Disco – Novo – GoGo
Integrating Local Search and Complete Search with Restarts *

Meinolf Sellmann
Department of Computer Science
Brown University, Box 1910
Providence, RI 02912, USA
sello@cs.brown.edu

Carlos Ansótegui
Artificial Intelligence Research Institute
(IIIA-CSIC)
Bellaterra, SPAIN
carlos@iiia.csic.es

Abstract

A hybrid algorithm is devised to boost the performance of complete search on under-constrained problems. We suggest to use random variable selection in combination with restarts, augmented by a coarse-grained local search algorithm that learns favorable value heuristics over the course of several restarts. Numerical results show that this method can speed-up complete search by orders of magnitude.

Introduction

Local search methods are known to perform well on under-constrained problems where they allow us to solve instances of sizes that are orders of magnitude larger than what any systematic search method is able to handle. Complete search methods on the other hand are suited to tackle critically constrained and over-constrained problems as well, due to their ability to prove unsatisfiability when no solution exists.

Related Work: Many efforts have been undertaken in the past to bring the respective strengths of both local and complete search approaches together. In (Fang and Ruml 2004) and in (Shen and Zhang 2005), local search approaches were presented that trade some of their time and space efficiency to also be able to prove unsatisfiability. Analogously, in (Prestwich 2002) and in (Prestwich 2000), the ideas of "stochastic local search" and "randomized backtracking" were introduced. In the same spirit as "greedy randomized adaptive search procedures (GRASPs)" (Hart and Shogan 1987; Pitsoulis and Resende 2001), these methods consist in tree-based search methods that achieve some speed-up on under-constrained problem instances by giving up the completeness of the underlying systematic algorithms.

While all these contributions try to overcome the limitations of the respective main approach by giving up some of its strength, there were also hybrid approaches proposed that actually combine local and systematic search. Probably the easiest hybridization idea is to use local search first and then to switch over to complete search later. This is a very common idea in constrained optimization, where the initial local search phase can help to find a near-optimal solution quickly, which can improve tremendously the pruning effectiveness in the following branch-and-bound approach. Another idea in this scenario is to use a very simple local search algorithm (like a quick greedy algorithm for instance) to improve on the performance of any feasible solution that is being found by the "master" tree-search approach. This idea was adapted by (Habet *et al.* 2002) for SAT where a depth-bounded systematic search triggers local searches at nodes that reach the depth-limit. (Fischetti and Lodi 2003) introduced the idea of "local branching", a method that partitions the search space by adding non-unary branching constraints that emulate the local search neighborhood of some "guiding solution".

In the realm of constraint satisfaction, a partially very successful technique is "large neighborhood search (LNS)" (Shaw 1997) which can be viewed as a special case of "variable neighborhood search" (Hansen and Mladenovic 2003). The idea here is to explore a potentially very large local search neighborhood by means of systematic search methods. Interestingly, discrepancy-based search methods like "limited discrepancy search" (Harvey and Ginsberg 1997) and other methods like "dynamic backtracking" (Ginsberg *et al.* 1996) can also be viewed as hybrids as they enforce search space exploration and effectively enable complete search methods to find feasible solutions very quickly if they exist. Many other hybrid local/complete search approaches have been developed, and thanks to new powerful tools like the constraint-based local search system "Comet" (Van Hentenryck and Michel 2005) and novel conceptual frameworks like the one developed by (Hooker 2005), we can hope for many more in the future. For a more thorough introduction to the topic of local and complete search hybridization we refer the reader to the excellent tutorial presented by (Focacci *et al.* 2001).

Contribution: Tree-search typically constitutes the core of constraint programming solvers and Davis-Putnam-Logewood-Loveland (DPLL) algorithms in satisfiability. These algorithms have shown to be well-suited for critically constrained and over-constrained problem instances, but for under-constrained problems they often lack the speed and efficiency that local search approaches exhibit. When a user is not really sure whether his/her problem instance has a solution, what is (s)he supposed to do? Today, an expert would probably recommend to try a local search first, and if that does not work for some time, to start over with a systematic search. An approach which strangely resembles the authors' own strategy when trying to find the latest credit card bill at home... Unfortunately, unlike for constrained optimization,

*Research partially supported by TIN2004-07933-C03-03, TIN2005-09312-C03-01 and TIC2003-00950 funded by the *Ministerio de Educación y Ciencia*.
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

in constraint satisfaction an initial local search phase really does not give us anything and is basically a waste of time if it ends unsuccessfully.

In this paper, we try to improve upon the situation by enhancing the ability of tree-search based methods to find feasible solutions for under-constrained problems quickly. We pursue this goal by exploiting the following observation: Between restarts, we have the freedom to change not only the variable ordering (which is of course being picked randomly in this setting), but we also have the freedom to change the value selection heuristic which tells our search which child-node to investigate first when diving deeper into the tree. The main contribution of this paper is an algorithm that uses local search to "learn" better value selection heuristics over the course of many restarts.

We describe the idea in detail in the following section. Then, we provide extensive numerical results that show that our method is able to speed-up tree-search by orders of magnitude on under-constrained problems.

Organizing Systematic Search

Whether we consider integer programming, constraint programming, or satisfiability: all three areas are dealing with finite yet NP-hard problems, and in all three areas the most efficient and most successful complete solvers are based on tree-search augmented by some inference methods such as linear programming relaxations, constraint propagation, or no-good learning. Consequently, research in the respective communities can roughly be separated in inference related or search related contributions.

This paper deals with search. Within any systematic search, there are some liberties that require us to make decisions on how we want to organize the search. The important questions that need to be answered are: How do we split the partition that we are currently investigating when our inference methods are inconclusive? And second, in which partition of the search space do we continue our search next? In classical backtrack search with unary branching, these questions simplify to the following questions: On which variable shall we branch next? And what value shall we assign to the branching variable first?

Interestingly, there are really two schools who advocate two radically different ways of answering these questions. The first group of researchers tries to devise "heuristics". Some try to come up with procedures that make decisions that appear reasonable from a human point of view, and others try to "learn" from the experience gathered during search to make better decisions in the future (see as an example (Epstein *et al.* 2002; Vidotto *et al.* 2005)).

The second school of researchers takes a completely different approach: instead of attempting to make "reasonable guesses", they have chance decide how the search space is partitioned. Now, there is a substantial probability that random choices result in very long runtimes — in practice we measure heavy-tailed runtime distributions (Gomes *et al.* 1997) — but there is also a good chance that we end up with a reasonably short run. Consequently the random-choice approach is complemented by the idea of restarting the search whenever we have reason to believe that we were probably just not lucky with the particular partitioning that we just tried (see as an example (Kautz *et al.* 2002)).

Random Variables — Heuristic Values

To speed-up complete search on under-constrained problem instances, we hybridize both ideas: We suggest to choose the branching variables randomly while trying to learn good value heuristics over the course of different restarts. The motivation for this is that a good value selection heuristic can guide us to a feasible solution effectively no matter how badly we happen to partition the search space. The idea is not unlike the motivation behind adaptive multistart methods for local search where starting states are generated by a generation method that is biased by the centroid of previous local optima (Boese *et al.* 1994). The approach, if it succeeds, has the advantage that, when we are insecure about the constrainedness of our problem instance, we could start out with one approach right away without wasting time on an initial, potentially unsuccessful, local search phase.

The question arises how we can improve our value selection heuristic by learning from previous searches? We decided to take a very conservative approach by trying to stick to most parts of the earlier value selection heuristic and to update it only where the search has proven it to be inconsistent. Let us explain this in a little more detail: To us, a value selection heuristic is simply an assignment of values to variables. It captures nothing else than the value that we intend to try on every variable first. By representing a value selection heuristic as an assignment, the entire endeavor to find a good value heuristic nicely aligns with our overall goal to find a feasible assignment.

Note that one can view the conservative update of the heuristic over the course of several restarts as a very coarse-grained local search (some may even view it as a large neighborhood search): The next heuristic evolves from the previous one by a very complex move that is the result of the previous restart of the randomized systematic search procedure. We said already that we want to keep as much of the previous heuristic assignment as possible. On the other hand, we also want to update the heuristic where search has revealed that it is in itself inconsistent.

So, what does an unsuccessful restart tell us with respect to the inconsistencies within our last value selection heuristic? Let us use the convention that a left-most branch corresponds to following the heuristic. Now, we consider the path that leads to the search node S *right before* backtracking fails and the cut-off limit is reached. Wherever that path does not follow the left-most branch, the search has revealed that, given the decisions taken before, the heuristic is inconsistent at this point. Furthermore, whenever constraint or unit propagation reveal that a value in the current assignment must be eliminated from the domain of the current variable, the heuristic is inconsistent.

Consequently, after a restart reaches its cut-off limit, we update the heuristic like this: We consider the domains of variables at S. For all variables, we check whether the value denoted in our current value selection heuristic is actually still in its domain. If yes, then we leave the heuristic untouched for this variable. And if no, then we choose a random value out of the variable's domain and make that value our next first choice value for this variable. We refer to this method as Disco-Novo-GoGo.[1]

[1]From (poor) Latin: "I learn, I start anew, I speed myself up"

We formalize the discussion by sketching the algorithms that we will use in our experimental evaluation. Assume our traditional tree-search approach with restarts works like this:

```
TR: bool Traditional (void)
InitFailLimit (failLimit)
while (true) do
   status ← TreeSearch(failLimit)
   if (status ≠ inconclusive) then
      return (status == solved)
   end if
   UpdateFailLimit (failLimit)
end while
```

Note that we leave it open according to which strategy the successive failLimits are chosen (see e.g. (Luby *et al.* 1993)), and that the call to TreeSearch may very well have global side-effects, e.g. due to no-good learning. Then, for our basic hybrid, we change the algorithm as follows:

```
BH: bool BasicHybrid (void)
InitFailLimit(failLimit), InitRandom (heuristic)
while (true) do
   status ← TreeSearch(failLimit,heuristic)
   if (status ≠ inconclusive) then
      return (status == solved)
   end if
   UpdateHeuristic (heuristic), UpdateFailLimit (failLimit)
end while
```

We assume that TreeSearch actually performs one more backtracking step after the fail limit has been reached. Then, at the choice point where the search ends, either all variables still have at least one value in their respective domains, or the search fails entirely, in which case we expect TreeSearch to return 'unsolvable'. Only when this is not the case, we update the heuristic as follows:

```
void UpdateHeuristic (ValueArray heuristic)
for all (x ∈ variableSet) do
   if (heuristic[x] ∉ domain[x]) then
      heuristic[x] ← ChooseRandomValue(domain[x])
   end if
end for
```

Now, depending on how the fail limit is updated after each restart, the length of each restart can actually become very long rather quickly. Obviously, the local search approach that learns a better value heuristic suffers dearly if there are only very few steps taken. Therefore, and also to account for the fact that local search algorithms are known to benefit from restarts themselves, we introduce two variants of the traditional and the basic hybrid approach:

```
MRT: bool MetaRestartTraditional (void)
InitMoveLimit(maxLocalMoves)
while (true) do
   InitFailLimit(failLimit), moves ← 0
   while (moves++ < maxLocalMoves) do
      status ← TreeSearch(failLimit)
      if (status ≠ inconclusive) then
         return (status == solved)
      end if
      UpdateFailLimit (failLimit)
   end while
   UpdateMovesLimit(maxLocalMoves)
end while
```

Note how a loop of "meta-restarts" is wrapped around our traditional approach now, whereby, at the beginning of every meta-restart, the fail limit for the tree-search procedure is reset to its original value. We introduce this variant so that we can distinguish between the effect of different restart strategies and the actual improvement caused by learning better value heuristics when experimenting with the following hybrid algorithm:

```
MRH: bool MetaRestartHybrid (void)
InitMoveLimit(maxLocalMoves)
while (true) do
   InitFailLimit(failLimit), InitRandom (heuristic), moves ← 0
   while (moves++ < maxLocalMoves) do
      status ← TreeSearch(failLimit,heuristic)
      if (status ≠ inconclusive) then
         return (status == solved)
      end if
      UpdateHeuristic (heuristic), UpdateFailLimit (failLimit)
   end while
   UpdateMovesLimit(maxLocalMoves)
end while
```

In every meta-restart, not only is the fail limit reset to the original start value, but also the heuristic is reset randomly. This results in shorter tree-searches and an effective restart of our local search for a good value selection heuristic. However, we do assume that global side-effects of TreeSearch are not affected by meta-restarts, i.e., if TreeSearch learns no-goods for example, then they are not lost by meta-restarts.

Numerical Results

We have introduced the idea of hybridizing complete and local search by using a random variable selection/restart approach augmented by a local search algorithm that learns better value selection heuristics over the course of several restarts. We outlined a basic hybrid algorithm (BH) and a variant with local search meta-restarts (MRH). In this section, we compare these two approaches with the corresponding traditional approaches (TR) and (MRT) on a number of applications modeled as satisfiability and constraint programming problems. Note that all four methods are complete and have the potential to solve critically constrained and over-constrained problem instances.

Diagonally Ordered Magic Squares

We start our experimentation on magic squares which represent nice under-constrained problems that are non-trivial for both complete and local search approaches. A magic square (Moran 1982) of order N is an N by N matrix with entries from 1 to N^2, such that the sum of the entries in each column, row, and the main diagonals is the same. In a Diagonally Ordered Magic Square (DOMS) (Gomes and Sellmann 2004) the entries on the main diagonals, when traversed from left to right, have strictly increasing values. To the best of our knowledge, no polynomial-time construction for DOMS is known.

We use a simple constraint programming approach to tackle this problem: There is one variable per cell with an associated domain of values between 1 and N^2, and all variables must take different values. The sum and ordering constraints are added to the model. We order the rows randomly and then traverse the square row by row choosing the left-most variable with the smallest domain as our branching variable. In our TR approach, the branching value is chosen randomly. The search proceeds in standard backtracking fashion ((Shaw 2005) suggests that fancier search meth-

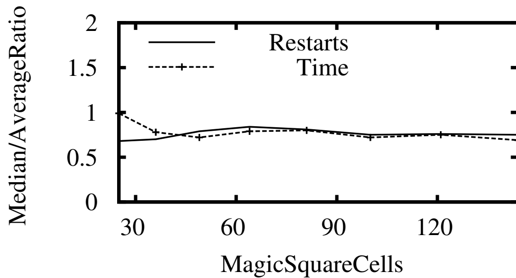

Figure 1: The ratio of average and median number of restarts and time used by method MRH for DOMS of orders 3 to 12.

ods like least discrepancy search do not perform better than depth-first search in restarting methods) and is interrupted when a fail-limit of $N * 80 + 20 * (restarts/5)$ is reached, where 'restarts' counts the number of restarts conducted so far. The MRT approach uses the very same setting, but after $N * 5 + 20 * (metaRestarts/3)$ have been conducted, 'restarts' is reset to zero, where 'metaRestarts' counts the number of such resets.

We compare these approaches with our MRH method where the value heuristic is chosen randomly at the beginning of each meta-restart and is updated according to the function given in the previous section after every (ordinary) restart. The algorithms were implemented in Ilog Solver 6.0 and experiments were conducted with an AMD Athlon 2 GHz Processor 3000+ with 1 GB of main memory.

Since we are dealing with randomized methods, we conduct 100 runs for each instance with magic square orders growing from 3 to 12. To assure that the restart strategies are actually doing their job, we first compare the median and average values of various parameters such as time, number of choice points, etc. For all methods and all such parameters we look at similar plots as the one shown in Figure 1. We see that the ratios of median over average of number of restarts and time are almost constant, which implies that the averages are well under control and do not grow arbitrarily as it would be typical for heavy-tailed distributions. Consequently, we conclude that the restarts are indeed helping us to avoid heavy-tailed runtime behavior.

Now, Figure 2 shows the average number of (ordinary) restarts needed by all three approaches when computing diagonally ordered magic squares of orders 3 to 12. We see how effectively our heuristic-learning algorithm can reduce

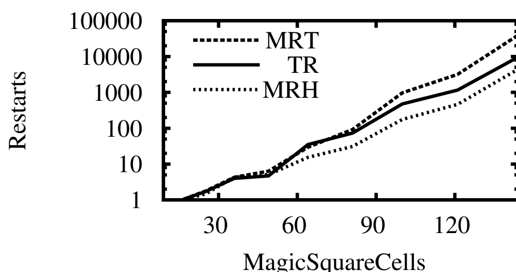

Figure 2: DOMS tackled with CP-based methods TR, MRT, and MRH. The picture shows the varying number of restarts [log-scale] needed by the different methods.

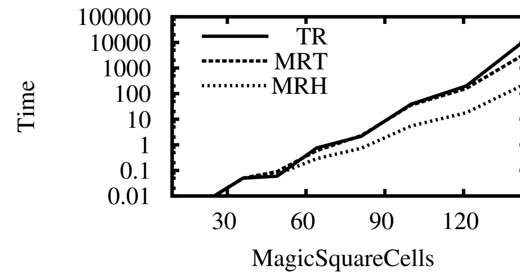

Figure 3: DOMS tackled with CP-based methods TR, MRT, and MRH. The picture shows the varying time [log-scale] needed by the different methods.

that number when comparing MRH and MRT that both follow the very same restart strategy. The average number of restarts is actually reduced by an order of magnitude. When comparing MRH with TR, on the other hand, we see that the reduction is not nearly that dramatic, the number of restarts is reduced by a factor of 2. However, we need to take into account that, in TR, the fail-limits grow linearly and that tree-searches become longer and longer over time. Consequently, when comparing the actual runtimes in Figure 3, we see that MRH outperforms both traditional algorithms by roughly a factor of 20 on DOMS of order 12.

We were curious to see how our new method would perform on more tightly constrained problems. Despite the great speed-ups that we were able to achieve, we would not have gained much if our improved performance on under-constrained problems would have to be paid for by an inferior performance on critically constrained and over-constrained problem instances where the true strength of complete search methods lies. Therefore, we conducted the same set of experiments on a related problem, namely that of computing Dumbledore Squares which are DOMS that hide a Latin square structure. Curiously, in (Gomes and Sellmann 2004) it was found that this special kind of DOMS could actually be solved faster than ordinary DOMS. Figure 4 shows that, for these tightly constrained problems, the performance of all three methods is essentially the same. It is not surprising that MRH does not yield any speed-ups for this problem that is hardly suited for local search approaches, but we are relieved to find that the frequent update of our value selection heuristic does not slow down the overall search either.

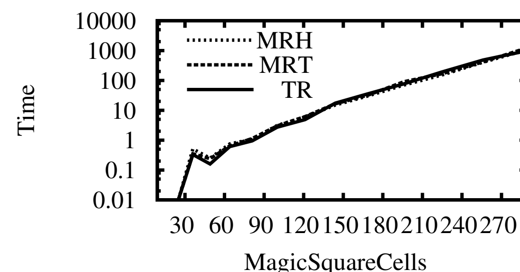

Figure 4: Dumbledore Squares tackled with CP-based methods TR, MRT, and MRH. The picture shows the varying time [log-scale] needed by the different methods.

v	cl	TR-exp	TR-linear	BH-exp	BH-linear
600	7000	29.3	18.1	4.37	2.6
600	7050	33	64	8.68	5.66
600	7100	46	88	18.72	6.54
600	7150	39.5	485	15.6	23
600	7200	171	847	92	78

Table 1: Median time [sec] for 5-SAT instances with 600 variables and number of clauses ranging from 7000 to 7200 (50 instances per data point).

v	cl	TR-exp	TR-linear	BH-exp	BH-linear
600	7000	29.38	18.1	4.37	2.6
660	7700	67	267	2.26	3.5
720	8400	98	874	9.4	7
780	9100	227	> 2000	31.9	35
840	9800	1062	> 2000	103	77

Table 2: Median time [sec] for 5-SAT instances at clause/variable-ratio 11.67 (50 instances per data point).

order	holes	TR-exp	TR-linear	BH-exp	BH-linear
38	1200	2.47	1.88	0.87	0.90
40	1200	1.36	1.26	0.77	0.68
42	1200	3.33	5.27	0.69	0.76
44	1200	21.29	15.76	5.3	4.11

Table 3: Median time [sec] for QCP instances with 1200 holes where order of the square ranges from 38x38 to 44x44 (100 instances per data point).

order	holes	TR-exp	TR-linear	BH-exp	BH-linear
44	1150	17.39	15.81	4.55	5.022
44	1200	21.29	15.76	5.3	4.11
44	1250	17.30	23.75	7.16	4.78
44	1300	19.5	21.44	4.54	2.83
44	1350	16.83	16.45	2.96	1.81
44	1400	15.69	31.01	3.23	2.19

Table 4: Median time [sec] for QCP instances of order 44 with the number of holes ranging from 1150 to 1400 (100 instances per data point).

Experimental Results on SAT Instances

After our encouraging results on problems that were modeled as constraint programs, we were curious to see how our method would perform on satisfiability instances. For our experiments, we decided to use MiniSAT (Eén and Sörensson 2003) as our "traditional solver" TR. It has been awarded several times at SAT competitions and its source code is publicly available.

MiniSAT is an interesting solver for us since, among other advanced methods like clause learning, MiniSAT takes advantage of randomized branching variable selection in combination with a restarting strategy. At the same time, the value selection heuristic is surprisingly simply: MiniSAT always tries setting the branching variable to 'false' first. Initially, the fail-limit is set to 100 and it is increased by a factor of 1.5 after every restart, i.e., the fail-limit grows exponentially over the number of restarts. We refer to the original MiniSAT solver as TR-exp and to our basic hybrid derived from MiniSAT as BH-exp. Obviously, an exponential growth of subsequent restarts is not preferable for our approach. Therefore, we also investigated variations of these solvers, TR-linear and BH-linear, where we increase the fail-limit after each restart by an increment of 100. Consequently, the fail-limits now grow just linearly.

The first set of SAT-related experiments was conducted on a set of randomly generated 5-SAT instances (where each clause consists of exactly 5 literals) that were produced by the generator from (Selman *et al.* 1992). This and all following SAT experiments were performed with an AMD Opteron 2 GHz Processor 248+ with 2 GB of main memory.

In Table 1, we give the median time in seconds needed by our four different solvers on 5-SAT instances with 600 variables and an increasing number of clauses. Like that, we hope to gain an idea how the solvers compare on increasingly constrained problem instances. First, we see that, for the pure MiniSAT solver, a linear restart strategy is only beneficial on rather under-constrained instances. As the instances become more and more constrained, MiniSATs exponential strategy is clearly the better choice. Now, when comparing against the respective versions of our hybrid solver, we see that learning favorable value selection heuristics is clearly a good idea: BH-exp and BH-linear are up to 7 times faster than TR-exp and TR-linear. However, as was to be expected, the runtime improvements become less prominent as the instances become more and more constrained.

What is curious here is that BH-linear actually outperforms BH-exp. Intuitively, it does make more sense to restart more rapidly if one would have expected to have found a solution already when the value heuristic is expected to work well. Also, faster restarts give more burden to the local search part of our hybrid algorithm which may be beneficial on under-constrained problems. However, this is speculation and more research is needed to explain this effect more fundamentally.

In order to see how the speed-ups scale with instance size, we conducted another set of experiments whose results are shown in Table 2. We fixed the clause/variable-ratio at 11.67 and increased the number of variables to 840 in increments of 60. We observe, the speed-ups gain in momentum for larger instances, and the hybrid algorithms outperform their traditional variants by an order of magnitude.

Finally, we wanted to investigate whether our method could also improve the performance on more structured problem instances. As our benchmark, we chose a set of SAT instances that model the Quasigroup Completion Problem (QCP): Given a square with N rows and columns, we are to fill the cells in each row with a permutation of the numbers 1 to N such that each column also shows a permutation of those numbers. This structure is known as a "Latin square". Now, in the QCP some of the cells in the square are already filled in, and we need to decide whether a completion to a full Latin square is possible or not. We use the QCP generator "lsencode" presented in (Kautz *et al.* 2001) in order to generate instances that exhibit a balanced structure of holes — which results in SAT instances that are known

to be much harder to solve than random instances with an arbitrary pattern of holes.

In Table 3, we report the median time in seconds needed by our four contestants on QCP instances with 1200 holes and varying order from 38x38 to 44x44. As we keep the number of holes fixed in this experiment, the instances become both larger and more constrained as we increase the order. We see that, for this problem instance, the hybrid methods still consistently outperform their traditional counterparts, even though the speed-ups are not quite as dramatic as we have seen them before.

The results of our final experiment are summarized in Table 4 where we vary the constrainedness of our problem instances again at a fixed size of order 44. Once more we see clearly that the speed-ups achieved by learning good value heuristics become larger the more under-constrained our instances get. Eventually, BH-exp outperforms TR-exp by a factor of 5, and BH-linear outperforms TR-linear by a factor of 10.

Conclusions

We introduced the idea to augment restarted randomized complete search algorithms with a simple local search procedure that learns favorable value-selection heuristics over the course of several restarts. In some sense this idea can be viewed as complimentary to the use of no-goods: the latter store information about parts of the search space that cannot contain solutions, whereas the value heuristics that we learn store information about promising parts of the search space. Experiments on constraint programming and satisfiability problems showed that the new method can lead to speed-ups of orders of magnitude on under-constrained problem instances.

Regarding future work, an anonymous reviewer suggested to leverage other ways to update the value heuristics. E.g., one could keep statistics how often a value is still in the domain of a variable after the fail limit is reached. With respect to domains of size larger than two, one could also keep an ordering of values for each variable rather than storing the first choice value only.

References

E. Balas and M. Carrera. A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, 44:875–890, 1996.

K. Boese, A. Kahng, S. Muddu. A New Adaptive Multi-start Technique for Combinatorial Global Optimizations. *Operations Research Letters*, 16:101–113, 1994.

N. Eén and N. Sörensson. An Extensible SAT-solver. *SAT*, 502–518, 2003.

S.L. Epstein, E.C. Freuder, R. Wallace, A. Morozov, B. Samuels ACE, The Adaptive Constraint Engine. *CP*, LNCS 2470:525–540, 2002.

H. Fang and W. Ruml. Complete local search for propositional satisfiability. *National Conference on Artificial Intelligence*, 161–167, 2004.

M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, B(98): 23–47, 2003.

F. Focacci, F. Laburthe, and A. Lodi. Local Search and Constraint Programming. `http://citeseer.ist.psu.edu/-focacci01local.html`, 2001.

M.L. Ginsberg, J.M. Crawford, D.W. Etherington. Dynamic Backtracking. `citeseer.ifi.unizh.ch/-ginsberg96dynamic.html`, 1996.

C.P. Gomes and M. Sellmann. Streamlined constraint reasoning. *CP*, LNCS 3258:274–289, 2004.

C.P. Gomes, B. Selman, N. Crato. Heavy-Tailed Distributions in Combinatorial Search. *CP*, LNCS 1330:121–135, 1997.

D. Habet, L.M. Chu, L. Devendeville, M. Vasquez. A Hybrid Approach for SAT. *CP*, LNCS 2470:172–184, 2002.

W.D. Harvey and M.L. Ginsberg. Limited discrepancy search. *International Joint Conference on Artificial Intelligence*, 607–613, 1997.

P. Hansen and N. Mladenovic. A tutorial on Variable Neighborhood Search. *Les cahiers du GERAD G-2003-46*, Universite de Montreal HEC, Montreal, Canada, 2003.

J.P. Hart and A.W. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

J.N. Hooker. A search-infer-and-relax framework for integrating solution methods. *International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, LNCS 3524:243–257, 2005.

Dynamic Restart Policies. H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, B. Selman. *National Conference on Artificial Intelligence*, 674–682, 2002.

H. Kautz and Y. Ruan and D. Achlioptas and C. Gomes and B. Selman and and M. Stickel. Balance and Filtering in Structured Satisfiable Problems. *International Joint Conference on Artificial Intelligence*, 193–200, 2001.

M. Luby, A. Sinclair, D. Zuckerman. Optimal speedup of las vegas algorithms. *Information Processing Letters*, 173–180, 1993.

J. Moran. The Wonders of Magic Squares. *New York: Vintage*, 1982.

L. Pitsoulis and M. Resende. Greedy randomized adaptive search procedures. *Handbook of Applied Optimization* 168–181, 2001.

S. Prestwich. Randomised Backtracking for Linear Pseudo-Boolean Constraint Problems. *International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, 7–19, 2002.

S. Prestwich. Stochastic Local Search in Constrained Spaces. *Practical Applications of Constraint Technology and Logic Programming*, 27–39, 2000.

B. Selman and H. Levesque and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. *National Conference on Artificial Intelligence*, 440–446, 1992.

P. Shaw. Comparison of Search Methods for the Talent Scheduling Problem. *Informs Proceedings*, 2005.

P. Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. *Technical report*, APES group, Department of Computer Sciences, University of Strathclyde, 1997.

H. Shen and H. Zhang. Another Complete Local Search Method for SAT. *Logic for Programming Artificial Intelligence and Reasoning*, 2005.

P. Van Hentenryck and L. Michel. *Constraint-Based Local Search*, The MIT Press, 2005.

A. Vidotto, K.N. Brown, J.C. Beck. Robust Constraint Solving Using Multiple Heuristics. *Irish Artificial Intelligence and Cognitive Science Conference*, 203–212, 2005.