Automated Negotiation for Package Delivery

Dave de Jonge and Carles Sierra
Artificial Intelligence Research Institute, IIIA-CSIC
Campus de la Universitat Autònoma de Barcelona
08193, Bellaterra, Catalonia, Spain
{davedejonge, sierra}@iiia.csic.es

***Abstract*—Package delivery companies compete with each other and have costumers spread over wide areas. We propose a negotiation algorithm that allows companies and individual postmen to negotiate over who delivers what package. This way, package delivery can be made more efficient, yielding a higher profit and/or lower costs for all parties. Our system does not force competing companies to co-operate, but proposes solutions that allow all parties to increase their individual profit.**

I. Introduction

Package delivery companies often operate in wide areas (such as entire countries or continents) that overlap with the areas of competing companies. Although this is convenient for customers that want to send packages over large distances, it is inefficient, since deliverers (postmen) may need to traverse unnecessary long distances. Efficiency would be improved if each postman could deliver all its packages in a small area.

Therefore, we propose an automated solution, that enables postmen to negotiate, in an automated way, with many other postmen at the same time over which package to deliver, in real time. This means that postmen can decide to exchange packages even when they are already on their way to deliver them. Our solution is decentralized and therefore scalable.

A possible alternative to automated negotiation would be to apply a Distributed Constraint Optimization Problem (DCOP) solver to find solutions that distribute the packages in a fair way. The problem with this however, is that fair solutions are not always feasible, since package delivery is a discrete domain so there might not be any solution that gives the same amount of profit to every party. Moreover, different companies might have different opinions about what can be considered 'fair'. This would make the package delivery companies distrust the system and could lead to conflicts.

Therefore, our proposed algorithm does not try to find a solution that is fair to everyone, but negotiates on behalf of one company (or one single postman), in order to maximize that company's profit. All the other companies are considered as opponents, and each one of them is responsible for accepting or rejecting possible solutions, depending on what they themselves consider profitable.

Our algorithm applies a Branch & Bound search through the space of all solutions, and estimates for every explored solution the utility gain of that solution to each negotiator. It then proposes some of the solutions it found according to a concession strategy.

Although automated negotiations have been studied extensively in the literature [1], [2], [3], [4], [5], most of these studies make unrealistic assumptions so they are not directly applicable to real-world problems. In particular, they cannot handle the following properties, which our algorithm does take into account:

- The utility functions are non-linear, hard to calculate and hard to invert.
- Solutions may involve many agents, including humans.
- The number of solutions is very large, so one cannot exhaustively explore the space of all solutions.
- The environment changes during the negotiations due to actions of others.
- Other agents in the system are unknown and cannot be blindly trusted.
- Decisions have to be made within limited time.

All of these features are indeed present in the package-delivery problem: the utility of delivering a package is a complex function of time and distance, and depends on the other packages to be delivered. The number of agents to negotiate with is large, as there can be many postmen working at the same time. The number of possible solutions is very large, as there are many ways to divide packages among postmen. The environment changes continuously, as new packages appear during the day and the postmen are moving around. The agents to negotiate with usually work for competing companies, thus we cannot blindly trust them. And finally, decisions should made quickly, as packages need to be delivered in time.

In the next section we present a simplified version of the package delivery problem. In Section III we explain our algorithm to solve the problem, in Section IV we show the results of our experiments, in Section V we explain how the problem can be adapted for more re-

alistic environments, and finally in VI we present our conclusions.

II. Negotiating Salesmen Problem

In order to do experiments with negotiation for package delivery we have defined a simplified problem which is an abstracted version of real-world package delivery. In section V we show how the problem can be adapted to represent the real world more realistically. The advantage of our simplified problem is that it allowed us to do experiments quickly to test our algorithm.

The problem we define here is a variant of the Traveling Salesman Problem (TSP) that we call the *Negotiating Salesmen Problem* (NSP). It resembles the multiple Traveling Salesmen Problem (mTSP) as described in [6], but with the difference that each agent in the NSP is only interested in minimizing its own path.

The idea is that several agents (the salesmen, or postmen) need to visit a set of cities. The salesmen all start in the same city[1], and all other cities should be visited by at least one agent. Initially, each city is assigned to one salesman that has to visit it. However, the salesmen are allowed to exchange some of their cities amongst each other, so that the agents can decrease the distances they have to travel. For example: if a city v is assigned to agent α, but α prefers to visit another city v' that is assigned to agent β, then α will propose β to exchange v for v'. If β however doesn't want v he will not accept this deal. And if no other agent wants to accept v either, then α is obliged to travel along city v. However, we impose the restriction that not all cities can be exchanged. This is because a postman might have some obligations that he cannot, or does not want to, transfer to anyone else. The cities that can be exchanged are called the *interchangeable cities*, while the cities that cannot be exchanged are the *fixed cities*.

The proposed exchanges of cities do not have to be bilateral, but may involve any number of salesmen.

In the following, all sets we mention are finite.

Definition 1: An instance of the NSP is a tuple $\langle G, v_0, A, F, I, \epsilon_0, t_{dead}\rangle$, which consists of: a weighted graph G, a marked vertex v_0 of the graph, a set of agents A, a set of fixed cities F, a set of interchangeable cities I, an initial distribution of cities ϵ_0 and a deadline t_{dead}. These components are further explained below.

G is a complete, weighted, undirected graph: $G = \langle V, w\rangle$ with V the set of vertices (the *cities*) and w the weight-function that assigns a cost to each edge: $w : V \times V \to \mathbb{N}$ such that it satisfies the triangle inequality[2]:

$$\forall a, b, c \in V : \quad w(a, c) \leq w(a, b) + w(b, c)$$

One of the vertices is marked as the *home city*: $v_0 \in V$. Each agent has to start and end its trajectory in this city. We use the symbol $\overline{V}$ to denote the set of *destinations*, that is: all cities except the home city: $\overline{V} = V \setminus \{v_0\}$. The set of destinations is partitioned into two disjoint subsets: F and I, so: $\overline{V} = F \cup I$ and $F \cap I = \emptyset$. They are referred to as the set of *fixed cities* and the set of *interchangeable cities* respectively.

The set of agents is denoted[3] by $A = \{\alpha, \beta, ...\}$. Each destination is initially assigned to an agent, by the function $\epsilon_0 : \overline{V} \to A$. We use the symbol $\overline{V}_i$ to denote the subset of $\overline{V}$ consisting of all cities that are assigned by ϵ_0 to agent i. $\overline{V}_i = \{v \in \overline{V} |\ \epsilon_0(v) = i\}$. $\overline{V}_i$ is referred to as agent i's set of preassigned cities. The definitions above imply that for each agent its set of preassigned cities can be further subdivided into: $\overline{V}_i = F_i \cup I_i$ where F_i is $\overline{V}_i \cap F$ and I_i is $\overline{V}_i \cap I$.

Finally, the instance includes a real number t_{dead} that represents the deadline for the negotiations. Agents are allowed to negotiate over the assignment of cities, until this deadline has passed.

Definition 2: A solution of an instance of NSP is a tuple $\langle \epsilon_s, T\rangle$ in which ϵ_s is a distribution of cities: $\epsilon_s : \overline{V} \to A$ such that the restrictions of ϵ_0 and ϵ_s to F are equal: $\forall v \in F : \epsilon_0(v) = \epsilon_s(v)$. $T = (T_\alpha, T_\beta, ...)$ is a tuple of finite sequences of cities, one for each agent, such that for each agent i, T_i contains v_0 and all cities in $\overline{V}'_i$, with $\overline{V}'_i = \{v \in V | \epsilon_s(v) = i\}$.

This means that in the solution, the cities are distributed between the agents according to ϵ_s, but the fixed cities F are still assigned to their original owners. So in the solution, the cities are redistributed: $\overline{V} = \overline{V}'_\alpha \cup \overline{V}'_\beta \cup ...$, but the fixed cities are not: $\overline{V}'_i \cap F_i = F_i$. A sequence T_i of the solution represents a cycle in the graph that starts and ends in v_0 and that passes all vertices in $\overline{V}'_i$. For each agent we then have a cost: $c(T_i) \in \mathbb{N}$. If T_i is given by $T_i = (v_0, v_1, v_2, ...v_k)$, then $c(T_i)$ is defined as:

$$c(T_i) = w(v_k, v_0) + \sum_{j=1}^{k} w(v_{j-1}, v_j) \qquad (1)$$

By definition, an agent i prefers a cycle T_i^1 over a cycle T_i^2 if and only if $c(T_i^1) < c(T_i^2)$. We assume all agents

[1]This assumption was made to stay close to the original mTSP, but should be dropped in realistic domains. See also Section V.

[2]The assumptions of completeness and triangle inequality are made to simplify the problem. A problem instance without these properties can be converted into an equivalent instance that does satisfy them, so these assumptions are made without loss of generality.

[3]We use Greek letters as the *names* of specific agents, while we use Latin letters as variables that can refer to any undetermined agent. The letter ϵ however does not represent an agent, because it is reserved for assignments of cities.

are rational and therefore a solution is only feasible if for each agent the cost of the solution is less then the cost it would incur from the original distribution of cities ϵ_0.

III. NB^3: Negotiation Based Branch and Bound

Branch & bound (BB) is a general algorithm to find optimal solutions in discrete domains [7]. The objective of a BB algorithm is to find a solution $x \in S$ to a problem that minimizes a given cost function $f : S \to \mathbb{R}$. The algorithm incrementally generates a tree where nodes represent sets of solutions in S. The algorithm splits the set of solutions represented by a node into a number of subsets $(S_1, \ldots, S_i, \ldots, S_k)$ that become the children of the node. For each node an upper bound and a lower bound for the values of f on the elements of S_i is calculated. When the lower bound of a node is higher than the upper bound of another node, then the former node can be ignored (*pruned*) as it will not contain the optimal solution.

In our problem however, there is not one single cost function, but a *set* of cost functions, one for each salesman in the NSP. Each agent is only interested in minimizing its own cost function and needs to negotiate with the other agents in order to achieve this. We propose an algorithm that is run by such an agent and that applies BB to explore the search space.

A. Actions and Plans

We assume a number of agents $A = \{\alpha, \beta, \ldots, \}$ an initial world state $\epsilon_0 \in \mathcal{E}$, and a set of actions $\mathcal{O}$ that these agents might execute. Each agent $i \in A$ can choose its actions only from a certain subset $\mathcal{O}_i$ of $\mathcal{O}$. Therefore we have $\mathcal{O} = \bigcup_{i \in A} \mathcal{O}_i$. We will use the notation $\overline{\mathcal{O}}_i$ to refer to the set of actions that all agents different from i might perform $\overline{\mathcal{O}}_i = \mathcal{O} \setminus \mathcal{O}_i$.

In the NSP, a world state is an assignment of interchangeable cities to agents: $\epsilon : I \to A$. An action $ac \in \mathcal{O}$ is a triple (i, v, j) with $i, j \in A, i \neq j$ and $v \in I$, representing an agent i giving a city v to another agent j. The set of actions $\mathcal{O}_j$ that agent j can execute consists of the actions in which a city is given to j

$$\mathcal{O}_j = \{(i, v, j) \in A \times I \times \{j\} | i \neq j\}$$

Note that such an action increases the cost of j and so it is up to j to decide whether or not it agrees with this. Therefore, we consider it as being executed by j rather than by i.

The execution of an action on a world state results in a new world state: $ac(\epsilon) = \epsilon'$. A set of actions, that is, a joint plan, $p \subseteq \mathcal{O}$ acts on a world state ϵ by letting all the actions $ac \in p$ act on ϵ (note that in the NSP the order of execution of the actions is irrelevant).

In the NSP, the execution of an action $ac = (i, v, j)$ means that the city v is re-assigned to agent j:

$$\epsilon'(v) = j \text{ and } \forall u \in \overline{V} \setminus \{v\} : \epsilon'(u) = \epsilon(u).$$

Each agent i aims to minimize its own cost function $f_i(\epsilon)$, which means they have conflicting interests. Agents therefore need to negotiate to find compromises. In the NSP $f_i(\epsilon)$ is the length of the shortest path through all cities assigned to i under assignment ϵ.

Not every plan is feasible. In the NSP, a plan is only feasible if a city is not given away twice, and cities are only given away by their respective owners. The set of all feasible plans in world state ϵ is denoted as *fea*, with: $p \in fea(\epsilon) \Leftrightarrow \forall (i, v, j) \in p : \epsilon(v) = i \wedge \forall (i, v, j), (i', v', j') \in p : v \neq v'$. We next explain the different components of NB^3 from the perspective of agent α. The other agents might also run a copy of NB^3, or any other negotiation algorithm, or they might even be human.

B. The Search Tree

An agent that runs the NB^3 algorithm builds a search tree that is explored according to a best-first strategy. Each arc between a node and its parent is labeled by a certain action. Every node n then represents a partial plan $path(n)$ that consists of all the actions along the path from the root to n. Equivalently, every node represents the set of all world states S_n that can be realized when the actions in $path(n)$ are executed.

For each node generated, the agent determines whether the corresponding partial plan is good enough to be proposed, taking the cost for himself into account, as well as the costs for the other participating agents. While waiting for the acceptance of issued proposals, NB^3 keeps on expanding the tree in search for alternative plans in case the proposed plan is not accepted. The question of whether a plan is considered 'good enough' depends on a time-dependent concession strategy. The closer we are to the deadline, the more the agent is willing to concede.

Note that applying BB makes the algorithm especially useful in non-linear domains, because for each plan that it explores it calculates the cost of that specific plan, rather than simply calculating the cost for each action individually and then summing them, as one could do in a linear domain.

C. Bounding

When applying BB to negotiations, it is required that each node n maintains bounds for the cost functions f_i of *every* agent. NB^3 therefore always needs to have a model[4] of these cost functions, and uses it to calculate

[4]For the NSP it is easy to make such a model because the cost of an agent is simply its path length. For more realistic scenarios, see Section V

for every agent $i \in A$ the following quantities (we assume the current state of the world is ϵ_0 and we define $\epsilon_1 = (path(n))(\epsilon_0)$):

A **global upper bound**: gub_i. The value of the best environment agent i can guarantee himself without cooperation from any other agent, given the current state of the world.

$$gub_i = \min_{p_1 \subset \mathcal{O}_i} \max_{p_2 \subset \overline{\mathcal{O}}_i} \{f_i((p_1 \cup p_2)(\epsilon_0)) \mid p_1 \cup p_2 \in fea(\epsilon_0)\}$$

For each node n an **intermediate value**: $e_i(n)$. The cost agent i incurs if exactly the actions in the plan $path(n)$ are executed and no other actions.

$$e_i(n) = f_i(\epsilon_n)$$

For each node n a **lower bound**: $lb_i(n)$. The minimum cost that i will incur given that the plan $path(n)$ is executed.

$$lb_i(n) = \min_{p_1 \subset \mathcal{O}_i} \min_{p_2 \subset \overline{\mathcal{O}}_i} \{f_i(p_1 \cup p_2(\epsilon_n)) \mid p_1 \cup p_2 \in fea(\epsilon_n)\}$$

These values cannot always be calculated exactly, because α might not have complete knowledge of the current world state or of the other agents' cost functions f_i and because the time restrictions might make it impossible for α to compute these quantities exactly. Therefore, α can only approximate them.

The global upper bound acts as the 'reservation value': an agent would never accept any deal that yields a cost higher than its global upper bound. The intermediate value of a node is the value that the agent would get if the actions in the path from this node to the root node are executed. So if $e_i(n) > gub_i$ the plan corresponding to node n is not profitable for agent i.

D. Searching and Pruning

When a plan p proposed by α is accepted by β, agent β commits itself to execute his part of the plan, that is: the actions $p_\beta = p \cap \mathcal{O}_\beta$. All actions that are incompatible with those in p_β then become unfeasible, so α can prune every node that has any of these actions in its path to the root.

Since NB^3 performs a best-first search, it uses an *expansion heuristic* h to rank the nodes: $h(n) \in \mathbb{R}_{\geq 0}$. In each cycle of the algorithm the node with the highest expansion heuristic is chosen to split.

We do not have the space here to give a full description of the calculation of the expansion heuristic, but the main idea is that we estimate the likelihood that a node will generate a child node representing a plan that would be accepted by the other agents. We calculate the likelihood that agent β will accept a plan by assuming it is 0 when the cost for β is higher than gub_β, and 1 when the cost is less than the highest cost β has already offered to pay. Furthermore, we assume this likelihood increases linearly when the cost is in between these two values.

The lower bound $lb_i(n)$ is the lowest cost agent i could possibly achieve in any descendant of node n. This means that if $lb_i(n) > gub_i$, any plan that extends $path(n)$ will be unprofitable for i. Therefore, it is useless to further expand n so the expansion heuristic $h(n)$ for such a node should be zero.

For more details on this we refer to [8].

We like to stress here the importance of the fact that the offers made by the other agents influence the expansion heuristic. If β concedes, it becomes more attractive for α to explore plans in which β is involved. This means that search and negotiation are intimately intertwined with each other; search influences negotiation and vice versa. This is a key point of NB^3 and forms a major difference with other existing negotiation algorithms.

IV. Experimental Results

We here present some results we have obtained by applying NB^3 to the NSP. Ideally, we should test our algorithm against other negotiation algorithms. However, as far as we know there are no algorithms developed that can handle the kind of domain we are considering. Therefore, in Section IV-C we test NB^3 by letting some agents running NB^3 negotiate with agents running a random search. Also, in order to see how the algorithm scales with increasing problem size, we have done some experiments in which *all* agents were running NB^3.

A. Evaluating Results

The NSP instances used for our experiments were created by choosing random points in a 2-dimensional Euclidean plane. Each such point represents a city and for each pair of cities the distance between them is given by their Euclidean distance. The cities were randomly divided among the agents. In all experiments each agent was assigned one fixed city.

For each run we stored for each agent the set of cities it initially owned, and the set of cities it owned when the deadline for negotiations had passed. The shortest paths through these sets of cities were then calculated by the *Concorde TSP Solver* [9]. The length of the shortest path through the initial set of cities owned by agent i is denoted as C_i^{in}, while the length of the shortest path through the final set of cities owned by agent i is denoted as C_i^{fin}. Our performance measure is then defined as the percentual cost reduction averaged over all agents:

$$Q = \frac{100}{|A|} \sum_{i \in A} \frac{C_i^{in} - C_i^{fin}}{C_i^{in}} \tag{2}$$

All results in this section were averaged over 25 runs, each with a different instance of the NSP.

The experiments were conducted on a computer with a Pentium 4, 3GHz CPU. The agents were implemented as Jade agents [10].

B. Varying the Complexity

To see how the algorithm scales with the complexity of the problem, we have performed six tests with different numbers of agents and 10 interchangeable cities per agent. Also we did six tests with 10 agents each but with different numbers of interchangeable cities assigned to the agents. The results are presented in Figure 1. From

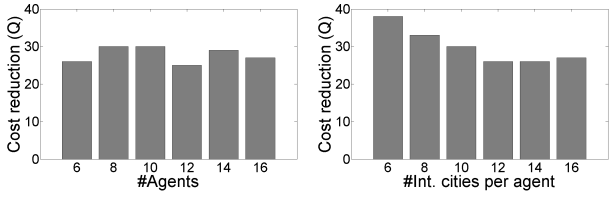

Figure 1. Cost reduction as a function of the number of agents and of the number of cities.

the graph on the left we see that *the results do not get any worse as the number of agents increases*. Apparently the increased complexity of the problem is compensated by the increased computing power resulting from the larger number of agents and the fact that an agent can profit from the plans discovered by other agents. From the graph on the right, we see that when the number of cities increases the results become worse, but the decrease is relatively small. This suggests that *the expansion heuristic successfully manages to limit the number of redundant nodes that are explored*, as it is supposed to do.

C. Comparison with Random Search

We now compare NB^3 against random search. That is: we let some agents running NB^3 (the "smart agents") negotiate with a number of agents (the "dumb agents") running an algorithm equivalent to NB^3, except that the expansion heuristic for each node was replaced by a random number.

We did four tests. Each test involved 10 agents, but for each test the number of dumb agents among those 10 was different. The results are presented in Figure 2. The graph on the left shows the average scores of the dumb agents and the one on the right shows the average scores of the smart agents. As expected the average result over all agents becomes worse as the number of dumb agents increases. Also, we conclude that *the smart agents perform significantly better than the dumb agents*. Since the smart agents are able to find better plans they have more bargaining power, and are thus able to exploit the dumb agents.

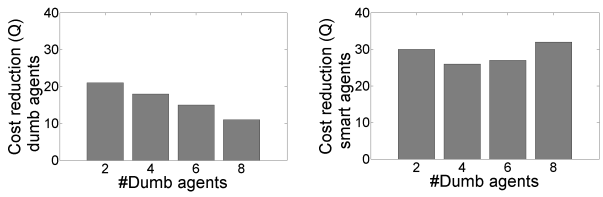

Figure 2. Cost reduction of dumb agents and smart agents as a function of the number of dumb agents.

V. Realistic Package Delivery Problems

The NSP is a highly simplified version of real package delivery. In order to apply NB^3 to real-world package delivery, we need to take a lot more factors into account. Some of these however can be easily incorporated into the NSP without changing the original problem much.

A. Utility

In the real world the cost of a postman is not simply given by the length of its trajectory, but rather by the financial cost of traversing its trajectory, which depends on the amount of gas used and the presence of toll booths. Taking this into account however does not change the essence of the NSP, since we could simply re-interpret the weights of the edges of the graph as the financial cost associated of traversing them. Of course, the true cost might not only be the financial cost, but could also include time, so we could define weights of the graph as a linear combination of time and money.

Furthermore, the postman may not receive the same amount of money for each package it delivers. Again, this is not a problem, since one could simply assign the value of a package to its destination vertex. This value is then subtracted from the cost of a path that passes this vertex.

B. Locations

To interpret the NSP as a real world problem, we should interpret the cities as generic locations, rather than real cities. In reality one can identify an infinite number of locations, but this can be overcome by only considering the destinations of packages that are currently in process to be delivered, the current locations of the postmen, and the locations of the post offices. This set of locations is dynamic, as clients might request the delivery of new packages at any moment.

Also, to apply our system to the real world we should drop the assumption that all agents start in the same home city, as real postman are spread out across their area of operation. The initial location of each postman is then simply its current location at the time of negotiations. Since the algorithm is running continuously, it should regularly update the positions of the postmen.

C. Constraints

Furthermore, one has to take into account that postmen are subject to certain constraints. For example, there is a maximum on the number of packages it can carry at the same time, depending on their weights and sizes. Moreover, postmen are constrained in the amount of time they can (or want to) work.

D. Non-linearity

Although the NSP is already a non-linear problem, since the length of a trajectory is a non-linear function of the co-ordinates of the visited cities, we encounter yet another form of non-linearity when we are dealing with real-world problems. This is because, given the weight of a certain trajectory, the true utility of this trajectory might be a non-linear function of this weight.

For example: a postman might be willing to work 8 hours a day for 80 Euro, but not be willing to work 12 hours a day for 120 Euro, because the 4 extra hours are much more tiresome then the first 8 hours were, so the postman demands a higher salary per hour for these extra hours. Utility is then a non-linear function of time and money.

Although this cannot be incorporated easily into the NSP, it is still not a problem for NB^3, as long as we have some means of aggregating the time-cost, the financial-cost and the constraints into a single utility value.

E. Utility Learning

Finally, we should take into account that a postman has emotions that determine how much it values time and money. This makes it almost impossible to find an exact utility function that expresses the postman's preferences. We are therefore planning to develop a system in which the user can express its preferences, in a discrete and qualitative way, from which the system can approximate a utility value. A postman could pre-define some of its preferences, but could also refine its preference representation during the negotiations. The system can suggest several solutions to the user, who may then react by indicating which of them it prefers. The system can use this information to dynamically learn and adapt a representation of the user's preferences. Since this happens in real time, the learned preference profile might even reflect the current emotional state of the user.

VI. Conclusions

We have introduced an algorithm called NB^3 to improve the efficiency of package delivery. It enables package delivery companies to negotiate over who delivers which package, but without neglecting the fact that companies are competitors and are therefore only willing to co-operate if it yields them an increase of profit.

We have introduced a test case problem called NSP to test our algorithm and we have performed some experiments with this problem. From their results we conclude that our algorithm decreases the costs of the postmen and scales well with increasing problem size.

Finally, we have explained how the NSP and NB^3 should be adapted in order to make them suitable for real-world package delivery.

VII. Acknowledgments

Supported by CHIST-ERA project ACE and the Spanish Ministry of Education and Science TIN2010-16306 project CBIT.

References

[1] N. Jennings, P. Faratin, A. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge, "Automated negotiation: Prospects, methods and challenges," *International Journal of Group Decision and Negotiation*, vol. 10, no. 2, pp. 199–215, 2001.

[2] G. Lai, K. Sycara, and C. Li, "A decentralized model for automated multi-attribute negotiations with incomplete information and general utility functions," *Multiagent Grid Syst.*, vol. 4, pp. 45–65, January 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1378675.1378677

[3] P. Faratin, C. Sierra, and N. R. Jennings, "Using similarity criteria to make negotiation trade-offs," 2000, pp. 119–126.

[4] ——, "Negotiation decision functions for autonomous agents," *Robotics and Autonomous Systems*, vol. 24, no. 3-4, pp. 159 – 182, 1998, multi-Agent Rationality. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889098000293

[5] T. Baarslag, K. Hindriks, C. M. Jonker, S. Kraus, and R. Lin, "The first automated negotiating agents competition (ANAC 2010)," in *New Trends in Agent-based Complex Automated Negotiations, Series of Studies in Computational Intelligence*, T. Ito, M. Zhang, V. Robu, S. Fatima, and T. Matsuo, Eds. Springer-Verlag, 2010.

[6] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, June 2006.

[7] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[8] D. de Jonge and C. Sierra, "Branch and bound for negotiations in large agreement spaces," IIIA-CSIC, Bellaterra (Barcelona), Spain, Tech. Rep., 2012. [Online]. Available: http://www.iiia.csic.es/files/mon_library/TR-IIIA-2012-01.pdf

[9] D. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, "http://www.tsp.gatech.edu/concorde," 2012.

[10] Jade, "Java agent development framework, http://jade.tilab.com," 2012.