# In-Time Agent-Based Vehicle Routing with a Stochastic Improvement Heuristic

**Robert Kohout and Kutluhan Erol**

Intelligent Automation, Inc
2 Research Place
Rockville, Md. 20850
{kohout,kutluhan}@i-a-i.com

**Abstract**

*Vehicle routing problems (VRP's)* involve assigning a fleet of limited capacity service vehicles to service a set of customers. This paper describes an innovative, agent-based approach to solving a real-world vehicle-routing problem embedded in a highly dynamic, unpredictable domain. Most VRP research, and all commercial products for solving VRP's, make a static-world assumption, ignoring the dynamism in the real world. Our system is explicitly designed to address dynamism, and employs an *in-time* algorithm that quickly finds partial solutions to a problem, and improves these as time allows. Our fundamental innovation is a stochastic improvement mechanism that enables a distributed, agent-based system to achieve high-quality solutions in the absence of a centralized dispatcher. This solution-improvement technology overcomes inherent weaknesses in the distributed problem-solving approach that make it difficult to find high-quality solutions to complex optimization problems. In previous work on similar problems, the MARS system of Fischer and Müller, *et al*., achieved an average route performance of roughly 124% of Solomon's algorithm for a VRP problem, which is known to achieve results that average roughly 107% of optimal. Our algorithm produces routes that average 106% those produced by an adaptation of Solomon's algorithm to a more general problem.

## The Pickup and Delivery Problem with Time Windows

*Vehicle routing problems (VRP's)* involve assigning a fleet of limited capacity service vehicles to service a set of customers. These problems have been well studied in the Operations Research literature (e.g.,Laport 1992.) Insofar as the Travelling Salesman Problem is embedded in virtually every VRP of practical interest, this entire class of problems is intractable. The vast majority of vehicle routing research addresses a static problem in which all of the relevant data is known in advance. Typically, service vehicles are assumed to start at some initial location (the *depot*), all customer demands and constraints are known, as are the distance and travel times between each pair of customers, and between each customer and the depot. There are a number of commercial systems for solving VRP's, and like the academic literature, all of them make a static-world assumption, ignoring the naturally occurring dynamism in the world. Customers can be late, or fail to appear. Traffic can be delayed for a variety of reasons. The time required to service a customer can be highly varying and unpredictable.

This paper describes an innovative approach to solving a vehicle routing problem embedded in a highly dynamic, unpredictable domain. It employs an *in-time* algorithm that quickly finds partial solutions to a problem, and improves these as time allows. In an online application for a dynamic environment, the problem itself may change over time, and the results need not be delivered all at once. "*In-time"* captures the essence of our algorithm in two ways: it is designed to run online in a dynamic world (i.e. in "real" time), and it returns partial results in time to execute them.

This work was conducted by the authors at Intelligent Automation, Inc. (IAI) as part of an effort to construct agent-based optimization systems for online problem solving. It was developed in collaboration with an actual airport shuttle company, and was designed to solve the Airport Shuttle Scheduling Problem, in which a fleet of limited capacity *shuttle* vehicles must be dispatched to service customers arriving at and departing from a set of regional airports. The goal of the system is to find a schedule of customer service times that minimizes the number of vehicles required to service a set of customers, with a secondary goal of minimizing the total time required to do so, while observing travel time constraints, vehicle capacity constraints and customer service time constraints. Our fundamental innovation is a stochastic improvement mechanism that enables a distributed, agent-based system to achieve high-quality solutions in the absence of a centralized dispatcher.

The airport shuttle business is highly dynamic. Customers can cancel and flights can be delayed. Traffic congestion varies routinely with the time of day, and less predictable delays due to accidents or weather are routine. Drivers get lost, and customers fail to appear as expected. Baggage delivery times vary widely by airline, time of day, and customer. Standard industry practice is to require a reservation 24 hours in advance, but this requirement is routinely waived, particularly for passengers arriving at an airport and requesting service "at the curb."

Shuttle companies typically require departing customers

to be available for shuttle pickup a minimum of two hours prior to the departure of their flight, but this time varies from company to company, and even between customers. There is generally no guarantee on how long an arriving customer's return from an airport may take, but shuttle companies have a long-term interest in ensuring that it is reasonable. The static problem, in which all of the relevant data is known is advance, is a specific instance of what the routing literature refers to as either the *Pickup and Delivery Problem With Time Windows (PDPTW),* or sometimes as the *Dial-a-Ride Problem*. A formalization of this problem can be found in (Dumas, Desrosier and Soumis 1991).

We know of no commercially available products for solving the static PDPTW, let alone the more realistic dynamic case that an airport shuttle company would require. However, there are a number of commercial products for solving the static version of a similar, related problem: the *Vehicle Routing Problem with Time Windows (VRPTW.)* In the PDPTW, customers have two associated locations: a pickup point and a delivery point, and time windows associated with each service location. In the VRPTW, there is a single service location, and one associated time window. Unlike the PDPTW, the VRPTW has been extensively studied, and optimal solutions are known for problem instances with as many as 100 customers. Most of the commercial products for vehicle routing are based upon Solomon's insertion heuristic (Solomon, 1987), which has demonstrated the ability to find high-quality solutions in a relatively short amount of time. We will discuss this algorithm in greater detail below.

## An Agent-Oriented Approach to Online Optimization

In order to address the dynamism of the airport shuttle application, we decided to design a system built around a set of cooperating software *agents*, each of which represents the interests and behavior of entities in the domain. These agents are organized into *contract nets* (Davis & Smith, 1983) that support rational, market-based assignment of resources. The use of agents for such problem-solving purposes has a number of potential advantages. Isolating system capabilities into independent processing units provides the foundation for distributing them over a large network of computers, thus allowing considerable computing power to easily be brought to bear on a given problem. Much of the promise of agent-based systems comes from the fact that software may be designed as individual capabilities and integrated into larger systems without having to reason about all of the ways in which a new capability may impact the behavior of the various other components of the system. Control is localized, and each agent can be designed to act independently, maximizing its own individual utility function without having to reason about the operation of the entire system.

While this localization of control simplifies the design of individual agents, it greatly complicates the problem of achieving high-quality global solutions. In previous work on agent-based vehicle routing systems, (Fischer, Müller, and Pischel, 1996) describe MARS (the *Modeling a Multi-Agent Scenario for Shipping Companies* system), which uses a hierarchical multi-agent system to solve the VRPTW. In OR parlance, MARS uses a heuristic local assignment procedure, followed by a 2-phase post-optimization process. Since both stages of post-optimization modify a schedule only if there is a net gain in global utility, the post-optimization (a.k.a. *local improvement)* qualifies as a hill-climbing algorithm. As noted in (Bentley, 1992) and elsewhere, such local improvement techniques are very sensitive to the quality of the initial solution, since they are only able find the local minimum of the solution space in which they begin their search. Thus, while MARS has a sophisticated market-based improvement algorithm that is well suited to agent-based routing applications, the crucial heuristic for determination of the initial solution set is less developed. (Fischer, Müller and Pischel, 1995) present a comparison of MARS with the known optimal results for 25-customer routing problems. In these cases, MARS achieves an average of roughly 133% of optimal. Solomon's algorithm is known to achieve an average of 107% of optimal prior to post-optimization, and 103% of optimal after post-optimization (Potvin and Rousseau, 1993). If we assume that the agent-based improvement mechanism is effective, MARS must be doing a poor job of initial customer insertion, and yet this is exactly the area that previous research indicates is crucial for obtaining high quality results. The significant advantages of distributed, agent-based systems for use dynamic application environments do not justify poor global performance, especially since, as we show in this paper, it is possible to obtain high-quality results in such an online control system.

## Multi-Agent Architecture

The architecture of our entire system is shown in Figure 1, below. Each pending group of customers[1] is represented by a *customer agent*, and each available vehicle is represented by a vehicle agent. There is no centralized scheduler, and even the GUI's were designed to run as agents under Cybele, our software infrastructure for agent-based applications. The route/address server agent performs address lookup and verification tasks, as well as the point-to-point routing required as part of the scheduling process.

When a customer requests service, the relevant information regarding number of passengers, itineraries, service address, etc. is entered via the GUI. At this time,

[1] Throughout the rest of this paper, references to "customers" will refer to groups of one or more persons who have indicated a desire to travel together, and share all relevant deadlines and constraints. We assume that no such group is larger than the maximum vehicle capacity, and our algorithm makes no attempt to split groups of more than one into smaller "lots."

the customer's address is verified with the route/address server, to ensure that the routing system can find the service address in its database. Once all this information is obtained, the routing system creates a customer agent, that will be responsible for seeing that the associated customer is serviced.

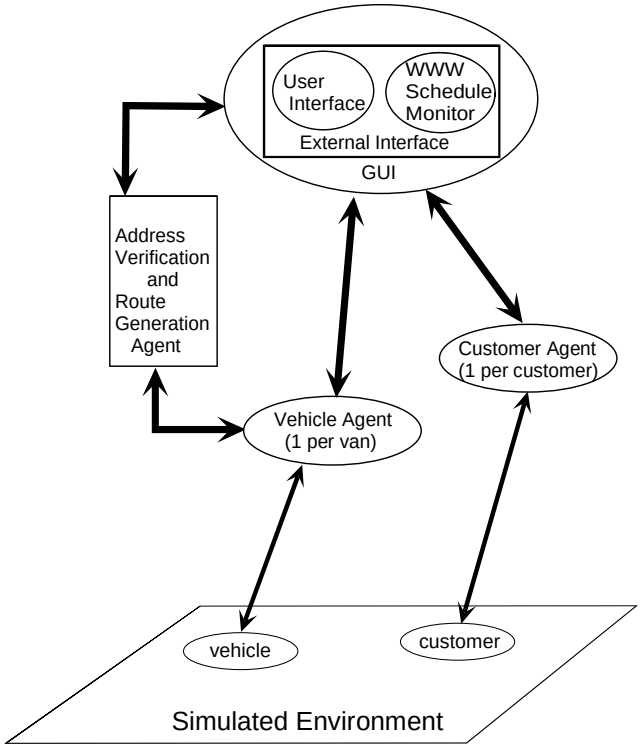


**Figure 1 Routing System Architecture**

The customer agent announces itself, and all vehicle agents compute the cost of carrying that customer with respect to their current schedules, and send these as quotes to the customer. Once a customer agent has received all of the expected quotes, it selects the low bidder and requests service from the associated vehicle agent. If the vehicle agent can still service the customer at the quoted price, the customer is inserted into the vehicle's schedule, and the customer agent is informed that the contract has been accepted. If the vehicle agent can no longer route the customer at the quoted price, presumably because another customer has been scheduled since the quote was sent, the vehicle agent returns a message indicating its new best price, and the customer agent repeats the process until it is scheduled.

## The Algorithm

Our vehicle agents make scheduling decisions by using Solomon's insertion heuristic, adapted for the PDPTW. For this reason, and because we use a centralized implementation of this same algorithm as the basis for evaluating our improvement technique, we summarize Solomon's algorithm as Algorithm 1. If $d_{ij}$ is the travel time between customers $i$ and $j$, and $b_j$ is the time scheduled to begin service at customer $j$, then Solomon's cost heuristic defines the cost of inserting customer $u$ between adjacent customers $i$ and $j$ in a route R as

$$H = (d_{iu} + d_{uj} - \mu d_{ij}) + \Delta b_j$$

Where $\mu$ is a non-negative parameter, and $\Delta b_j$ *is* the change in service time at customer $j$.

```
Let L be the list of unscheduled customers
R = new vehicle route

While there are still unassigned customers in L
  Do {
   For each unassigned customer C Do
     For each potential insertion position in R
     Do {
       If C can feasibly be served in this
          route position Then
          {
          Compute H = Solomon's heuristic value
          If  H is minimal Then {
            save H along with the
            corresponding position in the route
            }
         }
     }
   Assign customer with minimum H value to the
   Corresponding insertion position in R,
   and remove that customer from L.

   If NO Customer could be feasibly served in
    the current schedule Then
     {  /* start a new route */
     Save R
     R = new route
    }
  }
```

**Algorithm 1 - Solomon's Algorithm**

Recall that this algorithm was developed to solve the VRPTW. In order to apply it to the PDPTW, our modified algorithm first locates a slot where it is feasible to insert the pickup position, and then searches forward through a vehicle's schedule for a feasible slot in which to insert the delivery point, while accounting for the implications of pickup insertion. The ultimate cost of a customer's insertion into a schedule is a weighted sum of the costs of pickup insertion and delivery insertion.

Our agent-based insertion algorithm is summarized below as Algorithm 2. This algorithm uses the exact same cost metric and low-level data structures as our implementation of Solomon's heuristic for the PDPTW. The two main differences are that it builds the routes for the various vehicle agents in parallel, and schedules each customer in the order that they are presented to the system. Solomon's heuristic builds routes one at a time. Before inserting a customer into a route, it iterates through *all* of the remaining customers, and greedily chooses the cheapest remaining customer for insertion into the current

route. Only after no remaining customer can be inserted into a route does it initialize a new one.

The effect of the outer loop of Solomon's algorithm is to impose an ordering on the set of customers. If customers were presented to our agent-based system in the exact same order determined by this process, it would produce identical results. However, as we show below, in the absence of local improvement, our agent-based algorithm does quite poorly when customers are scheduled in a random order.

```
Create a vehicle route
While unscheduled customers remain Do
   Get the cost of inserting the next customer
   in each of the active vehicle routes

    If the customer cannot be inserted into any
    Vehicle Agent's Route Then
       Start a new Vehicle Agent and Route
    Else
       Insert the customer into the lowest cost
       Vehicle route
```

**Algorithm 2 -Abstract Agent-Based Insertion Algorithm**

We have developed a stochastic improvement heuristic that overcomes the order-sensitivity of the insertion algorithm. This improvement technique permits the system to converge to high quality solutions, in the absence of any centralized control, and without requiring any agents to maintain information about the state of any other agents in the system. A vehicle agent "sees" a customer only when the customer agent announces itself, and records information about a customer only if and when that customer is inserted into its schedule. Vehicle agents are not required or expected to cooperate with each other in any form or fashion.

The basic improvement mechanism is simple: we allow customer agents to stochastically request removal from a schedule, and go searching for a better deal by re-announcing their availability for service. In this way, we overcome the primary weakness of Algorithm 2, which is its sensitivity to the order in which customers are announced. In many cases, a customer is released from a given van only to be later rescheduled in the same vehicle, but there are enough opportunities for improvement that overall performance improves significantly.

The algorithm makes no distinction between scheduling and rescheduling. In fact "rescheduling" begins before scheduling has ever finished. The basic process can be described as follows:

1. Each vehicle agent has a "rescheduling interval", which is stochastically determined. This interval consists of some fixed period of time (the *baseline)* plus a random variable drawn from an exponential population. This period is set when a vehicle agent is initialized, and then reset to a new value at the end of each rescheduling interval. This is implemented as a timer that fires to begin a rescheduling interval.
2. During a rescheduling interval, a vehicle agent first stochastically selects an exponential random variable, which is the upper bound on the number of customers in that van that may be rescheduled, and which is bounded above by the total number of customers in the schedule.
3. Customers are selected from the vehicle, and each customer is selected for rescheduling with a probability $p_c(t)$ that goes to zero its earliest service time approaches. In the tests of the next section, $p_c(t)$ reaches zero thirty (simulated) minutes before the earliest possible time a customer could be serviced in a van.

This choice of $p_c(t)$ reflects the online application for which the algorithm was designed. In order to ensure that customers traveling early in the day are scheduled efficiently, we allow the system to run three hundred simulated minutes, plus an additional simulated minute per customer, prior to enforcing policy 3. Since customers are normally required to make reservations a day in advance, this early rescheduling is consistent with the target application.

## Empirical Evaluation

We obtained a small database of typical customer information from an airport shuttle company operating in the Washington, DC metropolitan area, where there are three large regional airports. We used this information to generate distributions of relevant customer information as a function of the time of day. For the purposes of the test described in this paper, this information includes a) whether a customer is arriving or departing, and b) which airport customers are flying into or out of. We also determined the distribution of customer group sizes. To determine point-to-point travel times and distances, we used Caliper Co.'s TransCAD® system to geo-code addresses, and compute the minimum pair-wise distances and travel times for 500 addresses from this database. Note that the minimum-travel-time route is not necessarily the same as the minimum-distance route. All tests reported in this paper used the 250,000 minimum-time values obtained in this way.

This information was used to randomly generate problem sets. To create a new customer, we first randomly selected an address id in the range 0-499. We assume that these addresses are independent from the other values in the data. After selecting an address, we use a Bernoulli test to determine whether or not the customer is arriving or departing. We then select the customer's airport, based upon our distribution analysis of arriving and departing customers. To determine the airport-arrival time, we randomly select the hour of the flight based upon our distribution analysis. We then use a random number uniformly distributed between 0-59 to decide the exact minute of the flight. Finally we determine the customer group size using the empirical distribution.

This process was used to generate two different test data sets, each of one hundred problem instances. The first test

set contains 100 problems of 100 customers each, and the second contains 100 problems of 200 customers each. While this data generation process is imperfect, we believe it provides an adequate model of a real-world shuttle-dispatching problem.

Recall that our system was designed to run online. In order to test this algorithm, we ran it in simulated time, as we will describe below. The results reported in this paper were run on a single 350M Hz PC with 256Mb of SDRAM, running Windows NT 4.0. Tests were run at one hundred and twenty times real time (i.e. 500 milliseconds = 1 simulated minute.) In order to prevent thrashing associated with having scores of customers simultaneously negotiating with as many as fifteen vehicle agents, we maintained a queue of unannounced customers. Every five hundred milliseconds, this queue was examined and if not empty, the customer agent at the head of the queue announced itself. Each problem was run for 1200 simulated minutes, because we observed that there was little, if any significant improvement after this cutoff.

The algorithm turns out to be very sensitive to tuning. There are a number of different tuning parameters, but for the purposes of this paper, we focused upon the two rescheduling interval parameters, and the exponentially distributed random variable that determines the maximum number of customer that will be rescheduled per van. We tuned the algorithm on ten 100-customer test problems that are not in the reported test set. The results reported in this paper use a rescheduling baseline of 5000 milliseconds, plus an exponential random variable with a mean of 5000 milliseconds. We use a value of 2.0 for the mean of the exponentially distributed number of deletions per van.

The object-oriented implementation of the algorithm makes it a simple matter to customize customer constraints. However, our tests assume a uniform set of constraints. For departing customers, we take the airport arrival time to be the latest time at which the customer can be delivered to the airport without violating a deadline. Normally, this is 30 minutes prior to flight departure. Consistent with industry practice, the earliest permissible pickup time for a departing customer is two hours prior to this time. The latest permissible pickup time is the time at which the van could arrive at the customer's site, load all passengers into the van and still be able to drive a direct path to the airport at the minimum travel time, and arrive there at the customer's airport arrival time. We allow one minute, plus one additional minute per passenger, to load and unload the van.

For arriving customers, we assume that the earliest possible pickup time is the airport arrival time, and that the customer must be picked up within two hours of this time. The earliest possible time that an arriving customer can be dropped off is in the case where the customer is picked up and loaded in to a van at the earliest possible time, and driven directly home. The latest allowable drop-off time is three hours later than this. Again, we assume one minute plus an additional minute per passenger is required to deliver a customer to his destination, but at the airport, we allow twenty minutes, plus an additional minute per passenger. All vehicles in these experiments were given a maximum capacity of six passengers.

## Results

We compare the performance of three algorithms on two sets of problems. The first algorithm is our centralized algorithm, which is an implementation of Solomon's insertion heuristic extended to the PDPTW. The second algorithm is our agent-based system in the absence of any post-optimization procedure, and corresponds to Algorithm 2 above. The third row summarizes the behavior of our agent-based scheduler with stochastic optimization.

| Algorithm | Vans | Travel Time | Waiting Time | Total Time |
|---|---|---|---|---|
| Centralized | 7.78 | 5291 | 1383 | 6674 |
| Distributed / No optimization | 9.77 | 6015 | 3036 | 9051 |
| Distributed with optimization | 7.60 | 4586 | 2545 | 7132 |

**Table 1 100-Customer Routing Results**

The primary optimization criteria in our target application was to minimize the number of delivery vehicles employed, with a secondary goal of minimizing the total time that vehicles were in use. Our results are presented in Tables 1 and 2. As Table 1 shows, in the 100-customer cases, our distributed algorithm uses 2.4% fewer vehicles, while producing routes that consume roughly 6.8% more time than those produced by the centralized algorithm. These differences are statistically significant. The hypothesis that the average difference in performance of these two algorithms is zero fails a paired, two-tailed T-test at the 0.04 confidence level ($t = 2.07$). The probability that the mean travel times found by the two algorithms are actually equal approaches zero ($p = 4.11E\text{-}11$, $t = 7.26$.) Note that the average *travel times* in the distributed routes are roughly 86.7% that of the centralized routes. The time advantage displayed by the centralized algorithm stems from the fact that the distributed algorithm finds routes with almost twice as most total *waiting time* which is essentially slack in the schedule. This is a desirable result in the highly dynamic and unpredictable application domain for which the algorithm was designed and tuned. With drivers working on commission, routes with fewer vans (and thus more customers per van), shorter total drive times and more schedule slack are ideal, in that they minimize company costs, as well as the likelihood that a service guarantee will not be met.

The total time of our solutions, with respect to the centralized algorithm, improves slightly as the number of customers increases. This is illustrated in Table 2, which presents summary results for the 200-customer problems. The distributed algorithm achieves similar vehicle savings

as the 100-customer case - approximately 2.3% fewer vehicles, while coming significantly closer to the total time performance of the centralized algorithm. Again, the observed differences are statistically significant, with the difference in vehicle usage approaching the 0.001 level of confidence (t = 3.26). The difference in route distances is significant at the 0.01 confidence level (t = 2.66.)

| Algorithm | Vans | Travel Time | Waiting Time | Total Time |
|---|---|---|---|---|
| Centralized | 12.67 | 9129 | 1878 | 11007 |
| Distributed / No optimization | 16.6 | 11092 | 4660 | 15753 |
| Distributed with Optimization | 12.36 | 8198 | 3038 | 11236 |

**Table 2 200-Customer Routing Results**

Our distributed, asynchronous implementation platform does not permit a direct comparison of the CPU requirements of the distributed and the centralized algorithms. Instead, we present analytical evidence that suggest our approach will scale well. Asymptotically, the centralized algorithm is $O(Cn^2)$, where $n$ is the number of customers and $C$ is the maximum vehicle capacity. The agent-based algorithm is $O(CV)$ where $V$ is the number of vehicles and $C$ is as above. In addition, the number of stochastic "give backs" in the 100-customer tests averaged 611, so the cost of improvement can be viewed as a small constant multiple of the initial cost of scheduling in the distributed case. Moreover, the increase in the number of the re-insertions is less-than-linear in the increase in the number of customers. In the 200 customer tests, the total number of re-insertions averaged 782. Taken together, we have every reason to believe that the optimization technique scales well as the number of customers increases. In addition, this algorithm was explicitly designed for and implemented as a distributed application. Assuming that we assign a separate processor to each vehicle agent, the asymptotic time-to-solution is $O(C)$.

## Conclusions and Future Work

The results we report in this paper were a bit of a surprise. Our goal was to design an agent-based online optimization system for vehicle routing. Like Fischer and Müller, we expected to sacrifice some level of global utility is return for the increased reactive capabilities of a distributed, agent-based system. Our hope in designing the stochastic improvement mechanism was to overcome an apparent limitation of previous approaches based upon local post-optimization, and to come relatively close to the level achieved by practical centralized heuristics. The results were surprising because they actually out-performed a centralized heuristic that is known to perform well. Note that there is nothing about our technique that explicitly precludes a market-based post-optimization mechanism such as the one used in MARS, and we hope to experiment with such a hybrid in the future.

Inasmuch as these results were unexpected, there remains a good deal to be done to prove the viability of the approach. In particular, there are no standard benchmarks for the PDPTW that we know of, but there are dozens of studies that have used Solomon's instances for the VRPTW. In addition, the nature of our implementation makes a direct analysis of the runtime performance of this algorithm impossible. From the perspective of the vehicle routing literature, a centralized implementation of this basic technique to solve the VRPTW is warranted.

The tests and results presented in this paper all address a static problem. The fact that our distributed agent-based solution to this problem performs comparably to a centralized algorithm on this problem is both surprising and noteworthy. However, our research and development focus is upon the dynamic problems associated with changes in the operating environment, such as customer cancellations and delays, that occur after the algorithm has begun to solve a problem. Evaluating the quality of performance in such dynamic systems presents a significant challenge to the research community. Our next obvious step in this direction is to introduce such dynamism into the problem, and we are already testing our system upon such problems

## Acknowledgements

This work was funded in part by NASA SBIR contract NAS8-9705. We would like to thank Ben Smith and Steve Chien of NASA JPL, and James Wentworth of the US FHwA for their encouragement and support, as well as Jun Lang, Prasad Narasimhan, Angela Malais, Geoff Bernstein, and Renato Levy for their roles in the implementation.

## References

Bentley,J.J. 1992. Fast algorithms for geometric travelling salesman problems. *Op. Rsrch. Soc. of America*, **4**: 387-411.

Davis,R. and Smith,R.G. 1983. Negotiation as a metaphor for distributed problem solving. Artificial Intelligence, **20**:63-109.

Dumas,Y.; Desrosier,J. and Soumis,F. 1991. The pickup and delivery problem with time windows. *European Journal of Operational Research*, **54**:7-22.

Fischer,K.; Müller,J.P.; and Pischel,M. 1996. Cooperative transportation scheduling: an application domain for DAI. *Journal of Applied Artificial Intelligence. Special Issue on Intelligent Agents*, **10**.

Fischer,K;, Müller,J.P.; and Pischel,M. 1995. A model for cooperative transportation scheduling. *Proceedings of the 1st Int.l Conf. on Multiagent Systems (ICMAS'95)*, pp. 109-116.

Laporte,G. 1992. The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operations Research*, **59**:345-358.

Potvin,J.; and Rousseau,J. 1993. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operations Research*, **66**:331-340.

Solomon,M. 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, **35**:254-265.