Computers & Operations Research 36 (2009) 3215 – 3223

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

An ant colony system (ACS) for vehicle routing problem with simultaneous delivery and pickup

Yuvraj Gajpal*, Prakash Abad

Michael G DeGroote School of Business, McMaster University, Hamilton, Ontario, Canada L8S4M4

ARTICLE INFO

Available online 5 March 2009

Keywords:
Metaheuristic
Ant-colony
Vehicle routing
Simultaneous delivery and pickup

ABSTRACT

In this paper we use an ant colony system (ACS) algorithm to solve the vehicle routing problem with simultaneous delivery and pickup (VRPSDP) which is a combinatorial optimization problem. ACS is an algorithmic approach inspired by the foraging behavior of real ants. Artificial ants are used to construct a solution for the problem by using the pheromone information from previously generated solutions. The proposed ACS algorithm uses a construction rule as well as two multi-route local search schemes. The algorithm can also solve the vehicle routing problem with backhaul and mixed load (VRPBM). An extensive numerical experiment is performed on benchmark problem instances available in literature. It is found that ACS gives good results compared to the existing algorithms.

1. Introduction

Traditional supply chain management has focused on the forward flow of materials. However, the new environmental regulations and the incentives for returning and reusing products have increased the reverse flows in supply chains in recent years. The need for managing the reverse flow of materials has opened a new field in supply chain management called reverse logistics. Reverse logistics attempts to generate additional revenue by recapturing the value otherwise lost or underutilized in the supply chain.

One of the problems in reverse logistic is that the vehicle planner and dispatcher needs to consider the delivery and pickup of materials from a customer by a single vehicle. The vehicle routing problem with simultaneous delivery and pickup (VRPSDP) is characterized by the following: a fleet of identical vehicles located at the depot to be used to serve customers distributed geographically in the area. The capacity of each vehicle is Q. A customer requires a given shipment to be delivered and another load to be picked up during the single visit of a vehicle. The objective is to design a set of minimum cost routes to serve all customers so that the load on a vehicle is below vehicle capacity Q at each point in the route. VRPSDP is an NP-hard problem in the strong sense because when we set either all pickup demands or all delivery demands to zero, the problem reduces to the capacitated vehicle routing problem (CVRP) which is a known NP-hard problem. VRPSDP is closely related to the vehicle routing problem with backhauls (VRPB) which requires that all deliveries to the linehaul customers must be made before any pickup from a backhaul customer. When the delivery or pickup is allowed in any order of sequence, the problem is called vehicle routing problem with backhaul and mixed load (VRPBM). VRPBM is considered to be a special case of VRPSDP because when we set either the delivery or pickup demand of each customer to zero, the problem reduces to VRPBM. See Dethloff [12] and Bianchessi and Righini [2] for mathematical formulation of VRPSDP.

Berbeglia et al. [1] have provided the classification scheme for a static pickup and delivery problem. In this classification scheme, the problems of VRPSDP and VRPBM fall under the category of one to one pickup and delivery problem.

2. Literature survey

We survey the literature in two parts. The first part is on the VRPBM where the linehaul and backhaul customers are served in any sequence. The second part is on the VRPSDP in which a customer requires some quantity of materials to be delivered as well as a load to be picked up during the same visit.

There are relatively few papers which deal with VRPBM. Deif and Bodin [10] extended the classical saving heuristic of Clarke and Wright to solve VRPBM. They calculate the saving between the linehaul and backhaul customer by adding a penalty term to postpone the insertion of the backhaul customer in the route. Cosco et al. [8] combined the saving method with load based insertion method to solve VRPBM. Golden et al. [18], Cosco et al. [8] and Salhi and Nagy [29] combined Clarke and Wright saving heuristic with an insertion

* Corresponding author. Fax: +1 9055218995.
E-mail address: gajpaly@mcmaster.ca (Y. Gajpal).

0305-0548/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2009.02.017

based heuristic. These papers first solve CVRP consisting only of the linehaul customers and then insert backhaul customers in the route. They differ in the way a backhaul customer is inserted in the existing route. Golden et al. [18] consider the number of linehaul customers left on the route after the insertion of a backhaul customer whereas Cosco et al. [8] consider the remaining load to be delivered after the insertion of a backhaul customer. Salhi and Nagy [29] adopted a similar approach but they insert two backhaul customers at a time in the route. Mosheiov [23] proposed a heuristic based upon the tour partitioning approach. Wade and Salhi [34] designed an ant colony system (ACS) algorithm for VRPBM. Recently Bianchessi and Righini [2] have proposed a tabu search algorithm using complex and variable neighborhood approach.

VRPSDP was introduced by Min [22] to solve a real life problem of transporting books between libraries. He used the cluster first and route second approach to solve the problem. A TSP problem is solved exactly during the routing phase and if the TSP route violates the vehicle capacity constraint then the TSP problem is resolved by penalizing the arc violating the vehicle capacity constraint. Salhi and Nagy [29] solved VRPSDP using the insertion based heuristic that they designed for VRPBM. They also extended the VRPBM heuristic to multi-depot version of the problem. Dethloff [12] described the application of VRPSDP in reverse logistics. He proposed an insertion based heuristic which uses the concept of the residual load.

Recently a number of authors have proposed metaheuristics—mainly tabu search—to solve VRPSDP (e.g. [9,5,32,2]). An exact method based upon the branch and price method is proposed for VRPSDP by DellAmico et al. [11].

In recent years, several authors have applied ACS metaheuristic to vehicle routing problems. For an introduction to the ACS approach see Dorigo et al. [16]. Bullenheimer et al. [4] designed an ACS for CVRP. This ACS is further improved by Doerner et al. [13] by combining it with the savings heuristic. Reimann et al. [27] proposed a decomposition approach (D-Ants) for CVRP. Wade and Salhi [34] designed an ant colony algorithm for VRPBM. In most of the above papers, local search is used to improve a given vehicle route. In this paper we present an ACS algorithm for solving VRPSDP. We use two multi-route local search schemes based upon interchange and insertion operations and an exchange of sub-paths.

This paper is organized as follows. Section 3 describes the local search schemes used in ACS. Section 4 describes the ACS algorithm for VRPSDP. An extensive computational experiment with benchmark problem instances available in literature is presented in Section 5. We compare the performance of different algorithms with ACS. Conclusions follow in Section 6.

3. Local search

Local search is an important part of metaheuristics including ACS. In this section we present three types of local search schemes: 2-opt scheme, the customer insertion/interchange multi-route scheme and the sub-path exchange multi-route scheme. Given that the load fluctuates over the route, one important issue in VRPSDP and VRPBM is checking the feasibility of a route during local search. In this paper, we use an approach based upon the notion of cumulative net-pickup load. The detail of the cumulative net-pickup approach can be found in Gajpal and Abad [17] which describes how the feasibility check is performed in constant time for the three local search schemes described in next sub-sections.

3.1. 2-Opt local search scheme

The 2-opt local search scheme was originally proposed by Lin [21] for improving the traveling salesman problem (TSP) solution. It is one of the best known local search schemes for TSP and it is based on the arc exchange approach. In CVRP, each vehicle route is a TSP. The 2-opt scheme starts with a given route and breaks it at two places to generate three segments. We view a route as a path that starts and ends at the depot. The route is then reconstructed by reversing the middle segment (the segment that does not consist the depot). A given route is broken at each combination of two places and is updated whenever there is an improvement. The process is continued until there is no further improvement. In CVRP, the new route is always feasible while performing the 2-opt local search since the total demand of a given route does not change. However, this is not true for VRPSDP. In VRPSDP, the load on the vehicle fluctuates and it may exceed the vehicle capacity at some arc. In this paper we define the feasible route as a route in which the load along each arc is less than or equal to the vehicle capacity and the total distance traveled is less than or equal to the allowable maximum distance.

3.2. Customer insertion/interchange multi-route scheme

This approach uses two types of operations for a given customer. Let R_r denote the route of vehicle r and let $S = \{R_1, R_2, \ldots, R_r, \ldots, R_s, \ldots, R_{v-1}, R_v\}$ represent a solution to VRPSDP. Here v is the number of vehicles used in solution S. Let the number of customers in route R_r be n_r. Route R_r thus contains the list of n_r customers plus two copies of the depot at position 0 and $n_r + 1$. Let $\sigma_r(i)$ be the customer in the i^{th} position in route R_r. Then, the two operations are described below for customer $\sigma_r(i)$.

1. Insertion operation: In this operation customer $\sigma_r(i)$ is removed from the i^{th} position in route R_r and is inserted at each other position in R_r as well as at each position in route R_s, $s \neq r$.
2. Interchange operation: In this operation customer $\sigma_r(i)$ from route R_r is moved to his best position k in route R_s and customer $\sigma_s(k)$ from route R_s is moved to his best position in R_r, $s \neq r$.

The basic steps in the local search are:

Step 1: Initialize $\Omega = \{1, 2, \ldots, n\}$.

Step 2: Randomly choose a customer, say $\sigma_r(i)$, in R_r from set Ω. Remove customer $\sigma_r(i)$ from set Ω.

Step 3: Perform the insertion of customer $\sigma_r(i)$ at his best position in R_r or R_s, $s \neq r$ and evaluate the total tour length of the resultant solution.

Step 4: Consider the interchange of customer $\sigma_r(i)$ and customer $\sigma_s(k)$ in R_s, $s \neq r$; $k = 1, 2, \ldots, n_s$. Identify the best interchange customer and the corresponding solution.

Step 5: Choose the best solution obtained from steps 3 and 4 and compare it with the current solution. If the best solution from steps 3 and 4 is better than the current solution, then update the current solution.

Step 6: If $\Omega = \phi$ then stop and report the current solution, else go to step 2.

In order to reduce CPU time, the following neighborhood criteria are used before performing an insertion/interchange operation. Let Γ be some fixed number and let N_Γ be the set of Γ nearest customers (in term of the distance) of customer $\sigma_r(i)$. The interchange operation between customer $\sigma_r(i)$ and customer $\sigma_s(k)$ is performed only if at least one of the Γ neighbors of customer $\sigma_r(i)$ is served by vehicle s and at least one of the Γ neighbors of customer $\sigma_s(k)$ is served by vehicle r. Similarly, the insertion of customer $\sigma_r(i)$ in the route of vehicle s isperformed only if one of the Γ neighbors of customer $\sigma_r(i)$ is served by vehicle s.

The customer insertion/interchange multi-route scheme is similar to the intersection/exchange heuristics used by other researchers (e.g., [25,33,20]). Our insertion operation is similar to the previous work but our interchange operation differs in the way customers are placed in their destination routes. In the previous work, the positions

of customer $\sigma_r(i)$ from vehicle r and customer $\sigma_s(k)$ from vehicle s are interchanged. In our approach, customer $\sigma_r(i)$ from vehicle r is placed in the route of vehicle s at his best insertion place. Similarly, customer $\sigma_s(k)$ from vehicle s is placed in the route of vehicle r at his best insertion place.

The cost of a solution generated by an interchange or insertion operation can be determined in constant time. In the absence of the customer neighborhood criteria, the theoretical computational complexity of one iteration of the insertion/interchange multi-route scheme can be calculated as follows.

The insertion operation has complexity of $O(n^2)$. The interchange operation considers a customer for interchange with each customer from another vehicle route. We assume that the n customers are evenly distributed among v routes. Hence there are in total $n \times (n - n/v)$ pairs for possible interchange. Each customer from a given pair is placed at his/her best insertion place and to find the best insertion place for a customer, we need to evaluate the customer's insertion at n/v places on average. In addition, finding the best insertion place for one customer in the pair is independent of finding the best insertion for the second customer. Therefore, the overall complexity of an insertion/interchange multi-route scheme is

$$O(n^2) + O(n) \times O(n - n/v) \times (O(n/v) + O(n/v)) = O(n^2) + O(n^3/v^2) = O(n^3/v^2).$$

Using the customer neighborhood criteria should reduce CPU time in practice. However, in the worst case scenario, there may still be $n \times (n - n/v)$ pairs of customers for interchange even in the presence of the customer neighborhood criteria. Thus, the theoretical computational complexity of the scheme remains the same with or without customer neighborhood criteria.

3.3. Sub-path exchange multi-route scheme

The customer insertion/interchange multi-route scheme considers moving a given customer to another vehicle route. The sub-path exchange multi-route scheme can move more than one customers from a given route to another route. Let $S = \{R_1, R_2, \ldots, R_r, \ldots, R_s, \ldots, R_{v-1}, R_v\}$ represent a solution to VRPSDP. The sub-path exchange multi-route scheme considers two routes R_r and R_s and combines them in two new routes R'_r and R'_s to produce a new solution $S' = \{R_1, R_2, \ldots, R'_r, \ldots, R'_s, \ldots, R_{v-1}, R_v\}$. Consider sub-path $\{\sigma_r(i), \ldots, \sigma_r(j)\}$ belonging to route R_r and sub-path $\{\sigma_s(k), \ldots, \sigma_s(l)\}$ belonging to R_s. The new route R'_r is obtained by replacing sub-path $\{\sigma_r(i), \ldots, \sigma_r(j)\}$ by either sub-path $\{\sigma_s(k), \ldots, \sigma_s(l)\}$ or reverse sub-path $\{\sigma_s(l), \ldots, \sigma_s(k)\}$. Similarly the new route R'_s is obtained by replacing sub-path $\{\sigma_s(k), \ldots, \sigma_s(l)\}$ by either sub-path $\{\sigma_r(i), \ldots, \sigma_r(j)\}$ or the reverse sub-path $\{\sigma_r(j), \ldots, \sigma_r(i)\}$.

We use the customer neighborhood criteria to reduce the CPU time. The exchange of sub-path $\{\sigma_r(i), \ldots, \sigma_r(j)\}$ of vehicle r with sub-path $\{\sigma_s(k), \ldots, \sigma_s(l)\}$ of vehicle s is considered only if customer $\sigma_s(k)$ is among the 10 nearest neighbors of customer $\sigma_r(i)$.

Each possible sub-path from route R_r and each possible sub-path from route R_s (subject to the nearest neighborhood criteria) is considered for possible exchange and the best pair is used to produce new routes R'_r and R'_s.

There are a total of v vehicles in a given solution. Assuming that n customers are evenly distributed among v routes, there are $n/v \times (n/v - 1)/2$ sub-paths for each vehicle route. Each sub-path from a vehicle route is exchanged with each other sub-path from another vehicle. Thus, the overall complexity of the sub-path exchange scheme in the absence of the neighborhood criteria is

$$O(v) \times O(n^2/v^2) \times O(v - 1) \times O(n^2/v^2) = O(n^4/v^2).$$

Let Γ be some fixed number and let N_Γ be the set of Γ nearest customers of customer $\sigma_r(i)$. With customer neighborhood criteria, a given sub-path from a vehicle route can be exchanged with the maximum of $\Gamma \times n/v$ sub-paths from all other vehicles. Thus, when a neighborhood criterion is used, the overall complexity of the sub-path exchange scheme is

$$O(v) \times O(n^2/v^2) \times O(\Gamma \times n/v) = O(n^3/v^2).$$

The sub-path exchange multi-route scheme is similar to the CROSS exchange heuristic of Taillard et al. [31] and ICROS heuristic of Braysy and Dullaert [3]. Our sub-path exchange multi-route scheme falls between the above two heuristics. Considering all possible sub-paths for exchange in ACS is similar to CROSS heuristic whereas connecting the reverse sub-paths in ACS is similar to ICROSS heuristic.

4. ACS for VRPSDP

An ACS algorithm makes use of simple agents called ants that iteratively construct solutions to a combinatorial optimization problem. The generation (or construction) of a solution by an ant is guided by pheromone trails and the problem-specific information. An individual ant constructs a complete solution by starting with the null solution and iteratively adding solution components. After the construction of a solution, each ant gives feedback by depositing pheromone (i.e., updating trail intensity) for each solution component. Typically, solution components that are part of better solutions or solution components that are used by ants over many iterations receive higher amounts of pheromone. These solution components in turn are more likely to be used in future iterations. This is achieved by using the evaporation factor or persistence of trail while updating trail intensities. See Stuetzle and Hoos [30] for details on the application of ACS to combinatorial optimization problems. Basic steps of an ACS algorithm are presented below. The detailed explanation of the steps is given in the next sub-sections:

Step 1: Initialize the trail intensities and parameters using an initial solution based upon the nearest neighborhood heuristic.

Step 2: While (termination condition is not met) do the following:

- Generate an ant-solution for each ant using the trail intensities.
- Improve each ant-solution by carrying out local search.
- Update elitist ants.
- Update trail intensities.

Step 3: Return the best solution found so far.

4.1. Initial solution

We construct an initial solution using the nearest neighborhood heuristic by iteratively adding customers to a vehicle route. We start a route from the depot and visit the nearest customer in terms of the distance. If visiting the nearest customer is feasible, we move to that customer; otherwise we return back to the depot and start another route from the depot. Likewise we proceed till all customers are served. Let L be the total tour length of the initial solution. The trail intensity of visiting customer b immediately after customer a is denoted by τ_{ab}. Initially all τ_{ab} are set to $1/L$.

4.2. Generation of a solution by an ant

An artificial ant constructs a complete solution (i.e., tour) by successively visiting customers until all customers are visited. Each ant begins the tour from either the first or the last served customer in a vehicle route in the best solution found to date. Thus if v is the number of vehicles used in the best solution to date, we use $2v$ ants

in the current iteration. Each ant constructs a complete solution. The choice of the next customer to be visited is affected by two factors: (1) the trail intensity τ_{ab} which stores the information on the selection of this solution component in previous iterations, (2) the saving value $s(a,b)$ which represents the saving (in terms of the distance traveled) when customers a and b are served jointly by one vehicle instead of two separate vehicles. Let the depot be customer 0 and let $c(a,b)$ be the distance between customers a and b. Then the saving value $s(a,b)$ is calculated as follows:

$$s(a,b) = c(0,a) + c(0,b) - c(a,b). \tag{1}$$

Here $s(a,b)$ is the problem-specific information used in the ACS algorithm. See Clarke and Wright [7] for a detailed description of the saving algorithm. The next customer to visit is chosen from m best candidates in terms of the attractive value ξ_{ab} which is calculated as follows:

$$\xi_{ab} = [\tau_{ab}]^{\alpha}[s(a,b)]^{\beta}. \tag{2}$$

Here, parameters α and β determine the relative biases for the trail intensity and the saving value, respectively.

Let Ω_m denote the set of m best customers ranked in terms of the attractive value. If the number of customers not yet visited is less than m, then Ω_m contains all customers not yet visited by the ant. The next customer to be visited is selected using the following probability:

$$P_{ab} = \begin{cases} \dfrac{\xi_{ab}}{\sum_{l\in\Omega_m}\xi_{al}} & \text{if } b \in \Omega_m, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Existing ant colony algorithms of Bullenheimer et al. [4] and Doerner et al. [13] allow Ω_m to include only the feasible customers. Since checking feasibility of a route is a time consuming process in VRPSDP, we consider all remaining customers for a possible move. Thus, we allow Ω_m to include all remaining customers and the feasibility of the route is checked only after the next customer to be visited is selected. If visiting customer b after visiting customer a leads to an infeasible solution, the ant returns to the depot and starts another route in which the first customer to be visited is customer b. A complete solution is built by choosing customers one by one and the construction process ends when all customers are served. During the construction phase, the next customer to visit is selected probabilistically. Thus in each iteration, each ant produces a different solution and this helps in jumping out of a local minimum.

4.3. Local search

After the construction of a solution by the ant, the solution is improved by local search. We use three local search schemes described in Section 3. The three local search schemes are used iteratively till there is no further improvement in the solution. During our experiment, we found that a local minimum is usually reached after three or four cycles.

4.4. Updating elitist ants

Elitist ants are used to update trail intensities. We use γ elitist ants which are distinct from one another. Elitist ants are defined as γ best ant solutions found so far. Elitist ants are updated by comparing the present elitist ant solutions with the current ant solution. If the current ant solution is found to be better than the μ^{th} elitist ant solution, then the current ant solution becomes the new μ^{th} elitist ant solution. The ranks of the previous μ^{th} elitist ant solution and elitist ant solutions below it are lowered by one. In order to keep γ elitist ants distinct from one another, the current solution is considered only if the total tour length in the current solution is not equal to the total tour length in any of the elitist ant solutions. The hashing function used by Osman and Wassan [26] can be used instead of the total tour length. We have used the tour length because it is easy to implement. Also, the probability is very low that two different solutions would have the same tour length.

4.5. Updating trail intensities

After all ants construct their solutions, trail intensities are updated using the solutions of γ elitist ants. The trail intensity of visiting customer b immediately after customer a is updated as follows:

$$\tau_{ab}^{new} = \rho \times \tau_{ab}^{old} + \sum_{\mu=1}^{\gamma} \Delta\tau_{ab}^{\mu}, \quad a = 1,\ldots,n, \;\; b = 1,\ldots,n, \;\; a \neq b. \tag{4}$$

Here ρ is the trail persistence factor (with $0 < \rho < 1$). The first term in Eq. (4) represents the information carried from previous iterations. The second term is the deposition of pheromone by γ elitist ants where

$$\Delta\tau_{ab}^{\mu} = \begin{cases} 1/L^{\mu} & \text{if arc } (a,b) \text{ is traversed by the } \mu\text{th elitist ant,} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Here L^{μ} is the total tour length in the μth elitist ant solution. In the updating scheme used by Bullenheimer et al. [4], the second term in Eq. (4) is multiplied by $(\gamma-\mu+1)$ (i.e. second term in their equation is $\sum_{\mu=1}^{\gamma}(\gamma-\mu+1)\Delta\tau_{ab}^{\mu}$). In our scheme, we use the same multiplier of 1 for each elitist ant. Thus, the existing updating scheme gives extra weight to higher ranked elitist ants. The strategy of extra weight to the best solution may divert the search toward the global minimum but it can also increase the risk of being trapped at a local minimum.

5. Numerical analysis

This section presents the numerical experiment of proposed ACS. We use the benchmark problem instances available in literature for checking the effectiveness of the proposed ACS.

5.1. Numerical analysis for VRPSDP

In this sub-section, we present the numerical results for the proposed ACS algorithm for VRPSDP and compare ACS with the existing metaheuristic approaches.

5.1.1. Parameter settings

The quality of the solution depends upon the number of ants used which in turn affects the CPU time of the algorithm. We have used $2v$ artificial ants. (Here v is the number of vehicles.) Thus, in each iteration, we generate $2v$ new solutions and each new solution is improved by local search. Our algorithm stops when the total number of iterations reaches 100. Parameters α, β, γ and ρ are set by performing sensitivity analysis within a limited CPU time. We set $\alpha=5$, $\beta=5$, $\gamma=10$ and $\rho=0.95$. We thus use 10 elitist ants to update the pheromone. Our customer interchanges multi-route local search scheme uses Γ of 10. Our sub-path exchange multi-route local search scheme uses Γ of size 5. See Dorigo and Stützle [15] for more insights on how to set parameter values and the effect the parameter values can have on solution quality.

In our experiment, we found that the local search has significant impact on the CPU time and the solution quality of the algorithm. The CPU time and the solution quality of the proposed ACS is more sensitive to parameter Γ (used in multi-route local search schemes) compared to the other parameters α, β, γ and ρ. Increasing the value of Γ improves the solution quality; however, it also increases the

CPU time. We found that increasing the value of Γ beyond 10 did not improve the solution quality significantly. Similarly, increasing the value of Γ beyond 5 in the sub-path exchange multi-route scheme did not improve the solution quality significantly.

5.1.2. Benchmark problems

The numerical experiment is performed using three sets of problem instances available in the literature for VRPSDP. The first data set is provided by Min [22] and it consists of the data for a real life problem of transporting books from the main library to branch libraries. There are two trucks available each with capacity of 10,500 pounds. The two trucks are used to deliver and pickup books from 22 branch libraries.

The second data set is provided by Dethloff [12]. This data set consists of 40 problem instances, each containing 50 customers. In this data set, two different geographical scenarios are used. In the first scenario referred to as SCA, both the x and y coordinates of the customers are uniformly distributed in the interval $[0, 100]$; thus customers are scattered in the square of size 100×100. In the second scenario referred to as CON, the coordinates of one half of the customers are distributed in the same way as in SCA and the coordinates of the other half of the customers are uniformly distributed over the interval $[100/3, 200/3]$. The delivery demands, d_a are generated from uniform distribution of $[0, 100]$ and the pickup demands, p_a are generated as $p_a = (0.5 + r_a)d_a$, where r_a is a random number uniformly distributed over interval $[0, 1]$.

The third data set is provided by Salhi and Nagy [29]. They generated problem instances from 14 benchmark problems of Christofides [6] using the following three steps: (1) keep x and y co-ordinates for each customer to be the same as in the original problem, (2) for customer a having co-ordinates x_a and y_a and the original demand d_a^{org}, compute the ratio $rt_a = \min(x_a/y_a, y_a/x_a)$, (3) calculate the new delivery demand for customer a as $d_a = rt_a \times d_a^{ori}$ and the new pickup demand for customer a as $p_a = (1 - rt_a) \times d_a^{ori}$. In this way, a set of 14 instances referred to as set A is generated. Another set of 14 instances referred to as set B is generated from set A by exchanging the pickup and delivery demand for each customer.

Altogether we use 69 problem instances for VRPSDP: 1 instance from Min [22], 40 instances from Dethloff [12] and 28 instances from Salhi and Nagy [29]. The problem instance by Min [22] is referred to as Min, Dethloff [12] instances are referred to as SCA and CON and instances of Salhi and Nagy [29] are referred to as CMTX/Y.

5.1.3. Computational experiment and the performance analysis of ACS

The proposed ACS was coded in C and implemented on an Intel Xeon with 2.4 GHz computer, single cpu cores with catch size of 512 KB. We compare ACS with the following algorithms for VRPSDP.

HA	the hybrid algorithm of Crispim and Brandao [9]
HeuSDP	the algorithm of Chen and Wu [5]
TS	tabu search of Tang and Galvao [32]
LNS	large neighborhood search by Ropke and Pisinger [28]

Generally it is difficult to compare CPU times since different algorithm are tested on different machines. A comparison of CPU times can be done using Mflops (million floating point calculation per second) values for different computers as given in Linpack benchmark report of Dongarra [14]. Table 1 presents Mflops values for the computers relevant to our study. Since some of the computers are not listed in the Linpack report, we used an equivalent machine listed in the report to obtain an approximate Mflops value for the particular machine. We use the Mflops values to scale the running time of an algorithm relative to our computer (i.e., Intel Xeon 2.4 GHz). The conversion factor r is the ratio of the Mflops value of a given computer to the Mflops value of our computer.

Table 1
Mflops of different computers and the conversion factor (r).

Algorithm	Actual computer used	Approx. equivalent comp. available in report	Mflops	r
HA	Pentium II 350 MHz	Pentium II 333 MHz	69	0.078054299
HeuSDP	Pentium IV 1.6 GHz PC	P 4 1.7 GHz	363	0.410633484
TS	Athlon 2 GHZ PC	Pentium IV 2.53 GHz	1190	1.346153846
LNS	Pentium IV at 1.5 GHz	Pentium IV at 1.5 GHz	326	0.368778281
ACS	Intel Xeon 2.4 GHz	Intel Xeon 2.4 GHz	884	1

Table 2
Comparison of ACS with TS over 40 VRPSDP instances of Dethloff [12] over single run.

	TS	ACS
Average solution	764.25	759.02
Average CPU time	3.73	9.29
Average scaled CPU time	5.02	9.29

Table 3
Comparison of ACS with LNS over 40 VRPSDP instances of Dethloff [12] over 10 runs.

	LNS	ACS
Average best solution	761.07	758.64
Average solution	767.4	759.1
Average CPU time	157.93	9.29
Average scaled CPU time	58.12	9.29

Generally, metaheuristic solutions tend to be better than the heuristic solutions. Hence, we do not include the heuristic solutions of Salhi and Nagy [29], Dethloff [12] and Nagy and Salhi [24] in the comparison. Ropke and Pisinger [28] carried out three large scale neighborhood search (LNS), namely standard, 6R-no learning and 6R-normal learning. Since on average, 6R-no learning yields a better solution than the others, we use 6R-no learning in our comparison. LNS uses the best solution over 10 runs while other algorithms use the single run solution. Thus, we report the ACS solution for the single run as well as the best solution over 10 runs for each problem instance. We also report the average solution over 10 runs for each problem instance. We use the best solution and the average solution over 10 runs to compare ACS with LNS while we use the single run solution to compare ACS with TS, HA and HeuSDP.

We did obtain the total tour length of 88.0 for the problem instance Min. This solution is proved to be the exact optimal solution as reported by Halse [19] (see [28]).

5.1.3.1. Numerical results for VRPSDP instances of Dethloff [12]. The average solution over 40 problem instances of Dethloff [12] from a single run for ACS and for TS is reported in Table 2. Performance measures for ACS and LNS are given in Table 3. The detailed results of different algorithms for 40 VRPSDP instances of Dethloff [12] are reported in Table 10 in Appendix A. All the CPU times reported in this paper are in seconds.

Table 2 shows that the average solution of ACS is better than the average solution of TS. Table 3 shows that ACS is better than LNS in terms of the average solution as well as the average best solution. Also, the scaled CPU time of ACS is much less than the scaled CPU time of LNS. The CPU time reported in Table 3 represents the average CPU time per run over 40 VRPSDP problem instances.

ACS has also produced nine *new best known solutions* for Dethloff [12] instances. The new best known solutions are reported in bold letter in Table 10 given in Appendix A.

Table 4
The convention used by different solution techniques.

Solution procedure	Delivery and pickup demand	Distance matrix	Fixed service time	Solution for maximum tour length constraint problem
HA	Fractional	Fractional	Not applicable	Not available
HeuSDP	Integer	Fractional	Not applicable	Not available
TS	Integer	Integer	Not included	Available
LNS	Fractional	Fractional	Included	Available
ACS	Fractional	Fractional	Included	Available

Table 5
Average solution of different algorithms over 14 problem instances (without maximum distance constraints) of Salhi and Nagy [29].

	HA	HeuSDP	TS	ACS
Average solution	829.93	770.13	777.86	755.84
Average CPU time	39	261.28	9.31	151.54
Average scaled CPU time	3.04	107.3	12.58	151.54

Table 6
Comparison between LNS and ACS over 28 problem instances of Salhi and Nagy [29].

	LNS	ACS
Average best solution	932.62	893.22
Overall average solution	949.6	895.31
Average CPU time	813.33	151.54
Average scaled CPU time	299.31	151.54

5.1.3.2. Numerical results for VRPSDP instances of Salhi and Nagy [29]. The average solutions produced by different algorithms for Salhi and Nagy [29] VRPSDP problem instances are reported in Tables 4 and 5. The numerical results for ACS and LNS over 28 VRPSDP instances of Salhi and Nagy [29] are reported in Table 11 in Appendix A. We do not include TS, HA and HeuSDP algorithms in Table 11 because of various reasons. It appears that the original data set for 28 VRPSDP instances of Salhi and Nagy [29] is not available. TS and HeuSDP have used integer values for generating delivery and pickup demands while the others have used fractional values for the same. TS solutions assume integer values for the distance matrix and do not consider fixed service time in problem instances with the maximum tour length constraint. HA and HeuSDP do not report the solutions for the problem instances with the maximum tour length constraint. The summary of different conventions used by different author is presented in Table 4.

We present average solutions in Table 5 for ACS, HA, HeuSDP and TS over only the 14 problem instances in which the maximum tour length constraint is not imposed. If one ignores the different rounding convention used, ACS has performed better than the other algorithms.

Summary results over 10 runs of ACS and LNS are given in Table 6. The CPU time reported in Table 6 represents the average CPU time per run over 28 VRPSDP problem instances. Results from Table 6 indicate that ACS is better than LNS in terms of solution quality as well as CPU time. ACS has also produced 22 *new best known solutions* and these new best known solutions are highlighted in bold letters in Table 11 in Appendix A.

5.2. Numerical analysis for VRPBM

We implemented the algorithm designed for VRPSDP also on VRPBM without any modification of the algorithm. We keep the parameter settings of ACS for VRPBM to be the same as for VRPSDP. This section compares numerical results of ACS with the existing metaheuristic approaches.

Table 7
Comparison between HA and ACS algorithms over 21 VRPBM instances (without maximum distance constraints) of Salhi and Nagy [29].

	HA	ACS
50%		
Average solution	797.43	734.44
Average CPU time	45.03	88.37
Average scaled CPU time	3.51	88.37
25%		
Average solution	879.57	813.63
Average CPU time	41.46	112.14
Average scaled CPU time	3.24	112.14
10%		
Average solution	945.29	873.2
Average CPU time	35.81	117.91
Average scaled CPU time	2.80	117.91

Table 8
Comparison between LNS and ACS algorithms over 42 VRPBM instances of Salhi and Nagy [29].

	LNS	ACS
50%		
Average best solution	880.93	881.02
Overall average solution	884.21	882.97
Average CPU time	178.14	150.74
Scaled CPU time	65.70	150.74
25%		
Average best solution	922.43	922.97
Overall average solution	925.30	924.49
Average CPU time	143.07	154.93
Scaled CPU time	52.76	154.93
10%		
Average best solution	955.43	954.74
Overall average solution	958.71	955.89
Average CPU time	133.21	148.47
Scaled CPU time	49.12	148.47

Table 9
Effect of new updating scheme and local search schemes on solution quality.

Setting	Overall average solution	% Deviation from the average best known solution
No. change from current ACS	895.67	0.29
By using Bullenheimer et al. [4] updating scheme of trail intensity	898.51	0.61
In the absence of local search scheme	1135.54	27.15
When 2-opt is applied on ACS	988.12	10.64
When sub-path exchange is applied on ACS	909.52	1.84
When insertion/interchange is applied on ACS	908.71	1.75

5.2.1. Benchmark problems

The numerical experiment is performed using the data set of Salhi and Nagy [29] for VRPBM. They generated three problem subsets referred to as T, Q and H from the 14 problem instances of VRP. In set T, every 10th customers is declared a backhaul customer and is assigned pickup demand equal to the original delivery demand. Similarly, for sets Q and H, every fourth and every second customer is declared to be a backhaul customer. Thus the proportion of backhaul customers is 10% in problem set T, 25% in problem set Q and 50% in problem set H. In our comparison, we refer to the three subsets T, Q and H as CMTT, CMTH and CMTQ, respectively.

5.2.2. Computational experiment and performance analysis of the algorithms

We compare ACS with the following algorithms for VRPBM.

HA the hybrid algorithm of Crispim and Brandao [9]
LNS large neighborhood search by Ropke and Pisinger [28]

The average solutions produced by different algorithms for Salhi and Nagy [29] VRPBM problem instances are reported in Tables 7 and 8. The numerical results of LNS and ACS for 42 problem instances are presented in Table 12 in Appendix B. We do not include HA in Table 12 because the solutions for 21 problem instances which include the maximum tour length constraint are not reported by HA. LNS reports the best solution over 10 runs while HA uses the single run solution. Hence, the solution from a single run as well as the best solution over 10 runs are reported in Table 12 for ACS. We also report the average solution over 10 runs for each problem instance for ACS.

Table 7 gives the average solution as well as the average scaled CPU time over 21 problem instances for HA and ACS. The average solution of ACS is much better than HA but the scaled CPU time for ACS is greater than the scaled CPU time for HA.

Table 8 gives the performance measures for ACS and LNS. The CPU time reported in Table 8 represents the average CPU time per run over 42 VRPBM problem instances. ACS is competitive with LNS in terms of the average best solution, however, in terms of the overall average solution, ACS is better than LNS. ACS has also produced six *new best known* solutions and these solutions are highlighted in Table 12 in Appendix B in bold letters. For comparison purposes, we calculated the average of the best known solutions over 42 VRPBM problem instances. This average is 918.47. Table 12 shows that the average best solution over 10 runs for LNS is 919.61 and it is 919.57 for ACS. Thus, the average best solutions for ACS and LNS are very close to the average best known solution.

Table 10
Total tour length obtained by different algorithms and corresponding CPU time for 40 instances of Dethloff [12].

No.	Inst.	TS		LNS		ACS		
		Single run solution	CPU time	Best solution	CPU time	Single run solution	Best solution	CPU time
1	SCA3-0	640.55	3.37	636.1	232	635.62	**635.62**	6
2	SCA3-1	697.84	3.25	697.8	218	697.84	697.84	6
3	SCA3-2	659.34	3.52	659.3	203	659.34	659.34	6
4	SCA3-3	680.04	3.31	680.6	241	680.04	680.04	6.1
5	SCA3-4	690.5	3.43	690.5	208	690.5	690.5	5.7
6	SCA3-5	659.9	3.67	659.9	226	659.9	659.9	5.1
7	SCA3-6	653.81	3.35	651.1	233	651.09	651.09	6.1
8	SCA3-7	659.17	3.33	666.1	206	659.17	659.17	6.8
9	SCA3-8	719.47	3.4	719.5	190	719.47	719.47	5.4
10	SCA3-9	681	3.41	681	220	681	681	6
11	SCA8-0	981.47	4.14	975.1	98	961.5	**961.5**	11
12	SCA8-1	1077.44	4.27	1052.4	95	1050.38	**1049.65**	11.5
13	SCA8-2	1050.98	4.2	1044.5	94	1044.48	1042.69	11.9
14	SCA8-3	983.34	4.17	999.1	94	983.34	983.34	11.3
15	SCA8-4	1073.46	4.13	1065.5	93	1065.49	1065.49	11.1
16	SCA8-5	1047.24	4.02	1027.1	96	1027.08	1027.08	11.3
17	SCA8-6	995.59	3.85	977	94	971.82	**971.82**	12
18	SCA8-7	1068.56	4.22	1061	92	1063.15	1052.17	12.5
19	SCA8-8	1080.58	3.85	1071.2	98	1071.18	1071.18	11
20	SCA8-9	1084.8	4.2	1060.5	92	1061.23	1060.5	11.5
21	CON3-0	631.39	3.64	616.5	215	616.52	616.52	8.3
22	CON3-1	554.47	3.31	554.5	245	554.47	554.47	7.1
23	CON3-2	522.86	3.45	521.4	232	519.11	**518**	6.9
24	CON3-3	591.19	3.28	591.2	231	591.19	591.19	7.2
25	CON3-4	591.12	3.47	588.8	221	588.79	588.79	6
26	CON3-5	563.7	3.38	563.7	209	563.7	563.7	6.9
27	CON3-6	506.19	3.32	500.8	225	499.05	499.05	7.3
28	CON3-7	577.68	3.51	576.5	227	576.48	576.48	7
29	CON3-8	523.05	3.66	523.1	237	523.05	523.05	7.4
30	CON3-9	580.05	3.36	586.4	207	578.25	578.25	6.8
31	CON8-0	860.48	4.19	857.2	94	857.17	857.17	12.3
32	CON8-1	740.85	3.89	740.9	94	740.85	740.85	12
33	CON8-2	723.32	3.76	716	94	712.89	**712.89**	13
34	CON8-3	811.23	4.12	811.1	98	811.07	811.07	13.9
35	CON8-4	772.25	3.75	772.3	95	772.25	772.25	11.9
36	CON8-5	756.91	3.99	755.7	94	754.88	**754.88**	12.4
37	CON8-6	678.92	4.04	693.1	96	678.92	678.92	12.4
38	CON8-7	814.5	4	814.8	94	811.96	**811.96**	13
39	CON8-8	775.59	3.74	774	94	767.53	**767.53**	12.5
40	CON8-9	809	4.13	809.3	92	809	809	12.9
Average		764.25	3.73	761.07	157.93	759.02	758.64	9.29

Table 11
Total tour length obtained by LNS and ACS and corresponding CPU time for 28 VRPSDP instances of Salhi and Nagy [29].

No.	Inst.	LNS		ACS		
		Best solution	CPU time	Single run solution	Best solution	CPU time
1	CMT1X	467	221	466.77	466.77	5.00
2	CMT2X	704	294	688.05	**684.21**	20.75
3	CMT3X	731	863	721.4	**721.4**	41.25
4	CMT4X	879	1676	857.19	**854.12**	131.75
5	CMT5X	1108	2340	1035.03	**1034.87**	377.50
6	CMT6X	559	113	555.43	**555.43**	14.00
7	CMT7X	901	167	901.11	**900.12**	47.75
8	CMT8X	866	413	865.5	865.5	80.75
9	CMT9X	1205	765	1061.97	**1161.54**	300.00
10	CMT10X	1462	1275	1401.13	**1386.29**	773.50
11	CMT11X	837	1821	844.52	839.66	57.25
12	CMT12X	685	684	663.09	**663.01**	36.25
13	CMT13X	1578	563	1542.86	**1542.86**	160.25
14	CMT14X	885	387	821.75	**821.75**	78.50
15	CMT1Y	467	235	466.77	466.77	5.00
16	CMT2Y	685	331	688.26	684.94	22.25
17	CMT3Y	738	708	724.54	**721.4**	43.75
18	CMT4Y	876	1788	860.85	855.76	140.25
19	CMT5Y	1146	2177	1039.99	**1037.34**	393.50
20	CMT6Y	559	101	555.43	**555.43**	13.75
21	CMT7Y	952	166	901.22	**900.54**	46.25
22	CMT8Y	873	398	865.50	**865.5**	77.75
23	CMT9Y	1271	757	1161.97	**1161.54**	291.75
24	CMT10Y	1552	1255	1400.68	**1395.04**	757.50
25	CMT11Y	920	1376	859.57	**840.19**	52.75
26	CMT12Y	675	539	663.50	**663.5**	39.25
27	CMT13Y	1602	547	1542.86	**1542.86**	160.25
28	CMT14Y	–	–	821.75	**821.75**	74.75
Average		932.70	813.33	895.67	893.22	151.54

Table 12
Total tour length obtained by LNS and ACS and corresponding CPU time for 42 VRPBM instances of Salhi and Nagy [29].

No.	Inst.	LNS		ACS		
		Best solution	CPU time	Single run solution	Best solution	CPU time
1	CMT1H	465	51	465.02	465.02	5.6
2	CMT2H	663	78	663.92	662.63	22
3	CMT3H	701	186	702.94	701.31	35.6
4	CMT4H	829	345	831.39	831.39	125.4
5	CMT5H	983	514	1007.99	992.37	351.4
6	CMT6H	555	31	555.43	555.43	13
7	CMT7H	900	54	900.84	900.84	50
8	CMT8H	866	95	865.5	865.5	85.6
9	CMT9H	1166	177	1163.85	1161.63	306.4
10	CMT10H	1393	296	1392.39	**1383.78**	791
11	CMT11H	818	303	823.06	820.35	45.8
12	CMT12H	629	150	646.74	629.37	32.8
13	CMT13H	1543	125	1542.97	1542.86	164.2
14	CMT14H	822	89	821.75	821.75	81.6
15	CMT1Q	490	41	489.74	489.74	6
16	CMT2Q	733	65	733.15	732.76	26.2
17	CMT3Q	747	128	747.46	747.15	39.8
18	CMT4Q	918	244	918.57	**913.93**	153
19	CMT5Q	1119	381	1137.58	1134.72	451.8
20	CMT6Q	555	30	555.43	555.43	12.8
21	CMT7Q	901	53	902.95	900.69	46.8
22	CMT8Q	866	93	865.5	865.5	74.4
23	CMT9Q	1162	171	1161.97	1161.51	289.6
24	CMT10Q	1389	288	1392.23	**1386.54**	730.2
25	CMT11Q	939	196	939.36	939.36	66.2
26	CMT12Q	729	108	729.55	729.46	42
27	CMT13Q	1543	120	1542.97	1542.97	157.8
28	CMT14Q	822	85	821.75	821.75	72.4
29	CMT1T	520	34	520.06	520.06	7
30	CMT2T	783	57	785	782.77	26
31	CMT3T	798	109	798.07	798.07	42.6
32	CMT4T	1000	212	990.39	**990.39**	166.8
33	CMT5T	1227	333	1232.53	1232.08	460.8
34	CMT6T	555	31	555.43	555.43	11.6
35	CMT7T	903	52	903.05	903.05	39
36	CMT8T	866	95	865.54	865.54	65.6
37	CMT9T	1164	178	1162.8	**1162.68**	261
38	CMT10T	1402	291	1401.56	1400.22	658.6
39	CMT11T	1000	164	998.8	998.8	70.2
40	CMT12T	788	96	787.52	787.52	52
41	CMT13T	1544	127	1542.97	**1542.97**	152.8
42	CMT14T	827	86	826.77	826.77	64.6
Average		919.60	151.48	921.25	919.57	151.38

5.3. Effect of new updating scheme and local search on ACS

To study the effect of the new updating scheme and local search schemes on ACS, we execute the algorithm for 28 VRPSDP problem instances of Salhi and Nagy [29] with different settings. The results are shown in Table 9. For each setting, we report the average solution for a single run of ACS and its percentage deviation from the average best-known solution. The average of the best-known solutions is 893.06 whereas the average solution generated by the proposed ACS is 895.67. This solution is 0.29% away from the average best-known solution. When we used the updating scheme of Bullenheimer et al. [4] in generating trail intensities, the percentage deviation increased to 0.61%.

In order to see the effect of the different local search schemes on ACS, we applied each local search scheme one at a time. In each setting, we varied the stopping criteria as well as the number of ants used to keep the CPU times comparable. In the absence of local search, the average solution deteriorated substantially and the percentage deviation increased to 27.15%. This result indicates that the local search schemes play a significant role in the proposed ACS. The 2-opt local search scheme reduced the percentage deviation to 10.64%. The sub-path exchange multi-route scheme reduced it to 1.84% whereas the insertion/interchange multi-route scheme reduced it to 1.75%. These results indicate that the multi-route local search schemes are more effective than the 2-opt local search scheme.

6. Conclusions

While there are many solution procedures for CVRP, only a few solution procedures exist for VRPSDP. This paper presents an ant colony system (ACS) algorithm for solving VRPSDP. An inherent part of an ACS is local search. We use two multi-route local search schemes, the customer insertion/interchange multi-route scheme and the sub-path exchange multi-route scheme.

Computational experiment with benchmark problem instances has shown that in general the proposed ACS gives better results compared to the existing solution methods for VRPSDP both in terms of the solution quality and the CPU time. Overall, ACS has produced 31 new *best known* solutions for VRPSDP problem instances available in the literature. Further research can be done to improve the local search schemes in ACS. Since VRPBM is a special case of VRPSDP, we tested our ACS on VRPBM instances as well. We have found that ACS is competitive to LNS metaheuristic in solving VRPBM problem instances. ACS has also produced six new *best known* solutions for VRPBM instances.

Acknowledgements

The authors are grateful to the two reviewers and the editor for their valuable comments. This research was supported by Canada NSERC discovery grant 1226-05.

Appendix A. The detailed results for VRPSDP problem instances

The detailed results of different algorithms for 40 VRPSDP instances of Dethloff [12] are reported in Table 10. The numerical results for ACS and LNS over 28 VRPSDP instances of Salhi and Nagy [29] are reported in Table 11.

Appendix B. The detailed results for VRPBM problem instances

The numerical results of LNS and ACS for 42 problem instances are presented in Table 12.

References

[1] Berbeglia G, Cordeau JF, Gribkovskaia I, Laporte G. Static pickup and delivery problems: a classification scheme and survey. TOP 2007;15(1):1–31.
[2] Bianchessi N, Righini G. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. Computers and Operations Research 2007;34(2):578.
[3] Braysy O, Dullaert W. A fast evolutionary metaheuristic for the vehicle routing problem with time windows. International Journal on Artificial Intelligence Tools 2003;12(2):153–72.
[4] Bullnheimer B, Hartl RF, Strauss C. An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research 1999;89:319–28.
[5] Chen JF, Wu TH. Vehicle routing problem with simultaneous deliveries and pickups. The Journal of the Operational Research Society 2006;57(5):579–87.
[6] Christofides N, Mingozzi A, Toth P. The vehicle routing problem. In: Combinatorial optimization. Chichester: Wiley; 1979. p. 315–8.
[7] Clarke G, Wright JW. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 1964;12(4):568–81.
[8] Cosco DO, Golden BL, Wasil EA. Vehicle routing with backhauls: models, algorithms and case studies. In: Vehicle routing: method and studies. Amsterdam: Elsevier; 1988. p. 127–47.
[9] Crispim J, Brandao J. Metaheuristics applied to mixed and simultaneous extensions of vehicle routing problems with backhauls. The Journal of the Operational Research Society 2005;56(11):1296.
[10] Deif I, Bodin L. Extension of the Clarke and Wright algorithm for solving the vehicle routing problem with backhauling. In: Kidder A, editor, Proceedings of the Babson conference on software uses in transportation and logistic management, Babson Park, USA, 1984. p. 75–96.
[11] DellAmico M, Righini G, Salani M. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection. Transportation Science 2006;40(2):235.
[12] Dethloff J. Vehicle routing and reverse logistics: the vehicle routing problem with simultaneous delivery and pick-up. OR Spektrum 2001;23(1):79.
[13] Doerner K, Gronalt M, Hartl RF, Reimann M, Strauss C, Stummer M. SavingsAnts for the vehicle routing problem. In: Applications of evolutionary computing, proceedings, vol. 2279, 2002. p. 11–20.
[14] Dongarra JJ. Performance of various computers using standard linear equation software. Technical Report, University of Tennessee; 2006.
[15] Dorigo M, Stützle T. Ant colony optimization. Bradford Book; 2004.
[16] Dorigo M, Maniezzo V, Colorni A. The ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics—Part B 1996;26:29–41.
[17] Gajpal Y, Abad P. Cumulative net-pickup approach for checking the feasibility of a route in vehicle routing problem with simultaneous pickup and delivery. Technical Report, MGD School of Business, McMaster University, Hamilton, Canada; 2007. p. 1–40.
[18] Golden B, Baker E, Alfaro J, Schaffer J. The vehicle routing problem with backhauling: two approaches. In: Hammesfahr R, editor, Proceedings of the XXI annual meeting of S.E. Times, 1985. p. 90–2.
[19] Halse K. Modeling and solving complex vehicle routing problems. Ph.D. Thesis, Institute of Mathematical Statistics and Operations Research, Technical University of Denmark, Lyngby; 1992.
[20] Kindervater GAP, Savelsbergh MWP. Vehicle routing: handling edge exchanges. Local Search in Combinatorial Optimization 1997:337–60.
[21] Lin S. Computer solutions to the travelling salesman problem. Bell System Technology Journal 1965;44:2245–69.
[22] Min H. The multiple vehicle-routing problem with simultaneous delivery and pick-up points. Transportation Research. Part A, Policy and Practice 1989;23(5):377–86.
[23] Mosheiov. Vehicle routing with pick-up and delivery: tour-partitioning heuristics. Computers Industrial Engineering 1998;34(3):669.
[24] Nagy G, Salhi S. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. European Journal of Operational Research 2005;162(1):126.
[25] Osman IH. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Annals of Operations Research 1993;41:421–51.
[26] Osman IH, Wassan NA. A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls. Journal of Scheduling 2002;5(4):263–85.
[27] Reimann M, Doerner K, Hartl RF. D-ants: savings based ants divide and conquer the vehicle routing problem. Computers and Operations Research 2004;31(4):563–91.
[28] Ropke S, Pisinger D. A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research 2006;171(3):750.
[29] Salhi S, Nagy G. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. The Journal of the Operational Research Society 1999;50(10):1034.
[30] Stuetzle T, Hoos HH. Max-min ant system. Future Generation Computer Systems 2000;16:889–914.
[31] Taillard EP, Badeau P, Gendreau M, Guertin F, Potvin JY. A tabu search heuristic for the vehicle routing problem with soft time windows. Transportation Science 1997;31:170–86.
[32] Tang FA, Galvao RD. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. Computers and Operations Research 2006;33(3):595–619.
[33] Van Breedam A. Improvement heuristics for the vehicle routing problem based on simulated annealing. European Journal of Operational Research 1995;86(3):480–90.
[34] Wade AC, Salhi S. An ant system algorithm for the mixed vehicle routing problem with backhauls. In: Metaheuristics computer decision-making. Dordrecht: Kluwer Academic Publishers; 2004. p. 699–719.