A Reactive Method for Real Time Dynamic Vehicle Routing Problem

Kenny Qili Zhu and Kar-Loon Ong
Dept. Computer Science, National University of Singapore, Singapore 119260
{kzhu, ongkarlo}@comp.nus.edu.sg

Abstract

Real Time Dynamic Vehicle Routing Problem (RTDVRP) is an extension to VRPTW, in which the problem parameters change in real time. We present a solution to RTDVRP: a concurrent, agent-based Reactive Vehicle Routing System (RVRS) and the implementation of the RVRS, which combines a generic, concurrent infrastructure and a powerful incremental local optimization heuristic.

1 Introduction

VRPTW is a well-known *NP*-hard problem. The objective of the problem is to find routes for the vehicles to service all the customers at a minimal cost (in terms of travel distance, etc.) without violating the capacity and travel time constraints of the vehicles and the time window constraints set by the customers. Almost all VRPTW methods proposed are devoted to a static problem as the problem parameters and constraints have to be known in advance. Any change in these data requires a complete re-calculation. Changes after the vehicles are deployed are forbidden. Until now, little research has been focused on the dynamic VRPTW, where problem size and parameters change after the vehicles are already commissioned. Some of the earliest work on dynamic vehicle routing problem was from Bertsimas and Ryzin [1] which is essentially a generic mathematical model with the waiting time as an objective function. Some further work can be found on Powell, Jaillet and Odoni [6], Psaraftis [7] and Gendreau [3]. The emphases were placed on the problem modeling, solving approaches and algorithms, and not on the real time concurrency and reactivity issues. Our work presents a new framework to deal with these issues. We also differ from the previous work in the problem modeling as well as objective function construction: we adopt the traditional VRPTW and maintain travel distance as the objective function.

In our research, we describe a *Real Time Dynamic Vehicle Routing Problem (RTDVRP)*, an extension to traditional VRPTW, whose problem parameters change in real time when vehicles are already commissioned. The problem is reactive in nature, and its solution benefits from a concurrent, reactive framework. Here we view vehicles as agents that communicate with each other through a common VRPTW solver store. The medium of communication is *reactors*, simple atomic programs that involve a *condition*, an *action* and a *time-out*. The relationship between vehicles can be represented by constraints as one vehicle's breakdown or slowdown can affect other vehicles' designated routes. In this sense, the VRPTW solver is also a constraint solver that governs the interactions between various vehicle agents. To facilitate the solving of RTDVRP, we introduce a *reactive vehicle routing (RVR)* infrastructure, which is derived from the *Open Constraint Programming (OCP)* [4] concept. The main component of the infrastructure is a registry that handles concurrent submission, serialization, delay and wake-up of the reactors. The VRPTW solver uses an *incremental local optimization (ILO)* algorithm, and a persistent memory store. It solves the VRPTW on a dynamic basis without solving the entire problem, hence it is quick and effective.

In this paper, we first introduce the concept of RTDVRP, followed by the RVR functional framework. Then we present *Reactive Vehicle Routing System (RVRS)*, an implementation of RVR infrastructure and a simulation based on RVRS. Finally we summarize the the experimental results.

2 Description of RTDVRP

The traditional VRPTW is given by a set of identical vehicles, a special node called the depot, a set of customers to be visited, a directed network connecting the depot and the customers [10]. A route is defined as starting from the depot, going through a number of customers and ending at the depot. A cost

d_{ij} and a travel time t_{ij} are associated with each arc of the network. Every customer in the network must be visited only once by one of the vehicles. The objective is to serve all customers with minimum vehicles and travel distance without violating time window and capacity constraints. In reality, traffic conditions change dynamically, causing initial best assignment to be invalid. For example, one vehicle may break down and another vehicle has to change its route to substitute for the breakdown vehicle. In addition, the time window of a customer can be changed on the fly, so original routes may not apply any more. RTDVRP is an extension to VRPTW with two key differences: (1) In RTDVRP, traveling time between nodes changes dynamically due to the change of traffic condition; (2) In VRPTW, all vehicles depart at time zero, whereas in RTDVRP, there are some dynamically appointed vehicles that have differing departure times.

The Model of RTDVRP (Z Notation)

M = number of customers
K = number of vehicles
Triptime = travel times allocated to each vehicle
C_k = maximum load of vehicle $k = C$
d_{ij} = distance from node i to node j
t_{ij} = travel time from node i to node j
m_i = demand of customer i
TReady_i = earliest serve time of customer i
TDue_i = latest serve time of customer i
Service_i = service time at customer i
Next_i = successor of node i
TServe_i = time start to serve customer i
Load_i = load of vehicle when it departs from i

$\text{Customers} = \{x : \mathbb{N} \mid 1 \leq x \leq M\}$

$\text{Depot} = \{x : \mathbb{N} \mid M+1 \leq x \leq M+K\}$

$t_{ij} = \{ i, j \in \text{Customers} \cup \text{Depot} \mid \forall i, j \bullet \exists\, \tau \in \mathbb{N}_1 \bullet t_{ij} = \tau \times d_{ij}\}$

$\text{Time} = \{x : \mathbb{N} \mid 0 \leq x \leq \text{Triptime}\}$

$\text{Next} = \{(i, x) : i \in \text{Customers}; x \in \text{Customers} \cup \text{Depot} \mid \forall i, j \in \text{Customers} \bullet (i \neq j) \bigwedge (\text{Next}(i) \neq \text{Next}(j)) \bigwedge (\text{Next}(i) = j \Rightarrow \text{Next}(j) \neq i) \bigwedge (\text{Next}(i) \neq i)\}$

$\text{Next} = \{(i, x) : i \in \text{Depot}; x \in \text{Customers} \mid \forall i, j \in \text{Depot} \bullet (i \neq j) \bigwedge (\text{Next(i)} \neq \text{Next(j)})\}$

$\text{TServe} = \{(j, x) : j \in \text{Customers}; x \in \text{Time} \mid (\text{TReady}_j \leq x \leq \text{TDue}_j) \bigwedge (x \leq \text{Triptime} - t_{j(\text{Depot})} - \text{Service}_j) \bigwedge (\forall i, j \in \text{Customers} \bullet \text{Next}(i) = j \Rightarrow \text{TServe}(j) \geq \text{TServe}(i) + t_{ij} + \text{Service}_i\}$

$\text{Load} = \{(j, x) : j \in \text{Customers}; x \in \mathbb{N} \mid (0 \leq x \leq C) \bigwedge (\forall i \in \text{Customers} \cup \text{Depot}; \forall j \in \text{Customers} \bullet \text{Next}(i) = j \Rightarrow \text{Load}(j) = \text{Load}(i) + m_j\}$

3 An RVR Framework for Solving RT-DVRP

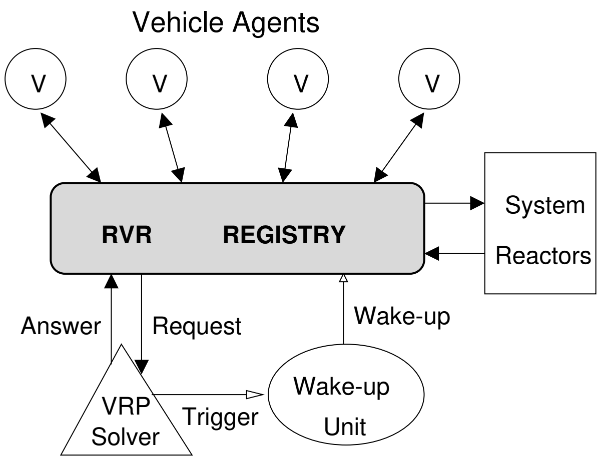

Figure 1. The infrastructure of RVR

In the following sub-sections, we present a framework that is capable of concurrent traffic updates and reactive routings. The Reactive Vehicle Routing model constitutes an concurrent registry, an incremental local optimization solver, a number of vehicle agents and reactors launched by the agents (Figure 1). The dark arrows in the figure denote the movement of reactors, and the hollow ones denote signals. The registry provides the interface and concurrency control to the agents. At the same time, it serializes the requests from the agents and active reactors and sends them to the VRPTW solver. Reactors interact with each other through the solver/store. A trigger mechanism facilitates the wake-up of hibernating reactors when some events occur to the database. The architecture of RVR is a special implementation of OCP [4], customized for solving RT-DVRP. In what follows, we provide detailed accounts for the theory and functionality of the components in RVR.

3.1 Vehicle Agents and Reactors

In the context of RTDVRP, vehicles can be viewed as distributed real time *agents*. Distributed agents are independent intelligent entities that are each loaded with some small programs to accomplish certain tasks [5]. These small programs are referred to as *reactors* because they usually react to events and perform some job. The vehicle agents are independent of each other once dispatched and they are self-controlled in the sense that they only follow the original routing plan given by the central server and next destinations given by their own reactor programs in case of traffic situation changes. Every vehicle is reactive because they issue a number of reactors ranging from simple short-lived update of location or cost to more sophisticated residual query of next moves as a result of some cost update from any vehicle agent. We can express the reactor invoked by an agent as

$$REACT\{\delta(\Delta) \rightarrow \delta'(\Delta)\} \quad (1)$$

where Δ denotes the database store, and $\delta(\Delta)$, $\delta'(\Delta)$ are updates to the data store, in that order. Formula (1) indicates that if and only if $\delta(\Delta)$ is true, both updates will be executed. In its simplest form, a reactor can be $REACT\{\delta'(\Delta)\}$, which means an update of the store Δ, or $QUERY\{\Delta\}$, which is a question about the state of the store. In the above cases, the updates of the store are performed *atomically*. In the OCP language, we can write the reactor as:

$$repeat\,(\delta_1 \Rightarrow \delta_2\ watching\,\epsilon) \quad (2)$$

Here we can treat δ_1 as condition, δ_2 as action and ϵ the timeout. The timeout determines how long the reactor will remain in the system. If ϵ is true, the reactor is timed out and no longer valid. Some reactors keep looping while others are run only once. In RVR model, we will have the following reactors:

new_soln: ONCE (*true*$\Rightarrow$ load new VRPTW problem and solve it, WATCHING *true*)

new_vehicles: LOOP (new solution with more vehicles $\Rightarrow$ dispatch new vehicles, WATCHING all customers visited)

next_move: LOOP (new solution $\Rightarrow$ ask for next moves, WATCHING all customers visited)

update_location: ONCE (true $\Rightarrow$ tell my present customer, WATCHING end of route)

update_cost: ONCE (true $\Rightarrow$ tell my new cost of running from customer i to j, WATCHING end of route)

The first two reactors are system reactors invoked by the RVR system. The latter three are issued by vehicle agents. The most important one is *next_move*, as it guides the route for a vehicle dynamically. *Update_cost* is the only vehicle reactor that may change the solution in the database. It represents the situation of a traffic jam or vehicle breakdown. Together these five reactors forms the foundation for solving RTDVRP problems.

3.2 RVR Registry

Concurrency is an apparent feature of RVR system. The underlying VRPTW solver, however, is sequential. We need concurrency controls to coordinate the external requests. On the other hand, we also need some mechanism to manage the repeating reactors internally. A proper architecture for the reactor circulation and trigger techniques are useful here. Figure 2 shows a schematic of the RVR registry. The RVR registry solves the above problems by taking a modular approach, specifically, a *Receptionist*, a *Delay* module, *Run* module and a *Wakeup* module. The registry functions as follows.

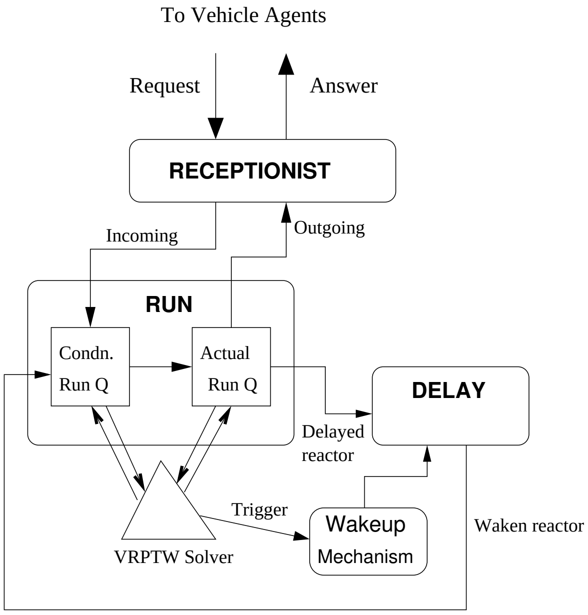

Figure 2. A schematic of RVR registry

The main characteristics of the RVR registry are summarized as follows: (1) the registry provides the concurrency control at the reception front-end to accommodate simultaneous agent requests; (2) it serves as a parser that translates the agent language to the solver language; (3) it coordinates the requests and queue them up according to certain priority policy, and then feeds the requests one by one to the solver; (4) it maintains a delayed queue that contains all the requests that cannot be completed or are looping; and (5) the registry is equipped with a trigger mechanism to wake up appropriate reactors for re-evaluation.

3.3 VRPTW Solver

The VRPTW solver can process one request at a time. A request can be solving for a particular VRP problem, updating the location of the vehicles, or reporting route condition, etc. The solver is also a data store that keeps the most up-to-date solution, current locations of the vehicles, the cost matrix and various constraints. The solver adopts an incremental local optimization routing scheme similar to that of Caseau's [2]. The solution is formed incrementally, while local optimization is done along the processes of constructing full solutions.

In the beginning, all customers are arranged into a stack. They are then served in order, to form the routes incrementally. Each time the heuristic looks for the best location for a customer, with fewer vehicles and less travel distance. After allocation of one customer, a series of optimization is performed involving operations on routes aiming at reducing the number of vehicles and travel distances. The solver ensures that all changes during optimization do not violate the constraints. Algorithm 1 summarizes the routing process.

Algorithm 1 Incremental Local Optimization

```
stack customer;
while customer not empty
begin
    cust = customer.pop_back()
    if(not insertion(cust, existing routes)) //no solution
      newroute(cust)
    else
      optimize(routes)
end
```

Routes optimizations are performed within a route and between routes. Routes operations include exchange operations, 3-opt operations and swap operations. The optimization on a route is performed whenever the route is modified. Other routes are modified with the above operations in a sequential manner, until no better global solution is found. This heuristic is greedy, as only downhill moves are allowed, which potentially causes the search to be trapped in a local minimum. The heuristic handles it by inserting the customers with different order based on one criteria. The criteria is associated with the priority of the customer to be served. Two factors are taken into consideration in determining the priority: (1) The distance of the node from the previous node in the priority queue (the nearer the higher priority: tendency to form a shorter route) and (2) The node's due time (the earlier the higher priority: tendency to be able to be served in time). The relative importance of these two factors is compromised with a weighting factor w, $w \in [0, 1]$. Thus the priority p formula is:

$$p(customer_j) = w \times d_{ij} + (1 - w) \times TimeDue_j$$

where i is the customer before j in the priority list (higher priority). Note that the lower the value of p, the higher priority is the customer. This weighting factor need to be tuned to suit different problems, because it has sizeable affect in the quality of the solution. The heuristic is simple yet produces good results. Compared to one of the best heuristics for VRPTW [8] on the Solomon benchmark, ILO's total distances are only 3% more in its worst case. The number of vehicles are the same for problem set C1, C2, R2 and RC2 and merely one more for problem set R1 and RC1.

4 RVRS: Implementation and Simulation

In what follows, we introduce the implementation details of Reactive Vehicle Routing System for solving RTDVRP's and a simulation we did with the help of RVRS as well as its computation results.

The RVRS reactor comes in the form of a character string. The string appears like macros in C, with "#" separating the condition, action and time out expressions. And the condition, action and time out are Prolog/CLP(R) like statements with a special RTDVRP predicate, for example:

```
#true#update_location(vehicle_no,
customer_no)#end_of_route(vehicle_no).
```

Every reactor object is given a unique identification code so that it can be tracked in the system and queried about if necessary. Noticeably, the querying agent is given a quick acknowledgment by the receptionist immediately after the reactor is received and appended to the input queue. The real answer from the solver is attached to the reactor object and circulated back the reception's output queue. It waits there until the original agent polls for that reactor's id for an answer.

The VRPTW Solver conceptually consists of a daemon, a router and a store. The router is the core of the solver, its effectiveness very much affects the overall system performance. Re-routing is required when a vehicle encounters a traffic jam and is unable to complete the route without violating constraints. The re-routing only applied to those unserved customers. Furthermore, in order to avoid disturbances to other vehicles, the immediate destination of the vehicles are not reallocated.

In the simulations, we evaluate the performance of the RVRS by the quality of the solutions under the effect of traffic jams, and the robustness of the RVR architecture. We introduce the notions of *network resistance* which measures how badly the network is congested, and *elasticity* which is an estimation of how stable the RVRS is. The parameters are defined formally:

Network resistance:

$$\Re = \sum_{i=0}^{n}\sum_{j=0}^{n} t_{ij} - \sum_{i=0}^{n}\sum_{j=0}^{n} d_{ij}$$

Total travel distance of vehicles:

$$D_r = \sum_{i=0}^{n}\sum_{j=0}^{n} d_{ij} \times x_{ijr}$$

where

$$x_{ijr} = \begin{cases} 0 & Next(i) \neq j \\ 1 & Next(i) = j \end{cases}$$

after r number of re-routings.

Elasticity :

$$\Omega = \Delta D / \Re$$

where $\Delta D = D_R - D_0$, and R is the number of reroutes done in the entire assignment. The smaller the Ω, the less affected is the overall solution under same network resistance.

The number of requests handled by the RVRS per second ranges from 14 to 50 in the simulation. Our experiment also shows that the average turnaround time for a reactor request, T_{ta}, is less than a second, where T_{ta} is defined as the time between an agent sends a reactor and it receives the answer by polling.

We experimented RVRS based on 6 Solomon problems, two from each category of C, R and RC. For problem sets C and R, $2 \leq \tau \leq 20$ and the probability of traffic jam between two consecutive customers $\varphi = 0.1$, while the values are $2 \leq \tau \leq 10$ and $\varphi = 0.05$ for problem sets RC. The results shows that additional distance is always less than the network resistance. Although the network resistance varies for a same problem, it is observed that the elasticity does not vary significantly. This exhibits the consistent performance of the solver, despite the randomness in traffic conditions. We also compare the value of standard deviation σ_R and $\sigma_{\Delta D}$ for each problem. And since $\sigma_{\Delta D} \ll \sigma_R$, we conclude that our simulation results are stable with regards to random environment changes.

5 Conclusion

We described a real time dynamic vehicle routing problem. The problem is so reactive and dynamic that re-routing the whole problem in real time with traditional local search VRPTW methods is neither useful nor practical. Instead, we proposed a Reactive Vehicle Routing framework for solving this type of problem. The framework consists of a number of vehicle agents which launch various reactors into a concurrent constraint architecture. The architecture combines a registry from the OCP concept and a solver using incremental local optimization. Reactivity and concurrency are the two most prominent features of the model. They offer the possibility for the agent-based constraint programming to solve for the complex real-time problems. Our implementation, RVRS, and a discretized simulation proved this model promising and it opens up a lot of opportunities for future research and applications in generic goods delivery.

References

[1] D. J. Bertsimas and G. V. Ryzin. "A stochastic and dynamic vehicle routing problem in the euclidean plane," *Operations Research*, vol. 39, pp. 601-614, 1991.

[2] Y. Caseau. "Heuristics for Large Constrained Vehicle Routing Problems," *Journal of Heuristics*, 5, 281-3-3, 1999.

[3] M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard. "Parallel tabu search for real-time vehicle routing and dispatching," *Transportation Science*, vol. 33, no. 4, pp. 381-390, 1999.

[4] J. Jaffar and R. Yap. "Open Constraint Programming," Invited Paper, *4th Intl. Conf on Principles and Practice of Constraint Programming (CP)*, Pisa, Oct. 1998.

[5] N. R. Jennings and M. J. Wooldridge. "Applications of Intelligent Agents". *Agent Technology: Foundations, Applications and Markets*, p.3-28, 1998.

[6] W. B. Powell, P. Jaillet, and A. Odoni. "Stochastic and dynamic network and routing," *Network Routing, Handbooks in Operations Research and Management Science*, vol. 8, pp. 141-295, 1995.

[7] H. N. Psaraftis. "Dynamic Vehicle Routing: Status and Prospects," *Annals of Operations Research*, vol. 61, pp. 143-164, 1995.

[8] E. Rochat and E. Taillard. "Probabilistic Diversification and Intensification in Local Search for vehicle Routing," *Journal of Heuristic*, 1, 147-167, 1995.

[9] M. Wooldridge and N. Jennings. "Intelligent agents: Theory and practice", *Knowledge Engineering Review*, 10(2), 1995.

[10] K. Zhu, K. C. Tan and L. H. Lee. "Heuristics for Vehicle Routing Problem with Time Windows," *6th AI and Math*, 2000.