A Simulated Annealing Algorithm for the Vehicle Routing Problem with Time Windows and Synchronization Constraints

Sohaib Afifi, Duc-Cuong Dang, Aziz Moukrim

A simulated annealing algorithm for the vehicle routing problem with time windows and synchronization constraints *

Sohaib Afifi[1], Duc-Cuong Dang[1,2], and Aziz Moukrim[1]

[1] Université de Technologie de Compiègne
Laboratoire Heudiasyc, UMR 7253 CNRS, 60205 Compiègne, France
[2] University of Nottingham, School of Computer Science
ASAP Research Group, Jubilee Campus
Wollaton Road, Nottingham, NG8 1BB, UK
{sohaib.afifi,duc-cuong.dang,aziz.moukrim}@hds.utc.fr

Abstract. This paper focuses on solving a variant of the vehicle routing problem (VRP) in which a time window is associated with each customer service and some services require simultaneous visits from different vehicles to be accomplished. The problem is therefore called the VRP with time windows and synchronization constraints (VRPTWSyn). We present a simulated annealing algorithm (SA) that incorporates several local search techniques to deal with this problem. Experiments on the instances from the literature show that our SA is fast and outperforms the existing approaches. To the best of our knowledge, this is the first time that dedicated local search methods have been proposed and evaluated on this variant of VRP.

Keywords: vehicle routing, synchronization, destruction/repair, simulated annealing.

1 Introduction

The vehicle routing problem (VRP) [9] is a widely studied combinatorial optimization problem in which the aim is to design optimal tours for a set of vehicles serving a set of customers geographically distributed and respecting some side constraints. We are interested in a particular variant of VRP, the so-called VRP with time windows and synchronization constraints (VRPTWSyn). In such a problem, each customer is associated with a time window that represents the interval of time when the customer is available to receive the vehicle service. This means that if the vehicle arrives too soon, it should wait until the opening of the time window to serve the customer while too late arrival is not allowed. Additionally, for some customers, more than one visit, e.g. two visits from two

* This work is partially supported by the Regional Council of Picardie and the European Regional Development Fund (ERDF), under PRIMA project.

different vehicles, are required to complete the service. Visits associated to a particular customer need to be synchronized, e.g. having the same start time.

VRPTWSyn was first studied in [3] with an application in health care services for elders. In such services, the timing and coordination are crucial and therefore the temporal constraints. The readers are referred to [4] for a complete review of those constraints involved in vehicle routing. As an extension of VRP, VRPTWSyn is clearly NP-Hard. There are only a few attempts to solve this problem in the literature [2, 3]. In those works, even heuristic ones, integer linear programming is the key ingredient and the methods often require much computational time to deal with large instances. Motivated by the potential applications and by the challenge of computational time, in this work we propose a Simulated Annealing algorithm (SA) for solving VRPTWSyn. Our SA incorporates several local search methods dedicated to the problem. It produces high quality solutions in a very short computational time compared to the other methods of the literature. New best solutions are also detected.

2 Simulated annealing algorithm

The main idea of a Simulated Annealing algorithm (SA) [6] is to occasionally accept degraded solutions in the hope of escaping the current local optimum. The probability of accepting a newly created solution is computed as $e^{-\frac{\Delta}{T}}$, where Δ is the difference of fitness between the new solution and the current one and T is a parameter called the current temperature. This parameter is evolved during the search by imitating the cooling process in metallurgy.

Our SA is summarized in Algorithm 1. The algorithm is implemented with a reheating mechanism, due to lines 6 and 25. The simulated annealing routine is from line 9 to line 23. In the algorithm, we use n to denote the number of visits. The other functions are described as follows.

2.1 Constructive heuristic

The procedure $BestInsertion(X)$ contains a constructive heuristic to build a solution from scratch ($X = \emptyset$) or from a partial solution. At each iteration of the heuristic, a visit with the less insertion cost is chosen to be inserted in the associated route. Extra routes will be added when it is impossible to insert the remained visits to the existing routes. The heuristic is actually terminated with all visits being routed.

In order to evaluate the insertion cost in constant time $O(1)$, some additional computations for each visit are archived and updated during the process. When an insertion is applied, the update is propagated through different routes because of the related synchronization constraints. The propagation may loop infinitely if the cross synchronizations are not prohibited, e.g. visiting u then v by the first vehicle, visiting p then q by the second one, and finally realizing that u and q are the same customer as well as v and p (see Fig. 1). In our implementation, such issues are avoided by carefully computing beforehand for each visit the set

Algorithm 1: Simulated annealing algorithm for VRPTWSyn.

Output: X_{best}, the best solution found so far by the algorithm;

```
1  X ← BestInsertion(∅);
2  X ← LocalSearch(X);
3  X_best ← X;
4  reheat ← 0;
5  while (reheat < rhmax) do
6      T ← T_0;
7      iter ← 0;
8      X_lbest ← X;
9      while (iter < itermax) do
10         X' ← Diversification(X, 1, d);
11         X' ← LocalSearch(X');
12         Δ ← Fitness(X') − Fitness(X);
13         iter ← iter + 1;
14         r ∼ U(0, 1);
15         if (r < e^{−Δ/T}) then
16             X ← X';
17             T ← α × T;
18             if (Fitness(X) < Fitness(X_lbest)) then
19                 iter ← 0;
20                 X_lbest ← X;
21                 if (Fitness(X) < Fitness(X_best)) then
22                     X_best ← X;
23                     reheat ← 0;
24     X ← Diversification(X, n/2, n);
25     reheat ← reheat + 1;
```

of valid positions (for insertion) from the existing routes. This process is known as the computation of transitive closures.

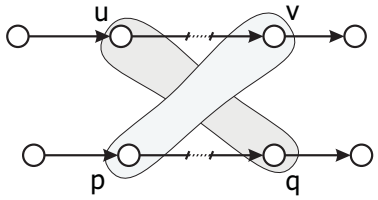

Fig. 1. A cross synchronization

2.2 Diversification process

The function $Diversification(X, d_{min}, d_{max})$ first removes a number (randomly generated between d_{min} and d_{max}) of visits from the current solution, then rebuilds it using the above constructive heuristic. This function is actually an implement of the *destruction/repair* operator [1]. The aim is to obtain a new

solution from the current one without losing much of the quality, thanks to the constructive heuristic.

In addition, a dynamic priority management is also administered to identify critical visits. Each visit is associated with a priority number initialized to 0. This number is increased by 1 unit whenever the insertion of the visit causes the creation of an extra route. Visits having the highest priority, i.e. frequently causing extra routes, are in fact critical. Therefore, they need to be inserted during the early stages of the constructive heuristic. With this dynamic management, the search is guided back to the feasible space whenever it hits the infeasible one. In general, we remarked that the portion of explored infeasible solutions over feasible ones is varied from one instance to another. This solely depends on the size of the time windows, e.g. the algorithm hits infeasible solutions more frequently with an instance having small time windows.

2.3 Local search procedure

The two following neighborhoods were adapted to the synchronization constraints and used in our local search procedure:

2-opt* (exchanges of paths between different routes [7]): in a 2-opt operator, we look for the possibility of exchanging two links with two others in the same route in order to find a local improvement. For the case of multiple vehicles, we use 2-opt* to denote the same principle of exchange but related to two distinct routes. This operator consequently implies the exchanges of paths between the two routes. It is particularly suitable for our case because it is hardly possible for the classical 2-opt to find an improvement due to preserved order of visits from the time windows. Our 2-opt* is implemented as follows: a subset of visits is randomly selected and for each couple of visits $\{r_i^k, r_j^{k'}\}$, we consider the arcs (r_i^k, r_{i+1}^k) and $(r_j^{k'}, r_{j+1}^{k'})$ (where r_i^k denotes the visit at position i in route k). If the exchange of these two arcs for $(r_i^k, r_{j+1}^{k'})$ and $(r_j^{k'}, r_{i+1}^k)$ ensures the feasibility then the associated cost is recorded. The feasibility check is handled by the same process as the one used in the constructive heuristic to avoid cross synchronizations. Therefore, the exchange cost is evaluated in constant time for each couple $\{r_i^k, r_j^{k'}\}$. The best one is then memorized and the exchange is applied.

or-opt (relocation of visits in the same route [8]): in this operator, we look for the possibility of relocating a sequence of (1, 2 or 3) visits from its original place to another one in the same route. The implementation of this operator is similar to 2-opt* operator: a random selection at the beginning then a feasibility check.

Our $LocalSearch(X)$ function is then the following: at each iteration, a random neighborhood w is chosen from the current set W, initialized to {2-opt*, or-opt}. Neighborhood w is then removed from W and repeatedly applied to the current solution until no improvement is found. If at least an improvement was detected by w, then the other neighborhood will be put back to W (in case it was removed). The procedure is terminated when W is empty.

3 Results

We tested our algorithm on the instances introduced by [3]. The benchmark comprises 10 sets grouped in 3 categories based on the number of customers. Each set has 5 varieties of instances, those are named after the width of the time windows. Our algorithm is coded in C++ and all experiments were conducted on an Intel Core i7-2620M 2.70GHz. This configuration is comparable to the computational environment employed by Bredström and Rönnqvist [2, 3] (a 2.67 GHz Intel Xeon processor). According to the protocol proposed in [2], all the methods were tested with the varieties of S (small), M (medium) and L (large) time windows. After several experiments on a subset of small instances, we decided to fix the parameters as follows: $T_0 = 20$, $\alpha = 0.99$, $d = 3$, $itermax = 10 \times n$ and $rhmax = 10$.

Table 1 shows our results compared to the literature. Instances, in which all methods report the same results, are discarded from this table. Columns n, m, s and *Best* show the number of visits, the number of vehicles, the number of synchronizations and the best known solution from all methods (including ours) respectively for each instance. A star symbol (*) is used in *Best* to indicate that the solution is proved to be optimal. The other column headers are: MIP for the results of the default CPLEX solver reported in [3]; H for the heuristic proposed in [3] which is based on the local-branching technique [5]; BP1 and BP2 for the results of the two branch-and-price algorithms presented in [2] and finally SA for our simulated annealing algorithm. Columns Sol and CPU correspond to the best solution found by each method and the associated total computational time. Bold numbers in Sol indicate that the solution quality reaches *Best*.

Data	n	m	s	*Best*	MIP		H		BP1		BP2		SA	
					Sol	CPU	Sol	CPU	Sol	CPU	Sol	CPU	Sol	CPU
1L	20	4	2	3.39*	3.44	3600.00	**3.39**	120.00	**3.39**	107.41	**3.39**	11.91	**3.39**	0.29
2L	20	4	2	3.42*	3.58	3600.00	**3.42**	120.00	**3.42**	2.72	**3.42**	7.41	**3.42**	0.64
3M	20	4	2	3.33*	3.41	3600.00	**3.33**	120.00	**3.33**	17.57	**3.33**	4.31	**3.33**	0.92
4M	20	4	2	5.67*	5.91	3600.00	5.75	120.00	**5.67**	27.53	**5.67**	2.55	**5.67**	0.72
4L	20	4	2	5.13*	5.83	3600.00	5.30	120.00	**5.13**	9.74	**5.13**	7.69	**5.13**	4.66
6S	50	10	5	8.14*	-	-	-	-	**8.14**	3600.00	**8.14**	197.92	**8.14**	93.78
6M	50	10	5	7.70	-	-	-	-	7.71	3600.00	**7.70**	3600.00	**7.70**	3358.60
6L	50	10	5	7.14*	-	-	-	-	**7.14**	3279.48	**7.14**	3600.00	**7.14**	2440.95
7S	50	10	5	8.39*	-	-	-	-	**8.39**	1472.39	**8.39**	169.30	**8.39**	163.03
7M	50	10	5	7.49	-	-	-	-	7.67	3600.00	7.56	3600.00	**7.49**	199.23
7L	50	10	5	6.86	-	-	-	-	6.88	3600.00	6.88	3600.00	**6.86**	144.94
8S	50	10	5	9.54*	-	-	-	-	**9.54**	931.95	**9.54**	850.52	**9.54**	149.95
8M	50	10	5	8.54*	-	-	-	-	**8.54**	3600.00	**8.54**	3490.57	**8.54**	276.46
8L	50	10	5	8.07	-	-	-	-	8.62	3600.00	8.11	3600.00	**8.07**	335.72
9S	80	16	8	12.13	-	-	-	-	-	3600.00	12.21	3600.00	**12.13**	397.876
9M	80	16	8	10.94	-	-	-	-	11.74	3600.00	11.04	3600.00	**10.94**	641.838
9L	80	16	8	10.67	-	-	-	-	11.11	3600.00	10.89	3600.00	**10.67**	376.24
10S	80	16	8	8.82	-	-	-	-	-	3600.00	9.13	3600.00	**8.82**	3099.28
10M	80	16	8	8.01	-	-	-	-	8.54	3600.00	8.10	3600.00	**8.01**	757.87
10L	80	16	8	7.75	-	-	-	-	-	3600.00	-	3600.00	**7.75**	3247.71

Table 1. Comparison of results and CPU times

From these results, we remark that SA finds all known optimal solutions (20 of 30) in very short computational times compared to the other methods. Quality of the other solutions is also better than the one found in the literature. The algorithm strictly improved the best known solutions for 9 instances of the data sets. Those instances are $7M$, $7L$, $8L$, $9S$, $9M$, $9L$, $10S$, $10M$ and $10L$. To summarize, our SA is clearly fast and efficient.

4 Conclusion

The paper presented a simulated annealing based heuristic for VRPTWSyn. Numerical results on the benchmark proposed by [3] demonstrate the competitiveness of the algorithm for such a problem. They also demonstrate that destruction/repair operator and local search methods can be efficiently adapted to the case of the synchronization constraints. As future work, we intend to investigate the performance of the SA on other variants of VRPTWSyn, such as the one with customer-driver preferences and the one with route balance constraints [3]. We also plan to investigate the use of the obtained solutions as a warm start for exact methods, such as mixed integer programming, to solve the open instances of VRPTWSyn to the optimality.

Bibliography

[1] Bouly, H., Moukrim, A., Chanteur, D., Simon, L.: An iterative destruction/construction heuristic for solving a specific vehicle routing problem (in French). In: MOSIM'08 (2008)

[2] Bredström, D., Rönnqvist, M.: A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints (Feb 2007)

[3] Bredström, D., Rönnqvist, M.: Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European Journal of Operational Research 191(1), 19–31 (Nov 2008)

[4] Drexl, M.: Synchronization in vehicle routing a survey of vrps with multiple synchronization constraints. Transportation Science 46(3), 297–316 (2012)

[5] Fischetti, M., Lodi, A.: Local branching. Mathematical Programming 98(1-3), 23–47 (Sep 2003)

[6] Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)

[7] Potvin, J.Y., Kervahut, T., Garcia, B.L., Rousseau, J.M.: The vehicle routing problem with time windows part I: tabu search. INFORMS Journal on Computing 8(2), 158–164 (1996)

[8] Solomon, M.M., Desrosiers, J.: Time window constrained routing and scheduling problems. Transportation science 22, 1–13 (1988)

[9] Toth, P., Vigo, D.: The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics (2002)