Agents Towards Vehicle Routing Problems*

Jiří Vokřínek
Agent Technology Center
Czech Technical University in Prague
vokrinek@agents.felk.cvut.cz

Antonín Komenda
Agent Technology Center
Czech Technical University in Prague
komenda@agents.felk.cvut.cz

Michal Pěchouček
Agent Technology Center
Czech Technical University in Prague
pechoucek@agents.felk.cvut.cz

ABSTRACT

A multi-agent VRP solver is presented in this paper. It utilizes the contract-net protocol based allocation and several improvement strategies. It provides the solution with the quality of 81% compared to the optimal solution on 115 benchmark instances in polynomial time. The self-organizing capability of the system successfully minimizes the number of vehicles used. The presented solver architecture supports great runtime parallelization with incremental increase of solution quality. The presented solver demonstrates applicability to the VRP problem and easy adaptation to problem variants.

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, heuristic, multi-agent solver, benchmarks

1. INTRODUCTION

The Vehicle Routing Problem (VRP) is a well-known optimization problem introduced in [3]. The problem is defined as routing of a fleet of gasoline delivery trucks between a terminal and a number of service stations. The trucks have load capacity limitations and deliveries have to be accomplished at minimum total cost (distance traveled).

This paper presents an agent-based solver producing feasible solution of the VRP instance in a polynomial time that doesn't use exhaustive searches or randomizing methods.

*Presented effort was sponsored by Czech Ministry of Education grant 6840770038 and by CERDEC, U.S. Army contract FSD BAA 8020902.A.

Cite as: Agents Towards Vehicle Routing Problems, Jiří Vokřínek, Antonín Komenda and Michal Pěchouček, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 773-780

1.1 Problem Statement

The VRP falls into the category of the NP-hard problems and so it is difficult to solve it in reasonable time. It is based on interdigitation of two underlaying problems that are also NP-hard – the Multiple Traveling Salesman Problem (MTSP) and Bin Packing Problem (BPP). A feasible solution to the VRP is a solution of MTSP that satisfies the capacity constraints (decision variant of BPP). By relaxation of one of the underlying problems (MTSP, BPP respectively), we can transform VRP into the other subproblem (BPP, MTSP respectively).

The Vehicle Routing Problem can be formalized as:

Definition 1. Let us have a set of cities $c_1, \ldots, c_k$ with known mutual distances and positive demands $d_1, \ldots, d_k$, the Vehicle Routing The problem is to find a set of m tours that together visit all nodes, each node is visited exactly once and by only one tour, the sum of the tours is minimal and the sum of the node demands served by each tour doesn't exceed the vehicle capacity C.

We define the set of tasks $T = \{n_1, ..., n_k\}$, where n_i is a doublet of a city and the corresponding demand:

$$n_i = (c_i, d_i), \tag{1}$$

$$\forall i \forall j : c_i \neq c_j \textbf{ iff } i \neq j.$$

The vehicle capacity constrain is defined for each route as:

$$\sum_{i=1}^{l} d_i \leq C, \tag{2}$$

where l is a number of tasks in the route and d_i is the demand of i^{th} task of the route. For ensuring feasibility of the solution we require:

$$\forall i : d_i \leq C. \tag{3}$$

It is obvious that for $d_i > C$, there is no way to handle this demand by a single truck and thus no solution for the problem exists.

In practical applications, the VRP is defined either with a fixed number of vehicles or as a problem with a minimal number of vehicles demanded. The determination of the minimal number of vehicles is a decision variant of BPP and is related also to determining the minimal number of routes for MTSP. Even though this problem is NP-hard, we can easily define the lower and upper bound of the number of vehicles (routes):

THEOREM 1. *The number of vehicles m in the feasible solution of the Vehicle Routing Problem is bounded by*

$$\frac{\sum_{i=1}^{k} d_i}{C} \leq m \leq k$$

where k is the number of cities, C is the capacity of the vehicle, and d_i is a demand of the i^{th} city.

PROOF. The lower bound of the number of vehicles respects the capacity constraints stated in Definition 1. For m vehicles the cumulative capacity of the whole fleet is $m \times C$ and the cumulative size of the demand is

$$\sum_{j=1}^{m} \sum_{i=1}^{l} d'_{i,j} = \sum_{i=1}^{k} d_i,$$

where $d'_{i,j}$ is a demand of the i^{th} city on the j^{th} route. Respecting Equation 2 the cumulative size of the demands in the feasible solution cannot exceed the cumulative capacity of the fleet. The upper bound is given by Equation 3 in the case where every demand is served by one route, i.e. $l = 1$ and $m = k$. □

1.2 Existing Solution Techniques

In the original Dantzig's article [3] the problem is formulated as a linear program providing a near-optimal solution. Classical solution techniques include wide variety of exact methods (branch and bound, branch and cut, set covering, spanning tree, shortest path relaxation, etc.), heuristics (constructive, two-phase, improvement, etc.), and metaheuristics (simulated annealing, tabu search, genetic algorithms, ant algorithms, neural networks, etc.). More information about solution techniques can be found for example in [6], [7], or [9]. This paper focuses on k-VRP (k is the number of cities), which is proven to be NP-hard and the best known approximation of the k-VRP is $\frac{5}{2} - \frac{3}{2n}$ for the metric case (triangle inequality is satisfied) [5].

The Agent-based approach to a variant of the VRP solver has been presented for example in [10]. Authors use three types of agents – Client, Bidder and Vehicle agents. The approach is based on the contract-net protocol (CNP) allocation and optimization based on exchange of tasks between the Vehicle Agents. The Vehicle Agents use an insertion heuristic and improvement strategy for task swapping between them. The error of the solution (compared to the optimal solution) presented in the paper is 4–29%.

A similar approach has been used in [1] for a dynamic variant of k-VRP (new tasks are added during the execution), where the initial allocation is generated using a centralized algorithm. The dynamic task allocation is made by the CNP protocol. Then two improvement phases are applied. The *intra-route optimization* is applied to each agent route and *inter-route optimization* is performed between Vehicle Agents (1 or 2 random tasks are moved between agents). The optimization is performed continuously during vehicle rides until all tasks have been fulfilled or a new dynamic task comes (1 hour interval in the experiments). The error of solution on all static requests has been reached 0–8%, but it is not described how much computation time has been used for static instances and also there is no discussion of stability and the speed of convergence of the solution quality (the optimization algorithm terminating condition is not defined).

In both [10] and [1] there is no discussion of the number of Vehicle Agents and handling of the potential allocation failure because of capacity constrains (no Vehicle Agent is capable of undertaking the next task) which can arise when the number of Vehicle Agents is lower than the upper bound defined by Theorem 1.

2. MULTI-AGENT SOLVER

The multi-agent planning approaches are used for solving a wide variety of planning problems. As analyzed in [2] the multi-agent planning techniques can be beneficial for such problems where the domain sizes of individual agents are considerably smaller (e.g. in logarithmic relation) than the overall size of the problem (even if the planning complexity of individual agent is exponential) and the number of dependencies between agents is low. Although the Vehicle Routing Problem consists of several NP-hard problems which may not satisfy the presented conditions (e.g. the BPP part produces an agent domain comparable to the overall problem size for the agent maintaining task distribution), this paper shows the applicability of the agent-based approach and discusses it's benefits and limitations.

2.1 Architecture

In this section we introduce the polynomial agent-based Vehicle Routing Problem solver producing a feasible solution, according to Definition 1, minimizing the routes' cost and the number of vehicles used. The solver is composed of three types of agents (see Figure 1). They are

- **Task Agent** for processing of demands and allocation invocation,
- **Allocation Agent** for maintaining allocation and the improvement process, and
- **Vehicle Agent** for route planning and optimization.

The multi-agent solver is composed of one Task Agent, one Planning Agent and a set V of Vehicle Agents (each agent utilizes one of the strategies described below). The number of Vehicle Agents respecting Theorem 1 is constrained by the lower and upper bound.

Since the agent-based solver doesn't use the exact method for solving the BPP part of the problem it may be impossible to find a solution using a minimal number of vehicles. In such a case, the whole process has to be restarted with an increased number of Vehicle Agents or an agent has to be added during the solving process. On the other hand, the upper bound of the number of Vehicle Agents guarantees the feasibility of solution but may produce worse results. Thanks to the agent paradigms, we can expect some sort of self-organizing behavior that emerges during the process of solution improvement. Let us formulate the following hypothesis:

HYPOTHESIS 1. *The implicit self-organizing ability of the agents emerges to the solution that corresponds to the minimal number of vehicles. The solutions produced by the system converge to the same number of vehicles regardless of the initial number of Vehicle Agents (provided that the solution is found).*

To solve the given VRP instance, we generate a set of solvers utilizing a different combination of strategies and

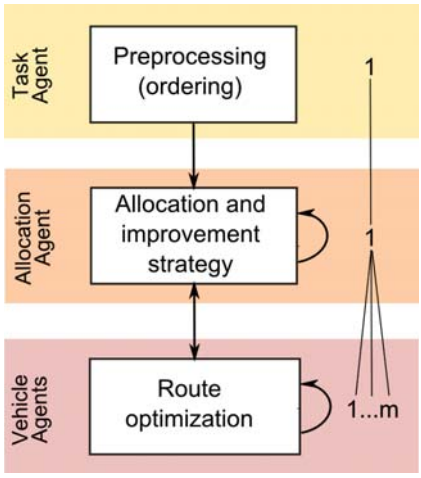

Figure 1: Generic architecture of agent-based VRP solver.

processing an input instance in parallel. The solvers produce solutions of the given problem independently and then the solutions are aggregated by the *Composite Solver* which selects the best one that minimizes the cost of the solution:

$$cost = \min_{S} cost(S), \tag{4}$$

where S is a solution provided by a solver and $cost(S)$ is the cost of the solution (see Equations 5 and 8). From the strategies described in the next subsections, we have built 42 solvers competing to answer the fundamental VRP question.

2.2 Task Agent

The Task Agent is responsible for collecting the demands and passing them to the Allocation Agent. It optionally applies the ordering preprocessing to the set of incoming tasks. It is able to pass the tasks to the Allocation Agent one by one (this represents the dynamic variant of VRP) or in a batch.

We suppose the ordering of the tasks directly influences the BPP part of the VRP. In the case of wrong ordering, the VRP solution may not be feasible for the lower bound of the number of Vehicle Agents because of capacity constraints (Equation 2). In the case of correct ordering, there may be a bigger chance to optimally allocate the tasks to the Vehicle Agents. These assumptions lead us to formulate the following hypothesis:

HYPOTHESIS 2. *Applying appropriate ordering to a set of tasks, the chance of finding the solution of VRP with a minimal number of vehicles is increased and the cost of the solution is decreased.*

Since the number of possible orderings increases rapidly with the size of the task set, we are not able to investigate all the orderings (in fact, the complexity of such an exhaustive search is comparable to solving the VRP itself). The First Fit Decreasing (FFD) heuristics is a classical BPP algorithm that has been recently proved [4] to provide the tight bound of $\frac{11}{9}OPT + \frac{6}{9}$ (e.g. when the instance of BPP can be optimally solved with 6 bins, the FDD is able to solve it with the maximum of 8 bins). Since it is a good algorithm for BPP, we decided to build the strategy of Task Agent inspired by this algorithm. The Task Agent strategy doesn't directly affect the allocation of the tasks to the Vehicle Agents, but it may have strong influence on the Allocation Agent strategy efficiency because of ordering capability. To inspect the influence of the Task Agent strategy, we have created also an opposite strategy to FFD for comparison.

The Task Agent uses the following three ordering strategies:

1. **Most Demand First** (MDF) based on the FFD, where the tasks are ordered with decreasing demands,
2. **Least Demand First** (LDF), which is opposite to MDF, so the tasks are ordered with increasing demands,
3. **First In First Out** (FIFO), where tasks are not ordered and their sequence corresponds to the time of arrival.

The time complexity of the Task Agent preprocessing (MDF and LDF strategies) corresponds to the complexity of standard sorting algorithms, which is $O(nlog(n))$. The complexity of the FIFO strategy is $O(1)$. Therefore the complexity of the Task Agent algorithm is $O^{TA} = O(nlog(n))$.

The task processing (passing them to the Allocation Agent) can be handled by one of two strategies:

1. **Batch processing** (NORM), where all available tasks are sent as one batch, and
2. **Iterative processing** (ITER), where task are sent one by one.

2.3 Allocation Agent

The Allocation Agent applies a defined strategy to allocate the tasks to the set of Vehicle Agents. The allocation strategy searches for the best suitable mapping of the tasks to the Vehicle Agents that minimizes the overall cost. The goal of Allocation Agent is to find such a partition $\mathcal{P}$ of the set of all tasks T that

$$\underset{\mathcal{P}}{\operatorname{argmin}} \sum_{i=1}^{v} cost(N_i), \tag{5}$$

where v is a number of Vehicle Agents, N_i is a subset of tasks allocated to the i^{th} Vehicle Agent, $cost(N_i)$ is a cost function computed by the i^{th} Vehicle Agent (Equation 8), and $N_i \subseteq T, \bigcup_{i=1}^{v} N_i = T$ where $\forall i, j : N_i \cap N_j = \emptyset$ **iff** $i \neq j$.

Equation 5 conforms to Definition 1, where Equation 2 is covered by Vehicle Agents.

The Allocation Agent algorithm consists of two phases: (i) *allocation phase* and (ii) *improvement phase*.

The first phase builds a feasible solution and the second performs incremental improvement. Both phases are captured by Algorithm 1.

Two allocation strategies implemented by the Allocation Agent are:

1. **Contract-net based** (CNP) – is based on the well-known contract-net protocol. For every task the best Vehicle Agent is selected according to insertion estimation (see Section 2.4) satisfying capacity constrains (Equation 2). This strategy contains no backtracking and in case of allocation failure (because of capacity constraint of Vehicle Agents) the whole process is restarted with a higher number of Vehicle Agents. This strategy is described by Algorithm 2.

2. **Capacity backtracking strategy** (BC) – is based on the previous strategy, but with backtracking in case of allocation failure. In case when no Vehicle Agent can undertake a new task because of the capacity constraint, the best Vehicle Agent is selected regardless of capacity limitations. This agent removes the worst tasks until the new task fits the increased free space. After that, the removed tasks are allocated again. The reallocation counter controls the number of reallocations and when it reaches the pre-defined maximum, the number of Vehicle Agents is increased and the process is restarted. This strategy is described by Algorithm 3.

The CNP strategy can fail because of capacity constrains and can be restarted up-to $k - \frac{\sum_{i=1}^{k} d_i}{C}$ times when the number of Vehicle Agents reaches upper bound (see Theorem 1).

The BC strategy can provide feasible allocation with a lower number of Vehicle Agents because of backtracking, but also can be trapped in an infinite reallocation loop. The reallocation counter helps to recover from this loop. In the worst case, the strategy is being restarted until the number of Vehicle Agents reaches the upper-bound and the BC provides the same result as CNP.

After successful allocation, the improvement phase takes place. We have designed three improvement strategies that Allocation Agent can use. The strategies are improvement heuristics that produce the same or better solution after each run. In all cases, the strategy is repetitively executed on all vehicle Agents until the solution overall cost (see Equation 5) stops improving (see Algorithm 1).

The improvement strategies are (see Algorithm 4 for more details):

- **Delegate worst** (DW) – each Vehicle Agent identifies it's worst task and tries to delegate it to another agent if the savings are higher than the insertion cost (see Equations 10 and 9).
- **Delegate all** (DA) – each Vehicle Agent delegates all its tasks (only if the savings are higher than the insertion cost).
- **Reallocate all** (RA) – each Vehicle Agent successively removes all its tasks from the plan and allocates it again using the CNP strategy. The result of the allocation can be the same as before task removing, or a change of the position of the task in the current agent plan, or delegation to another agent.

The results produced by the two-phase algorithm are influenced by the Task Agent processing strategy. The NORM strategy allows to allocate all tasks and then perform the improvement phase. The ITER strategy provides a high degree of *dynamism* where tasks are allocated one by one and the optimization is performed after allocation of each task.

The worst-case time complexity of the Allocation Agent algorithm is

$$O^{AA} = n \times O^{alloc} + n^2 \times m \times O^{impr}, \qquad (6)$$

where n is the number of tasks, m is the number of agents, O^{alloc} is the complexity of the allocation strategy, and O^{impr} is the complexity of the improvement strategy. The complexity of individual strategies is shown in Table 1 where rc is the reallocation counter threshold (constant) and O^{estI},

Algorithm 1 The Allocation Agent main algorithm.

```
function solve(T, V) begin
  forall t : T begin
    run allocation strategy for task t
    if allocation not successful then
      restart solver with increased
        number of Vehicle Agents
    end
  end
  improvement := true
  repeat until improvement is false
    improvement := false
    forall v : V begin
      run improvement strategy for agent v
      if solution has been improved then
        improvement := true
      end
    end
  end
end
```

Algorithm 2 Contract-net based allocation strategy.

```
function allocateCNP(t, V) begin
  forall v : V begin
    find winner with the lowest insertion
      estimation of t not exceeding capacity C
  end
  if winner is found then
    assign t to the winner
  else
    allocation not successful
  end
end
```

Algorithm 3 Capacity backtracking allocation strategy.

```
function allocateBC(t, V) begin
  allocateCNP(t, V)
  if allocation not successful then
    if reallocation counter is reached then
      allocation not successful
      return
    end
    forall v : V begin
      find winner with the lowest insertion
        estimation of t ignoring capacity constrain
    end
    while winner has not enough capacity for t begin
      remove the worst task of winner
        and put it to REALOC
    end
    assign t to the winner
    forall r : REALOC begin
      allocateBC(r,V)
    end
  end
end
```

Algorithm 4 Improvement strategies of Allocation Agent.

```
function improveDW(v,V) begin
  t = the worst task of agent v
  forall a : V \ v begin
    find winner with the lowest ins. estimation of t
  end
  if ins. cost of winner is lower then savings of v then
    swap t from v to winner
  end
end

function improveDA(v,V) begin
  forall t : tasks of agent v begin
    forall a : V \ v begin
      find winner with the lowest ins. estimation of t
    end
    if ins. cost of winner is lower then savings of v then
      swap t from v to winner
    end
  end
end

function improveRA(v,V) begin
  forall t : tasks of agent v begin
    remove t from agent v
    allocateCNP(t,V)
  end
end
```

O^{estR}, O^{worst}, O^{ins}, and O^{rem} are complexities of Vehicle Agent algorithms for task insertion/removal estimation, task insertion, finding the worst task, and task removal defined in the next section.

2.4 Vehicle Agent

The Vehicle Agent represents a single truck and is responsible for optimization of the route cost through the assigned tasks. It starts and finishes the route at the depot. In fact, this routing problem corresponds to the traveling salesman problem, where new customers come successively (as the Allocation Agent progress with task allocation).

Let $N = (n_1, \ldots, n_l)$ be a set of l tasks allocated to the agent, $I = (i : 1, \ldots, l)$ is an ordered index set, where $I \in \mathcal{I}, |I| = l$ and $\mathcal{I}$ is a set of all index permutations.

According to Definition 1 the objective function of Vehicle Agent can be formalized as follows:

$$\operatorname*{argmin}_{I \in \mathcal{I}} d(n_d, n_{I_1}) + \sum_{j=1}^{l-1} d(n_{I_j}, n_{I_{j+1}}) + d(n_{I_l}, n_d), \quad (7)$$

where $d(n_i, n_j)$ is a distance traveled between task i and j, and n_d is the depot, ensured that Equation 2 holds.

Given the I minimizing Equation 7, the cost function of the Vehicle Agent is then

$$cost(N) = d(n_d, n_{I_1}) + \sum_{j=1}^{l-1} d(n_{I_j}, n_{I_{j+1}}) + d(n_{I_l}, n_d). \quad (8)$$

The Vehicle Agent is able to compute the cost function during interactions with the Allocation Agent in the case of addition of new tasks, removal of an already assigned task, and estimation of adding/removing a task.

O^{CNP}	$m \times O^{estI} + O^{ins}$
O^{BC}	$rc \times (O^{CNP} + m \times O^{estI} + n \times O^{rem} + O^{ins})$
O^{DW}	$O^{worst} + (m-1) \times O^{estI} + O^{ins} + O^{rem}$
O^{DA}	$n \times O^{estR} + (m-1) \times O^{estI} + O^{ins} + O^{rem}$
O^{RA}	$O^{rem} + m \times O^{CNP}$

Table 1: Complexity of Allocation Agent strategies.

The algorithm used by Vehicle Agent is based on the well-known cheapest-insertion heuristics [8]. When inserting new tasks n_{l+1} the algorithm searches (in case of satisfying Equation 2) for the best suitable index j, where

$$\operatorname*{argmin}_{j \in I'} d(n_{I'_{j-1}}, n_{l+1}) + d(n_{m+1}, n_{I'_j}) - d(n_{I'_{j-1}}, n_{I'_j}), \quad (9)$$

and $I' = (d) \cup I \cup (d)$. The inner part of the Equation 9 is the insertion cost estimation. The new task n_{l+1} is then inserted on the position $k = j - 1$ in the agent's plan, e.g.:

$$\begin{aligned} I &:= (i_1, \ldots, i_{k-1}, l+1, i_k, \ldots, i_l), \\ N &:= N \cup n_{l+1}, \\ m &:= l + 1. \end{aligned}$$

The same heuristics is used for identification of the worst task (invoked by the delegation strategy of Allocation Agent), so the worst task n_j is such a task where:

$$\operatorname*{argmax}_{j \in I'} d(n_{I'_{j-1}}, n_{I'_j}) + d(n_{I'_j}, n_{I'_{j+1}}) - d(n_{I'_{j-1}}, n_{I'_{j+1}}), \quad (10)$$

i.e. the savings (the right part of the equation) of removing such tasks are maximized.

In the ***dynamic*** variant of the VRP, the algorithm is simply modified by constructing $I' = (c) \cup (i_p, \ldots, i_l) \cup (d)$ where n_c is the current position of the agent and n_{i_p} is the next task to be serviced. The plan is not searched from the beginning but from the current point of execution – new task cannot be inserted to the already traveled path $(n_{i_1}, \ldots n_{i_{l-1}})$. This modification has minimal impact on the functionality or the source codes of the whole system. The Vehicle Agent is also able to easily cover modifications of VRP such as the multi-depot variation, heterogenous capacities, additional route constraints, etc.

The complexity of insertion heuristics for inserting the i^{th} task is $O^{ins} = O(i)$, so the overall complexity of the planning l tasks allocated to an agent is $O(\frac{l^2}{2})$. The complexity of finding the worst task is the same as planning a task and insertion estimation, thus $O^{worst} = O^{estI} = O(i)$, removing one task costs only one operation as does removal estimation, thus $O^{rem} = O^{estR} = O(1)$.

2.5 Complexity

The general computational complexity of the multi-agent solver is introduced in [2]. Using transformation of the multi-agent planning problem to the distributed constraint satisfaction problem, the worst-case time complexity of the multi-agent planning is upper-bounded by

$$f(\mathcal{I}) \times exp(comm) + exp(int), \quad (11)$$

where $f(\cdot)$ is the factor inducted by requesting each agent to plan while committing to a certain sequence of actions, $\mathcal{I}$ is the complexity of an individual agent's planning, $exp(comm)$

represents a factor exponential in min-max number of per-agent commitments, and an additive factor $exp(int)$ represents the interactions of agents.

The consequences of Equation 11 lead to interesting features of the multi-agent solver, such as (i) no direct exponential dependence on the number of agents, (ii) no direct exponential dependence on the size of the planning problem or size of the joint plan, and (iii) no direct exponential dependence on the length of individual agent plans [2].

In our case the feature (ii) does not have a strong impact on the BPP part of the problem because in the worst case the size of the problem of our agents is the same as the size of the overall problem (for the MTSP part of the problem the feature holds). On the other hand the exponential factors are reduced by the polynomial heuristics – allocation and improvement strategies of the Allocation Agent and the insertion heuristic of the Vehicle Agent. The ordering strategy of the Task Agent does not have a strong influence on the worst case complexity because of its additive nature and low complexity.

By combining complexities of the individual agent strategies described earlier, we can define the complexity of the overall solver. Because of the space limitations of the article, we are not able to describe the complexity of every combination of strategies, so we present only the results here. The upper-bound of the worst-case time complexity of the solver is (taking into account the worst-case number of restarts, reallocations and backtracking)

$$O(n^3), \tag{12}$$

where n is the number of tasks (i.e. nodes or demands). There is no influence of the number of agents, but the number of Vehicle Agents is reflected by its dependency on the number of tasks (see Theorem 1) and increase the worst-case complexity exponent by one because of restarts towards the upper-bound number of vehicles.

2.6 Experiment Baseline Solver

For the evaluation purposes we have created a baseline solver reconstructing the optimal (or the best known) solution. It is based on the known results of the benchmark instances. It is composed of the presented three types of agents with special strategies. For the Task Agent it is:

- **Optimal order**, where tasks are ordered in an optimal way. This strategy is based on the previously known optimal solution. The order of the tasks is given by merging optimal routes, where tasks are ordered by their distance from the depot to the corresponding city computed on the vehicle route.

The Allocation Agent algorithm is composed only of the following allocation strategy (no optimization strategy is needed):

- **Optimal allocation**, where tasks are allocated to the Vehicle Agents according to routes in the known optimal solution.

The Vehicle Agent optimal strategy is bounded to the optimal allocation (it is obvious that with non-optimal allocation, the Vehicle Agent strategy is not able to produce the globally optimal solution). So the optimal Vehicle Agent strategy is:

- **Optimal route**, where tasks are ordered in the same way as in the known optimal solution.

This baseline solver is used for comparison in the experiments as a whole (reconstructing the best known solution) or as a part of the standard solver for investigating influences of the individual agents.

3. EXPERIMENTS

The presented solver has been evaluated on the VRP benchmark instances from two sources. The first source is VRPLIB[1] (s̈ymetric CVRP instances), and the second is a compilation of instances from the COIN-OR project[2]. All the instances use the Euclidean distance for edge weights and the number of nodes varies from 16 to 484. We have used 115 instances with a known solution (optimal or best known).

The comparison of solutions has been made against solutions reconstructed by the baseline solver (see Section 2.6). We measure the computation time and quality of the solution defined as:

$$\frac{cost_{solver}}{cost_{optim}} \times 100[\%], \tag{13}$$

where $cost_{solver}$ is the solution cost provided by the solver (see Equation 4) and $cost_{optim}$ is the cost of the best known solution. All the experiments have been run on a standard laptop with 3GB of RAM and 2.5GHz dual core processor. The solver has been implemented as JAVA application with no performance optimizations.

3.1 Solution Quality

The quality of the solution has been evaluated for the composite solver (using 42 parallel solvers with combination of strategies as described in Section 2). The aggregated results of all experiments show that the best solution quality reached is 100%, the lowest quality is 81.3% and the average solution quality over all instances is 91.3%. The results per instance (the x-axis represents instances with an increasing number of nodes to the right) can be seen in Figure 2 – the dots are the best solutions of the composite solver, the vertical lines represent the span of results of other solvers (e.g. non-winning strategies combinations). The results of the experiment also indicate that there is no dependence of the solution quality on tightness of the instance. The composite solver has reached the solution quality of more than 81% on all the benchmark instances and the average quality has been 91%. The quality of 100% has been reached for 3 instances, for 28 instances we reached quality better than 95% and there have been 63 instances with solution better than 90%.

3.2 Ordering Strategy

These experiments evaluate the influence of the Task Agent ordering strategy on the quality of the solution provided by the solver. The strategies of Task Agent are compared to the baseline Optimal Order strategy. Table 2 shows the solution quality of the solver through all instances aggregated for ordering strategies (solvers are divided into the groups according to the ordering strategy used and the best solution

[1]http://www.or.deis.unibo.it
page /research_pages/ORinstances/VRPLIB/VRPLIB.html
[2]http://www.coin-or.org
page /SYMPHONY/branchandcut/VRP/data/

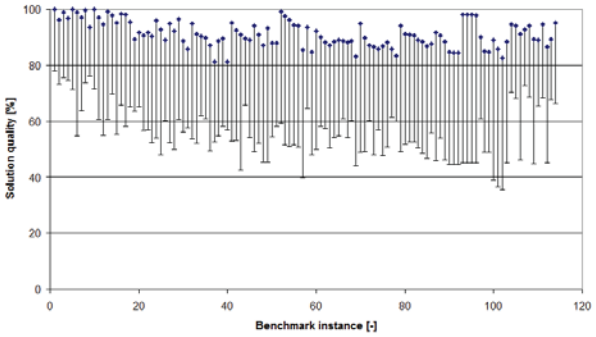

Figure 2: Solution quality of the composite solver per each benchmark instance.

Task Processing	max	min	average	deviation
MDF	100.0	75.83	87.62	6.30
LDF	99.1	68.46	87.06	7.39
FIFO	100.0	75.87	88.45	5.87
Optimal Ordering	100.0	69.8	88.84	6.39

Table 2: A comparison of the task processing methods. The values are the best results in form of solution quality.

is used for each instance and ordering). The best solution for all ordering strategies reaches 100% of solution quality and the average quality is almost the same for all evaluated ordering strategies. The worst performance provides LDF (also the biggest deviation of results). It seems to be provably worse than MDF, which looks like potential support of Hypothesis 2 and good influence of FFD heuristics for BPP. Unfortunately, the FIFO strategy provides a similar statistical performance as MDF. Moreover, the Optimal Ordering does not show big difference of the solution quality, so we can state that **Hypothesis 2 is refused**. We have found no strong influence of the selection of the ordering strategy on efficiency of the solver, we have not even found influence of usage of the FFD heuristic to the solution quality.

3.3 Insertion Heuristic

This experiment demonstrates the error of the Vehicle Agent insertion heuristic. The solution of a baseline solver is compared to the solution of the solver with the baseline Task Agent and Allocation Agent (i.e. optimal Order and optimal Allocation) and the optimal number of regular Vehicle Agents.

The maximum reached solution quality for this solver setting has been 100%, minimum 80% and the average quality has been 95%. The average error of the insertion heuristics computed for each Vehicle Agent route's individually has been 0.8%, the maximal single route error has been 9%.

3.4 Strategies Composition

This experiment evaluates the efficiency of Allocation Agent strategies combined with a different task processing strategy of Task Agent. The effectiveness of the used strategies can be expressed as an aggregation of the ranks across the benchmark instances and solvers (see Figure 3).

The strategies utilized by the solver are expressed using their abbreviations as defined in Section 2 in the form:

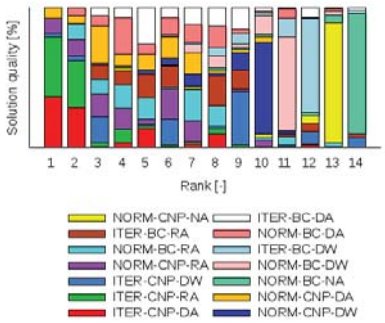

Figure 3: A histogram showing which task processing, allocation, and optimization strategies are more effective in all instances.

task processing strategy – allocation strategy – improvement strategy. If no improvement strategy is used, it is denoted as `NA`. We have run the solvers for all task ordering strategies and selected the best results to eliminate the influence of the task ordering strategy. The histogram in Figure 3 shows the `ITER-CNP-DA` together with `ITER-CNP-RA` are in 78% of the instances ranked as the best generic solvers. On the other hand, the worst strategies are `NORM-BC-NA` and `NORM-CNP-NA` which are ranked as last and one before last in 86% and 85% of the instances. Strategies `NORM-CNP-DW`, `NORM-BC-DW`, and `ITER-BC-DW` also show considerably bad effectiveness. These occupy ranks from 10–12 (out of 14) of the chart in 66% of instances.

3.5 Computation Time

This experiment focuses of the computation time of the composite solver. Figure 4 shows the average computation time of all combinations of strategies and orderings on the benchmark instances. During the time, the composite solver provides the solution presented in Figure 2. The x-axis represents the number of nodes in the instance. The y-axis represents the processing time of the composite solver (the aggregated time of 42 solvers for each instance). For all instances, the first solution (from one of the 42 solvers) has been obtained in a few milliseconds (11 milliseconds for the largest instance with 484 nodes) and the longest processing time of a single solver is close to 10 seconds (again for the instance with 484 nodes). The lowest processing time of the composite solver has been 6 milliseconds (*P-n16-k8*) and the longest time has been 49 seconds (*E484-19k*). The computation time of the solver in the experiments is upper-bounded by $\frac{n^3}{1500}$, which is better than the worst case complexity denoted by Equation 12.

3.6 Number of Vehicle Agents – Optimization

In this experiment we investigate the influence of the initial setting of the number of Vehicle Agents. We have compared two settings – the lower-bound and upper-bound.

Thanks to improvement strategies the solution provided by the solver starting with a lower-bound number of Vehicle Agents (L-B) and the solver with an upper-bound number of Vehicle Agents (U-B) is the same. On all instances the solution of U-B solver converged to the exactly the same solution as L-B solver. The only difference was the instance *A-n62-k8*, where the U-B solver provides the solution of quality

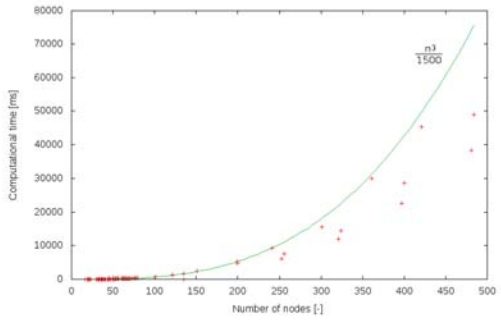

Figure 4: Computation time (aggregated) of the algorithm for all instances.

Number of vehicles	max	min	average	deviation
Unlimited	100.0	81.82	92.22	4.95
Minimal	100.0	76.95	91.0	5.64

Table 3: A comparison of solution quality of the best solutions by the numbers of vehicles.

0.88%, but the L-B solver stuck on 0.86% (both produce the solution using 8 vehicles, which is the optimal value). The computation time of the U-B solver was 2 times longer than L-B solver on average, in the worst case it was 5.4 times longer and in the best case 1.6 times faster (it was slightly faster in 6% of all cases). In the case of the U-B solver, the allocation and improvement strategies reduce the number of used Vehicles Agents and the solution converge to the solution of L-B solver. This experiment demonstrates the emergent self-organization ability of the agent-based system and **strongly supports Hypothesis 1**.

3.7 Number of Vehicle Agents – Constraint

This experiment focuses on the constrained number of vehicles. The solution obtained by the composite solver doesn't always meet the potential constraint for a minimal number of vehicles used (the BPP part of the VRP). A solution using the minimal number of vehicles has been produced by the solver for 92 instances. Table 3 shows the difference between the best solutions produced by the composite solver constrained by the minimal number of vehicles (given by the known optimal solution) and the solution with an unlimited number of vehicles. In the constrained case, some of the solvers do not provide the solution and thus the composite solver provides worse result. For three particular instances (*E421-41k*, *P-n55-k8*, *P-n55-k15*), the solution with an optimal number of vehicles could not be found using any combination of solver strategies (those three are not included in the computation of values for the case with minimal number of vehicles in Table 3).

4. CONCLUSIONS

We have designed and evaluated a multi-agent VRP solver that provides a feasible solution in polynomial time. The quality of the solution has been over 81% of all benchmark instances that greatly outperforms the known lower-bound approximation. The time complexity of the solver on experimental instances has been upper-bounded by $O(n^3)$. We have defined the bounds for the Vehicle Agents and shown that thanks to self-organization principles the presented agent-based solver converges to the same solution when starting from the lower bound and upper bound number of vehicles.

The presented solver architecture supports great runtime parallelization and it is able to provide the first solution in a very short time with good solution quality (for the biggest instance the first solution is produced under 10 milliseconds with 50% solution quality).

The results show that iterative task processing provides better results than batch processing and the backtracking strategy is not very efficient. The best performance has been reached with iterative task processing and ordinary CNP allocation with Delegate All and Reallocate All improvement strategies. The possible extension of the solver would be to introduce more improvement strategies and the mechanism of their runtime combination but in exchange of the computation time (and also worst-case complexity) increase. Another extension should be randomization of the task ordering strategies.

The presented solver demonstrates very good applicability to the VRP problem and easy adaptation to problem variants. Generalization of this approach to other problems (e.g. multiple traveling repairman problem and its variants) seems to be a promising way of future research.

5. REFERENCES

[1] D. Barbucha and P. Jedrzejowicz. Agent-based approach to the dynamic vehicle routing problem. In *7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009)*, volume 55 of *Advances in Soft Computing*, pages 169–178. Springer Berlin / Heidelberg, 2009.

[2] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS*, pages 28–35, 2008.

[3] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

[4] G. Dósa. The tight bound of first fit decreasing bin-packing algorithm is ffd(i) <= 11/9opt(i) + 6/9. In *ESCAPE*, pages 1–11, 2007.

[5] M. Haimovich, R. Kan, and L. Stougie. *Vehicle routing : methods and studies*, chapter Analysis Of Heuristics For Vehicle Routing Problems, pages 47–61. Elsevier, Amsterdam, 1988.

[6] G. Laporte, M. Gendreau, and J.-Y. Potvin. Classical and modern heuristics for the vehicle routing problem. Technical report, GERAD, 1999.

[7] Y. Marinakis and A. Migdalas. Annotated bibliography in vehicle routing. *Operational Research*, 7(1):27–46, January 2007.

[8] D. J. Rosenkrantz, R. E. Stearns, and P. M. L. II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581, 1977.

[9] P. Toth and D. Vigo. *The Vehicle Routing Problem, Monographs on Discrete Mathematics and Applications*. Society for Industrial and Applied Mathematics, U.S., 2001.

[10] B. Zeddini, M. Temani, A. Yassine, and K. Ghedira. An agent-oriented approach for the dynamic vehicle routing problem. *International Workshop on Advanced Information Systems for Enterprises*, 0:70–76, 2008.