitively with the state-of-the-art approaches for each considered CO problem. In fact, for solving the MPIDS problem, $\text{Aco}_{\text{neg}}^+$ is currently the state-of-the-art approach. All in all, it can be said that we have proven the general applicability and the effectiveness of our negative learning Aco proposal in this work.

9.2 OUTLOOK

As mentioned above, this thesis has served to demonstrate the effectiveness of the proposed negative learning Aco proposal in solving several CO problems. Moreover, we have demonstrated that our proposal works well with *Cplex* and other algorithms as options for the additional algorithmic component that provides negative feedback to the baseline Aco algorithm. An extension of this thesis could be imagined along the following lines.

- Our negative learning Aco variants consistently improved over the standard Aco algorithm; however, we found that one of them— Aco_{neg} —did not perform as consistently as the other one— $\text{Aco}_{\text{neg}}^+$ —, especially in our work concerning the MDS problem. Aco_{neg} outperformed $\text{Aco}_{\text{neg}}^+$ significantly when applied to instances from the random geometric graph type; yet,

it obtained poor results when applied to those from the random graph type. Although this finding might be explained by using the *no free lunch theorem* [192, 193], investigating further into the correlation between the performance of this negative learning Aco variant and the problem instance characteristics will be an essential development path of this thesis.

- A natural extension of our work on MaxSAT is to adapt our negative learning ACO for weighted MaxSAT and partial MaxSAT, which is the variant of MaxSAT that declares some clauses as hard and imposes that hard clauses must be satisfied by any valid solution. Since industrial instances are generally encoded using weighted and partial MaxSAT, it might be interesting to use a SAT-based MaxSAT solver or a branch-and-bound MaxSAT solver with clause learning as the additional algorithmic component. These solvers are particularly competitive for industrial instances and our negative learning Aco might help to improve their performance. Finally, another extension of this work is to incorporate a decimation approach [152] in the generation of solutions.
- In Chapter 8 we described the application of the *MMAS* algorithm to the *Multi-head Weigher Machine* (MWM) optimization problem that has many applications in food packaging industries. This CO problem is concerned with applying the combinatorial weighing method in which an optimization algorithm selects a subset of products—from a set of n available products—that results in a total product weight closest to (but not less than) a specific target weight. We generated MWM test cases with several configurations in this work. The results show that the standard *MMAS* already successfully solved these optimization problems. It produces optimal solutions in 97.3% of all test cases, all within excellent execution times. Furthermore, we used the results obtained by the optimization software to analyze the relationship between the test case configuration and the quality of an optimal solution. The finding shows that the quality of an optimal solution generally declines as the size and complexity of the test case increases. Note that even though the optimization software may find the optimum total weights for the available test cases, the results may not be sufficiently good in terms of the practical requirement, which is determined by how close they are to a specified target weight. Hence, optimizing the test case configurations is an inherent aspect of solving this CO problem for its practical use. Accordingly, this work will be an important area for the future development of this thesis. In fact,

it will be a good starting point for implementing hybrid metaheuristics, particularly the negative learning Aco, to actual problems in engineering.

References

- [1] Marco Dorigo and Thomas Stützle. Ant colony optimization: Overview and recent advances. In *Handbook of Metaheuristics*, pages 311–351. Springer, 2019.
- [2] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.
- [3] Orhan Engin and Abdullah Güçlü. A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. *Applied Soft Computing*, 72:166–176, 2018.
- [4] Erfan Babaee Tirkolaee, Mehdi Alinaghian, Ali Asghar Rahmani Hosseinabadi, Mani Bakhshi Sasi, and Arun Kumar Sangaiah. An improved ant colony optimization for the multi-trip capacitated arc routing problem. *Computers and Electrical Engineering*, 77:457–470, 2019.
- [5] Raka Jovanovic, Milan Tuba, and Stefan Voß. An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research*, 274(1):78–90, 2019.
- [6] Huijun Peng, Chun Ying, Shuhua Tan, Bing Hu, and Zhixin Sun. An improved feature selection algorithm based on ant colony optimization. *IEEE Access*, 6:69203–69209, 2018.
- [7] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Positive feedback as a search strategy. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.6342>, 1991. Accessed: 2020-03-15.
- [8] Alberto Colorni, Marco Dorigo, Vittorio Maniezzo, et al. Distributed optimization by ant colonies. In *Proceedings of The First European Conference on Artificial Life*, volume 142, pages 134–142. Paris, France, 1991.

- [9] Marco Dorigo, Vittorio Maniezzo, and Alberto Coloni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.
- [10] Bernd Bullnheimer, Richard F Hartl, and Christine Strauß. A new rank based version of the ant system. A computational study. <https://epub.wu.ac.at/id/eprint/616>, 1997. Accessed : 2020-03-16.
- [11] Bernd Bullnheimer, Richard F Hartl, and Christine Strauß. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.
- [12] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.
- [13] Thomas Stützle and Holger H Hoos. MAX–MIN ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.
- [14] Christian Blum and Marco Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):1161–1172, 2004.
- [15] Y Schlein, R Galun, and MN Ben-Eliahu. Abstinons. *Journal of Chemical Ecology*, 7(2):285–290, 1981.
- [16] M Giurfa. The repellent scent-mark of the honeybee *Apis mellifera* tigustica and its role as communication cue during foraging. *Insectes Sociaux*, 40(1): 59–67, 1993.
- [17] Elva JH Robinson, Duncan E Jackson, Mike Holcombe, and Francis LW Ratnieks. ‘No entry’ signal in ant foraging. *Nature*, 438(7067):442–442, 2005.
- [18] Elva JH Robinson, Duncan E Jackson, Mike Holcombe, and Francis LW Ratnieks. No entry signal in ant foraging (hymenoptera: Formicidae): new insights from an agent-based model. *Myrmecological News*, 10, 2007.
- [19] Christoph Grüter, Roger Schürch, Tomer J Czaczkes, Keeley Taylor, Thomas Durance, Sam M Jones, and Francis LW Ratnieks. Negative feedback enables fast and flexible collective decision-making in ants. *PLoS One*, 7(9):e44501, 2012.

- [20] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.
- [21] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.
- [22] Darrell Whitley. An overview of evolutionary algorithms: Practical issues and common pitfalls. *Information and Software Technology*, 43(14):817–831, 2001.
- [23] Stefan Boettcher and Allon G Percus. Extremal optimization: Methods derived from co-evolution. *arXiv preprint math/9904056*, 1999.
- [24] Stefan Boettcher. Extremal optimization: Heuristics via coevolutionary avalanches. *Computing in Science and Engineering*, 2(6):75–82, 2000.
- [25] Stefan Boettcher and Allon Percus. Nature’s way of optimizing. *Artificial Intelligence*, 119(1-2):275–286, 2000.
- [26] Stefan Boettcher and Allon G Percus. Optimization with extremal dynamics. *Complexity*, 8(2):57–62, 2002.
- [27] Stefan Boettcher and Allon G Percus. Extremal optimization at the phase transition of the three-coloring problem. *Physical Review E*, 69(6):066703, 2004.
- [28] Peter J Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *International Conference on Evolutionary Programming*, pages 601–610. Springer, 1998.
- [29] Maurice Clerc. *Particle Swarm Optimization*, volume 93. John Wiley and Sons, 2010.
- [30] Yann Cooren, Maurice Clerc, and Patrick Siarry. Initialization and displacement of the particles in TRIBES, a parameter-free particle swarm optimization algorithm. In *Adaptive and Multilevel Metaheuristics*, pages 199–219. Springer, 2008.
- [31] Yann Cooren, Maurice Clerc, and Patrick Siarry. Performance evaluation of TRIBES, an adaptive particle swarm optimization algorithm. *Swarm Intelligence*, 3(2):149–178, 2009.

- [32] Yann Cooren, Maurice Clerc, and Patrick Siarry. MO-TRIBES, an adaptive multiobjective particle swarm optimization algorithm. *Computational Optimization and Applications*, 49(2):379–400, 2011.
- [33] Hamid R Tizhoosh. Opposition-based learning: A new scheme for machine intelligence. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 1, pages 695–701. IEEE, 2005.
- [34] Sedigheh Mahdavi, Shahryar Rahnamayan, and Kalyanmoy Deb. Opposition based learning: A literature review. *Swarm and Evolutionary Computation*, 39:1–23, 2018.
- [35] Christian Blum and Günther R Raidl. *Hybrid Metaheuristics: Powerful Tools for Optimization*. Springer, 2016.
- [36] Steven S Skiena. *The Algorithm Design Manual*, volume 2. Springer, 1998.
- [37] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson Education India, 2006.
- [38] Richard Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 38(8):716, 1952.
- [39] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.
- [40] Joxan Jaffar, Andrew E Santosa, and Razvan Voicu. Efficient memoization for dynamic programming with ad-hoc constraints. In *AAAI*, volume 8, pages 297–303, 2008.
- [41] Guido Moerkotte and Thomas Neumann. Dynamic programming strikes back. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 539–552, 2008.
- [42] Juan Pablo Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57, 2015.
- [43] IBM. ILOG CPLEX optimization studio documentation. <http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer>, 2020. Accessed : 2020-02-06.

- [44] Rimmi Anand, Divya Aggarwal, and Vijay Kumar. A comparative analysis of optimization solvers. *Journal of Statistics and Management Systems*, 20(4): 623–635, 2017.
- [45] LLC Gurobi Optimization. Gurobi optimizer reference manual. <http://www.gurobi.com>, 2020. Accessed : 2020-02-06.
- [46] Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.
- [47] Robert Ashford. Mixed integer programming: A historical perspective with Xpress-MP. *Annals of Operations Research*, 149(1):5, 2007.
- [48] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. Choco: an open source java constraint programming library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Constraint Programming (OSSICP'08)*, pages 1–10, 2008.
- [49] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018.
- [50] David S Johnson, Christos H Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, 37(1):79–100, 1988.
- [51] Nenad Mladenović and Pierre Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.
- [52] Thomas A Feo and Mauricio GC Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.
- [53] Rubén Ruiz and Thomas Stützle. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049, 2007.
- [54] Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer, 2003.
- [55] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [56] Rob A Rutenbar. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine*, 5(1):19–26, 1989.

- [57] Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29–57, 1993.
- [58] Fred Glover. Tabu search—part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
- [59] Fred Glover. Tabu search—part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
- [60] Fred Glover and Manuel Laguna. Tabu search. In *Handbook of Combinatorial Optimization*, pages 2093–2229. Springer, 1998.
- [61] Roberto Battiti and Giampietro Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.
- [62] Pierre Hansen and Nenad Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3): 449–467, 2001.
- [63] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report*, 826:1989, 1989.
- [64] Pablo Moscato and Carlos Cotta. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, pages 105–144. Springer, 2003.
- [65] Yuhui Shi and Russell Eberhart. Particle swarm optimization: Developments, applications and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No. 01TH8546)*, volume 1, pages 81–86. IEEE, 2001.
- [66] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE, 1995.
- [67] Xiao-Hu Zhi, XL Xing, QX Wang, LH Zhang, XW Yang, CG Zhou, and YC Liang. A discrete PSO method for generalized TSP problem. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No. 04EX826)*, volume 4, pages 2378–2383. IEEE, 2004.
- [68] Xiaohu H Shi, Yanchun Chun Liang, Heow Pueh Lee, C Lu, and QX Wang. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, 103(5):169–176, 2007.

- [69] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. An overview. *Swarm Intelligence*, 1(1):33–57, 2007.
- [70] Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 69–73. IEEE, 1998.
- [71] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality. *Physical Review A*, 38(1):364, 1988.
- [72] Per Bak and Kan Chen. Self-organized criticality. *Scientific American*, 264(1):46–53, 1991.
- [73] Per Bak and Kim Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71(24):4083, 1993.
- [74] Yu-Wang Chen, Yao-Jia Zhu, Gen-Ke Yang, and Yong-Zai Lu. Improved extremal optimization for the asymmetric traveling salesman problem. *Physica A: Statistical Mechanics and its Applications*, 390(23-24):4459–4465, 2011.
- [75] Stefan Boettcher. Extremal optimization for Sherrington-Kirkpatrick spin glasses. *The European Physical Journal B-Condensed Matter and Complex Systems*, 46(4):501–505, 2005.
- [76] Edward D Weinberger. Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63(5):325–336, 1990.
- [77] Peter F Stadler. Towards a theory of landscapes. In *Complex Systems and Binary Networks*, pages 78–163. Springer, 1995.
- [78] Vittorio Maniezzo. Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS Journal on Computing*, 11(4):358–369, 1999.
- [79] Éric Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991.
- [80] Yong Li, Panos M Pardalos, and Mauricio GC Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. *Quadratic Assignment and Related Problems*, 16:237–261, 1993.

- [81] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In *Machine Learning Proceedings 1995*, pages 38–46. Elsevier, 1995.
- [82] Oscar Cordón, Inaki Fernández de Viana, Francisco Herrera, and Llanos Moreno. A new ACO model integrating evolutionary computation concepts: The best-worst ant system. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.30.6593>, 2000. Accessed : 2020-03-16.
- [83] Oscar Cordón, Inaki Fernández de Viana, and Francisco Herrera. Analysis of the best-worst ant system and its variants on the QAP. In *International Workshop on Ant Algorithms*, pages 228–234. Springer, 2002.
- [84] James Montgomery and Marcus Randall. Anti-pheromone as a tool for better exploration of search space. In *International Workshop on Ant Algorithms*, pages 100–110. Springer, 2002.
- [85] Steffen Iredi, Daniel Merkle, and Martin Middendorf. Bi-criterion optimization with multi colony ant algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 359–372. Springer, 2001.
- [86] Chris Simons and Jim Smith. Exploiting antipheromone in ant colony optimisation for interactive search-based software design and refactoring. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 143–144, 2016.
- [87] Vitorino Ramos, David MS Rodrigues, and Jorge Louçã. Second order swarm intelligence. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 411–420. Springer, 2013.
- [88] Alice R Malisia and Hamid R Tizhoosh. Applying opposition-based ideas to the ant colony system. In *2007 IEEE Swarm Intelligence Symposium*, pages 182–189. IEEE, 2007.
- [89] Zhaojun Zhang, Zhaoxiong Xu, Shengyang Luan, Xuanyu Li, and Yifei Sun. Opposition-based ant colony optimization algorithm for the traveling salesman problem. *Mathematics*, 8(10):1650, 2020.
- [90] Nicolás Rojas-Morales, María-Cristina Riff, Carlos A Coello Coello, and Elizabeth Montero. A cooperative opposite-inspired learning strategy for ant-based algorithms. In *International Conference on Swarm Intelligence*, pages 317–324. Springer, 2018.

- [91] Nicolás Rojas-Morales, María-Cristina Riff Rojas, and Elizabeth Montero Ureta. A survey and classification of opposition-based metaheuristics. *Computers and Industrial Engineering*, 110:424–435, 2017.
- [92] Inès Alaya, Christine Solnon, and Khaled GHÉDIRA. Ant algorithm for the multidimensional knapsack problem. In *International Conference on Bioinspired Methods and their Applications (BIOMA 2004)*, pages 63–72, 2004.
- [93] Ke Ye, Changsheng Zhang, Jiayu Ning, and Xiaojie Liu. Ant-colony algorithm with a strengthened negative-feedback mechanism for constraint-satisfaction problems. *Information Sciences*, 406:29–41, 2017.
- [94] Antonio Gonzalez-Pardo and David Camacho. A new CSP graph-based representation for ant colony optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 689–696. IEEE, 2013.
- [95] Wei Xu and Fuzhou Gong. Performances of pure random walk algorithms on constraint satisfaction problems with growing domains. *Journal of Combinatorial Optimization*, 32(1):51–66, 2016.
- [96] Takuya Masukane and Kazunori Mizuno. Solving constraint satisfaction problems by cunning ants with multi-pheromones. *International Journal of Machine Learning and Computing*, 8(4), 2018.
- [97] Shigeyoshi Tsutsui. cAS: Ant colony optimization with cunning ants. In *Parallel Problem Solving from Nature-PPSN IX*, pages 162–171. Springer, 2006.
- [98] Takuya Masukane and Kazunori Mizuno. Refining a pheromone trail graph by negative feedback for constraint satisfaction problems. In *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 1–6. IEEE, 2019.
- [99] Teddy Nurcahyadi and Christian Blum. A new approach for making use of negative learning in ant colony optimization. In *International Conference on Swarm Intelligence*, pages 16–28. Springer, 2020.
- [100] Pedro Pinacho-Davidson, Salim Bouamama, and Christian Blum. Application of CMSA to the minimum capacitated dominating set problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 321–328, 2019.
- [101] Teddy Nurcahyadi and Christian Blum. Negative learning in ant colony optimization: Application to the multi dimensional knapsack problem.

In *2021 5th International Conference on Intelligent Systems, Metaheuristics and Swarm Intelligence*, pages 22–27, 2021.

- [102] Xiangjing Lai, Jin-Kao Hao, Fred Glover, and Zhipeng Lü. A two-phase tabu-evolutionary algorithm for the 0–1 multidimensional knapsack problem. *Information Sciences*, 436:282–301, 2018.
- [103] Xiangjing Lai, Jin-Kao Hao, Zhang-Hua Fu, and Dong Yue. Diversity-preserving quantum particle swarm optimization for the multidimensional knapsack problem. *Expert Systems with Applications*, 149: 113310, 2020.
- [104] Teddy Nurcahyadi and Christian Blum. Adding negative learning to ant colony optimization: A comprehensive study. *Mathematics*, 9(4):361, 2021.
- [105] Yiyuan Wang, Shaowei Cai, and Minghao Yin. Local search for minimum weight dominating set with two-level configuration checking and frequency based scoring function. *Journal of Artificial Intelligence Research*, 58:267–295, 2017.
- [106] Yiyuan Wang, Shaowei Cai, Jiejiang Chen, and Minghao Yin. A fast local search algorithm for minimum weight dominating set problem on massive graphs. In *IJCAI*, pages 1514–1522, 2018.
- [107] David Chalupa. An order-based algorithm for minimum dominating set with application in graph mining. *Information Sciences*, 426:101–116, 2018.
- [108] Yi Fan, Yongxuan Lai, Chengqian Li, Nan Li, Zongjie Ma, Jun Zhou, Longin Jan Latecki, and Kaile Su. Efficient local search for minimum dominating sets in large graphs. In *International Conference on Database Systems for Advanced Applications*, pages 211–228. Springer, 2019.
- [109] Shaowei Cai, Wenying Hou, Yiyuan Wang, Chuan Luo, and Qingwei Lin. Two-goal local search and inference rules for minimum dominating set. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, pages 1467–1473, 2020.
- [110] Albert López Serrano, Teddy Nurcahyadi, Salim Bouamama, and Christian Blum. Negative learning ant colony optimization for the minimum positive influence dominating set problem. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1974–1977, 2021.

- [111] Geng Lin, Jian Guan, and Huibin Feng. An ILP based memetic algorithm for finding minimum positive influence dominating sets in social networks. *Physica A: Statistical Mechanics and its Applications*, 500:199–209, 2018.
- [112] Geng Lin, Jinyan Luo, Haiping Xu, and Meiqin Xu. A hybrid swarm intelligence-based algorithm for finding minimum positive influence dominating sets. In Yong Liu, Lipo Wang, Liang Zhao, and Zhengtao Yu, editors, *Proceedings of ICNC-FSKD 2019 – Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery*, pages 506–511. Springer International Publishing, 2020.
- [113] Fahiem Bacchus, Matti Järvisalo, and Martins Ruben. Maximum satisfiability. In *Handbook of Satisfiability, second edition*, pages 929–991. IOS Press, 2021.
- [114] Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In *Handbook of Satisfiability, second edition*, pages 903–927. IOS Press, 2021.
- [115] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Rubens Martins. MaxSAT evaluation 2020: Solver and benchmark descriptions. <https://helda.helsinki.fi/bitstream/handle/10138/318451/mse20proc.pdf?sequence=1>, 2020. Accessed : 2021-06-03.
- [116] Arnaud Fréville. The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, 155(1):1–21, 2004.
- [117] Paul C Chu and John E Beasley. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4(1):63–86, 1998.
- [118] Xiangyong Kong, Liqun Gao, Haibin Ouyang, and Steven Li. Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm. *Computers and Operations Research*, 63:7–22, 2015.
- [119] Ling Wang, Sheng-yao Wang, and Ye Xu. An effective hybrid eda-based algorithm for solving multidimensional knapsack problem. *Expert Systems with Applications*, 39(5):5593–5599, 2012.
- [120] Yannick Vimont, Sylvain Boussier, and Michel Vasquez. Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem. *Journal of Combinatorial Optimization*, 15(2):165–178, 2008.

- [121] Sylvain Boussier, Michel Vasquez, Yannick Vimont, Saïd Hanafi, and Philippe Michelon. A multi-level search strategy for the 0–1 multidimensional knapsack problem. *Discrete Applied Mathematics*, 158(2): 97–109, 2010.
- [122] Renata Mansini and M Grazia Speranza. Coral: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24 (3):399–415, 2012.
- [123] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3: 43–58, 2016.
- [124] Borja Calvo and Guzmán Santafé Rodrigo. scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal*, Vol. 8/1, Aug. 2016, 2016.
- [125] Salvador Garcíá and Francisco Herrera. An extension on “statistical comparisons of classifiers over multiple data sets” for all pairwise comparisons. *Journal of Machine Learning Research*, 9(Dec):2677–2694, 2008.
- [126] Gabriela Ochoa, Katherine M Malan, and Christian Blum. Search trajectory networks of population-based algorithms in continuous spaces. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 70–85. Springer, 2020.
- [127] Gabriela Ochoa, Katherine M Malan, and Christian Blum. Search trajectory networks: A tool for analysing and visualising the behaviour of metaheuristics. *Applied Soft Computing*, page 107492, 2021.
- [128] Michael R. Garey and David S. Johnson. *Computers and Intractability*, volume 174. Freeman San Francisco, 1979.
- [129] Ruizhi Li, Shuli Hu, Huan Liu, Ruiting Li, Dantong Ouyang, and Minghao Yin. Multi-start local search algorithm for the minimum connected dominating set problems. *Mathematics*, 7(12):1173, 2019.
- [130] Fuyu Yuan, Chenxi Li, Xin Gao, Minghao Yin, and Yiyuan Wang. A novel hybrid algorithm for minimum total dominating set problem. *Mathematics*, 7(3):222, 2019.

- [131] Yupeng Zhou, Jinshu Li, Yang Liu, Shuai Lv, Yong Lai, and Jianan Wang. Improved memetic algorithm for solving the minimum weight vertex independent dominating set. *Mathematics*, 8(7):1155, 2020.
- [132] Habiba Drias, Amine Taibi, and Sofiane Zekour. Cooperative ant colonies for solving the maximum weighted satisfiability problem. In *International Work-Conference on Artificial Neural Networks*, pages 446–453. Springer, 2003.
- [133] Habiba Drias and Sarah Ibri. Parallel ACS for weighted Max-Sat. In *International Work-Conference on Artificial Neural Networks*, pages 414–421. Springer, 2003.
- [134] Pedro C Pinto, Thomas A Runkler, and Joao MC Sousa. An ant algorithm for static and dynamic MAX-SAT problems. In *Proceedings of the 1st International Conference on Bio Inspired Models of Network, Information and Computing Systems*, pages 10–es, 2006.
- [135] Marcos Villagra and Benjamín Barán. Ant colony optimization with adaptive fitness function for satisfiability testing. In *International Workshop on Logic, Language, Information, and Computation*, pages 352–361. Springer, 2007.
- [136] João Guerra and Inês Lynce. Reasoning over biological networks using maximum satisfiability. In *Proceedings of the 18th International Conference on Principles and Practice of Constraint Programming, CP*, pages 941–956, 2012.
- [137] João Marques-Silva, Josep Argelich, Ana Graça, and Inês Lynce. Boolean lexicographic optimization: algorithms and applications. *Annals of Mathematics and Artificial Intelligence*, 62(3-4):317–343, 2011.
- [138] Carlos Ansótegui, Felip Manyà, Jesus Ojeda, Josep M. Salvia, and Eduard Torres. Incomplete MaxSAT approaches for combinatorial testing. *Journal of Heuristics*, 2022. In press.
- [139] Saïd Jabbour, Nizar Mhadhbi, Badran Raddaoui, and Lakhdar Sais. A SAT-based framework for overlapping community detection in networks. In *Proceedings of the 21st Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, Part II, PAKDD*, pages 786–798, 2017.
- [140] Dominique D’Almeida and Éric Grégoire. Model-based diagnosis with default information implemented through MAX-SAT technology. In *Proceedings of the IEEE 13th International Conference on Information Reuse and Integration, IRI*, pages 33–36, 2012.

- [141] Lei Zhang and Fahiem Bacchus. MAXSAT heuristics for cost optimal planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, pages 1846–1852, 2012.
- [142] Miquel Bofill, Marc Garcia, Josep Suy, and Mateu Villaret. MaxSAT-based scheduling of B2B meetings. In *Proceedings of the 12th International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR*, pages 65–73, 2015.
- [143] Felip Manyà, Santiago Negrete, Carme Roig, and Joan Ramon Soler. Solving the team composition problem in a classroom. *Fundamenta Informaticae*, 174(1):83–101, 2020.
- [144] Josep Argelich, Chu-Min Li, Felip Manyà, and Jordi Planes. The first and second Max-SAT evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, 4(2-4):251–278, 2008.
- [145] Holger H Hoos and Thomas Stützle. Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481, 2000.
- [146] Holger H Hoos and Kevin O’Neill. Stochastic local search methods for dynamic SAT—an initial investigation. In *AAAI-2000 Workshop on Leveraging Probability and Uncertainty in Computation*, pages 22–26, 2000.
- [147] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise strategies for improving local search. In *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI’94*, pages 337–343, 1994.
- [148] David McAllester, Bart Selman, and Henry Kautz. Evidence for invariants in local search. In *Proceedings of the 14th National Conference on Artificial Intelligence, AAAI’97*, pages 321–326, 1997.
- [149] Shaowei Cai, Chuan Luo, John Thornton, and Kaile Su. Tailoring local search for partial MaxSAT. In *The Twenty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2014*, pages 2623–2629, 2014.
- [150] Chuan Luo, Shaowei Cai, Kaile Su, and Wenxuan Huang. CCEHC: An efficient local search algorithm for weighted partial maximum satisfiability. *Artificial Intelligence*, 243:26–44, 2017.
- [151] Zhendong Lei and Shaowei Cai. Solving (weighted) partial MaxSAT by dynamic local search for SAT. In *The Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 1346–1352, 2018.

- [152] Shaowei Cai and Zhendong Lei. Old techniques in new ways: Clause weighting, unit propagation and hybridization for maximum satisfiability. *Artificial Intelligence*, 287:103354, 2020.
- [153] Chu Min Li, Felip Manyà, and Jordi Planes. Exploiting unit propagation to compute lower bounds in branch and bound Max-SAT solvers. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP-2005*, pages 403–414, 2005.
- [154] Chu Min Li, Felip Manyà, and Jordi Planes. Detecting disjoint inconsistent subformulas for computing lower bounds for Max-SAT. In *Proceedings of the 21st National Conference on Artificial Intelligence, AAI-2006*, pages 86–91, 2006.
- [155] Chu Min Li, Felip Manyà, Nouredine Ould Mohamedou, and Jordi Planes. Resolution-based lower bounds in MaxSAT. *Constraints*, 15(4):456–484, 2010.
- [156] Chu Min Li, Felip Manyà, and Jordi Planes. New inference rules for Max-SAT. *Journal of Artificial Intelligence Research*, 30:321–359, 2007.
- [157] André Abramé and Djamel Habet. Ahmaxsat: Description and evaluation of a branch and bound Max-SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 9(1):89–128, 2014.
- [158] Chu-Min Li, Zhenxing Xu, Jordi Coll, Felip Manyà, Djamel Habet, and Kun He. Combining clause learning and branch and bound for MaxSAT. In *Proceedings of the 27th International Conference on Principles and Practice of Constraint Programming, CP*, volume 210 of *LIPICs*, pages 38:1–38:18, 2021.
- [159] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7(2-3):59–6, 2010.
- [160] Tobias Paxian and Bernd Becker. Pacose: An iterative SAT-based MaxSAT solver. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, page 12, 2020.
- [161] Ruben Martins, Vasco M. Manquinho, and Inês Lynce. Open-WBO: A modular MaxSAT solver. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing, SAT*, pages 438–445, 2014.
- [162] Carlos Ansótegui and Joel Gabàs. WPM3: An (in)complete algorithm for Weighted Partial MaxSAT. *Artificial Intelligence*, 250:37–57, 2017.

- [163] Alexey Ignatiev, António Morgado, and João Marques-Silva. RC2: an efficient MaxSAT solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 11(1):53–64, 2019.
- [164] Paul Saikko, Jeremias Berg, and Matti Järvisalo. LMHS: A SAT-IP hybrid MaxSAT solver. In *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing, SAT*, volume 9710 of *LNCS*, pages 539–546, 2016.
- [165] Fahiem Bacchus. MaxHS in the 2020 MaxSAT Evaluation. In *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*, pages 19–20, 2020.
- [166] Anupama Potluri and Alok Singh. Metaheuristic algorithms for computing capacitated dominating set with uniform and variable capacities. *Swarm and Evolutionary Computation*, 13:22–33, 2013.
- [167] Feng Wang, Erika Camacho, and Kuai Xu. Positive influence dominating set in online social networks. In *International Conference on Combinatorial Optimization and Applications*, pages 313–321. Springer, 2009.
- [168] Feng Wang, Hongwei Du, Erika Camacho, Kuai Xu, Wonjun Lee, Yan Shi, and Shan Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412(3):265–269, 2011.
- [169] Angela K. Fournier, Erin Hall, Patricia Ricke, and Brittany Storey. Alcohol and the social network: Online social networking sites and college students’ perceived drinking norms. *Psychology of Popular Media Culture*, 2(2):86, 2013.
- [170] Cheng Long and Raymond Chi-Wing Wong. Minimizing seed set for viral marketing. In *2011 IEEE 11th International Conference on Data Mining*, pages 427–436. IEEE press, 2011.
- [171] Dilek Günneç, Subramanian Raghavan, and Rui Zhang. Least-cost influence maximization on social networks. *INFORMS Journal on Computing*, 32(2):289–302, 2020.
- [172] Guangyuan Wang. *Domination Problems in Social Networks*. PhD thesis, University of Southern Queensland, 2014.
- [173] Amir Afrasiabi Rad and Morad Benyoucef. Towards detecting influential users in social networks. In *International Conference on E-Technologies*, pages 227–240. Springer, 2011.

- [174] Hassan Raei, Nasser Yazdani, and Masoud Asadpour. A new algorithm for positive influence dominating set in social networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 253–257. IEEE, 2012.
- [175] Mai Fei and Chen Weidong. An improved algorithm for finding minimum positive influence dominating sets in social networks. *Journal of South China Normal University*, 48(3):59–63, 2016.
- [176] Jiehui Pan and Tian-Ming Bu. A fast greedy algorithm for finding minimum positive influence dominating sets in social networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops*, pages 360–364. IEEE, 2019.
- [177] Salim Bouamama and Christian Blum. An improved greedy heuristic for the minimum positive influence dominating set problem in social networks. *Algorithms*, 14(3):79, 2021.
- [178] Helge A Wurdemann, Vahid Aminzadeh, Jian S Dai, John Reed, and Graham Purnell. Category-based food ordering processes. *Trends in Food Science and Technology*, 22(1):14–20, 2011.
- [179] Parag Narkhede, Ritesh Dhawale, and B Karthikeyan. Microcontroller based multihead weigher. *Indian Journal of Science and Technology*, 9(30): 1–5, 2016.
- [180] Enrique Del Castillo, Alessia Beretta, and Quirico Semeraro. Optimal setup of a multihead weighing machine. *European Journal of Operational Research*, 259(1):384–393, 2017.
- [181] Rafael García-Jiménez, J Carlos García-Díaz, and Alexander D Pulido-Rojano. Packaging process optimization in multihead weighers with double-layered upright and diagonal systems. *Mathematics*, 9(9):1039, 2021.
- [182] Alexander Pulido-Rojano, J Carlos García-Díaz, and Vicent Giner-Bosch. A multiobjective approach for optimization of the multihead weighing process. In *2015 International Conference on Industrial Engineering and Systems Management (IESM)*, pages 426–434. IEEE, 2015.
- [183] Yoshiyuki Karuno and Oki Nakahama. Performance of a heuristic total weight in combinatorial mixture packaging of two types of items. In

Proceedings of the Sixth International Conference on Industrial Application Engineering (ICIAE 2018, IIAE), pages 235–238, 2018.

- [184] Yoshiyuki Karuno and Oki Nakahama. A requirement for the number of items in a package produced by multi-headweighers. In *2018 18th International Conference on Control, Automation and Systems (ICCAS)*, pages 1397–1402. IEEE, 2018.
- [185] Yoshiyuki Karuno and Oki Nakahama. An improved performance of greedy heuristic solutions for a bi-criteria mixture packaging problem of two types of items with bounded weights. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 14(5):JAMDSM0066–JAMDSM0066, 2020.
- [186] Alessia Beretta, Quirico Semeraro, and Enrique del Castillo. On the multihead weigher machine setup problem. *Packaging Technology and Science*, 29(3):175–188, 2016.
- [187] Dor Ma’ayan and Itai Dabran. Case study: Implementing an industrial iot solution for a multihead weighing machine (MWM). In *2019 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, pages 1–4. IEEE, 2019.
- [188] J Carlos García-Díaz and Alexander Pulido-Rojano. Performance analysis and optimisation of new strategies for the setup of a multihead weighing process. *European Journal of Industrial Engineering*, 14(1):58–84, 2020.
- [189] Alexander Pulido-Rojano and J Carlos García-Díaz. Optimisation algorithms for improvement of a multihead weighing process. *International Journal of Productivity and Quality Management*, 29(1):109–125, 2020.
- [190] Enrique del Castillo, Alessia Beretta, and Quirico Semeraro. Analysis and optimal targets setup of a multihead weighing machine. *arXiv preprint arXiv:1511.07504*, 2015.
- [191] Enrique del Castillo, Alessia Beretta, and Quirico Semeraro. Optimal targets setup of a multihead weighing machine. <https://sites.psu.edu/engineeringstatistics/files/2016/09/WeigherPaperSeptember2016-1trrc2d.pdf>, 2016. Accessed : 2021-09-24.
- [192] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

- [193] Yu-Chi Ho and David L Pepyne. Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications*, 115(3):549–570, 2002.