Internal Report 2011–14 August 2011

Universiteit Leiden
Opleiding Informatica

Active Guided Evolution Strategy

for

Dynamic Vehicle Routing Problems

Maarten Groeneweg

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Active Guided Evolution Strategy for Dynamic Vehicle Routing Problems

Maarten Groeneweg
Supervisor: Michael Emmerich

August 17, 2011

Universiteit Leiden

ABSTRACT

This thesis presents an implementation and the adaptation of the active guided evolution strategies to solve dynamic vehicle routing problems. In the dynamic routing problem new requests and uncertain values are revealed and handled during execution of a routing schedule. Heuristics for creating solutions as well as adapting them to events are described. The used heuristics consist of a combination of route construction, guided local search and evolution strategies.

1. INTRODUCTION

Vehicle routing problems (VRP) describe the challenges of determining the routing schedule of a fleet of vehicles visiting customers at their locations. These challenges are widely faced in the logistics of transportation and delivery of goods or persons, as well as in the work of mobile service providers. In a vehicle routing problem the routes of vehicle are planned, where vehicles have to start from a depot location and visit a number of customers to complete their requests before returning to the starting depot. Each of the customer requests have to be completed by exactly one of the vehicle to satisfy the problem. Determining the number of vehicles, distributing the requests among them and planning the order of the visits gives a solution for the problem. The large amount of possible combinations possible in these solutions leads to a complex problem for which no optimal solution can be found through deterministic algorithms.

The vehicle routing problem is an optimisation problem with the goal of minimising the cost of the vehicles and travel. Usually this requires minimising the number of vehicles and the length of the routes. Besides the requirement of visiting each request the problems are commonly limited by additional constraints:

Capacitated Vehicle Routing Problems (CVRP) [5] model the limitations of vehicle capacity. The sum of the demands on that capacity of the requests visited by a vehicle can not exceed that vehicles capacity. Normally vehicle fleets are used with a uniform capacity for each vehicle.

Vehicle Routing Problems with Time Windows (VRPTW) [4] model the constraints customers place on the time of delivery. A time window is given for each request during which the visiting vehicle must arrive. Vehicles arriving early can wait until the customer is ready for them, but late arrivals violate the constraint. This is taken as a soft constraint by adding a penalty to the route cost.

Vehicle Routing Problems with duration limits have an upper bound on the total time or distance for each route. Vehicles have to be back at the depot before reaching this limit.

Pick-up and Delivery Vehicle Routing Problems [2] have paired requests, requiring a pick-up of goods at one location before the delivery of those goods to the second location by the same vehicle. This problem type will not be subject of this study.

The common availability of mobile communication has led to increasing options to deal with the uncertainties of real world use and to provide customers with faster service by scheduling new or changed requests as soon as possible. Dynamic vehicle routing problems (DVRP) allow the problem instance to be

changed at any time while the solution schedule is being executed. New information that was not available while planning the solution can be revealed at a later time. This allows new requests from customers to be included in an active route. Uncertain values in the input data can also be changed as the early estimations are adjusted to the final actual values. When new information becomes available events are generated to notify the solving algorithm. In response the solution has to be adjusted to match the changes. The demand for fast response times to these events places increased importance on the execution time of a solving algorithm. On the other hand real-time methods can continue to improve solutions during execution after the initial construction and after first event responses.

This study of dynamic routing problems will focus on mixed instances with part of the requests known a priori and part of the requests revealed during execution. During simulation of the solutions execution, travel times, service times and demands are given stochastic values that will be revealed after travel or processing of a request is completed. To avoid repeated course changes, vehicles are given their next destination after completing the previous request. When facing waiting time due to time window constraints vehicles may spend some of the waiting time before receiving travelling orders. During this time the route can still be changed to insert a different request with less waiting time.

2. PROBLEM FORMULATION

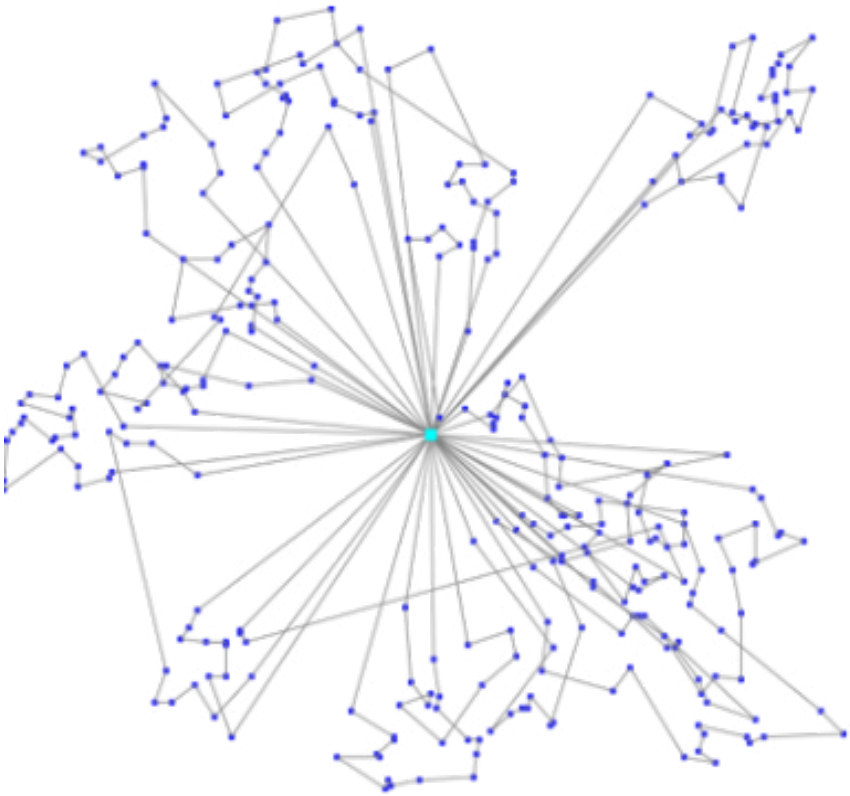

Fig. 2.1: A vehicle routing solution graph

The vehicle routing problem is modelled as a complete undirected graph $G = (V, E)$ following the notation of Mester and Bräysy [5] and using large parts of the problem formulation from Bent and Van Hentenryck [2]. The set of vertexes $V = \{v_0, v_1, \ldots, v_n\}$ represent the locations of the problem instance with the depot at v_0 and customers at the remaining vertices. Associated with

each customer index are the variables with the other input data.

$c_{i,j}$ = The distance cost between vertices v_i and v_j given as travel time.
q_i = The demand of goods for customer v_i ($q_i \geq 0, q_0 = 0$)
s_i = The service time to complete the request of customer v_i ($s_i \geq 0, s_0 = 0$)
e_i = The earliest time service can start for customer v_i ($e_i \geq 0$)
l_i = The latest time service can start for customer v_i ($l_i \geq e_i$)
l_0 = The duration limit.
Q = The maximum capacity of the vehicles.

A solution S for a vehicle routing problem is a routing schedule, consisting of a set of routes where each route is the path for one vehicle. A route is given as a sequence of locations to visit, starting and ending at the depot.

$$S = \{r_1, r_2, \ldots, r_v\} \tag{2.1}$$
$$r_i = \langle v_0, v_{i_1}, v_{i_2}, \ldots, v_{i_{n-1}}, v_{i_n}, v_0 \rangle \tag{2.2}$$
$$customers(r_i) = \langle v_{i_1}, v_{i_2}, \ldots, v_{i_{n-1}}, v_{i_n} \rangle \tag{2.3}$$

This schedules the vehicle to visit the $customers(r_i)$ in the given order. With the distance cost between the locations given, the travel cost of each route can be calculated as the sum of the costs of travelling from each visit to the next.

$$t(r_i) = c_{0,i_1} + c_{i_1,i_2} + \cdots + c_{i_{n-1},i_n} + c_{i_n,0} \tag{2.4}$$

A valid solution has each customer scheduled in a route while avoiding repeated visits to the same customer. With the problem being modelled as a complete graph, vertices only need to be visited to service requests, not as intermediate points in a travelling path.

$$\bigcup_{i=1}^{v} customers(r_i) = V \setminus v_0 \tag{2.5}$$
$$customers(r_i) \cap customers(r_j) = \emptyset \;\; \forall i, j \in \{1, \ldots, v\}, i \neq j \tag{2.6}$$

The used capacity q of a vehicle by the serviced requests is the sum of the demands of those requests. To allow more flexibility in intermediate solutions used during event handling in real-time VRPs, the capacity constraint is used as a soft constraint. The capacity violation v_c adds a penalty to the cost of a solution.

$$q(r) = \sum_{v_i \in r} q_i \tag{2.7}$$
$$v_c(r) = w_{v_c} \cdot \min(q(r) - Q, 0) \tag{2.8}$$
w_{v_c} = Penalty weight for capacity violations.

To check the time windows the arrival time a_i and departure time δ_i of visited requests have to be found. The earliest possible times can be derived from the departure time of the preceding requests in the route noted as i^-.

$$a_i = \max(\delta_{i^-} + c_{i^-,i}, e_i) \qquad 0 < i \leq n \tag{2.9}$$

$$\delta_i = \begin{cases} e_0 & i = 0 \\ a_i + s_i & 0 < i \leq n \end{cases} \tag{2.10}$$

Arriving later than the end of the time window adds a lateness violation penalty v_l to the solution cost.

$$v_l(r) = w_{v_l} \cdot \sum_{v_i \in r \setminus v_0} \min(a_i - l_i, 0) \tag{2.11}$$

$$w_{v_l} = \text{Penalty weight for time window violations.}$$

Looking at the time window limit for the remainder of the route can help skip some computations while evaluating operations. The time limit is the latest arrival time at a request which allows the remainder of the route including that request to be completed without exceeding time window constraints.

$$l_i^* = \min(l_i, l_{i^+}^* - c_{i,i^+} - s_i) \tag{2.12}$$

The cost of routes $C(r)$ adds the travel cost and the penalty costs. To find better solutions the number of vehicles $|S|$ and the total cost $C(S)$ have to be minimised. Solutions are evaluated by an objective function $g(S)$, using a lexicographic order to prioritise vehicle count over travel cost.

$$C(r_i) = t(r_i) + v_l(r_i) + v_c(r_i) \tag{2.13}$$

$$C(S) = \sum_{i=1}^{v} C(r_i) \tag{2.14}$$

$$g(S) = \big(|S|, C(S)\big) \tag{2.15}$$

3. IMPROVEMENT HEURISTICS

To solve complex vehicle routing problems in real time, heuristic methods are needed that provide good solutions with a limited amount of calculations. Based on the literature review of Van Wezel [7] the fastest performing algorithm is the active guided evolution strategy (AGES) from Mester and Bräysy [4, 5]. Since that algorithm also provided the second highest quality solutions, this algorithm has been selected for further study. The objective of this thesis is to adapt the AGES algorithm from the static VRP to a real time algorithm for the dynamic vehicle routing problem.

Active guided evolution strategies are a metaheuristic combining a construction heuristic and two improvement heuristics to apply the different strengths of the methods during different stages of solution construction. In the construction phase a solution is created from scratch based on a problem instances input data. The best solution from the construction phase forms the starting point for the improvement phase. The improvement heuristics explore the solution space of a current solution further by allowing non-improving operations. The current solution is compared by the objective function $g(S)$ (2.15) to the best known solution to detect new best solutions. The improvement phase uses one heuristic focussed mainly on improvement to rapidly optimise a solution. When that stage reaches a local minimum the second stage switches to a exploration focussed heuristic to find other minima.

3.1 Local Search

The basis for all stages is a local search for better solutions in the neighbourhood of a current solution. Whenever a local search for improvements is performed, all possible single applications of an improvement operator are evaluated. The operation with the highest cost savings is selected and applied to create the new current solution. These steps are reiterated as long as the best operation is improving the solution.

The operators used for the improvements are the relocate and 1-interchange operators illustrated in Figure 3.1. The relocate operator removes one request and reinserts it in a new position in a route. This route can be the same as or different from its previous route. The 1-interchange operator exchanges the route positions of two requests. This operation is used for inter-route moves only. Since the constraints time windows place on the order of requests, operators making larger changes provide less benefit on those problems and have not been used.

Because the evaluation of operations is repeated for all possibilities, they are the focus of the algorithms efficiency. Rather than re-evaluating the full solution for every operation each insertion or removal of a request from a route sequence is evaluated to efficiently find the effect of an operation. The cost change of the

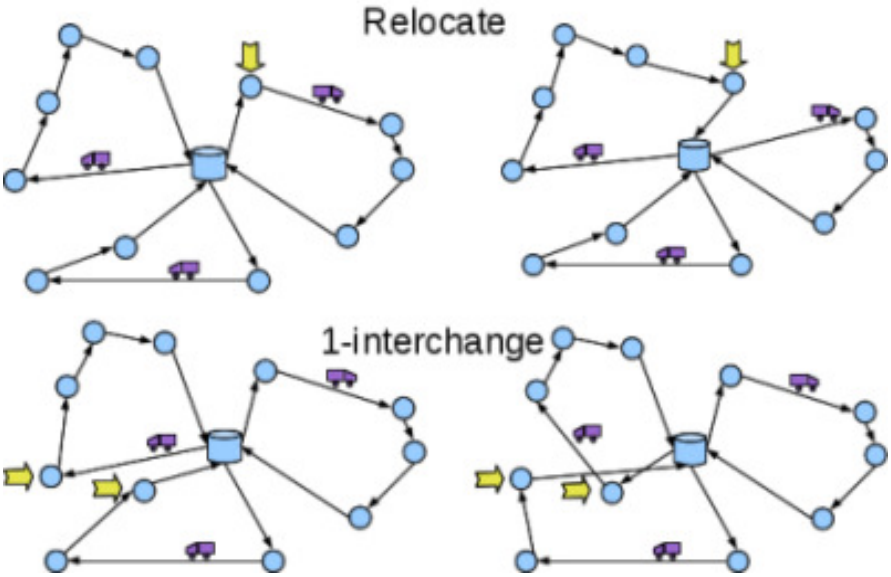

Fig. 3.1: Relocate and 1-interchange operators

travel time for an insertion of request i between p and n adds two new edges to and from the new request and removes the old edge. The cost is given by Osman [6] as $c_{p,i} + c_{i,n} - c_{p,n}$. For removals the cost is the negative of that and multiple steps taken for an operation add their cost changes together. By keeping track of the arrival and departure times of requests and the remaining free space of a vehicles capacity the cost of penalties can also be found as the difference between the before and after situations. All requests following an insertion can be affected by the changes in the time schedule however. The evaluation has to step through the following requests as well to find increased violations of the time windows. Storing the time limits l^* (2.12) for the remaining route can be used to stop that evaluation early. If the new arrival time of a request is below the remaining routes time limit and it was not exceeding its time window before, then evaluation of the change can stop there. By using these methods the evaluation of operations can be calculated in constant time for many common cases, although it has a worst case complexity in the order of the number of requests in the affected routes.

3.2 Cheapest reinsertion heuristic

To create an initial solution a simplified implementation of Mesters [4, 5] cheapest reinsertion heuristic is used. The solution is created initially with all known requests served by a separate route. For each request, all relocation positions to routes later in the search are evaluated. The request is then reinserted to the position with the cheapest cost if it is an improvement over the current position. The heuristic reiterates from the start as long as more emptied routes can be removed from the solution.

The cost evaluations for these reinsertions are modified by a weight parameter to influence the likelihood of reinsertions. Multiple starting solutions are

created with different values of the parameter to find the best basis for the rest of the algorithm. The cost changes for the removal and reinsertion are combined so that for the reinsertion of request i from in between h and j to a position between p and n the saving cost is:

$$(c_{h,i} + c_{i,j} + c_{p,n}) - \alpha(c_{p,i} + c_{i,n} + c_{h,j}) \tag{3.1}$$

The first term has the costs of edges removed by the reinsertion and the second term has the new costs. Unlike in Mesters original proposal of the algorithm, the difference of the costs before and after the reinsertion is taken similar to the other operations. In the original the costs of a position were added together, but this did not seem to make much difference. Another difference is that the solutions are not improved during construction. After the best solution is chosen from the different weighted constructions, the full solution is improved by the local search heuristic. By avoiding earlier searches more parameter values can be attempted resulting in better solutions, using 25 values between 0.1 and 1.5. However the full local search is often a performance bottleneck for large scale problems.

3.3 Guided local search

The guided local search metaheuristic is an adaptation of the local search by influencing it with an altered objective function. While the local search is still based on improving moves, improvements based on the modified objective function can be non-improving for the original objective. This gives a improvement focused heuristic that can escape from some local minima. The method of the guided local search is similar to tabu search, it tries to minimise costs by removing the graph edges contributing the most to the cost. The heuristic guides the local search by increasing the cost of edges it wants to discourage further. Higher costs of an edge increase the saved cost of its removal which increases the possibilities of those operations.

The GLS objective function is based on the regular objective function with the travel cost values replaced. The modified travel costs are the original values with penalties added to them.

$$c'_{i,j} = c_{i,j} + p_{i,j}\lambda L \tag{3.2}$$

$p_{i,j}$ = The number of penalties applied to the edge between vertices i and j.

$\lambda = 0.01$, A penalty weight parameter.

L = The average travel cost per edge in the initial solution.

To decide which edges should be penalised all edges in paths used by the current solution are evaluated for the utility of penalising them:

$$U_{i,j} = \frac{c_{i,j}}{(1 + p_{i,j})} \tag{3.3}$$

The edge with the highest utility is penalised by incrementing its penalty count by one. The utility function selects edges with a high travel cost, but reduces the utility when the edge has been tried repeatedly. This avoids getting stuck on a costly but necessary edge.

The two vertices connected by the edge are then used as the centre of the local search. To limit the number of possible operations to evaluate, the improvement phase restricts its search to one variable neighbourhood at a time. These neighbourhoods are formed by selecting a subset of routes from the solution. For the guided local search the neighbourhood is called a penalty variable neighbourhood (PVN) and is selected from routes close to the latest penalised edge. The neighbourhood start from the route containing the penalised edge. One of the vertices connected by the penalised edge is selected in alternating order. A search finds the vertex geographically nearest to the penalised vertex which is in a route that is not already present in the neighbourhood. This route is added to the neighbourhood. More routes are added until the total of the requests of the selected routes reaches l_{PVN}, a desired amount of requests. The target amount is chosen before every iteration of the GLS based on an upper bound set once.

$$l_{PVN} = l_{PVN}^{upper} a^2 \tag{3.4}$$

$$l_{PVN}^{upper} = (0.2 + 0.5 \times a^2) \times \text{nr. customers} \tag{3.5}$$

$$a = \text{Random value between 0 and 1.}$$

A local search is performed on the penalty variable neighbourhood, using the penalised objective function for its evaluations. When the local search can find no further improvements, the guided search can be repeated by finding a new edge to penalise in the full solution.

3.4 Evolution strategy

When the guided local search fails to find its way out of local minima as well, the second improvement metaheuristic is engaged for a slower but wider exploration of the search space. The evolution strategies search stage reuses the penalty variable neighbourhood from the earlier stage to restrict the size of the problem. The amount of data to work on is also limited to the use of a (1+1)-evolution strategy, from the one original neighbourhood a single new solution is generated through mutation of the route representation. A best-accept selection picks the best of those two solutions as the parent for the next iteration of the evolution strategies search.

For the evaluation of fitness used by the selection, the penalised objective function of the guided search is used, letting it guide the direction of the evolution strategies search as well. The mutation method used is a remove-insert scheme. A portion of the requests in the variable neighbourhood is removed, and are later reinserted one by one in the remaining routes. The requests to be removed are chosen randomly, and in each iteration a new number of requests to remove is picked. There is a 5% chance of selection all requests in the neighbourhood to remove, otherwise a random fraction between 0.2 and 0.7 of all the requests in the neighbourhood is chosen. The reinsertion of requests is done with the cheapest insertion heuristic of the construction phase. The parameter weighing the cheapest reinsertion costs is used to vary the mutation rate of the evolution strategy. The evolution strategies search is always executed as a set of 5 iterations, using one of the reinsertion parameters $(0.6, 0.8, 1.0, 1.2, 1.4)$ for each iteration. The child solution created is improved with the local search heuristic before getting evaluated and compared against its parent solution.

4. EVENT HANDLING

For dynamic vehicle routing problems a real-time optimisation agent is needed to maintain the solution. Every time a change occurs during execution of a routing schedule the optimising agent is notified through an event. The agent must apply the change and create a new solution feasible for the changed circumstances in as short a time as possible.

For the developed dynamic AGES algorithm event handling is divided in two stages. The first stage applies the events changes and re-optimises the altered solution through improving operations only. This stage creates the first response to the event. The second stage optimises the solution further by also using the non-improving operations of the guided local search and evolution strategies search. Better solutions found later can replace the routes being executed as long as only unvisited requests are changed. The event types investigated initially are based on the regular operation of routing executions.

Arrival events notify the arrival of a vehicle at the location of a request. At this point the final travel time from the vehicles previous location to that request is known and the time schedule for the remaining route can be updated.

Processed events notify the completion of service for a request. After completion the service time and demand used by the request are known. The updated solution for this event is used to provide the next destination of the relevant vehicle.

Departure events notify the optimising agent of the destination a vehicle has chosen as its next target and the time it has departed towards that destination. Once the next destination has been determined it can no longer be changed. Solutions should match this decision and avoid relocation the destination in improvement operations.

Dynamic request events are new requests revealed during execution. The new requests must be inserted into the pending portion of one of the routes in the solution. The larger changes induced by these events lead to most of the challenges of dynamic routing problems.

4.1 Applying events

To create an updated solution for an event the last best solution is used as the basis. The current solution undergoing improvements is discarded and the improvements are paused. The changes needed to apply the event are made to the solution, replacing changed values and cascading their effects through the rest of affected routes. For the insertion of new requests all insertion positions

in the pending sections of all routes are examined with the cheapest reinsertion evaluation. The weight parameter is not used for this case.

After the event is applied a first round of improvements is made on the solution to adjust it to the changes. Based on the same ideas of the penalty variable neighbourhood used by the guided local search, an event variable neighbourhood is defined. This neighbourhood is centred on the request affected by the event and consists of the geographically closest routes to that request. To create a new solution suitable for direct replacement only improving operations are used. This requires resetting all penalties applied by the guided local search to avoid them influencing improvements. The event variable neighbourhood is improved by the local search heuristic until no new improvements are found. A suitable response solution is ready at this point.

4.2 Improving events

After the event is applied improvement can continue on the affected routes. Because non-improving operations are used again these improvements keep a copy of the last best solution to compare against. The event improving stage keeps the event variable neighbourhood of an event to focus the follow-up improvements on that area. The same metaheuristics used for solution optimising are used to improve the event neighbourhoods. Guided local search is used by selecting an edge used in the neighbourhoods routes to penalise. Rather than forming a penalty variable neighbourhood the event neighbourhood is used for the following local search. When no improvements have been found for a number of iterations, the evolution strategies search is used instead. The evolution strategy is repeated until that fails to improve as well. At this point the events improvement is ended. When no event applications or improvements are remaining the regular improvement phase resumes to work on the full solution.

4.3 Slack requests

In dynamic problems with part of the requests revealed later, the solution created before execution of the routes start is based on limited information. With a smaller number of initial requests the solution will likely be over optimised. Requests added later can then not be inserted without large violations of capacity and time window constraints. Slack space has to be added in vehicle capacity and in the routes to reserve room for future requests. To do this, dummy requests are generated as a representation of the slack space. The created solutions will have to include those requests in their routes together with the regular requests. The advantage of these dummy requests is that slack space can be replaced with the real dynamic request on a one to one basis. This keeps the total number of real requests and slack space requests correct. Each time a dynamic request is inserted, the geographically closest slack request to it is dropped from the solution first.

The number of slack requests generated is based on an estimation of the number of dynamic requests. The slack requests are spread over the area of the regular requests, placed on random uniformly distributed coordinates. An even spread of locations encourages slack space to be divided equally among the routes, although this is not enforced. Basing the locations on a regular grid

distribution or on historical data could be investigated as well. The averages of the service times and demands of the initial requests are used to seed the slack requests. They do not use time windows.

5. EXPERIMENTS

The effectiveness of the algorithm was tested on the large scale test sets created by Gehring and Homberger [3]. The benchmark has test instances of 200, 400, 600, 800 and 1000 requests. Six problem types are used; the C-instances consists of clustered locations, the R-instances consists of randomly spread locations and the RC-instances have a mix of those two. The C1, R1 and RC1-instances have narrow constraints on the time windows, leading to routes servicing few requests in an order with little flexibility. The C2, R2 and RC2-instances have wider constraints, allowing long routes servicing many requests. For each combination of size and type there are 10 test instances. However, only the first of each has been used in tests due to time pressure. For comparison with other algorithms, the best known results from the SINTEF transportation optimisation portal [1] were used.

The test instances were turned into dynamic problems by taking 50% of the requests in the test set as dynamic requests, and leaving the remainder as initial requests. The dynamic requests are revealed at a uniformly distributed submission time between the start of the simulation and shortly before the end of the requests time window. A minimum submission time of 300 seconds before the closing of the time window is given to allow some time for scheduling. The travel distances, service times and demands of the tests sets are used as the input data for the algorithm. During simulation these values are randomized with a normal distribution with a mean centred on the original value and a standard derivation scaled to the size of that value. The travel time uses a standard derivation of 10% of its original value, service time and demand use 5% of its value.

The experiments ran on a PC with a Intel Core 2 dual core 2.66 GHz processor and 2 GB of memory. The optimising agent got 15 minutes to create an optimised solution based on the initial data. After that the execution of that solution was simulated, with communication of events and updated solutions continuing between simulator and optimising agent. The simulator used the time values interpreted as seconds and ran the time at double speed. The 1000 customer problems had trouble keeping up at that rate and were run with 30 minutes preparation time and normal speed simulation. To allow route changes while a vehicle is waiting for the opening of time windows, the vehicle spends the excess time at its previous location while leaving its next destination open to change. The estimation of waiting time is done with a safety margin of twice the standard derivation of the travel time to the current destination.

Problem	Vehicles	Distance	Overcapacity	Lateness
$w_{v_l} = 200, w_{v_c} = 10000$				
C1-6-1	65	17964	0	2235
C2-6-1	21	10069	0	82
R1-6-1	60	30243	3	577
R2-6-1	13	20663	0	18
RC1-6-1	64	24086	2	444
RC2- 6-1	16	14735	0	320
$w_{v_l} = 500, w_{v_c} = 100000$				
C1-6-1	64	18656	0	3406
C2-6-1	23	10909	0	40
R1-6-1	60	25642	3	271
R2-6-1	12	20667	0	34
RC1-6-1	62	21908	3	334
RC2-6-1	15	14408	0	262

Tab. 5.1: Effect of penalty weights.

Problem	Vehicles	Distance	Overcapacity	Lateness
Minimal submission time: 100s.				
C1-6-1	65	17964	0	2235
C2-6-1	21	10069	0	82
R1-6-1	60	30243	3	577
R2-6-1	13	20663	0	18
RC1-6-1	64	24086	2	444
RC2- 6-1	16	14735	0	320
Minimal submission time: 200s.				
C1-6-1	68	18351	0	356
C2-6-1	22	9663	0	51
R1-6-1	66	28451	0	250
R2-6-1	12	19730	0	38
RC1-6-1	80	24760	0	149
RC2-6-1	15	14189	0	211
Minimal submission time: 300s.				
C1-6-1	63	16554	0	165
C2-6-1	23	9268	0	1
R1-6-1	58	23319	4	128
R2-6-1	12	19771	0	14
RC1-6-1	60	19241	0	141
RC2-6-1	16	14075	0	133

Tab. 5.2: Effect of minimum submission time before closing of time windows.

5.1 Results

Some initial tests were done to sample the effect of some of the parameters on solution quality. The median problem sets of 600 requests were taken for these tests, with a simulation of the six instance types. These tests were mainly done to reduce the amount of lateness, the violation of time window constraints. The first attempt was to increase the weights for penalties. The results of this is in Table 5.1. Larger penalties give some improvements, but one large deterioration as well.

More significant results have been gotten from increasing the minimal time between submission and the end of the time window. These results are shown in Table 5.2. These constraint violations have gotten to an acceptable level. It does demonstrate a difficulty in responding to urgent new requests.

The effect of varying the ratio of dynamic and initial requests is recorded in

Problem	Vehicles	Distance	Overcapacity	Lateness
25% dynamic				
C1-6-1	65	16235.15	0	89
C2-6-1	23	9174.41	0	6
R1-6-1	61	23235.93	1	124
R2-6-1	13	19181.48	0	36
RC1-6-1	58	18395.04	1	184
RC2-6-1	18	13342.72	0	20
50% dynamic				
C1-6-1	61	15373.38	0	61
C2-6-1	23	9014.29	0	0
R1-6-1	59	23164.95	1	150
R2-6-1	12	19687.62	0	13
RC1-6-1	58	18634.74	2	67
RC2-6-1	16	14793.65	0	123
75% dynamic				
C1-6-1	63	15859.29	0	104
C2-6-1	22	9328.48	0	0
R1-6-1	58	22850.43	7	109
R2-6-1	11	20727.04	0	21
RC1-6-1	58	18689.07	0	140
RC2-6-1	13	14671.75	0	1243

Tab. 5.3: Effect of the percentage of dynamic requests.

Table 5.3. Surprisingly this does not appear to have much effect on the results. The numbers vary between different runs, but no clear direction to improvement or degradation can be found.

The quality of the algorithm without the effects of the dynamic problem was measured by running the static test instances until no improvements could be found. In Table 5.4 the results of the static tests are compared with the best known results from SINTEF [1]. A few matches with the best results where achieved. Most are slightly worse, requiring a few more vehicles, on average 6.19% more. With the extra vehicles the distances can often be reduced a bit, although the large 50% savings are due to the results from Blocho and Czech having much longer distances than other methods.

In Table 5.5 the results of the dynamic simulations are shown. Compared to the best known static results, the travel distances are quite a bit larger now. With the additional difficulties of a dynamic problem, this was to be expected. Most of the distances are about 10% larger, which seems reasonable enough. The total number of vehicles needed is very close to the results of the static problem with only 3 more vehicles used. Violations of constraints, especially of time windows, is unfortunately common. This should be a goal for further improvement.

The time taken by event handling is listed in Table 5.6, giving times for the first response after applying the event, and for finishing all improvements. The first response should usually come well within 100 milliseconds, which is a good result. Only for the largest problems does the maximum start going over a second. The finishing times for the improvement rounds are much larger, but these are not critical. Running over real world travel times instead of short simulations will free up even more time for event handling.

Problem	AGES Solution			Best known solution		Difference	
	Vehicles	Distance	Time	Vehicles	Distance	Vehicles	Distance
C1-2-1	20	2704.57	0:11	20	2704.57	0.00%	0.00%
C2-2-1	6	1931.44	3:57	6	1931.44	0.00%	0.00%
R1-2-1	22	4733.71	1:41	19	5024.65	15.79%	-5.79%
R2-2-1	5	4080.27	7:42	4	4483.16	25.00%	-8.99%
RC1-2-1	20	3552.86	1:18	18	3602.80	11.11%	-1.39%
RC2-2-1	7	2958.94	12:37	6	3099.53	16.67%	-4.54%
C1-4-1	40	7152.06	1:53	40	7152.02	0.00%	0.00%
C2-4-1	12	4116.33	11:02	12	4116.05	0.00%	0.01%
R1-4-1	41	10567.38	4:00	38	11084.00	7.89%	-4.66%
R2-4-1	10	8718.49	24:53	8	9213.68	25.00%	-5.37%
RC1-4-1	39	8775.06	11:34	36	8630.94	8.33%	1.67%
RC2-4-1	15	6316.50	14:59	11	6688.31	36.36%	-5.56%
C1-6-1	60	14095.64	9:27	60	14095.64	0.00%	0.00%
C2-6-1	22	8512.26	28:16	18	7774.10	22.22%	9.50%
R1-6-1	59	21907.18	15:10	59	21131.09	0.00%	3.67%
R2-6-1	14	17512.17	38:39	11	18291.18	27.27%	-4.26%
RC1-6-1	57	17536.25	22:25	55	17317.13	3.64%	1.27%
RC2-6-1	21	12347.89	1:02:47	14	25524.13	50.00%	-51.62%
C1-8-1	80	25184.38	5:13	80	25030.36	0.00%	0.62%
C2-8-1	24	11703.70	2:17:05	24	11654.81	0.00%	0.42%
R1-8-1	80	38653.40	48:31	79	39612.20	1.27%	-2.42%
R2-8-1	17	28031.67	2:11:56	15	28392.87	13.33%	-1.27%
RC1-8-1	77	31323.95	51:03	72	35102.79	6.94%	-10.77%
RC2-8-1	26	19676.26	1:27:02	18	42243.22	44.44%	-53.42%
C1-10-1	100	42529.43	1:19:27	100	42478.95	0.00%	0.12%
C2-10-1	34	18129.29	2:44:32	30	16879.24	13.33%	7.41%
R1-10-1	100	55251.62	33:10	100	53904.23	0.00%	2.50%
R2-10-1	20	44410.67	3:15:20	19	42467.87	5.26%	4.57%
RC1-10-1	92	48113.40	3:22:43	90	47143.90	2.22%	2.06%
RC2-10-1	29	29729.47	3:28:40	20	63373.15	45.00%	-53.09%
Total	1149	550256.24		1082	620148.01	6.19%	-11.27%

Tab. 5.4: Results of static vehicle routing problem instances.

Problem	AGES Solution		Constraint violations		Best known solution		Difference	
	Vehicles	Distance	Capacity	Lateness	Vehicles	Distance	Vehicles	Distance
C1-2-1	20	2748.80	0	15	20	2704.57	0.00%	1.64%
C2-2-1	8	2054.72	0	0	6	1931.44	33.33%	6.38%
R1-2-1	21	4864.02	0	6	19	5024.65	10.53%	-3.20%
R2-2-1	5	4116.14	0	2	4	4483.16	25.00%	-8.19%
RC1-2-1	20	3598.91	0	11	18	3602.80	11.11%	-0.11%
RC2-2-1	6	3453.84	0	4	6	3099.53	0.00%	11.43%
C1-4-1	41	7775.80	0	77	40	7152.02	2.50%	8.72%
C2-4-1	15	4713.67	0	0	12	4116.05	25.00%	14.52%
R1-4-1	42	10820.67	2	69	38	11084.00	10.53%	-2.38%
R2-4-1	9	9254.28	0	17	8	9213.68	12.50%	0.44%
RC1-4-1	41	9523.66	1	123	36	8630.94	13.89%	10.34%
RC2-4-1	10	7496.46	0	570	11	6688.31	-9.09%	12.08%
C1-6-1	61	15373.38	0	61	60	14095.64	1.67%	9.06%
C2-6-1	23	9014.29	0	0	18	7774.10	27.78%	15.95%
R1-6-1	59	23164.95	1	150	59	21131.09	0.00%	9.62%
R2-6-1	12	19687.62	0	13	11	18291.18	9.09%	7.63%
RC1-6-1	58	18634.74	2	67	55	17317.13	5.45%	7.61%
RC2-6-1	16	14793.65	0	123	14	25524.13	14.29%	-42.04%
C1-8-1	85	28383.13	0	342	80	25030.36	6.25%	13.39%
C2-8-1	32	15734.63	0	19	24	11654.81	33.33%	35.01%
R1-8-1	78	40972.93	0	294	79	39612.20	-1.27%	3.44%
R2-8-1	15	32075.80	0	54	15	28392.87	0.00%	12.97%
RC1-8-1	76	32908.67	1	542	72	35102.79	5.56%	-6.25%
RC2-8-1	20	22064.58	0	119	18	42243.22	11.11%	-47.77%
C1-10-1	103	48920.02	3	13082	100	42478.95	3.00%	15.16%
C2-10-1	40	21932.43	0	59	30	16879.24	33.33%	29.94%
R1-10-1	96	59854.60	5	592	100	53904.23	-4.00%	11.04%
R2-10-1	20	46068.62	0	68	19	42467.87	5.26%	8.48%
RC1-10-1	96	52564.38	2	941	90	47143.90	6.67%	11.50%
RC2-10-1	24	33298.86	0	108	20	63373.15	20.00%	-47.46%
Total	1152	605868.25			1082	620148.01	6.47%	-2.30%

Tab. 5.5: Results of 50% dynamic vehicle routing problem instances.

Problem	Event application		Event improvement	
	Average (ms)	Maximum (ms)	Average (ms)	Maximum (ms)
C1-2-1	2	29	77	1.518
C2-2-1	2	17	67	598
R1-2-1	2	25	47	438
R2-2-1	4	69	269	2.751
RC1-2-1	1	27	48	359
RC2-2-1	5	110	285	3.965
C1-4-1	6	50	93	1.657
C2-4-1	16	206	1.419	62.095
R1-4-1	9	103	327	6.851
R2-4-1	13	253	751	42.799
RC1-4-1	37	1.002	50.450	86.914
RC2-4-1	16	383	995	19.745
C1-6-1	23	227	741	28.264
C2-6-1	42	838	63.288	213.077
R1-6-1	20	317	468	9.399
R2-6-1	30	920	3.275	65.008
RC1-6-1	28	325	20.592	58.022
RC2-6-1	58	2.236	110.417	225.515
C1-8-1	52	1.206	41.060	119.962
C2-8-1	49	2.005	38.420	191.869
R1-8-1	86	3.069	114.290	211.250
R2-8-1	75	1.932	134.556	396.650
RC1-8-1	48	1.186	38.583	84.006
RC2-8-1	59	867	79.509	208.905
C1-10-1	104	2.953	128.630	287.871
C2-10-1	75	951	26.906	178.082
R1-10-1	92	1.068	50.962	159.695
R2-10-1	100	3.273	125.118	574.476
RC1-10-1	195	7.170	292.949	515.590
RC2-10-1	276	7.909	513.091	1.230.120

Tab. 5.6: Event handling times in milliseconds for dynamic problem instances.

6. CONCLUSIONS

The developed method of solving dynamic vehicle routing problems with a real-time optimising agent is functioning. It can create a solution and keep it updated to match events. First response times are usually a fraction of a second, the effort invested in the implementation of efficient evaluations and concurrent processing of events has paid off here. The use of dummy slack request to reserve space for dynamic requests also does its job, resulting in a similar amount of vehicles used in static and dynamic problems.

The quality of the generated solutions has more room for improvement to be able to match other implementations. Travel distances do get increased a bit from the disruptions of dynamic insertions and stochastic travel times. The initial full improvement search in the construction phase could also use improvement in its execution time. The enforcement of soft constraints on the solution has turned out fairly unreliable despite high penalty weights. The use of hard constraints might have been a better choice in hindsight, perhaps using a combination of enforcing hard constraints during the construction and improvement phases while allowing soft constraints for the insertion of dynamic requests.

BIBLIOGRAPHY

[1] Transportation optimization portal of sintef applied mathematics. `http://www.sintef.no/Projectweb/TOP/`, retrieved on 8-8-2011.

[2] R. Bent and P. van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & OR*, 33:875–893, 2006.

[3] H. Gehring and J. Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In K. Miettinen, M. Mäkelä, and J. Toivanen, editors, *Proceedings of EUROGEN99*, number A 2 in Series A. Collections, pages 57–64. University of Jyväskylä, 1999. Benchmark data: `http://www.fernuni-hagen.de/WINF/touren/inhalte/probinst.htm`, retrieved on 11-5-2011.

[4] D. Mester and O. Bräysy. Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Computers & OR*, 32:1593–1614, 2005.

[5] D. Mester and O. Bräysy. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers & OR*, 34:2964–2975, 2007.

[6] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[7] M. van Wezel. Literature review, problem variants, algorithms and research directions for the real-time vehicle routing problem in the deliver project, 2010.