An Iterated Local Search for the Split Delivery Vehicle Routing Problem

Z.Z. Wen, X.Y. Dong, S. Han
Beijing Key Lab of Traffic Data Analysis and Mining, School of Computer and IT
Beijing Jiaotong University
Beijing, China

***Abstract*--A multi-restart iterated local search (MRSILS) algorithm is introduced for the Split Delivery Vehicle Routing Problem (SDVRP). The initial solution is generated by the GENIUS and applied the local search procedure by removing one node from its current route and inserting it into the best locations, with the possibility of splitting its demand. A node inserting algorithm is proposed for SDVRP, trying to find a better solution by an insertion with the consideration of splitting the demand. The perturbation is applied while certain predefined continuous no improvement local search steps are occurred. In order to extend the search space and keeping the quality of the restart solution, a method is designed for the perturbation that the perturbed solution is chosen from an elite solution set, while the best solution is preferred. The object of this work is to minimize the total travel distance. Experimental results on a benchmark set show that the MRSILS is competitive with a state of the art algorithm.**

Keywords-SDVRP; local search; perturbation; multi-restart

I. INTRODUCTION

Vehicle Routing Problem with split delivery (SDVRP) is a variant of capacitated vehicle routing problem (CVRP). The demand of each node in SDVRP can be served by more than one vehicle. In SDVRP, there is a set of nodes *N* with service demand, infinite homogeneous vehicles *V* with capacity constraint *k* to provide service and one depot *D*. The vehicles start and end their tours at the depot withoutthe load surpassing*k*during the tour and the demand of all the nodes must be satisfied[1].

Compared with other VRPs, SDVRP problem has not been enough concerned. There are still some excellent works published. Dror and Trudeau [1], [2]analyze the savings achieved by allowing deliveries split in some VRP instances and provide a heuristic algorithm for SDVRP. Nowak et al. [3] show the potential benefit of split loads and prove the most benefit occurring with delivery of node just one half of vehicle capacity. Drorand Trudeau [1] describe valid inequalities for the SDVRP and Archetti et al. [4]describe a tabu search algorithm for SDVRP. Chen et al. [5] and Archetti et al. [6] put forward hybrid heuristics for SDVRP combined with the advantages ofheuristic and optimization methods to improve calculation efficiency of the single algorithm. Derigset al.[7] present several local-search-based metaheuristics including simulated annealing (SA), threshold accepting (TA), record-to-record travel (RRT), the attribute based hill climber heuristic (ABHC) and the attribute based local beam search heuristic (ABLBS) for the SDVRP, and the ABHC get the best experimental results.

Boudia et al. [8] point out that the using of dynamic programming increases the running time by 30% without significant improvement than the one found by a greedy heuristic.In this work, we propose a greedy iterated local search (ILS) algorithm in which the insertion move in local search phase follows the simple principle that always do the best insertion, considering the possible splitting of the demand. In order to extend the search space effectively, without deteriorating the quality of the restart solution, a multi-restart perturbation method is designed, which is derived from the idea of Dong et al. [9].

The rest of this paper is organized as follows. In section 2, we present our multi-restart local search algorithm, including the method for the initial solution, local search structure and perturbation method. In section 3, the experimental resultsare reported and analyzed. In section 4, we put forward the innovation and deficiency point of our algorithm.

II. MULTI-RESTART ITERATED LOCAL SEARCH ALGORITHM

The followingtermsare used in this paper. The number of nodes is *N*. Direct trip is a tour in which a vehicle starts from the depot, goes directly to a customer, delivers*k* units and then turns back directly to the depot [4]. Remove savingof node*i*is denoted by RS_i and computed by $RS_i=\sum_{r\in R(i)}(c_{pi}+c_{iq}-c_{pq})$, where $R(i)$ is all the cars that visit node i in current solution; p and q are the predecessor and the successor of node i in route r; c_{pi}denotes the distance between node p and i and so on. Remove saving list*RSL*is defined as a non-increasing list of RS_i for all node *i*.The minimum cost for inserting node i into route r is denoted by IC_{ir} =min $\{c_{si} + c_{it} - c_{st}\}$, where s and t are two adjacent nodes in route r. The minimum costs for inserting node i into all the routes in Rformingthe insert cost list of node*i*, denoted by ICL_i, and the elements in which are sorted in ascending way.

There are two phases in local search. In phase 1, if the demands of some nodes are larger than k, then the nodes will be divided into two parts $I_K$and I_R. Part $I_K$contains the nodes with demand surpassing k, and the demand of the node i in I_K is set to$k\lfloor d_i/k\rfloor$. Then nodes are created as I_K' for the nodes in I_K with non-zero remaining demand. Part $I_R$is the remaining nodes by removing I_K from I, i.e. $I_R = I\backslash I_K$. Initial solution for I_K is generated by adding direct trips for every node. We use GENIUS [10] for $I_R \cup I_K$' to generate a big TSP tour, and then split the tour by k. The initial solution consists of the above two initial solutions.

© 2015. The authors - Published by Atlantis Press

In phase 2, we propose a multi-restart iterated local search for SDVRP. We do the local search by removing node *i*and then reinserting it into another position according to the method presented in section 2.1. The searching order of node *i* is arranged by *RSL*. This arrangement is based on the hypothesis that searching the node with larger remove saving can achieve a better solution with greater possibility. A qualified solution pool is also maintained to enlarge the restart-solution space. The pool is designed for the perturbation function, which is the solution chosen from the pool is perturbed to generate the start solution for the next local search. The algorithm is names MRSILS and the pseudo-code is shown in algorithm 1, where s_0 is the initial solution; s' is the current solution; s^* is the current best solution; s'' is the solution which is the result of local search starting from s'; the number of nodes is denoted by N; $S$is the solution poolstoring a set of solutions; *pool_size*represents the maximum size of S.

Algorithm 1: Multi-restart local search frame

1. Generate s_0, $s' \leftarrow s_0$, $s^* \leftarrow s_0$, i, *flag* ← false, $S \leftarrow \phi$, *pool_size*;
2. **while** (termination criterion is not satisfied) **do**
3. Compute the *RSL* for s';
4. $i \leftarrow 0$;
5. **while** ($i<N$) **do**
6. $s'' = LocalSearch(s', i)$;
7. **if** s'' is better than s' **then**
8. $s' \leftarrow s''$;
9. **endif**
10. **if** s' is better than s^* **then**
11. $s^* \leftarrow s'$, *flag*←true;
12. **endif**
13. **if** the perturbation limitation is met **then**
14. **if***flag* is true **then**
15. $S \leftarrow \phi$, *flag*← false;
16. **endif**
17. $S \leftarrow accept(S, s')$;
18. $s' \leftarrow perturb(S, s^*)$;
19. **endif**
20. i++;
21. **endwhile**
22. **endwhile**
23. Stop and output s^*.

In step 17 of the algorithm 1, the accept(S, s') is used to estimate whether s' is qualified to be added to S: if the size of S is less than pool_size, add s'to S; otherwise if s' is better than the worst one in S, then remove the worst and insert s' into S. The solutions in Sare several good solutions around the current best solution s*, so choosing a solution from it to generate the restart solution is a reasonable option. The perturbation method in step 18 is shown in section 2.2.

A. Insertion Method

In the localsearch, we successively remove a node from the current solution and reinsert it to its best location. The insertion is designed as follows to support the possible split operations. As for the selection of the least cost move of node *i*, all the routes with surplus capacity larger than the demand of node *i* are checked, in addition to the combination of several routes with least insertion cost. The whole insertion procedure is shown as below.

Step1: Remove node *i* from s′ and compose route list RL_i;

Step 2: Compute the insertion cost IC_{ir} for route r in RL_i, whose surplus capacity can satisfy the demand of node *i*, denote the route with minimumIC_{ir} by r_{min}, and set min_{ic} to IC_{ir}; if it does not exist, set$min_{ic}$to a very large number;

Step 3: Compute the minimum cost $min_{is}$for the case of the sum of the surplus capacity of the first z routes in RL_i is larger than the demand of node *i*, while the first z– 1 routes can not satisfy the demand, where the routes considered above have most least insertion cost and the surplus capacity of each one is smaller than the demand of node *i*; if such case does not exist, the min_{is} is a very large number;

Step 4: Insert the node *i* to the case indicated by the smaller min_{is} and min_{ic};

B. Perturbation Strategy

The perturbed solution and the perturbation method have a great influence on the quality of the final best solution. There are two steps for the perturbation strategy. Firstly choose the solution which will be perturbed; secondly perturb the chosen solution. As for the first step, when the S is full, we choose one randomly from it; otherwise the current best solution s^* is chosen. The perturbation method is shown as *algorithm* 2.

Usually, perturbation is occurred when the local search is trapped in a local optimum, i.e. the current solution can not be improved by re-inserting any node.

Algorithm 2: Perturbation strategy

1. Choosing two routes r_1 and $r_2$from the solution being perturbed randomly;
2. Generate start nodes A_i and $B_i$for these two routes randomly;
3. Randomly generate$C_A$the numberof being perturbed nodes for r_1;
4. Mark C_A nodes starting from A_i on path r_1 as set N_{RA};
5. Suppose the total demand of node set N_R is denoted by d_R, and the surplus capacity of route r is denoted by s_r, then take the maximal number of nodes in r_2 starting from B_i as N_{RB}, satisfying $s_{r1} + d_{RA} - d_{RB} > 0$;
6. If all of the nodes in N_{RA} can be served by r_2, the perturbation can be implemented by exchanging the N_{RA} and N_{RB}directly ; otherwise, exchange the N_{RA} and N_{RB} with the splitting of the last node in N_{RA} .

However, it can be assumed that the local search can hardly improve the solution further after checking certain number of nodes without improvement, and so make a perturbation to continue the local search is reasonable in such a case. We set the above number of insertion nodes as the criterion for the perturbation in step 13 in algorithm 1, which is denoted by L.

III. EXPERIMENTAL RESULTS

The problems 1-5 and 11 from Gendreauet al.[11] are considered in this work. The customer number of these instances is between 50 and 199. As proposed by Archetti et al.[4], the demand of the customers is changed by introducing

two parameters $\alpha$and γ, where $\alpha$and γin the interval [0, 1] and $\alpha \leq \gamma$. The demand d_i of customer i is set to$d_i = \alpha k + \delta (\gamma - \alpha) k$, where δ is a randomly number in [0, 1]. The (α, γ) are set as follows: (0.1, 0.3), (0.1, 0.5), (0.1, 0.9), (0.3, 0.7), (0.7, 0.9). The instance set used in this work is provided warmheartedly by C. Archetti, and it is the same with what is used in Derigset al. [7]. We get the results with 1.3GHz CPU with 3GB memory and the algorithm is implemented in C++.

In order to optimize the perturbation, the parameter L is set to $\lceil 0.01N \rceil$, $\lceil 0.1N \rceil$, $\lceil 0.5N \rceil$ and N, respectively. The MRSILS is run 5 independent times for each case with the five CPU minutes. The best solution is chosen from these five runs, and the comparison is shown in fig. 1 in average relative percentage deviation (ARPD) over the case of $0.01N$, and the RPD is computed as:

$$RPD = (f - f_{0.01N})/ f_{0.01N} \times 100$$

where f denotes the objective achieved by the compared case, and $f_{0.01N}$ denotes the objective gotten by setting the L to $0.01N$. From this figure, it can be seen that the results are generally the best with $L = 0.01N$, other than the setting of $L = 0.1N$ on the original benchmark set, in which case the setting of $L = 0.1N$ is the best. In the following experiment, the L is set to $0.01N$.

The ABHC proposed by Derigset al. [7] is one of the state of the art algorithms. In their work, the algorithm is run five times independently with 5, 10, 15, 30 and 60 CPU minutes, respectively, and they report the best results among these runs. They carried out the experiments on 3GHz PCs with 2 GB memory. In order to make a relative fair comparison, the MRSILS is run in thesame settings and the comparison results are listed in Table 1, where NV denotes the number of vehicles, TD denotes the total distance, and TD% denotes the relative percentage deviation of the MRSILS over the ABHC. The results show that the MRSILS is comparable to the ABHC, with better for 17 out of 36 instances, and better in overall performance.

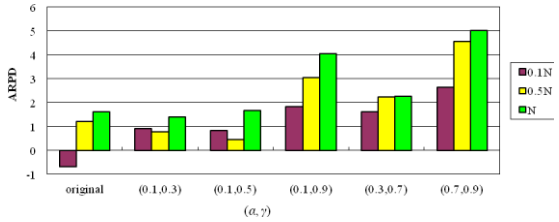

FIGURE I. OPTIMIZATION OF L

TABLE I. COMPARISON RESULTS FOR INSTANCES FROM ARCHETTIET AL.[6]

Instance				ABHC		ILS		
Name	|C|	α	γ	NV	TD	NV	TD	TD%
P01	50	—	—	5	524.61	5	531.03	-1.22
P02	75	—	—	10	829.89	10	831.85	-0.24
P03	100	—	—	8	826.14	8	834.52	-1.02
P04	150	—	—	12	1028.42	12	1066.04	-3.66
P05	199	—	—	16	1302.90	17	1343.67	-3.13
P11	120	—	—	7	1042.12	7	1048.00	-0.57
P01	50	0.1	0.3	11	776.42	11	764.90	1.48
P02	75	0.1	0.3	16	1123.98	16	1130.05	-0.54
P03	100	0.1	0.3	22	1478.59	22	1487.30	-0.59
P04	150	0.1	0.3	32	2055.18	32	2060.06	-0.24
P05	199	0.1	0.3	40	2540.06	41	2546.80	-0.27
P11	120	0.1	0.3	27	2913.09	27	2948.99	-1.23
P01	50	0.1	0.5	16	1012.56	16	1019.95	-0.73
P02	75	0.1	0.5	24	1508.73	24	1512.78	-0.27
P03	100	0.1	0.5	33	2035.91	33	2026.81	0.45
P04	150	0.1	0.5	49	2912.08	49	2895.16	0.58
P05	199	0.1	0.5	63	3581.66	63	3579.67	0.06
P11	120	0.1	0.5	41	4270.38	41	4299.16	-0.67
P01	50	0.1	0.9	26	1489.64	27	1495.85	-0.42
P02	75	0.1	0.9	41	2340.09	41	2317.31	0.97
P03	100	0.1	0.9	56	3145.33	56	3133.99	0.36
P04	150	0.1	0.9	84	4638.74	84	4618.62	0.43
P05	199	0.1	0.9	105	5669.26	105	5635.13	0.60
P11	120	0.1	0.9	67	6890.39	67	6884.85	0.08
P01	50	0.3	0.7	26	1488.28	26	1497.15	-0.60
P02	75	0.3	0.7	39	2243.93	39	2253.36	-0.42
P03	100	0.3	0.7	53	3014.08	53	3020.04	-0.20
P04	150	0.3	0.7	79	4435.95	79	4376.86	1.33
P05	199	0.3	0.7	102	5541.09	102	5516.43	0.45

P11	120	0.3	0.7	64	6671.04	64	6704.80	-0.51
P01	50	0.7	0.9	41	2174.54	41	2164.65	0.46
P02	75	0.7	0.9	61	3266.78	61	3240.56	0.80
P03	100	0.7	0.9	82	4447.47	82	4406.68	0.92
P04	150	0.7	0.9	122	6467.17	122	6447.69	0.30
P05	199	0.7	0.9	161	8297.71	162	8283.38	0.17
P11	120	0.7	0.9	98	10233.37	99	10239.82	-0.06
total					114217.55		114163.9	0.05

IV. CONCLUSION

When using local search methods, an intensive exploration of the solution space is performed by moving at each step from the current solution to another promising solution in its neighbors[12]. The proposed MRSILS removes a node and reinserts it to the best location according to SDVRP constraints. It performs well, achieves good solution on standard SDVRP data set[6], and gets the best solution for 17 instances. It is also easy to implement.

ACKNOWLEDGMENTS

Prof. Y. Lin provides lots of support. Thanks to Prof. C. Archetti for the data set and the help of U. Derigs. This work is supported by The FundamentalResearchFunds for the Central Universities of China (Project Ref. 2014JBM034, Beijing Jiaotong University).

REFERENCES

[1] Dror M. and P. Trudeau, Savings by split delivery routing. *TransportationScience*, 23, pp. 141–145, 1989.

[2] Dror M. and P. Trudeau, Split delivery routing. *NavalRes.Logistics*, 37, pp. 383–402, 1990.

[3] Nowak M, Ergun O, White C, Pickup and Delivery with Split Loads. *TransportationScience*, 42, pp. 32–43, 2008.

[4] Archetti C, Speranza M and Hertz A, A tabu search algorithm for the split delivery vehicle routing problem. *TransportScience*, 40, pp. 64–73, 2006.

[5] Chen S, Golden B and Wasil E, The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results. *JournalofNetworks*, 49, pp. 318–329, 2007.

[6] Archetti C, Speranza M and Savelsbergh M, An optimization based heuristic for the split delivery vehicle routing problem. *TransportScience*, 42, pp. 22–31, 2008.

[7] U Derigs, B Li and U Vogel, Local search-based metaheuristics for the split delivery vehicle routing problem.*Journalofthe OperationalResearchSociety*, 61, pp. 1356–1364, 2010.

[8] Boudia M., Prins C., Reghioui M., An effective memetic algorithm with population management for the split delivery vehicle routing problem. *LectureNotesinComputerScience*, 4771, pp. 16–30, 2007.

[9] Xingye Dong，Houkuan Huang and Maciek Nowak, A Multi-Restart Iterated Local Search Algorithm for the Permutation Flow Shop Problem Minimizing Total Flow Time.*ComputerOperationalResearch*, 40(2), pp. 627–632, 2013.

[10] Gendreau, M., A. Hertz, G. Laporte, New insertion and postoptimization procedures for the traveling salesman problem. *OperationalResearch*, 40, pp. 1086–1094, 1992.

[11] Gendreau M., A. Hertz, G. Laporte, A tabu search heuristic for the vehicle routing problem. *ManagementScience*, 40, pp. 1276–1290, 1994.

[12] Xingye Dong，Houkuan Huang and Ping Chen, An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion.*ComputerOperationalResearch*, 36(5), pp. 1664–1669, 2008.