Solving the One-Commodity Pickup and Delivery Problem Using an Adaptive Hybrid VNS/SA Approach

Manar I. Hosny and Christine L. Mumford

Cardiff School of Computer Science & Informatics
M.I.Hosny,C.L.Mumford@cs.cardiff.ac.uk

Abstract. In the One-Commodity Pickup and Delivery Problem (1-PDP), a single commodity type is collected from a set of pickup customers to be delivered to a set of delivery customers, and the origins and destinations of the goods are not paired. We introduce a new adaptive hybrid VNS/SA (Variable Neighborhood Search/Simulated Annealing) approach for solving the 1-PDP. We perform sequences of VNS runs, where neighborhood sizes, within which the search is performed at each run, are adaptable. Experimental results on a large number of benchmark instances indicate that the algorithm outperforms previous heuristics in 90% of the large size test cases. Nevertheless, this comes at the expense of an increased processing time.

1 Introduction

The One-Commodity Pickup and Delivery Problem (1-PDP) is an important problem in transportation and logistics. The problem deals with supplying and collecting one type of commodity from a number of customers, some of them are designated as pickup customers and the others as delivery customers. Each pickup customer provides a certain amount of the commodity, while each delivery customer consumes a certain amount of the same commodity, i.e., goods collected from pickup customers can be delivered to any delivery customer. All customers are served by one vehicle with a limited capacity, and the journey of the vehicle should start and end at a central depot. The depot can supply or consume any additional amount of the commodity that is not supplied or consumed by the customers. Our goal is to find a feasible and minimum cost route for the vehicle, such that all customers are served without violating the vehicle capacity constraint (see [4] for a formal definition of the 1-PDP).

The 1-PDP has many applications in practice. For example, the commodity could be milk that should be collected from farms and delivered to factories with no restriction on the origin and the destination of the product, or it could be money that should be distributed between the branches of a bank [5]. It can also model any logistic situation in which some warehouses have an extra supply of some commodity, while others are in short of the same commodity. A typical situation is when some hospitals need to transfer a certain medicament

to other hospitals, which are in short of this medicament. For example, an H1N1 vaccination could be transferred in urgent epidemic circumstances [7].

In our research we apply an adaptive hybrid VNS/SA (Variable Neighborhood Search/Simulated Annealing) approach to the 1-PDP. The algorithm is distinguished by performing the VNS repeatedly, each time starting from the final solution obtained from the previous VNS run. Also, the algorithm is *adaptive*, in the sense that the maximum neighborhood size allowed in each VNS run is not fixed and depends on the current stage of the search. Early runs are allowed to perform wider jumps in the solution space from the current solution, using large neighborhood sizes. Later runs, on the other hand, are only allowed smaller explorations of the search space, in the vicinity of the current solution, to maintain the solution quality. The stopping criterion for each VNS run is also adaptive and depends on the improvement realized in the current solution. The basic VNS meta-heuristic systematically increases the neighborhood size, within which the search is performed, up to a pre-specified maximum size. In our approach, nevertheless, each VNS run is also terminated when a further increase in the neighborhood size seems not beneficial, even if the maximum neighborhood size was not reached. During each VNS run, an SA acceptance criterion is used to allow the algorithm to escape local optima.

The rest of this paper is organized as follows. Section 2 is a summary of some related work. Section 3 briefly describes the VNS meta-heuristic and highlights previous 1-PDP research that utilizes this approach. Section 4 explains in detail the main components of our proposed heuristic, which we will call an **Adaptive Hybrid VNS/SA (AVNS-SA)** technique for solving the 1-PDP. The complete AVNS-SA algorithm is shown in Section 5. Experimental results are presented in Section 6, and a summary and future directions are given in Section 7.

2 Related Work

Since the 1-PDP is $\mathcal{NP}$*-hard*, exact algorithms are only suitable for small problem sizes. For example, [5] presented a branch-and-cut exact algorithm to solve instances of up to 60 customers. To deal with large size problems, the same authors tried two heuristic approaches in [6]. The first approach is a nearest-neighbor insertion heuristic followed by an improvement phase, using 2-Opt and 3-Opt edge exchanges. The second approach is an incomplete optimization procedure, to find the best solution in a restricted feasible region.

In [4] a heuristic approach, named hybrid GRASP/VND, is proposed. The approach combines two optimization heuristics. The first is called GRASP (Greedy Randomized Adaptive Search Procedure), and is based on a repetition of a construction phase and a local search phase. On the other hand, Variable Neighborhood Descent (VND) is a variant of VNS, which is simply a local search that gradually increases the neighborhood size whenever a local optimum is reached. The hybrid GRASP/VND is basically a GRASP, with an added local search

using VND. After the basic GRASP/VND, a further post-optimization phase is performed using *move forward* and *move backward* operators.[1]

On the other hand, a GA approach was introduced in [9] to solve the 1-PDP. The algorithm first starts by creating a population of feasible solutions using a new nearest-neighbor construction heuristic. The initial population is then optimized using a 2-Opt neighborhood move. The most distinguishing feature of the algorithm is a new *pheromone-based* crossover operator, where the selection of the next node to be inserted in the child is based on a probabilistic rule that takes into account the pheromone trail of the edge connecting the last inserted node and the potential new node, such that edges that have proved successful in the past are favored by an increased pheromone value. The offspring are further optimized using a 2-Opt local search. The mutation operator is based on a 3-exchange procedure. The algorithm was tested on the benchmark instances created by [6], producing the best so far results in most test cases.

3 Variable Neighborhood Search (VNS) and its Application to the 1-PDP

VNS is a relatively new meta-heuristic that was introduced in [2] and [3]. The idea is to generate new solutions that are distant from the incumbent solution, by systematically increasing the neighborhood size within which the search is performed. In addition, a local search is performed on the new solution to reach a local optimum within the current neighborhood. After the local search phase, the new solution replaces the current solution if it is better in quality. The basic steps of the VNS algorithm, as described in [3], are shown below:

- *Initialization*: Select the set of neighborhood structures N_k, $(k = 1, ...k_{max})$; find an initial solution x; choose a stoping condition;
- *Repeat* the following until the stopping condition is met:
  1. Set $k \leftarrow 1$;
  2. Repeat the following steps until $k = k_{max}$:
     (a) *Shaking*: Generate a point x' at random from the k^{th} neighborhood of x $(x' \in N_k(x))$;
     (b) *Local Search*: Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;
     (c) *Move or not*: if the local optimum x'' is better than the incumbent, move there $(x \leftarrow x'')$, and continue the search with N_1 $(k \leftarrow 1)$; otherwise set $k \leftarrow k + 1$.

As previously mentioned in Section 2, a variant of VNS,called Variable Neighborhood Decent (VND) has been tried for the 1-PDP as part of the hybrid GRASP/VND heuristic proposed in [4]. The VND heuristic does not include a shaking phase, but only a local search that involves a 2-Opt followed by a 3-Opt move. The algorithm achieved promising results that were better than the

[1] More details about the heuristic in [4] will be given in Section 3.

results obtained by the same authors in their previous heuristics suggested in [6]. However, their algorithm apparently was not fully capable of escaping the trap of local optima. This is evident by the fact that they had to use a post-optimization phase to improve the final result. According to the authors, this post-optimization often made the difference between beating the results obtained by their previous heuristic in [6] or not. The hybrid GRASP/VND heuristic was also outperformed by the GA in [9], in most test cases. A possible shortcoming of this heuristic is the absence of a shaking phase, which should help the diversification of the incumbent solution and allow the succeeding local search phase to escape local optima. In our proposed approach, we try to apply the basic VNS, with both the shaking and the local search, hoping to overcome the limitation of the previous VND attempt on the 1-PDP.

4 The AVNS-SA Heuristic

The main features of our proposed AVNS-SA approach are described below:

The Initial Solution: The construction algorithm we use is the same as the algorithm proposed by [9], which is a nearest-neighbor construction heuristic. However, in [9], they stop the construction process when infeasibility is encountered and try a new starting node. In our approach, we allow violations in the capacity constraint by continuing the construction process despite infeasibility.

The Objective Function: The objective function we use to estimate the quality of a solution S is set to: $F = (NCV(S)+1) \times Dist(S)$, where $NCV(S)$ is the number of capacity violations along the route, and $Dist(S)$ is the total distance of the solution, given the current visiting order of nodes. If there are no capacity violations in the solution, i.e., the solution is feasible, the total distance will be the sole measure of the solution cost.

The Shaking Procedure: The shaking procedure is used for *diversification* of the search in the VNS. We chose as a shaking procedure a displacement of a sequence of nodes with some probability of inverting this sequence. Our VNS algorithm passes the current neighborhood size ($NhSize$) as a parameter to the shaking procedure, which will in turn use this parameter as the *maximum* possible number of nodes that will be displaced. Specifically, the number of nodes to be displaced is a random number between 1 and $NhSize$. So even for large values of $NhSize$, small sequences of nodes could still be displaced, and in fact there is a 'bias' toward such small moves, since they have a chance of being executed in all neighborhood sizes. This is intended to prevent a large disruption of the current solution, and is recommended by some VNS implementations, as in [8].

The local search procedure: This procedure is the tool that VNS uses for *intensification* of the search. In our algorithm, we chose as a local search a simple 2-Opt edge exchange algorithm. Our local search exhaustively tests all possible edge exchanges, and uses *best improvement* as a replacement strategy.

Multiple VNS Runs and the Maximum Neighborhood Size: A sequence of several runs of the VNS procedure is performed to achieve the best result. Each run starts from the final solution obtained in the previous run. The initial maximum neighborhood size ($NhSize_{max}$) value, sent to the first VNS run in the sequence, was set to $NhSize_{max} = 2 \times \sqrt{n}$, where n is the total number of nodes.

However, we realized that during the first VNS run, improvement happens quickly for most $NhSize$ values, even for the large ones among them. Subsequent VNS runs, though, usually respond only to smaller changes in the solution. In other words, smaller neighborhood sizes seem to be more beneficial in subsequent VNS runs, since larger changes seem to cause a disturbance of the current solution and may reduce its quality. Accordingly, after each VNS run, $NhSize_{max}$ was reduced by 1/4 of its value. The reduction is repeated until $NhSize_{max}$ reaches a minimum value of $NhSize_{max}/4$, at which stage no further reduction is performed, and the VNS procedure uses the current $NhSize_{max}$ for all remaining runs. Thus, the algorithm is adaptive in the sense that $NhSize_{max}$ passed to the VNS is not fixed and depends on the current stage of the search.

The VNS could be repeated for a fixed number of iterations, or until no improvement is realized in the current solution for a number of attempts. We chose the second approach, and stopped the repetition of the VNS when no improvement happens in 5 consecutive attempts.

Stopping and Replacement Criteria for Individual VNS Runs: As explained in Section 3, VNS is based on systematically increasing the neighborhood size ($NhSize$), within which a new solution is generated, from 1 up to $NhSize_{max}$. Thus, the shaking and the local search procedures are repeated for all values of $NhSize = 1, 2, 3...NhSize_{max}$. However, we found that in some cases, the current solution may not respond to changes in the neighborhood size and reach a stage of stagnation. Therefore, rather than indiscriminately increasing $NhSize$ up to the pre-specified maximum, we chose to also end each VNS run when the solution has not changed for a certain number of consecutive attempts of increasing $NhSize$. The number of attempts was again chosen to be $NhSize_{max}/4$. Thus, the VNS will be *adaptive* in the sense that it will stop the shaking and the local search cycle, when no benefit seems to be realized from increasing $NhSize$.

In our algorithm, the VNS procedure repeats the shaking and the local search for the ***same*** $NhSize$ for a number of trials. When the number of trials reaches a certain pre-defined limit, the shaking and the local search cycle stops for the current $NhSize$, and the VNS moves on to the next $NhSize$.

The SA Temperature Value: As previously mentioned, we use an SA acceptance criterion within the VNS to replace the current solution. To allow for an adaptive calculation of the SA starting temperature for each instance individually, we adopted the approach in [1]. The temperature is calculated based on the average value of $\Delta cost$, where $\Delta cost$ is the difference in the objective value between some randomly generated solutions for the current problem instance.

Normally, by the end of each complete VNS run, the SA temperature would have reached a small value that should not permit the acceptance of any worse solutions. If we were then to start the next VNS run in the sequence with such small value, there would be no benefit to the SA acceptance criterion, since all worse solutions would be rejected. On the other hand, starting a new VNS run with the initial temperature too high is not advisable, since many worse solutions would be accepted, possibly causing serious loss of solution quality. To achieve a balance between these two situations, the final temperature value reached in the current VNS run is *doubled* before the beginning of the next VNS run.

5 The AVNS-SA Algorithm

To put it all together, Algorithm 1 shows the main Adaptive VNS-SA (AVNS-SA) heuristic, which will invoke the VNSSA procedure (Algorithm 2).

Algorithm 1 Adaptive VNS-SA (AVNS-SA) Algorithm

1: Find an initial solution ($InitSol$) and calculate the starting SA temperature ($StartTemp$).
2: $NhSize_{max} \leftarrow 2 \times \sqrt{n}$, where n is the number of nodes
3: $MaxStagnation \leftarrow NhSize_{max}/4$
4: $Decrement \leftarrow NhSize_{max}/4$
5: Initialize $MaxAttempts$ to a small number (e.g. 5).
6: $NoImprovement \leftarrow 0$
7: **repeat**
8: $NewSol = VNSSA(InitSol, NhSize_{max}, StartTemp, MaxStagnation)$
9: **if** ($NhSize_{max} > Decrement$) **then**
10: $NhSize_{max} \leftarrow NhSize_{max} - Decrement$
11: **else**
12: $NhSize_{max} \leftarrow Decrement$
13: **if** ($NewSol$ is not better than $InitSol$) **then**
14: $NoImprovement++$
15: **else**
16: $NoImprovement \leftarrow 0$
17: $InitSol \leftarrow NewSol$
18: $StartTemp \leftarrow StartTemp \times 2$
19: **until** ($NoImprovement$ reaches $MaxAttempts$)

6 Experimental Results

The algorithm was tested on instances created by [6]. There are 2 types of problem instances. Small instances have a number of customers n in {20, 30, 40, 50, 60}. For these instances, the optimum is known and was obtained using the exact method proposed in [5]. There are also large instances with n in {100, 200,

Algorithm 2 The VNSSA Algorithm

Input: $InitSol$, $NhSize_{max}$, $StartTemp$, $MaxStagnation$
Output: a new, possibly improved, solution X
$k \leftarrow 0$ { Initialize the current neighborhood size k }
$Stagnation \leftarrow 0$
$NumTrials \leftarrow LIMIT$ {$LIMIT$ is the maximum allowed number of trials}
$X \leftarrow InitSol$
repeat
  $k++$ { Increment the current neighborhood size}
  $Trials \leftarrow 0$
  while ($Trials < NumTrials$) **do**
    $Shaking(X, XI, k)$ {Displace a sequence of nodes in X up to a maximum of k, with or without inversion. Result stored in XI}
    $LocalSearch(XI, XII)$ {2-Opt applied on XI. Result stored in XII}
    if ($Objective(XII) < Objective(X)$) **then**
      $X \leftarrow XII$
    else
      Accept XII using SA acceptance probability
      $StartTemp \leftarrow StartTemp \times \alpha$ {Decrement current temperature}
      $Trials++$ {Increment number of trials only when a worse solution is found}
    end while
  if (X did not change in the last iteration (i.e., for the current neighborhood size k)) **then**
    $Stagnation++$
  else
    $Stagnation = 0$
until ($Stagnation = MaxStagnation$) or ($k = NhSize_{max}$)

300, 400, 500}. For each combination of n and a different vehicle capacity Q in {10, 15, 20, 25, 30, 35, 40, 45, 1000}, 10 problem instances have been created and given the letters {'A' to 'J'}. The data set and the results obtained in [4] can be downloaded from the **Pickup and Delivery Site** of Hernández-Pérez[2]: http://webpages.ull.es/users/hhperez/PDsite/index.html

We chose the test cases with the smallest vehicle capacity ($Q = 10$), i.e, the hardest instances. The algorithm was run 5 times on each test case from 20-300 customers. On the other hand, only one run was performed on test cases of 400 and 500 customers, due to time limitation. Also, a number of computers with different specifications were used to run the algorithm. For this reason, the run times we quote here will vary according to the platform.

Results on Problem Sizes 20-60 customers: In this experiment, the algorithm was able to achieve the optimum results at least once in the 5 runs for 39 out of the 50 test cases. The maximum relative difference to the optimum was less than 2% among the 11 cases where the optimum was not found, which was

[2] New best results were obtained by the GA in [9] for vehicle capacity $Q = 10$, but they do not appear in the pickup and delivery site yet.

for test case N50q10D. The processing time ranged on average from 0.66 seconds for 20 customers problems to 47.79 seconds for 60 customers problems.

Results on Problem Sizes 100-500 customers: Table 1 shows the results of the AVNS-SA algorithm on large size problems, from 100 to 500 customers. The table shows the best result achieved and the average result of the 5 runs. The average value is replaced by the best result for problems of size 400 and 500, since the algorithm was run only once on these problems. Finally, the previous best known results are also shown in the table. Most of the best known results were found by the GA in [9]. The best result achieved by the AVNS-SA algorithm is shown in boldface if it was better than the best known result.

Table 1. AVNS-SA Results (100-500 customers)

Name	**Best**	**Avg**	**Prev-Best**	**Name**	**Best**	**Avg**	**Prev-Best**
N100q10A	**11741**	12175.8	11828	N300q10F	**24042**	24290.6	24826
N100q10B	**13066**	13410.6	13114	N300q10G	**23683**	23945	23868
N100q10C	**13893**	14073.8	13977	N300q10H	**21555**	21824.6	21625
N100q10D	14328	14597	**14253**	N300q10I	**23871**	24110.2	24513
N100q10E	11430	11823.6	**11411**	N300q10J	**22503**	22688.8	22810
N100q10F	11813	11947	**11644**	N400q10A	**30657**	30657	31486
N100q10G	**12025**	12118	12038	N400q10B	**24248**	24248	24262
N100q10H	12821	12844	**12818**	N400q10C	**27853**	27853	28741
N100q10I	**14025**	14278.6	14032	N400q10D	**23750**	23750	24508
N100q10J	13476	13642.8	**13297**	N400q10E	**24798**	24798	25071
N200q10A	17690	17849.2	**17686**	N400q10F	**26625**	26625	26681
N200q10B	**17618**	17887.8	17798	N400q10G	23925	23925	**23891**
N200q10C	16535	16626.6	**16466**	N400q10H	25628	25628	**25348**
N200q10D	**21228**	21594.2	21306	N400q10I	**28262**	28262	28714
N200q10E	**19220**	19485.2	19299	N400q10J	**24847**	24847	26010
N200q10F	**21627**	21677.4	21910	N500q10A	**27904**	27904	28742
N200q10G	**17361**	17634	17712	N500q10B	**26612**	26612	26648
N200q10H	**20953**	21191.4	21276	N500q10C	**30247**	30247	30701
N200q10I	**18020**	18328.2	18380	N500q10D	**29875**	29875	30794
N200q10J	19016	19240.4	**18970**	N500q10E	**29978**	29978	30674
N300q10A	**22940**	23163	23242	N500q10F	**28527**	28527	28882
N300q10B	**22473**	22920.4	22934	N500q10G	**26171**	26171	27107
N300q10C	**21183**	21454	21800	N500q10H	**35805**	35805	36857
N300q10D	**25220**	25500.6	25883	N500q10I	**30247**	30247	30796
N300q10E	**26636**	26934	27367	N500q10J	**30428**	30428	31255

Table 1 shows that the algorithm was able to improve best known results for 50% of the 100 test cases, 70% of the 200 test cases, 100% of the 300 test cases, 80% of the 400 test cases, and finally 100% of the 500 test cases.

The overall average of the 5 runs for all test cases of size 100 is 13091.12, which is only 1% worse than the average result of the GA in [9], having a value of

12954.16. Moreover, our overall average for the 200 test cases is 19151.44, while the average of the heuristic in [9] for the same test cases in 10 runs is 19339.48, i.e., our results account for an improvement of approximately 1%. On the other hand, the overall average of our results for the 300 test cases was 23683.12, with an improvement of more than 2% compared the average of their results for the same test cases, which is 24224.28.

In addition, the average result of the 10 instances of size 500 achieved by our algorithm was 29579.4. This is an improvement of approximately 3% over the average of the best results of [9], having the value 30377.1. These results may also indicate that our algorithm seems to perform even better on larger size problems. The average processing time in this experiment ranged from approximately 542.22 seconds for 100 customers instances to 151103.04 seconds for 500 customers instances.

To further test the robustness of the AVNS-SA algorithm, we performed an additional experiment by running the algorithm on 100-customers problems vehicle capacities of 20 and 40, running the algorithm 5 times on each test case. In this experiment, the algorithm outperformed previous heuristics in 4 out of 10 test cases, for vehicle capacity $Q = 20$, and was able to match the best known result in 6 out of 10 cases, for vehicle capacity $Q = 40$. The average processing time for $Q = 20$ instances was 155.76 seconds, and for $Q = 40$ instances was 146.17 seconds.

Contrary to the exceptional results achieved by our AVNS-SA algorithm, its processing time in general was to a large extent disappointing. For example, the average processing time for 100 customers problems was 542.22 seconds, while the processing time reported by [9] was 21.12 seconds for the same category.

7 Summary and Future Work

In this research, we investigated a new adaptive hybrid VNS/SA approach to the 1-PDP. Adaptation is applied in both the maximum neighborhood size allowed in each VNS run, and in the stopping condition for each VNS run. Searching within smaller neighborhood sizes is preferred in our approach, and larger sizes are only attempted when this looks promising from the search perspective.

Experimental results on a large number of problem instances indicated that our algorithm outperforms previous heuristics in most hard test cases, where the vehicle capacity is smallest. This is especially noticeable for large problem sizes. The algorithm was able to achieve the optimum results for all but few test cases in the small size problems, and was able to achieve new best known results for 90% of the large test cases. The algorithm is also robust enough, since it performs equally well on a wide range of problem instances, e.g. instances with a different vehicle capacity, without the need for any parameter tuning.

These distinguished results, though, come at the expense of computation time. Although we cannot provide an accurate analysis at this stage, because of the use of different processors to run the experiments, we recognize that the running time of the algorithm is rather too long.

In the future, we will continue investigating possible techniques to reduce the run time, for example by reducing the number of VNS runs, or changing the stopping condition for each individual run. Our computational experimentation indicates that some problem instances need fewer than 5 consecutive attempts (without improvement) to reach the best results. However, for other instances, reducing the maximum number of attempts to less than 5 may cause the algorithm to stop prematurely and produce lower quality result. More investigation of the best termination criterion is therefore needed to reduce the overall processing time. Other possible improvement attempts, with respect to the run time, should be oriented towards the local search procedure, since it is the most time consuming part of the algorithm. We can try to reduce the number of calls to this algorithm, or make it optimize only part of the solution rather than whole solution. For example, exchanges can be only restricted to edges with a certain number of closest neighboring nodes.

References

1. Dorband, J., Mumford, C. L., and Wang, P. Developing an ace solution for two-dimensional strip packing. In *18th International Parallel and Distributed Processing Symposium Workshop on Massively Parallel Processing* (2004).
2. Hansen, P., and Mladenović, N. An introduction to variable neighborhood search. In *Meta-heuristics, Advances and trends in local search paradigms for optimization*, S. Voss, S. Martello, I. H. Osman, and C. Roucairol, Eds. Kluwer Academic Publishers, 1998, pp. 433–458.
3. Hansen, P., and Mladenović, N. Variable neighborhood search: Principles and applications. *European Journal of Operational Research 130*, 3 (2001), 449 – 467.
4. Hernández-Pérez, H., Rodríguez-Martín, I., and Salazar-González, J.-J. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research 36*, 5 (2008), 1639 – 1645.
5. Hernández-Pérez, H., and Salazar-González, J.-J. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics 145*, 1 (2004), 126 – 139.
6. Hernández-Pérez, H., and Salazar-González, J.-J. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science 38*, 2 (2004), 245–255.
7. Martinović, G., Aleksi, I., and Baumgartner, A. Single-commodity vehicle routing problem with pickup and delivery service. *Mathematical Problems in Engineering 2008* (2008), 1–17. doi:10.1155/2008/697981.
8. Placek, M., Hartl, R. F., and Doerner, K. A variable neighborhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics 10* (2004), 613–627.
9. Zhao, F., Li, S., Sun, J., and Mei, D. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Industrial Engineering 56*, 4 (2008), 1642 – 1648.