Algorithm for Vehicle Routing Problem with Time Windows Based on Agent Negotiation

Petr Kalina
Department of Cybernetics
Faculty of Electrical Engineering
Czech Technical University in Prague
peta.kalina@gmail.com

Jiří Vokřínek
Agent Technology Center
Faculty of Electrical Engineering
Czech Technical University in Prague
jiri.vokrinek@fel.cvut.cz

ABSTRACT

We suggest an efficient algorithm for the vehicle routing problem with time windows (VRPTW) based on agent negotiation. The algorithm is based on a set of generic negotiation methods and state-of-the-art insertion heuristics. Experimental results on well known Solomon's and Homberger-Gehring benchmarks demonstrate that the algorithm outperforms previous agent based algorithms. The relevance of the algorithm with respect to the state-of-the-art centralized solvers is discussed within a comprehensive performance and algorithmic analysis, that has not been provided by previous works. The main contribution of this work is the assessment of general applicability of agent based approaches to routing problems in general providing for a solid base for future research in this area.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent agents, Multiagent systems*

General Terms

Algorithms, Measurement, Performance, Experimentation

Keywords

Vehicle routing problem with time windows, Multi-agent problem solving, Agent negotiation

1. INTRODUCTION

The vehicle routing problem with time windows (VRPTW) is one of the most important and widely studied problems in the the transportation domain. For a comprehensive literature review refer e.g. to surveys presented by [1, 2]. The VRPTW is a problem of finding a set of routes from a single depot to serve customers at geographically scattered locations. Each customer is visited by exactly one route with each route starting and ending at the depot. For each route the sum of demands of the customers served by the route must not exceed the capacity of the vehicle serving the route (capacity constraint). Also, the service at each customer must begin within a given time interval (time window constraints). The primary objective of the VRPTW is to find the minimum number of routes servicing all customers. Usually a secondary optimization objective is to minimize the total distance traveled. The primary objective corresponds to solving the underlying multiple bin-packing problem while the secondary objective corresponds to a variant of the multiple traveling salesman problem — both solved in a state space constrained by the time windows. Traditionally the VRPTW (together with the closely related pickup and delivery problem with time windows - PDPTW) is a problem area dominated by centralized solvers e.g.[9, 11].

Real world applications of routing algorithms are often very complex with highly dynamic, heterogenous and potentially non-cooperative or privacy conscious environments having to be captured and processed, being part of the higher level transactions e.g. general supply chain management processes etc. The multi-agent systems are an emerging choice for modeling systems with attributes similar to those mentioned above. An interesting survey on real-world applicability of agent based approaches in the transportation domain is presented in [13].

The aim of this paper is not, however, to stress the real-world applicability of presented algorithm. On the other hand, we present a thorough assessment of the agent based algorithm in terms of overall performance in an effort to establish its position among the state-of-the-art algorithms.

2. RELATED WORK

As already mentioned, a thorough survey of VRPTW algorithms is presented by [1, 2]. Thus we only refer to the two currently leading state-of-the-art algorithms.

In [9] the authors present an algorithm based on the *ejection pools* principle. The algorithm is based on performing very good unfeasible insertions of customers to individual routes, followed by an ejection procedure in which the feasibility is recovered by ejecting some other customers from the unfeasible routes. The algorithm equals the best known cumulative number of vehicles (CVN) of 405 on the Solomon's instances with new best known cumulative travel time (CRT) of 57233.

An improved algorithm presented in [11] further employs a specific local search strategy guiding the ejections. Also, a feasible insertion mechanism denoted as *squeeze* as well as a search diversification *perturb* procedure are employed throughout the solving process boosting the algorithm's convergence. The algorithm provides for the contemporary best known CVN of 10290 over the whole extended Homberger-Gehring benchmark set.

A number of approaches have been suggested for solving the VRPTW and routing problems in general by means of multi-agent negotiation profiting from well known multi-agent based approaches to general task allocation problems [16]. On simple VRP good results have been reported by an

agent based solver presented by [15]. In general, however, there has been very few works trying to rigorously establish the position of agent negotiation in a field dominated by centralized solvers, with most contributions focusing on the real-world applicability rather than outright performance.

An agent based algorithm for VRPTW is presented in [5], built around the concepts of a Shipping Company and underlying Shipping Company Truck. The planning is done dynamically and is based on the well known contract net protocol (CNP) accompanied by a "simulated trading" improvement strategy based on finding the optimal customer exchanges by solving a maximal pairing problem on a graph representing the proposed exchanges. No relevant performance assessment is provided and the algorithm is found to be sensitive to the ordering of routed tasks.

The algorithm for PDPTW presented by [7] is essentially a parallel insertion procedure based on CNP with subsequent improvement phase consisting of reallocating some randomly chosen tasks from each route. Used cost structure is based on the well known Solomon's I1 insertion heuristic [14]. The performance is assessed on an ad-hoc dataset.

The algorithm for VRPTW presented by [8] is based on agents representing individual customers, individual routes and a central planner agent. A sequential insertion procedure based on Solomon's I1 heuristic is followed by an improvement phase in which the agents propose moves gathered in a "move pool" with the most advantageous move being selected and performed. Additionally, a route elimination routine is periodically invoked — which is not well described in the text. Experimental assessment is based on Solomon's instances [14] with a CVN of 436 and CRT of 59281. No runtime information is provided.

In [4] the authors propose a VRPTW algorithm based on Order agent — Scheduling agent — Vehicle agent hierarchy. The algorithm is based on a modified CNP insertion procedure limiting the negotiation to agents whose routes are in proximity of the task being allocated in an effort to minimize the number of negotiations. Again no relevant performance information is provided.

3. NOTATION

Let $\{1..N\}$ represent the set of customers with the depot denoted as 0. Let a sequence of customers $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$ denote a route served by a single vehicle with c_0 and c_{m+1} corresponding to the depot. For each customer c_i on the route let (e_i, l_i, s_i, d_i) denote the earliest/latest service start times (the *time window*), service time and demand at the customer respectively. For simplicity, we will use the term *task* to denote a customer and all accompanying service information. Let D denote the vehicle capacity and let $t_{i,j}$ correspond to the travel time between customers c_i and c_j (in an Euclidian space). We use the term *partial solution* to denote a solution with some unserved customers.

Given a route $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$ let (E_i, L_i) correspond to the *earliest* and *latest possible service start* at customer c_i computed recursively according to:

$$\begin{aligned} E_1 &= max\,(e_1, t_{0,1}) \\ E_i &= max\,(e_i, E_{i-1} + s_{i-1} + t_{i-1,i}) \end{aligned} \quad (1)$$

and

$$\begin{aligned} L_m &= l_m \\ L_i &= min\,(l_i, L_{i+1} - t_{i,i+1} - s_i) \end{aligned} \quad (2)$$

As shown in [3], the time window constraints are satisfied when $E_i \leq L_i$ for all $i \in 1..m$. The capacity constraint is satisfied when $\sum_1^m d_i \leq D$.

4. ALGORITHM BASED ON AGENT NEGOTIATION

As mentioned above, a relevant rigorous assessment of the key properties of the respective agent-based algorithms e.g. runtime, convergence, etc. has not been provided by neither of the previous studies. Thus the main contribution of this work is: (i) the establishment of a general framework for agent based approaches based on the state-of-the-art knowledge, (ii) assessment of its algorithmic properties on known widely used benchmark sets using a performance conscious prototype implementation and (iii) the discussion of important areas for future research in an effort to provide a sound alternative to traditional solvers.

The presented algorithm is similar in its approach to the generalizad algorithmic framework for task allocation based problem solving described in [16, 15]. Thus the presented framework is generic and can adopt a number of approaches as far as the actual negotiation/allocation process is concerned. It provides a base for formalizing agent negotiation based solving approaches to routing problems in general.

A three layer basic architecture features a top layer represented by a Task Agent, middle layer represented by an Allocation Agent and a fleet of Vehicle Agents present at the bottom level of the architecture.

Task Agent acts as an interface between the algorithm's computational core and the surrounding infrastructure. It is responsible for registering the tasks and submitting them to the underlying Allocation Agent.

Allocation Agent instruments the actual solving process by negotiating with the Vehicle Agents. The negotiation is conducted based upon task commitment and decommitment cost estimates provided by the Vehicle Agents.

Vehicle Agent represents an individual vehicle serving a route. It provides the Allocation Agent with the above mentioned inputs. These are computed based on local (private) Vehicle Agent's plan processing.

Fiugure 1 illustrates the allocation algorithm process implemented by the Allocation Agent. The process is started given a partial solution σ and a set of unallocated tasks T. For the dynamic problem variant the set T corresponds to a one-element set with the actually processed task and σ represents the partial solution σ at the time of allocation. In static case the set T corresponds to an ordered set of all instance tasks, while σ is a partial solution representing a set of empty routes.

In essence, the allocation process consists of a series of *negotiation* interactions between the Allocation Agent and the Vehicle Agents serving the routes within the partial solution σ. In various places of the algorithm the Allocation Agent may require the Vehicle Agents to: (i) estimate the cost of

```
Input: Ordered set of tasks T, Partial solution σ
Output: Solution σ after task allocation

Procedure allocate(T, σ)
begin
 1:   Init reallocate counters r[t] := 0 for all t ∈ T;
 2:   while (exists(t ∈ T), r[t] ≤ reallocationLimit)
 3:     dynamicImprove(σ);
 4:     Select first t ∈ {t ∈ T, r[t] minimal};
 5:     I := {v ∈ Ins^feas(σ, t), costCommit(t, v)
              is minimal};
 6:     if (I ≠ ∅) then
 7:       Randomly select v ∈ I;
 8:       commit(t, v);
 9:       remove t from T;
10:     else
11:       r[t] := r[t] + 1;
12:     endif
13:   endwhile
14:   finalImprove1(σ, T);
15:   finalImprove2(σ, T);
16:   ..
17:   return σ;
end
```

Figure 1: The Allocation Agent main algorithm.

committing to a given task, (ii) estimate the gain resulting from dropping some commitment, (iii) identify the most costly task within their respective routes or (iv) commit to or decommit from a given task.

The interactions with the Vehicle Agents are represented by the $costCommit(t, v)$ and $commit(t, v)$ functions (lines 5 and 8) corresponding to the cost estimate of agent v committing to task t and the actual commitment. From the Allocation Agent's point of view these are Vehicle Agent's private operations. Thus they may reflect various aspects and constraints the vehicles need to consider (e.g. loading constraints, vehicle actual position, vehicle shift times, etc.), potentially reflecting the heterogeneity of the real-world problem being solved. Similarly, various semantics of the actual commitments may be introduced (e.g. revocable/irrevocable etc.). The other interactions mentioned above are carried out within the improvement methods (lines 3, 14 and 15) using the corresponding $gainDecommit(t, v)$, $worstCommitment(v)$ and $decommit(t, v)$ methods and will be described later in the text.

The process begins with resetting the reallocate counters (line 1) and runs in a loop that is terminated when the limit on unsuccessful allocation retries has been reached for all unallocated tasks or until no such tasks exist (line 2). In the former case the allocation has not been successful. Depending on the real world problem semantics the Task Agent may instantiate (dispatch/require) another vehicle and restart the process using either an empty solution or reusing the partial solution returned by the previous run.

The $dynamicImprove(\sigma)$ function (line 2) corresponds to a particular *dynamic improvement method* being executed iteratively throughout the allocation process. Later in the text we describe a set of applicable methods e.g. *ε-ReallocateWorst* or *ReallocateAll*. At this stage the application of a specific improvement method may enhance the partial solution σ and therefore: (i) potentially increase the chance of success for the latter stages of the allocation process and (ii) possibly modify the solution in a way that enables allocation of tasks that were not successfully allocated in previous attempts. As the individual routes get denser, the space for changing the solution decreases. Thus the ability to improve the solution in the early stages of the allocation process is an important feature of the algorithm.

The tasks with the lowest number of retries are processed first in the order in which they are encountered in the set T (line 4). An auction in which each of the Vehicle Agents provides a commitment cost estimate for the currently processed task is carried out on behalf of the Allocation Agent. Thus the agents that can feasibly undertake the task (the set Ins^{feas}) with the best commitment costs are identified (line 5). In a distributed environment the auction process is carried out using the CNP protocol. A randomly chosen agent from this set then commits to the task (lines 7 and 8) and the task is marked as allocated (line 9). In case no agent can feasibly undertake the task (line 6), the reallocate counter for the task is incremented (line 11).

The $finalImprove1, 2(\sigma, T)$ (lines 14, 15) correspond to the final improvement strategies being applied. Just like the $dynamicImprove(\sigma)$ function these correspond to a certain improvement method being applied here. However, the method used for final improvement may differ from the one used for the dynamic improvement. For example it may be advantageous to employ a route length conscious improvement method at the end of the allocation process to address the secondary optimization criteria of the problem. The difference in signature between the two functions illustrates the fact that throughout the final improvement we may still try to allocate the unallocated tasks.

Thus a particular algorithm is instantiated by supplying the actual fleet of Vehicle Agents with respective cost estimation functions and specifying the individual improvement methods for the $dynamicImprove$ and $finalImprove1, 2$ functions.

For the static variant of the VRPTW discussed within this work, the primary optimization criteria is addressed by running the allocation process with an initial solution σ corresponding to an appropriately small fleet of homogenous empty vehicles represented by Vehicle Agents. In case the process fails, a new Vehicle Agent is instantiated and added to the fleet and the process is restarted.

Within the next sections we present two different variants of VRPTW Vehicle Agent implementations based on the state-of-the-art insertion heuristics and three improvement methods for the static VRPTW problem variant, as well a theoretically sound setting for the initial size of the fleet. Several ways in which the set of tasks T can be ordered are discussed as well.

4.1 Insertion Heuristics

The two Vehicle Agent implementations presented within this study are based on the well known *cheapest insertion* principle. Let c_j be the customer associated with the task t, let $\langle c_0, c_1, ..c_m, c_{m+1}\rangle$ be the corresponding route of the agent v. Let $costIns(t, v, i)$ represent the cost estimate of inserting t between the customers c_{i-1} and c_i. The cost estimate for agent v committing to t is thus given by

$$costCommit(t, v) = \underset{i \in fi(1..m)}{\operatorname{argmin}} (costIns(t, v, i)) \qquad (3)$$

where $fi(1..m)$ represents the set of all feasible insertion points on the route.

Given an insertion index i, let (E_j, L_j) represent the *earliest possible* and *latest possible service start* at c_j when inserted at index i. The E_j and L_j values can be computed according to Equations 1 and 2 as

$$E_j = max\left(e_j, E_{i-1} + s_{i-1} + t_{i-1,j}\right) \tag{4}$$

and

$$L_j = min\left(l_i, L_i - t_{j,i} - s_j\right). \tag{5}$$

The insertion is feasible when both the time window constraint $E_j \leq L_j$ and the capacity constraint $\left(\sum_1^m d_i\right) + d_j \leq D$ are satisfied. By storing agent's cumulative demand $D^c = \sum_1^m d_i$ alongside the agent's plan the capacity constraint can be checked trivially by verifying that $D^c + d_j \leq D$.

Given the identified best insertion index i, the actual commitment of the agent v to the task t requires the $E_k, k = i..m$ and $L_l, l = 1..i-1$ values to be updated according to Equations 1 and 2 as well as the agent's cumulative demand D^c.

As mentioned above, we evaluated two respective implementations of the Vehicle Agent's interface functions based on two well known insertion heuristics.

4.1.1 Travel Time Savings Heuristic

The *travel time savings* heuristic is notoriously known to the routing community. Using the same example as in the previous section, the insertion cost corresponds to

$$costIns^{TT}(t, v, i) = t_{i-1,j} + t_{j,i} - t_{i-1,i}. \tag{6}$$

Let c_k denote a customer corresponding to a task t' already within v's plan. The decommitment gain is computed accordingly as

$$gainDecommit^{TT}(t', v) = t_{k-1,k} + t_{k,k+1} - t_{k-1,k+1}. \tag{7}$$

The travel time savings heuristic leverages the spatial aspects of the problem, with a cost structure corresponding to the impacts of agent commitments or decommitments on the travel time of the agents. It has been shown [14, 10], however, that an insertion heuristic exploiting the temporal relations of the tasks given by their respective time windows can yield significantly better results.

4.1.2 Slackness Savings Heuristic

The *slackness savings heuristic* thus introduces elements to the cost structure based on the interactions between individual time windows constraints caused by their respective widths and placements within the agent's route. It is a simplified adaptation of PDPTW heuristic presented by [10] for the VRPTW problem.

Given c_k corresponding to a customer on agent v's route from previous examples, let $sl_k = L_k - E_k$ represent the *slack time* at customer c_k. An insertion of c_j requires the L_k and E_k values to be updated along the route possibly reducing the corresponding sl_k values. Reductions to slack times correspond to the constraining effects an insertion of a customer imposes on the rest of the agent's route. Let $sl'_j = L_j - E_j$ represent the slack time at the inserted customer c_j after the insertion. We denote $sl_j = l_j - e_j$ the slack time at c_j *prior* to the insertion. Given $sl'_k = L'_k - E'_k, k = 1..m$ being the updated slack times after the insertion, the overall reduction in route slackness is given by

$$SLR(t, v, i) = \left(\sum_1^m \left(sl_k - sl'_k\right)\right) + \left(sl'_j - sl_j\right). \tag{8}$$

The i variable in function's signature corresponds to the fact the $sl'_k, k \in 1..m$ and sl'_j are particular for the insertion index i.

The $costIns^{SL}(t, v, i)$ for the slackness savings heuristic is based on both the spatial and the temporal aspects of the insertion with

$$costIns^{SL}(t, v, i) = \alpha.SLR(t, v, i) + \beta.costIns^{TT}(t, v, i). \tag{9}$$

where α and β, $\alpha + \beta = 1$ correspond to the respective weights of the two criteria being considered.

The $removalGain(t', v)$ is computed using an analogous approach as

$$\begin{aligned} removalGain^{SL}(t, v) = \alpha.SLI(t, v) \\ +\beta.removalGain^{TT}(t, v). \end{aligned} \tag{10}$$

where $SLI(v, t)$ corresponds to a slack time increase resulting from updated sl'_i values as a result of removing customer c_j with the updated slack time at customer c_j given by $sl'_j = l_j - e_j$.

4.2 Improvement Methods

Within the *dynamicImprove* and $finalImprove1, 2$ functions each Vehicle Agent decommits from some of its tasks, based on the particular negotiation method being applied. Each decommitment is followed by an auction process in which an agent with the lowest $costCommit(t, v)$ commitment cost estimate commits to the task. Thus the task can be reinserted to exactly the same position within the same agent's plan, to a different position within the agent's plan or a different agent can commit to the task. We refer to a decommitment and subsequent task reinsertion as the *reallocation* of a single task t. In the first above mentioned case we consider the reallocation unsuccessful as the solution was not changed. Following three negotiation methods were considered:

- ***ReallocateAll***: For each Vehicle Agent all of its tasks are reallocated. The tasks are processed in the order in which they appear in respective agents' routes. The agents are processed in the order in which they are added to the partial solution σ being processed.
- ***ε-ReallocateRandom***: For each Vehicle Agent a portion of its tasks corresponding to $\varepsilon \in (0, 1)$ is reallocated. For $\varepsilon = 1$ this corresponds to the previous method but for an individual agent the tasks are processed in a random order. The order in which the agents are processed is the same as above.
- ***ε-ReallocateWorst***: Each agent drops commitments to a portion of its tasks corresponding to $\varepsilon \in (0, 1)$. The tasks processed in decreasing order based on their respective decommitment gain estimates. The ordering is achieved via the *worstCommitment* agent interface function. For presented implementation this consists of the agent privately invoking the *gainDecommit* function for each task within its route and returning the most costly one.

The interactions between the Allocation Agent and the Vehicle Agents are carried out using the Vehicle Agent interface functions: *costCommit*, *commit*, *gainDecommit*, *decommit* and *worstCommitment*. As already mentioned, these are Vehicle Agent's private operations potentially reflecting the real world problem semantics.

4.3 Algorithm Initial Settings

The remaining settings necessary for instantiating a particular algorithm for the static VRPTW case are: (i) the count of empty Vehicle Agents in the initial partial solution σ and (ii) the ordering of the set of tasks T being passed to the allocation process.

4.3.1 Initial Vehicles Count

As already mentioned, the static VRPTW case primary optimization criteria is addressed by instantiating a fleet of empty vehicles and restarting the allocation process with increased number of vehicles in case of failure until a feasible solution is found.

A sound setting for the initial vehicles count should correspond to the lower bound number of vehicles for the problem instance being solved. Such a number can be computed based on the mutual incompatibilities of the tasks given their respective time windows and travel times. Two tasks are incompatible if they cannot be served by a single vehicle, that is when it is impossible to start the service at either of the customers at the earliest possible moment and reach the other customer within the corresponding time window. The minimal number of vehicles necessary for solving the instance is bound by the size of the maximal set of mutually incompatible tasks, providing for the time windows based lower bound on the number of vehicles. Also, the cumulative demand of all customers has to be lower than the cumulative capacity of all the vehicles combined providing for a capacity based lower bound on the number of vehicles.

Within this work thus the vehicles count in the initial partial solution σ is set to the bigger of the two above mentioned lower bounds. The size of the maximal set of mutually incompatible tasks is estimated using a graph based algorithm approximately solving a maximal clique problem on a graph with edges corresponding to the mutually incompatible tasks. The algorithm is described in [10].

4.3.2 Initial Task Ordering

Within the $allocate(T, \sigma)$ function the task to be processed next is chosen based upon the ordering of the set T. We considered the following settings:

- **Most Demand First** (MDF): Tasks are ordered decreasingly by the volume of their demands, according to the well known *most-constrained-first* greedy allocation principle applied to the underlying multiple bin packing problem.
- **Tightest Time Window First** (TTF): Tasks are ordered increasingly by the duration of their time windows following the *most-constrained-first* approach, this time based on the time windows of the individual tasks.
- **Earliest First** (EF): Tasks are ordered increasingly by the beginning time of their time window. This setting causes naturally competing tasks to be allocated in close succession.

During our experimental assessment of the presented algorithm we found that none of the orderings is dominant. To the contrary each of the orderings performed well on different subset of benchmarked instances. Thus finding a fitting task ordering for a particular problem instance is an interesting problem in its own right, however, it is outside the scope of this study. Instead, in an effort to appropriately illustrate the limits of the presented algorithm, we treated the *set of orderings* to be used as an additional parameter of the algorithm, running the $allocate(T, \sigma)$ function for each of the orderings and choosing the best result from these runs.

4.4 Complexity Analysis

Given N is number of tasks for a given problem instance and assuming the number of reallocation retries is constant, the asymptotic complexity of the allocation process corresponds to

$$O^{clique} + O^{ordering} + N \times \left(O^{dyn} + O^{alloc}\right) + O^{fin1,2} \quad (11)$$

where O^{clique} is the complexity of estimating the initial Vehicle Agents count, $O^{ordering}$ is the complexity of the initial ordering of the set of tasks T prior to the allocation process, O^{dyn} is the complexity of the *dynamicImprove* function, O^{alloc} corresponds to the complexity of the auction process of finding the agent with the minimal $costCommit(v, t)$. The $O^{fin1,2}$ corresponds to the complexity of the corresponding $finalImprove1, 2$ functions.

The complexity of O^{alloc} is inherent to the heuristic being used. Given m is the number of tasks within agent v's route, the complexity of the $costCommit^{TT}(t, v)$ function is $O(m)$. The $costCommit^{SL}(t, v)$ requires the updated slack times to be computed for each insertion point thus resulting in a complexity of $O(m^2)$. As the total number of tasks in agents' routes is bound by N, the worst case complexity of the O^{alloc} is $O(N)$ for the travel time savings heuristic and $O(N^2)$ for the slackness savings heuristic. Similarly the $decommitGain(t, v)$ function complexity depends on the insertion heuristic being used with $decommitGain^{TT}(t, v)$ having an O(1) complexity while $decommitGain^{SL}(t, v)$ being $O(N)$ in the worst case. The subsequent $commit(t, v)$ results in $O(N)$ worst case complexity due to the need to update the E_i and L_i values stored alongside the winning agent's route having N tasks in the worst case. For the same reasons the complexity of the $decommit(t, v)$ function is $O(N)$ as well. The improvement functions correspond to the application of a particular improvement method. With $\varepsilon = 1$ all of the methods consist of reallocating all tasks within the partial solution σ under construction. A single task reallocation consists of a $decommit(t, v)$ function being invoked, followed by an auction process and the subsequent invocation of $commit(t, v)$. Thus the complexity of an improvement strategy being applied is given by $O^{imp} = N \times \left(O^{decommit} + O^{alloc}\right)$ resulting in $O(N^2)$ and $O(N^3)$ worst case complexities for the two presented heuristic. Within the *ε-ReallocateWorst* method, the task with the maximal $decommitGain(t, v)$ value has to be identified prior to each reallocation. The complexity of such an operation is identical to the corresponding O^{alloc} however, so it does not affect the above mentioned conclusion. Thus $O^{fin1,2}$ and O^{dyn} correspond to the complexity of O^{imp} for respective heuristics.

As the $O^{clique} = O(N^3)$ [10] and $O^{ordering} = O(N\,log(N))$ the overall worst case complexity of presented algorithm is

Table 1: Performance of presented algorithm compared to best known results

Type	Best	Agents	*Algorithm-B*	*Algorithm-FI*	*Algorithm-DI*	*Algorithm-DIA*
All	10695, 5049252	–	+1110, +3375926 10.4%, 66.9%	+703, +2254681 6.6%, 44.7%	+343, +1962274 3.2%, 38.9%	+343, +944053 3.2%, 18.7%
100	405, 57233	+31, +2048 7.7%, 3.6%	+67, +39144 16.5%, 50.6%	+49, +23847 12.1%, 36.0%	+24, 20595+ 5.9%, 33.6%	+24, +4040 5.9%, 7.1%
200	694, 168307	–	+53, +128545 7.6%, 76.4%	+38, +89906 5.5%, 53.4%	+21, +83809 3.0%, 49.8%	+21, +29828 3.0%, 17.7%
400	1380, 389688	–	+120, +312576 8.7%, 80.2%	+73, +220114 5.3% /56.5%	+38, +201421 2.8%, 51.7%	+38, +84058 2.8%, 21.6%
600	2065, 823937	–	+194, +596673 9.4%, 72.4%	+127, +411362 6.2%, 49.9%	+56, +365305 2.7%, 44.2%	+56, +173849 2.7%, 21.1%
800	2734, 1478704	–	+278, +881897 10.2%, 59.6%	+180, +564243 6.6%, 38.1%	+89, +482700 3.3%, 32.6%	+89, +221219 3.3%, 14.7%
1000	3417, 2131385	–	+398, +1417088 11.6%, 66.5%	+236, +945209 6.9%, 44.3%	+115, +809443 3.4%, 38.0%	+115, +431058 3.4%, 20.2%
C1	2914, 952995	–	+470, +490242 16.1%, 51.4%	+333, +326360 11.4%, 34.2%	+151, +246114 5.2%, 25.8%	+151, +131224 5.2%, 13.8%
C2	895, 443144	–	+158, +456219 17.7%, 103.0%	+113, +283553 12.6%, 64.0%	+48, +246208 5.4%, 55.6%	+48, +90111 5.4%, 20.3%
R1	2881, 1121497	–	+136, +712998 4.7%, 63.6%	+67, +482043 2.3%, 43.0%	+48, +386052 1.7%, 34.4%	+48, +263126 1.7%, 23.5%
R2	600, 720454	–	+18, +799141 3.0%, 110.9%	+7, +594375 1.2%, 82.5%	+3, +564348 0.5%, 78.3%	+3, +190399 0.5%, 26.4%
RC1	2801, 1064510	–	+218, +483182 7.8%, 45.4%	+120, +304586 4.3%, 28.6%	+65, +267166 2.3%, 25.1%	+65, +166324 2.3%, 15.6%
RC2	603, 746602	–	+110, +434145 18.2%, 58.1%	+63, +263764 10.4%, 35.3%	+28, +252385 4.6%, 33.8%	+28, +12792 4.6%, 1.7%

$O(N^3)$ for the travel time savings heuristic and $O(N^4)$ for the slackness savings heuristic.

5. EXPERIMENTAL EVALUATION

The experiments were carried out using the set of well-known Homberger-Gehring benchmark instances [6]. To provide reference to previous agent-based approaches we also included the original Solomon's instance set [14], sharing the same basic attributes as the formerly mentioned set. Thus the complete benchmark set consists of 6 sets of instances with 100, 200, 400, 600, 800 and 1000 customers respectively, with 60 instances in each set (except Solomon's with 56 instances). For each set there are 6 instance types provided — the R1, R2, RC1, RC2, C1, and C2 type, each with a slightly different topology and time windows properties. For C1 and C2 types the customer locations are grouped in clusters, unlike the R1 and R2 classes where the customers are randomly placed. The RC1 and RC2 instance types combine the previous two types with a mix of both random and clustered locations. The C1, R1 and RC1 also differ from C2, R2 and RC2 in terms of the scheduling horizon, the former having a shorter horizon resulting in routes of about 10 customers on the average, the latter having a longer horizon providing for routes of around 50 customers.

The reason for using these particular widely used benchmark sets was to provide a relevant comparison with the state-of-the-art centralized solvers that has been missing from previous agent-based studies. Therefore, the inclusion of the extended Homberger-Gehring benchmarks is one of the unique assets setting this work apart.

5.1 Algorithm Configurations

We examined four different settings for the suggested algorithm. The simplest *Algorithm-B* setting refers to a baseline algorithm not employing neither the *dynamicImprove* nor the $finalImprove1,2$ functions. Such a setting corresponds to the simple parallel cheapest insertion procedure. *Algorithm-FI* extends the previous setting by employing the $finalImprove1$ function using one of the presented negotiation methods. The *Algorithm-DI* refers to a setting with both the *dynamicImprove* and $finalImprove$ functions being used. For all three mentioned settings the cost structure used throughout the whole algorithm corresponds to the slackness savings heuristic. We present these configuration to provide an insight into the role of the *dynamicImprove* and $finalImprove$ functions within the solving process.

The full fledged algorithm as presented within this study is denoted as *Algorithm-DIA*. It further extends the *Algorithm-DI* setting with a $finalImprove2$ function using the route length savings heuristic in an effort to address the secondary optimization criteria.

We used the *ReallocateAll* improvement method for the $finalImprove1,2$ functions in all applicable settings. With respect to the *dynamicImprove* function, we tested all three presented negotiation methods in the applicable *Algorithm-DI* and *Algorithm-DIA* settings. The presented results correspond to the best of these three runs.

We used $\varepsilon = 0.3$ setting for the ε parameter affecting two of the three presented improvement methods. With respect to the α and β parameters affecting the slackness savings heuristic cost structure we used $\alpha = \beta = 0.5$ setting. The

Table 2: Equalled best known solutions per instance types

Instance Type	VN Equal	VN and RT Equal	RT Error on VN Equal
All	48.6%	8.1%	24.6%
C1	33.9%	18.6%	9.9%
C2	51.7%	27.6%	9.9%
R1	30.6%	1.6%	28.4%
R2	95.1%	1.6%	26.7%
RC1	6.9%	0.0%	10.9%
RC2	72.4%	0.0%	29.3%

choice of these particular values is discussed later in the text.

5.2 Evaluation of Results

The performance of our algorithm is illustrated by Table 1. Two commonly used metrics of quality are presented: (i) the cumulative number of vehicles (CVN) and (ii) the cumulative travel time (CRT). The "Best" column presents the best known CVN and CRT for given set of instances taken from [11, 12]. The rest of the columns corresponds to the absolute and relative errors in both criteria listed for previous agent based studies [8] (the *Agents* column) and for the four settings of the presented algorithm. The results are presented for individual instance sizes ("100" corresponds to the Solomon's instances, while the "200 – 1000" sets correspond to the Homberger-Gehring instances) as well as for individual instance types that are common among both benchmark sets. Thus, for example, the second row of the last column shows that on Solomon's instance set the *Algorithm-DIA* setting achieved a CVN of 24 more than the CVN of the best known solutions, with a CRT of 4040 higher, resulting in a 5.9% and 7.1% respective relative errors.

Table 2 further illustrates the success of the full *Algorithm-DIA* settings in terms of being able to match the best known solutions in terms of: (i) the number of vehicles (VN), (ii) both criteria (VN and total travel time - RT). Complementing this information is the relative error in RT for the solutions matching the best known VN.

5.2.1 Overall Quality Analysis

In overall, the presented algorithm in the full *Algorithm-DIA* setting achieved a 3.2% CVN and 18.7% CRT average relative error when compared to the best known solutions. The algorithm was able to match the best known solutions in 48.6% in terms of the primary VN optimization criteria. In 8.1% of the cases both VN and RT criteria of the best known solutions were achieved. The average relative RT error for the VN best known matching instances was 26.4%. The algorithm outperforms all previous agent-based approaches, achieving a CVN of 429 compared to 436 presented by [8] on the Solomon's instances. The algorithm sets new best known solutions for agent-based approaches on the extended Homberger-Gehring datasets.

The performance is consistent across all instance sizes. The difference in performance between the Solomon's and the extended Homberger-Gehring datasets corresponds to the fact that slower solvers are typically not tested on the extended datasets. Such is the case, for example, with the previously presented agent-based algorithm featuring within the comparison. The results achieved on both datasets thus suggest, that an agent based approach to VRPTW is a sound alternative to the traditional centralized solvers.

Table 3: Insertion heuristic relative errors over 200 customer instances

Algorithm setting	Slack savings	Travel time savings
Algorithm-B	7.6%	25.1%
Algorithm-FI	5.5%	11.1%
Algorithm-DI	3.0%	5.2%

5.2.2 Dynamic and Final Improvements Analysis

The results for the individual algorithm settings illustrate the significance of both the $dynamicImprove$ and the $finalImprove1, 2$ functions. With the *Algorithm-B* setting there is no possibility to recover from a potentially bad allocation taking place in the earliy stages of the allocation process. For example, an early allocation may render some of the subsequent allocations infeasible due to the time window or capacity constraints, effectively preventing some parts of the search space to be traversed. In overall the *Algorithm-B* setting achieved an error of 10.4% in the VN criteria.

The *Algorithm-FI* setting extends the *Algorithm-B* setting by allowing some exchanges of the tasks within and between the routes during the final stage of the allocation process. At this stage, however, as a result of previous allocations, the partial solution σ is already tightly constrained. Thus the chance of reallocating a task already in σ is correspondingly small, resulting in a relative VN error of 6.6% across all instances.

With an average relative VN error of 3.2% the *Algorithm-DI* setting significantly outperforms the *Algorithm-FI* setting. Arguably this is due to the fact that the improvements are performed dynamically throughout the allocation process on smaller and therefore less constrained partial solutions. The slackness savings heuristic specifically tries to minimize the constraining effects of the insertions. Therefore, the *Algorithm-DI* setting dynamically improves the partial solution σ in an effort to increase the chance of future advantageous allocations or reallocations being performed.

Finally, by employing the $finalImprove2$ function, the *Algorithm-DIA* traverses the feasible neighborhood of the resulting solution σ using a travel-time driven cost structure in an effort to find a local travel-time minima. A success of such an adaptive strategy is illustrated by reducing the 38.9% relative RT error from the previous *Algorithm-DI* setting to only 18.7%.

There is a notable difference in performance of *Algorithm-B* and *Algorithm-FI* settings on clustered (C1, C2, RC1, RC2) and non-clustered (R1, R2) instances. The customers in clusters are temporally and spatially very close while the distances between clusters are much higher. A good solution is thus characterized by minimizing the number of travels between the clusters. In an early partial solution σ where not all customers are yet known the *Algorithm-B* may easily make very bad decisions like having a vehicle visit more clusters — a situation from which the final improvement of the *Algorithm-FI* cannot recover. The dynamic improvements performed within the *Algorithm-DI* setting help to counter this to some extent.

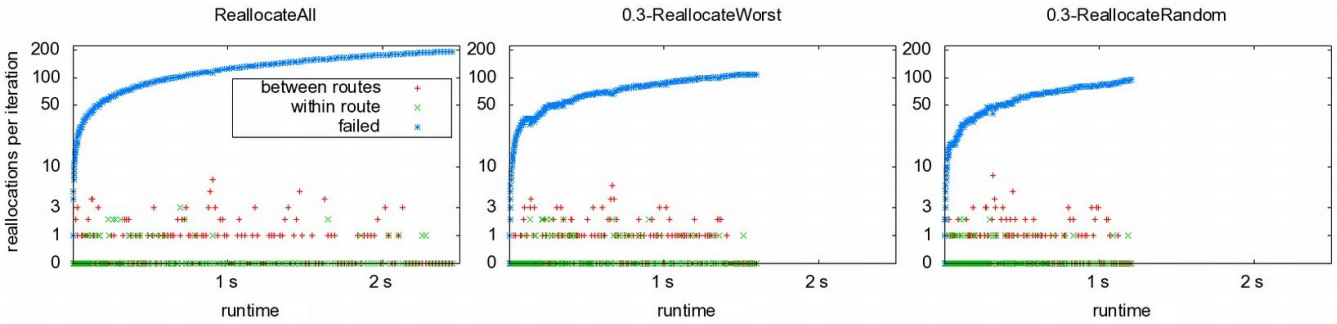

Figure 2: Improvement methods reallocation success

Table 4: Relative errors for insertion heuristics and individual orderings

	Algorithm-B		*Algorithm-DI*	
Ordering	Travel time	Slack	Travel time	Slack
MDF	46.0%	27.9%	14.6%	9.5%
TTF	28.2%	18.5%	11.4%	8.7%
EF	25.7%	12.3%	8.5%	6.4%
BEST	23.1%	7.6%	5.5%	3.0%
RAND	36.3%	23.0%	14.8%	9.7%

5.2.3 Insertion Heuristics Analysis

The commitment/decommitment cost structure provided by the insertion heuristics is the sole input for the otherwise abstract allocation process. Table 3 lists relative errors of the two presented heuristics measured for the 200 customer benchmark set. The results show that the slackness savings heuristic outperforms the traditional travel time savings heuristic in all three relevant algorithm settings (the $Algorithm-DI A$ setting actually uses both heuristics). The difference is most pronounced with the *Algorithm-B* setting while being less pronounced in *Algorithm-FI* and *Algorithm-DI* settings. Thus, interestingly, the improvement methods are able to exploit both of the heuristics with similar success. The results correspond to $\alpha = \beta = 0.5$ slackness savings heuristic parameters that has proved to be the most efficient in the computational tests that are outside the scope of this study.

Not surprisingly the slackness savings heuristic proved to be significantly slower of the two, with runtime in the *Algorithm-DI* setting being approximately 3 times longer.

5.2.4 Ordering Sensitivity Analysis

The relative errors for various initial orderings of the set of tasks T corresponding to the 200 customer benchmark set are listed by Table 4. The results for both the baseline *Algorithm-B* and the full-fledged *Algorithm-DI* settings and for each of the used insertion heuristics are presented. The BEST ordering row corresponds to the best results of MDF, TTF and EF orderings, while the RAND ordering corresponds to a baseline random ordering of the tasks.

The results suggest that neither of the proposed orderings is dominant in terms of outperforming the remaining two across the whole range of instances. To the contrary, the fact that the BEST results are significantly better than the results of either of the orderings proves that each of the orderings performs well on a different subset of instances. The results thus suggest that the individual instances differ in their nature, potentially favoring some particular ordering. For example, the MDF ordering attributed for 8 wins across the 60 measured instances suggesting that these may be the instances where leveraging the capacity aspect is beneficial, while in the rest of the cases the best results were achieved using orderings leveraging the temporal aspects of the problem. Finding an ordering that is fitting for a particular problem instance is an interesting problem in its own right that has not yet been addressed. To overcome this, we treated the set of orderings to be used as an additional parameter of the algorithm, running the *allocate* function for each ordering and choosing the best result from these runs. The presented results and runtimes correspond to the MDF, TTF and EF orderings being used.

The results further show that out of the two heuristics the slackness savings heuristic is clearly the less sensitive to the ordering of the two in the baseline *Algorithm-B* setting. In the *Algorithm-DI* setting the difference is less marked. This suggest that the dynamic improvement methods are successful in offsetting the sensitivity of used heuristic to ordering, a result that supports previous findings of [7].

5.2.5 Improvement Methods Analysis

We analyzed the respective performance of individual improvement methods used within the *dynamicImprove* function of the *Algorithm-DI* setting varying the ε parameter where applicable. Surprisingly, beginning with $\varepsilon = 0.3$, the quality of the resulting solution did not improve with bigger ε while the runtime did increase linearly. For the final improvement, we found the *ReallocateAll* method to achieve marginally better results. In overall, we found that the number of reallocations does not have a strong positive influence on the quality of resulting solution.

Figure 2 illustrates the number of reallocations performed within the individual calls of the *dynamicImprove* function for a 200 customer instance and the runtime in which they occurred within an invocation of the *allocate* function. Note that these results correspond to the slackness savings heuristic based implementation of the *costCommit* and *costDecommit* functions. Note also that the y axis is presented in logarithmic scale. The three types of points correspond to: (i) reallocations of tasks between routes, (ii) reallocations of tasks within a single route and (iii) reallocations that failed to find a feasible improving allocation for the task being reallocated. The three graphs correspond to

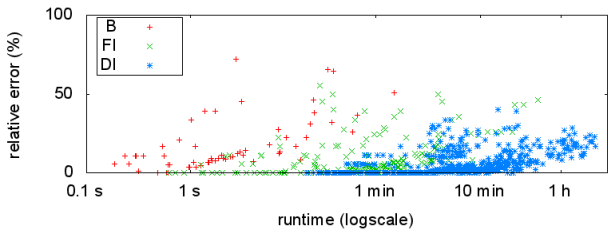

Figure 3: Results for individual algorithm settings for 1000 customer instances

Table 5: Cumulative and worst runtimes for individual instance sizes

Size	Nagata [11] Avg. RT	Lim [9] Avg. RT	*Algorithm-DIA* Avg. RT	*Algorithm-DIA* Worst RT
200	1 min	10 min	3 s	13 s
400	1 min	20 min	22 s	3 min
600	1 min	30 min	2 min	19 min
800	1 min	40 min	6 min	47 min
1000	1 min	50 min	8 min	74 min

the three presented improvement methods with $\varepsilon = 0.3$ for the two ε methods. The rising curve shape for the number of failed reallocations corresponds to the fact that throughout the process more tasks are being allocated and processed by the respective improvement methods.

The results suggest that neither of the methods is dominant terms of reallocation success. Contrary to our expectations, the ε-ReallocateWorst method did not succeed in selecting the most likely to be reallocated tasks. Furthermore, the number of successful reallocations drops towards the end of the solving process, suggesting that the cost structure provided by the slackness savings heuristic together with the proposed improvement methods get stuck in local optima. The inability to further transform the solution towards higher quality is arguably due to the fact that the number of feasible task reallocations drops rapidly as the solution gets denser.

The presented negotiation methodology only allows for traversing the solution space of partial solutions that are feasible in terms of time window and capacity constraints. We argue that for the negotiation based methodology to achieve stronger results than the arguably very promising results presented within this study, a method allowing for traversing also the infeasible space or performing more complex moves would have to be developed. Such a methodology could be embedded to the presented general solving architecture in form of some backtracking strategy and accompanying ejection based heuristic, allowing for temporal infeasibility of individual routes.

5.2.6 Runtime and Convergence Analysis

The convergence of *Algorithm-B, Algorithm-FI* and the full *Algorithm-DIA* settings is illustrated by Figure 3. The results correspond to the 1000 customers benchmark set. Note that the x-axis uses a logarithmical scale. The results confirm that the quality and robustness of the algorithm increase with more complex setting being used with obvious penalty in terms of runtime. Also the difference in terms of runtime between individual algorithm settings is dramatic. Interestingly, in many cases the very short-running settings produce a very good quality results matching even the best known solutions. This feature of the algorithm can be exploited by running various strategies in parallel competition returning an improving sequence of results over time. Based on previous evidence, using a wide set of task orderings in combination with the shorter running solvers might, for example, produce a very efficient and robust strategy with ideal parallelization features.

The comparison in terms of runtime of the full *Algorithm-DIA* setting with the currently leading algorithms is presented by Table 5. The average runtime-per-instance as well as the worst runtime recorded by presented algorithm is listed for individual instance sizes of the extended benchmark sets. The results correspond to a C++ implementation run on AMD Opteron 2.4G system for [11], a Java implementation run on a Intel Pentium 2.8G system for [9] and a normalized (single threaded) runtime on a 4G RAM AMD Athlon 2G Gentoo system running the 64-bit Sun JRE 1.6.0_22. The approach to parallelization is not mentioned in neither [9] nor [11]. Also, the secondary travel time minimization criteria is not addressed by [11]. We must note, however, that: (i) compared algorithms outperform presented algorithm in terms of CVN and (ii) are not computationally bound. Therefore to be able to draw a more relevant conclusions, settings with similar solution quality would have to be compared.

With respect to previously presented agent-based algorithms, no comparable data were provided by any of the previous works. The likely cause for that is that agent-based approaches typically rely on agent execution platforms corresponding to a loosely coupled distributed environment (JADE etc.) making them extremely inefficient.

The results show that there is a striking difference between the average and the worst runtime for the presented algorithm. The worst results correspond to the instances where the initial vehicles count estimation was much lower than the VN of the particular solution being found. In such a case the *allocate* function is restarted with sequentially increasing empty vehicles count until a feasible solution is found. The effect is most pronounced given an unfitting ordering is used for the particular run. Also, the effect increases the size of the instance. Looking back at Figure 3, the above mentioned observation is clearly illustrated. Apparently, for each respective algorithm setting, the runtimes for the individual results increase with decreasing quality of the corresponding solutions. We suggest that an improved restarts strategy could be developed, addressing this shortcoming. Such a strategy could benefit from performing variable size steps based upon analysis of the partial solution at the end of last step — the number of remaining unallocated tasks in particular. Also the set of orderings could be pruned based on their respective performance in shorter running settings e.g. *Algorithm-B, Algorithm-FI*. Another option is a restart strategy reusing the partial solution σ from the previous step (keeping the agents' commitments), but such an approach didn't prove successful in our testing.

The last interesting conclusion concerns the overall convergence attributes of the presented algorithm. Considering the previously discussed high ratio of unsuccessful reallocations, the above mentioned high number of restarts and the

fact that the implementation is a prototypal one, rather than a fully optimized one, the listed runtimes actually correspond to a very limited portion of the search space being traversed in comparison with the competing algorithms. This suggests that the algorithm navigates through the search space very efficiently providing for a very good convergence. This finding provides further evidence of the potential of the method and suggests that further research is needed to unlock its full potential.

6. CONCLUSION

This paper describes an algorithm for the VRPTW based on agent negotiation. The performance of the algorithm is evaluated using the well known benchmark sets in an effort to assess the relevance of agent based approaches to routing problems in general. The algorithm outperforms all previously presented agent based algorithms, being also the first agent based algorithm to be tested using the extended benchmark sets typically used by centralized performance optimized solvers. Experimental results show that the algorithm is able to match the best known solutions achieved by the centralized solvers in 48.6% of the cases with an average relative error of 3.2% across all tested instances with respect to the VRPTW primary optimization criteria.

The algorithm uses a generic negotiation based task allocation process embedded in a multi-agent hierarchy that promises to be flexible in terms of capturing the semantics of typically heterogenous, dynamic and privacy conscious systems modeled within the transportation domain. For purposes of this paper, the adaptation to the VRPTW is achieved by supplying specific cost estimation functions for agent commitments and decommitments that can be easily extended or modified.

A comprehensive analysis of the algorithm is presented suggesting promising future research opportunities in: (i) modifying presented allocation process to employ more complex moves or allow for traversing non-feasible search space, (ii) developing a method to identify the best fitting ordering for a particular VRPTW instance, (iii) improving the restarts strategy for the VRPTW case and (iv) adapting the system for more challenging problem variants exploiting its inherent flexibility.

7. ACKNOWLEDGEMENTS

This work was supported by the Ministry of Education, Youth and Sports of Czech Republic within the Research program MSM6840770038: Decision Making and Control for Manufacturing III and also within the grant no. LD12044.

8. REFERENCES

[1] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part I route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[2] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II metaheuristics. *Transportation Science*, 39(1):119–139, 2005.

[3] A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38:369–378, August 2004.

[4] Z. Dan, L. Cai, and L. Zheng. Improved multi-agent system for the vehicle routing problem with time windows. *Tsinghua Science Technology*, 14(3):407–412, 2009.

[5] K. Fischer, J. P. Müller, and M. Pischel. Cooperative transportation scheduling: an application domain for dai. *Journal of Applied Artificial Intelligence*, 10:1–33, 1995.

[6] J. Homberger and H. Gehring. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, 162(1):220–238, 2005.

[7] R. Kohout and K. Erol. In-time agent-based vehicle routing with a stochastic improvement heuristic. In *11th Conference on Innovative Applications of Artificial Intelligence*. AAAI/MIT Press, 1999.

[8] H. W. Leong and M. Liu. *A multi-agent algorithm for vehicle routing problem with time window*, page 106Ű111. ACM, 2006.

[9] A. Lim and X. Zhang. A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 19(3):443–457, 2007.

[10] Q. Lu and M. M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with hard time windows. *European Journal of Operational Research*, 175:672–687, 2005.

[11] Y. Nagata and O. Bräysy. A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 37(5):333–338, 2009.

[12] Sintef. Sintef web pages - transportation optimization portal, problems section.

[13] P. Skobelev. Multi-agent systems for real time resource allocation, scheduling, optimization and controlling: Industrial applications. In V. Marík, P. Vrba, and P. Leitao, editors, *Holonic and Multi-Agent Systems for Manufacturing*, volume 6867 of *Lecture Notes in Computer Science*, pages 1–14. Springer Berlin / Heidelberg, 2011.

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

[15] J. Vokřínek, A. Komenda, and M. Pěchouček. Agents towards vehicle routing problems. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1 - Volume 1*, AAMAS '10, pages 773–780, Richland, SC, 2010. International Foundation for Autonomous Agents and Multiagent Systems.

[16] J. Vokřínek, A. Komenda, and M. Pěchouček. Abstract architecture for task-oriented multi-agent problem solving. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 41(1):31 –40, January 2011.