An Ant Colony Optimization Approach for Solving the Multidimensional Knapsack Problem

Soh-Yee Lee, Yoon-Teck Bau
Faculty of Computing and Informatics
Multimedia University
Cyberjaya, Malaysia
lee.soh.yee08@mmu.edu.my, ytbau@mmu.edu.my

Abstract-**Ant Colony Optimization (ACO) is a metaheuristic that has been used to solve variety of optimization problems. In this paper, an ACO approach is proposed to solve the Multidimensional Knapsack Problem (MKP). The algorithm proposed in this paper is called preference-list ACO algorithm with mutation (PACOM). A preference-list is introduced to determine the number of items that should be considered by an ant. In addition, infeasible solutions are allowed to be constructed to reduce the time complexity to generate a solution. We compare the proposed algorithm with several ACO algorithms applied on the MKP. The experimental results of the benchmark problems show PACOM outperforms other ACO algorithms.**

I. INTRODUCTION

The multidimensional knapsack problem (MKP) is one of the most well-known NP-hard optimization problems. MKP modeling has been investigated widely in number of practical applications such as problems in project selections, cargo loading, cutting stock, bin packing, budget control, and the scheduling problem for multi-processor. Given a set of items $J = \{1, 2, \ldots, n\}$ and a set of resource constraints $I = \{1, 2, \ldots, m\}$, the objective of the MKP is to find a set of items from J, such that the total profit of the selected items is maximized while satisfying all the resource constraints in set I. Mathematically, MKP can be expressed as follows:

$$\begin{aligned} &\text{maximize } \sum_{j=1}^{n} p_j x_j \\ &\text{subject to } \sum_{j=1}^{n} r_{ij} x_j \le c_i, \quad i \in I = \{1,\ldots,m\}, \\ &\qquad x_j \in \{0,1\}, \quad j \in J = \{1,\ldots,n\}. \end{aligned} \tag{1}$$

where n is the number of items, m is the number of resources, each item $j \in J$ yields profits p_j upon inclusion. r_{ij} represents the required amount of i-th resource by item j, c_i is the capacity of each resource, and x_j is the decision variable associated with item j ($x_j = 0$ if the item is not selected, $x_j = 1$ if the item is selected). Assumption is made without loss of generality, where $p_j > 0$, $c_i > 0$, $0 \le r_{ij} \le c_i$ and $\sum_{j=1}^{n} r_{ij} > c_i$.

Ant Colony Optimization (ACO) is a recently developed metaheuristic which is inspired by the foraging behaviour of some ant species to solve hard combinatorial optimization problems [1]. The cooperation among the ants and the usage of indirect communication are the basic idea underlying all ACO algorithms [2]. In real world, the ants communicate indirectly via chemical substance, called *pheromone*. The ants lay pheromone on the ground to mark some favourable paths that could serve as guidance to other ants in the colony. ACO exploits a similar mechanism. In ACO, the *artificial ants* build solutions to an optimization problem and update the pheromone trails based on the quality of the solution. The pheromone trails are a numerical value which would be used by the ants to construct a solution. Ants make the probabilistic moves according to the possibly available heuristic information and the pheromone trails. The heuristic information is based on the input data of the problem to be solved.

ACO had been applied to solve various NP-hard problems, including travelling salesman problem [3, 4], vehicle routing problem [5], quadratic assignment problem [6, 7], and the multidimensional knapsack problem [8, 9, 10, 11, 12, 13].

In this paper, we present our algorithm – preference-list ACO algorithm with mutation (PACOM), which is an algorithm that improves Ant System (AS) in certain ways. It is applied to solve MKP. A new array, called *preference-list* is maintained to guide an ant during the construction of a solution. *Preference-number*, which is the number of items that should be considered by an ant, or also known as the construction steps, is set randomly according to the preference-list. PACOM allows the generation of infeasible solutions to avoid from checking the constraints violation while building the solutions. This could significantly reduce the time complexity of the algorithm. The infeasible solution is then converted to a feasible solution using the repair operator. Finally, a mutation scheme is done on the generated solution to avoid from getting into a local maximum.

This paper is organized as follows. Section II describes the AS for MKP. In section III, we present our proposed algorithm. The computational results on the benchmark problems are given in section IV. Lastly, the conclusion is done in section V.

II. ACO ALGORITHM FOR MKP

The earliest ACO algorithm to solve MKP is given in [8]. In [8], the pheromone trails are associated with each item. The ants start with an empty solution. At each construction step, the ants choose an item to be added to the solution probabilistically according to a *random proportional* rule, which considers the heuristic information and the pheromone trails. Given that $\widetilde{S}_k(t)$ is the partial solution constructed by ant k at iteration t, the selection probability $P_j^k(t)$ of the k-th ant selecting item j as the next item is given by:

978-1-4673-1938-6/12/$31.00 ©2012 IEEE

$$P_j^k(t)=\begin{cases}\dfrac{[\tau_j(t)]^\alpha[\eta_j(\widetilde{S}_k(t))]^\beta}{\sum_{l\in allowed_k(t)}[\tau_l(t)]^\alpha[\eta_l(\widetilde{S}_k(t))]^\beta}, & \text{if } j\in allowed_k(t)\\ 0 & \text{, otherwise.}\end{cases};\quad (2)$$

where $allowed_k(t) \subseteq S-\widetilde{S}_k(t)$ is a set of remaining feasible items, $\tau_j(t)$ is the value of pheromone trail for item j at time t, $\eta_j(\widetilde{S}_k(t))$ represents the heuristic information for item j, α and β are the parameters to control the relative importance of the pheromone trail and heuristic information of item j respectively. According to (2), the higher the value of pheromone trail $\tau_j(t)$, the higher the chance for item j to be included in the partial solution. Same applies to the heuristic information, $\eta_j(\widetilde{S}_k(t))$.

There are two types of heuristics used in MKP, namely, static heuristics and dynamic heuristics [11]. The heuristic information, $\eta_j(\widetilde{S}_k(t))$ in (2) is a dynamic heuristic, in which the value will change at run-time. Its value is depending on the partial solution, $\widetilde{S}_k(t)$. Since $\widetilde{S}_k(t)$ changes after an ant added a new item into the solution, therefore, recalculation of $\eta_j(\widetilde{S}_k(t))$ value is needed at each construction step. Let δ_{ij} (k, t) be the *tightness* of item j on the constraint i according to the $\widetilde{S}_k(t)$ at iteration t for ant k. It can be defined as the ratio of the required amount of i resource by j item, r_{ij} to the remaining capacity of i resource. It is expressed as follows:

$$\delta_{ij}(k,t)=\frac{r_{ij}}{c_i-\sum_{l\in\widetilde{S}_k(t)} r_{il}}. \quad (3)$$

Thus, the *average tightness* on all constraints i if item j is being chosen to be included in $\widetilde{S}_k(t)$ can be obtained by:

$$\bar{\delta}_j(k,t)=\frac{\sum_{i=1}^m \delta_{ij}(k,t)}{m}. \quad (4)$$

We can get the *pseudo-utility* of an item by dividing its profit to the average tightness. The result is the local heuristic information for item j, mathematically written as:

$$\eta_j(\widetilde{S}_k(t))=\frac{p_j}{\bar{\delta}_j(k,t)}. \quad (5)$$

For each cycle, the pheromone trails update is carried out after all ants have built a solution. The pheromone trail associated with each item is updated using the formula:

$$\tau_j(t+N_{\max})=(1-\rho)\tau_j(t)+\Delta\tau_j(t,t+N_{\max}). \quad (6)$$

where N_{max} is the maximum number of construction steps among all ants, i.e., at time $(t + N_{max})$, all ants have completed the construction of a solution. $0 \le \rho \le 1$ represents the pheromone evaporation rate. $\Delta\tau_j(t,t+N_{\max})$ is the amount of pheromone to be deposited by all ants on item j, given by:

$$\Delta\tau_j(t,t+N_{\max})=\sum_{k=1}^{a}\Delta\tau_j^k(t,t+N_{\max}). \quad (7)$$

where a is the number of ants, and $\Delta\tau_j^k(t,t+N_{\max})$ represents the quantity of pheromone deposited by ant k on item j. This quantity can be obtained by the following formula:

$$\Delta\tau_j^k(t,t+N_{\max})=\begin{cases}QL_k, & \text{if ant } k \text{ puts item } j \text{ in solution;}\\ 0, & \text{otherwise.}\end{cases} \quad (8)$$

In (8), Q is the constant parameter, L_k is the value of the objective function acquired by ant k, i.e., $\Sigma_{j=1}^n p_j x_j$. Q is set to $1/\Sigma_{j=1}^n p_j$.

The outline of the algorithm in [8] is given below:

Ant System Algorithm for the MKP:

```
1.  Set parameters, initialize pheromone trails.
2.  for $t = 1$ to number of cycles do
3.    for $k = 1$ to $a$ do
4.      while $allowed_k$ is not empty do   // Construction
                                              steps
5.        Select item $j$ according to $P_j^k(t)$ given by (2).
6.      end while
7.      Calculate the profit of the generated solution.
8.      Save the best solution so far.
9.    end for
10. Update pheromone trail $\tau_j$ on all items using (6).
11. end for
12. Print the best solution found.
```

III. Proposed Algorithm - PACOM

PACOM, the proposed algorithm, follows the ACO metaheuristic framework. Several modifications are done to AS [8] to obtain better results and enhance time performance. In the following, we describe the proposed algorithm in detail.

A. Heuristic information

Heuristic information used in [8] is a dynamic heuristic, which means recalculation of the value is necessary at each construction step. This leads to a longer computational time and slows down the algorithm. To improve this, we use a static heuristic in our proposed algorithm. The pseudo-utility of an item is determined by using the surrogate duality approach introduced by Pirkul [14]. The general idea of this approach is described briefly as follows.

The surrogate relaxation problem of the MKP (denoted as SR-MKP) is defined as:

$$\begin{aligned}&\text{maximize } \sum_{j=1}^{n} p_j x_j\\ &\text{subject to } \sum_{j=1}^{n}\left(\sum_{i=1}^{m}\mu_i r_{ij}\right)x_j \le \sum_{i=1}^{m}\mu_i c_i,\\ &\qquad x_j\in\{0,1\},\quad j\in J=\{1,\ldots,n\}.\end{aligned} \quad (9)$$

where $\mu = \{\mu_1, \ldots, \mu_m\}$ is a set of positive surrogate multipliers (or weights). SR-MKP is often solved to get the upper bound on the original MKP [15]. The best bound can be acquired by finding a set of multipliers that gives the minimum solution for (9). To obtain good values for μ, we could solve the linear programming relaxation (LP relaxation) of (1) and use the dual variables from that solution as surrogate multipliers [14]. In other words, μ_i is set to be the *shadow price* of *i*-th constraint in the LP relaxation of the MKP [16]. We can get the LP relaxation of the MKP by replacing the constraint $x_j \in \{0,1\}$ in (1) to $x_j \in [0,1]$. It means that x_j can take any value between 0 and 1 (inclusive) instead of constraining x_j to take either integer 0 or 1 only.

Since the SR-MKP is essentially a single-dimensional knapsack problem, the pseudo-utility, which is also employed as the heuristic information for each item, is defined as:

$$\eta_j = \frac{p_j}{\sum_{i=1}^{m} \mu_i r_{ij}}. \quad (10)$$

B. Pheromone trails

There are two kinds of pheromone models [12]. The first pheromone model is a model that associates pheromone trail with every pair of the items, which reflects the desirability of both items. This model requires $O(n^2)$ space to store all the pheromone trails and $O(n^2)$ time complexity to update the trails. The other one is a pheromone model that associates a pheromone trail with each item, which gives the desirability of each item. It only requires $O(n)$ space and $O(n)$ time complexity to update the trails. We follow the pheromone model from [8], where the pheromone trail is associated with each item. It is because the other model needs longer computational time to update the pheromone trails and requires much more memory to store all the pairs. Inherits from AS, all ants are allowed to deposit a certain amount of pheromone after the end of each cycle in PACOM. The constant parameter Q in pheromone trails update from (8) is slightly modified. Q is set to $1/(\Sigma_{j=1}^{n} p_j(1-x_j)) \times (L_k / L_{global\_best})$, where L_{global_best} is the profit value of the best solution generated so far. This parameter allows the ant with better solution to deposit more pheromone, in the hope that the optimal solution could be found.

C. Solution construction

To reduce the computational time and facilitate parallel implementation, we abolish the typical method to construct a solution. In the typical AS method, a dynamic heuristic is employed where an ant chooses the next item according to (2) while building a solution. The ants have to re-compute the selection probability of every item according to the partial solution at each construction step, until a complete solution is built. This leads to a higher time complexity. For AS, the time complexity for an ant to generate a solution is $O(n^2m)$. If there are a ants, the time complexity to generate solutions at one cycle is $O(an^2m)$.

To reduce the time complexity, we use a static heuristic. Since the heuristic value for each item calculated from (10) remains the same throughout the entire program, we only have to compute it once in the beginning of the program. The selection probability in (2) is changed to:

$$P_j^k(t) = \frac{[\tau_j(t)]^\alpha [\eta_j]^\beta}{\sum_{l\in J} [\tau_l(t)]^\alpha [\eta_l]^\beta}. \quad (11)$$

The pheromone trails value will only change after all ants have constructed a solution, i.e., after one cycle. Thus, we only need to compute the selection probability of each item once for one cycle using (11) and all the ants use the same value to build a solution. After a cycle, the selection probability of all items is re-computed.

We maintain an array of size *n*, called *preference-list*. Assuming the index of the array starts at 1. The aim of introducing this list is to randomly select a number of items to be considered by an ant, denoted as *preference-number*. In other words, the *preference-number* is the number of construction steps. The higher the value of element index *i* in the array at cycle *t*, denoted as *preference-list*$(t)_i$, the higher the chance that the *preference-number* will be set to *i*. For example, at cycle *t*, if *preference-list*$(t)_{99}$ has the highest value in the array, then the *preference-number* would be most probably set to 99 at that cycle. Only the iteration-best ant (ant with the best solution in the cycle) is allowed to update this list. Initially, the value of all elements in *preference-list* is set 0. Let *y* be the number of items selected by the best ant in the cycle *t*, we update the list using the formula:

$$\mathit{preference\text{-}list}(t)_i = \begin{cases} V + \mathit{preference\text{-}list}(t)_i, & \mathit{if}\ i = y; \\ \mathit{preference\text{-}list}(t)_i, & \mathit{otherwise}. \end{cases} \quad (12)$$

V is the constant value to be added to *preference-list*$(t)_y$, defined as:

$$V = \frac{L_{iteration\_best}}{\sum_{j=1}^{n} p_j - L_{iteration\_best}} \times \frac{L_{iteration\_best}}{L_{global\_best}}. \quad (13)$$

where $L_{iteration\_best}$ is the profit generated by the iteration-best ant. To reduce the computational time, the selection probability of each item would not be calculated again at each construction step. Thus, if a chosen item is already included in the partial solution, we will move on to the next construction step, i.e., x_j is set to 1 regardless $x_j = 0$ or 1. This allows us to select the items randomly, but without repetitive calculation at each construction step. Thus, the number of the selected items would be most likely less than the *preference-number* after the selection is done. We may still have enough resources to include other items into the solution after the selection is ended. This problem will be solved after we executed the repair operation. The unselected "good" items will be selected by the repair operator if it fits the constraints. "Good" items are the items with high profits but consume fewer resources. The repair operator will be discussed next.

The avoidance of checking the violation of the constraints during the solution construction could enhance the performance speed. Therefore, we allow an infeasible solution to be generated to skip the checking. Infeasible solutions are repaired using the repair operator. Without considering the time complexity of the repair operation, *a* ants need *O*(*an*) to generate solutions for one cycle in PACOM.

D. Repair operator

We adopted the repair operator introduced by Chu and Beasley [16], which consists of two phases: DROP and ADD. The items need to be sorted first before running the repair operator. We had tried to sort the items by two different criteria: sort the items according to their pseudo-utility value, which is also the heuristic information value, η_j in this case, and sort the items according to their selection probability value at cycle *t*, denoted as $P_j^k(t)$. We use *U-sort* to represent the sorting according to the pseudo-utility value of the item, and *P-sort* to represent the sorting according to the items' selection probability. From the result of the experiments shown in next section, we choose U-sort to be applied in our algorithm.

The DROP phase removes any item that violates the constraints from the solution by setting $x_j = 0$. The items are considered in the increasing order of η_j. The ADD phase examines each item in the decreasing order of η_j and changes x_j from 0 to 1 as long as the solution is feasible after the item is being selected. The pseudocode of this repair operation is given below, assuming that the items are already sorted according to the decreasing order of their η_j:

```
Repair Operator for the MKP:
1.  Initialize $R_i = \sum_{j=1}^{n} r_{ij}x_j$ .
2.  for $j = n$ to 1 do                        // DROP Phase
3.    if ($x_j = 1$) and ($R_i > c_i$ for any $i \in I$) then
4.      Set $x_j = 0$, $R_i = R_i - r_{ij}$ for all $i \in I$.
5.    end if
6.  for $j = 1$ to $n$ do                      // ADD Phase
7.    if ($x_j = 0$) and ($R_i + r_{ij} \le c_i$ for all $i \in I$) then
8.      Set $x_j = 1$, $R_i = R_i + r_{ij}$ for all $i \in I$.
9.    end if
10. end for
```

The time complexity of the repair operation is *O*(*amn*). Therefore, considering the repair operation, the time complexity for *a* ants to generate *a* solutions in PACOM are *O*(*an*) + *O*(*amn*) = *O*(*amn*), which is lesser than AS.

E. Mutation

In a single-dimensional knapsack problem, Balas and Zemel noticed that the solution acquired by packing the knapsack in the decreasing order of bang-for-buck ratios (p_j / r_j) differed from the optimal solution by only few variables [17]. In MKP, the bang-for-buck ratios can be defined as the pseudo-utility value obtained from (10) [15]. By applying a mutation scheme in the proposed algorithm, we could probably get an optimal solution for some problems. In addition, the mutation scheme can prevent the algorithm from getting stuck into a local maximum.

The mutation scheme would be applied after an ant has generated a solution. We use a simple random 4-flip method presented in [10], which is the core part of their local search. Their number of flips was set by going through a massive number of experiments. We randomly select four variables from the generated solution, and flip their x_j value from 0 to 1, or from 1 to 0, and repair the new solution if necessary. If the new solution is better, the original solution is replaced with the new solution.

F. Outline of the Proposed Algorithm

The pseudocode of the proposed algorithm is as below:

```
Proposed Algorithm for the MKP – PACOM:
1.  Set parameters, initialize pheromone trails.
2.  Calculate $\eta_j$ for each item and sort them according to
    the decreasing order of $\eta_j$.
3.  for $t = 1$ to number of cycles do
4.    Compute the selection probability of each item using (11).
5.    for $k = 1$ to $a$ do
6.      Set preference-number randomly according to the
        preference-list.
7.      while preference-number > 0 do // Construction steps
8.        Select item $j$ according to $P_j^k(t)$ given by (11)
          regardless $x_j = 0$ or $x_j = 1$.
9.        preference-number--;
10.     end while
11.     Repair the infeasible solution using the repair operator.
12.     Calculate the profit of the generated solution.
13.     Run the mutation scheme on the generated solution. Repair the solution if necessary.
14.     Save the best solution so far.
15.   end for
16.   Update pheromone trail $\tau_j$ on all items using (6).
17.   Update preference-list.
18. end for
19. Print the best solution found.
```

IV. Experimental Study

The algorithm proposed in this paper is tested on the benchmarks of MKP selected from OR-library. The instances for the testing are normally denoted as *m.n*. The tested instances are: 100 items with 5 constraints, denoted as 5.100, and 100 items with 10 constraints, denoted as 10.100. The algorithm was coded in C++ and all the tests were conducted on a PC running Microsoft Windows XP with 2.8 GHz Pentium IV CPU.

A. Parameters setting

The parameters were set after testing on a huge number of experiments. The parameters setting are: $\alpha = 1$, $\beta = 2$, $\rho = 0.4$, the number of ants, $a = n$. β and ρ values were chosen from $\beta = \{2, 3, 4, 5\}$ and $\rho = \{0.3, 0.4, 0.5\}$ after comparing the result of the experiments. The initial value of the pheromone trail, τ_j for all $j \in J$, is set to 1. Initial value of the *preference-number* = *n*. The maximum cycles are set to 100. Each instance is run 10 times, and the experimental results are shown below. For all the tables below, the average of the solutions are rounded off, the best average found among different approaches are in boldface, and the best result of the approach for each instance would be in boldface if it is equal to the best known solution.

B. *Comparison of results between different sorts*

The items need to be sorted before executing the repair operator. We compared two different ways to sort the items: P-sort and U-sort. Both sorting are in decreasing order. For P-sort, the item with the highest $P_j^k(t)$ value at cycle t would be first considered by the ADD operator in the repair operation. Similarly, the item with lower $P_j^k(t)$ value would have a higher chance to be removed from the solution in the DROP operation. Similar applies to U-sort. The parameters setting for comparing the two different sorts is the same.

Table I shows the comparison result between the two different sorts for 3 instances from 5.100 dataset. The table reports the best known solutions from the OR-library, the best and the average solutions obtained by P-sort and U-sort.

From Table I, we notice that the U-sort gives better result compared to P-sort, in terms of best solution found and the average of the solutions. It is because $P_j^k(t)$ is dependent to the past choices of the ants. Some "good" items may have lower $P_j^k(t)$ value if they are not chosen to be part of the solution. Similarly, "bad" items may have higher value of $P_j^k(t)$ compared to some more profitable items if some ants select them. Therefore, U-sort gives a better result as it is not affected by the bad choices done by the ants. Other than providing better results, employing the U-sort in our algorithm also save computational time as we just need to sort the items once only, i.e., before any construction of the solution starts. In another case, P-sort needs be performed at every cycle because the value of the selection probability for each item changes after a cycle. Thus, U-sort is definitely a better choice.

TABLE I
COMPARISON OF THE RESULTS OBTAINED USING P-SORT AND U-SORT

Instance	Best known	P-sort		U-sort	
		Best	Average	Best	Average
06	25591	**25591**	25516	**25591**	**25584**
07	23410	**23410**	23385	**23410**	**23392**
08	24216	24204	24173	**24216**	**24205**

C. *The performance of the proposed algorithm*

We compared our proposed algorithm with other ACO approaches, which are the algorithm proposed by Leguizamon and Michalewicz (denoted as Ant System in the table) [8], the approach proposed by Alaya, Solnon and Ghéira (represented as Ant-knapsack in the table) [9], and the algorithm proposed by Fidanova (shown as Fidanova in the table) [13]. The test results of 10 instances for benchmark 5.100 are summarized in Table II. For each instance, the table gives the best known solution from the OR-library, the best solution and the average solution found by Leguizamon and Michalewicz, the best solution and the average solution found by Alaya et al., the best solution found by Fidanova without the average solution as there is no result reported for this column, and finally our proposed algorithm – PACOM's best result and its average solution. It can be seen that PACOM hits all the best known solution for all instances, and outperforms the other algorithms in terms of average solutions.

Table III shows the results of 10 instances for benchmark 10.100. For each instance, the table gives the best known solution from the OR-library, the best solution and the average solution found by Leguizamon and Michalewicz, the best solution and the average solution found by Alaya et al., and the best solution and the average solution of our proposed algorithm, PACOM. It shows that PACOM obtained the best average solutions for all instances among the three algorithms. PACOM outperforms the Ant System for the best result found as PACOM can acquire the best solution better or equal to the best solution of the Ant System for all instances except one.. Comparing with Ant-knapsack, both PACOM and Ant-knapsack can give the best known solution for six different instances. However, it should be mentioned that Ant-knapsack generates 60000 solutions for an instance, and the results are obtained from 50 runs. PACOM generates only 10000 solutions for an instance, and the results are taken only from 10 runs. In addition, PACOM has a lower time complexity to generate a solution because Ant-knapsack uses a dynamic heuristic, while PACOM uses a static heuristic.

V. CONCLUSION

In this paper, we proposed a new ACO algorithm – PACOM, which has a preference-list to set the preference-number and applied it on MKP. We had tested two kinds of sorts, which are P-sort and U-sort for the repair operation. The result shows that U-sort is a better choice for the repair operation.

The experimental results for the benchmark problems show that the proposed algorithm is competitive. In addition, it is designed to have a lower time complexity to fit the practical applications of MKP. Our future work includes further study of the effectiveness of PACOM in theory, and adds additional ideas to improve the algorithm to provide better result.

REFERENCES

[1] M. Dorigo, T. Stüzle, *Ant Colony Optimization.* MIT Press, 2004.

[2] E. Bonabeau, G. Theraulaz and M. Dorigo, *Swarm Intelligence: From Natural to Artificial Systems.* New York: Oxford University Press, 1999.

[3] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 53-66, 1997.

[4] M. Dorigo, G. D. Caro and L. M. Gambardella, "Ant Algorithms for Discrete Optimization," *Artificial Life*, vol. 5, no. 3, pp. 137-172, 1999.

[5] L. M. Gambardella, E. D. Taillard and G. Agazzi, "MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds., London, U.K.: McGraw-Hill, pp. 63-76, 1999.

[6] L. M. Gambardella, E. D. Taillard and M. Dorigo, "Ant colonies for the quadratic assignment problem," *J. Oper. Res. Soc.*, vol. 50, no. 2, pp. 167-176, 1999.

[7] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *IEEE Trans. Data Knowl. Eng.*, vol. 11, pp. 769-778, 1999.

[8] G. Leguizamon and Z. Michalewicz, "A new version of ant system for subset problems," *Proc. 1999 Congr. Evolutionary Computation*, pp. 1459-1464, 1999.

[9] I. Alaya, C. Solnon and K. Ghéira, "Ant algorithm for the multi-dimensional knapsack problem," *Int. Conf. on Bioinpired Optimization Methods and Their Application*, pp. 63-72, 2004.

[10] M. Kong, P. Tian and Y. Kao, "A new ant colony optimization algorithm for the multidimensional knapsack problem," *Comput. and Oper. Research.*, vol. 35, pp. 2672-2683, 2008.

[11] S. Fidanova, "Heuristics for multiple knapsack problem," *IADIS Applied Computing 2005 Conf.*, pp. 255-260, 2005.

[12] S. Fidanova, "Ant colony optimization for multiple knapsack problem and model bias," *Int. Conf. Numerical Analysis and Its Applications*, pp. 280-287, 2005.

[13] S. Fidanova, "Evolutionary algorithm for the multidimensional knapsack problem," *Int. Conf. Parallel Problems Solving from Nature, Real World Optimization Using Evolutionary Computing*, pp. 831-840, 2002.

[14] H. Pirkul, "A heuristic solution procedure for the multiconstraint zero-one knapsack problem," *Naval Research Logistics*, vol. 34, pp. 161-172, 1987.

[15] B. Gavish and H. Pirkul, "Efficient algorithms for solving multiconstraint zero-one knapsack problems to optimality," *Mathematical Programming*, vol. 31, pp. 78-105, 1985.

[16] P. Chu and J. Beasley, "A genetic algorithm for the multidimensional knapsack problems," *J. Heuristics*, vol. 4, no. 1, pp. 63-86, 1998.

[17] E. Balas and E. Zemel, "An algorithm for large zero-one knapsack problems," *Oper. Research,* vol. 28, no. 5, pp. 1130-1154, 1980.

TABLE II
COMPARISON OF THE RESULTS FOR DIFFERENT ALGORITHMS ON 5.100 INSTANCES

Instance	Best known	Ant System		Ant-knapsack		Fidanova	PACOM	
		Best	Average	Best	Average	Best	Best	Average
5.100-00	24381	**24381**	24331	**24381**	24342	23984	**24381**	**24372**
5.100-01	24274	**24274**	24246	**24274**	24247	24145	**24274**	**24274**
5.100-02	23551	**23551**	23528	**23551**	23529	23523	**23551**	**23533**
5.100-03	23534	23527	23463	**23534**	23462	22874	**23534**	**23485**
5.100-04	23991	**23991**	23950	**23991**	23946	23751	**23991**	**23955**
5.100-05	24613	**24613**	24563	**24613**	24587	24601	**24613**	**24602**
5.100-06	25591	**25591**	25505	**25591**	25512	25293	**25591**	**25584**
5.100-07	23410	**23410**	23362	**23410**	23371	23204	**23410**	**23392**
5.100-08	24216	24204	24173	**24216**	24172	23762	**24216**	**24205**
5.100-09	24411	**24411**	24326	**24411**	24356	24255	**24411**	**24376**

TABLE III
COMPARISON OF THE RESULTS FOR DIFFERENT ALGORITHMS ON 10.100 INSTANCES

Instance	Best known	Ant System		Ant-knapsack		PACOM	
		Best	Average	Best	Average	Best	Average
10.100-00	23064	23057	22996	**23064**	23016	23057	**23025**
10.100-01	22801	**22801**	22672	**22801**	22714	**22801**	**22743**
10.100-02	22131	**22131**	21980	**22131**	22034	**22131**	**22089**
10.100-03	22772	**22772**	22631	22717	22634	**22772**	**22703**
10.100-04	22751	22654	22578	22654	22547	22603	**22582**
10.100-05	22777	22652	22565	22716	22602	22675	**22646**
10.100-06	21875	**21875**	21758	**21875**	21777	**21875**	**21821**
10.100-07	22635	22551	22519	22551	22453	**22635**	**22552**
10.100-08	22511	22418	22292	**22511**	22351	22423	**22403**
10.100-09	22702	**22702**	22588	**22702**	22591	**22702**	**22702**