Proceedings of 13th International Conference on Computer and Information Technology (ICCIT 2010)
23-25 December, 2010, Dhaka, Bangladesh

A novel ACO technique for Fast and Near Optimal Solutions for the Multi-dimensional Multi-choice Knapsack Problem

Shahrear Iqbal, Md. Faizul Bari, Dr. M. Sohel Rahman
AℓEDA Group
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
Dhaka-1000, Bangladesh.
{shahreariqbal,faizulbari,msrahman}@cse.buet.ac.bd

Abstract

In this paper, we have proposed a novel algorithm based on Ant Colony Optimization (ACO) for finding near-optimal solutions for the Multi-dimensional Multi-choice Knapsack Problem (MMKP). MMKP is a discrete optimization problem, which is a variant of the classical 0-1 Knapsack Problem and is also an NP-hard problem. Due to its high computational complexity, exact solutions of MMKP are not suitable for most real-time decision-making applications e.g. QoS and Admission Control for Adaptive Multimedia Systems, Service Level Agreement (SLA) etc. Although ACO algorithms are known to have scalability and slow convergence issues, here we have augmented the traditional ACO algorithm with a unique random local search, which not only produces near-optimal solutions but also greatly enhances convergence speed. A comparative analysis with other state-of-the-art heuristic algorithms based on public MMKP dataset shows that, in all cases our approaches outperform others. We have also shown that our algorithms find near optimal (within 3% of the optimal value) solutions within milliseconds, which makes our approach very attractive for large scale real time systems.

I. Introduction

The classical 0–1 Knapsack Problem (KP) is to pick up items for a knapsack to maximize the total profit, satisfying the constraint that, the total resource required does not exceed the resource constraint R of the knapsack. This problem and its variants are used in many resource management applications such as cargo loading, industrial production, menu planning, and resource allocation in multimedia servers [1]. The Multidimensional Multiple-choice Knapsack Problem (MMKP) is a variant of the classical 0–1 KP. Here we have n groups of items. Group i has ℓ_i items. Each item of the group has a particular value and it requires m resources. The objective of the MMKP is to pick exactly one item from each group for maximum total value of the collected items, subject to m resource constraints of the knapsack. In mathematical notation, let v_{ij} and $\overrightarrow{r_{ij}} = (r_{ij1}, r_{ij2}, \ldots, r_{ijm})$ be the value (profit) and required resource vector of the object o_{ij}, i.e., j-th item of the i-th group. Also assume that

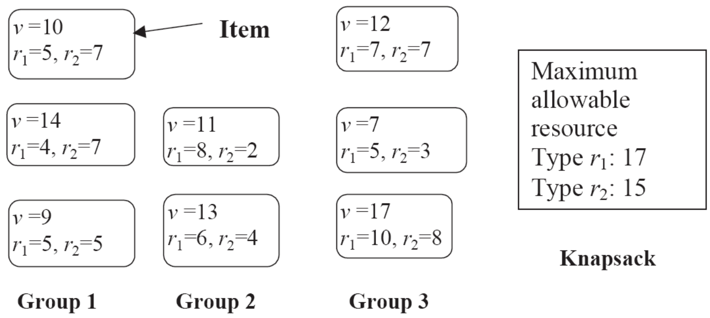

Fig. 1. Multidimensional Multiple-choice Knapsack Problem (figure borrowed from [2]).

$\overrightarrow{R} = (R_1, R_2, \ldots, R_m)$ be the resource bound of the knapsack. Now, the problem is to

$$maximize \sum_{i=1}^{n} \sum_{j=1}^{\ell_i} x_{ij} v_{ij} \ (objective\ function),$$

$$subject\ to \sum_{i=1}^{n} \sum_{j=1}^{\ell_i} x_{ij} r_{ijk} \leq R_k \ (resource\ constraints)$$

where $k = 1, 2, \ldots, m$, $x_{ij} \in \{0, 1\}$ are the picking variables, and for all $i \in 1\ to\ n$, $\sum_{j=1}^{li} x_{ij} = 1$.

Fig 1 illustrates an MMKP. We have to pick exactly one item from each group. Each item has two resources, r_1 and r_2. Clearly we must satisfy $\sum(r_1$ of picked items) ≤ 17 and $\sum(r_2$ of picked items) ≤ 15 and maximize the total value of the picked items. Notably, it may happen that no set of items satisfying the resource constraints exists implying that no solution will be found.

In this paper, we have described a new algorithm for solving MMKPs. The algorithm is based on Ant Colony Optimization (ACO), which is a recently developed, population-based stochastic meta-heuristic [3], [4]. ACO has been successfully applied to solve several NP-hard combinatorial optimization problems [5], [6], such as traveling salesman problem [7], [4], vehicle routing problem [8], and quadratic assignment problem [9], [10].

This meta-heuristic belongs to the class of problem-solving strategies derived from nature. The ACO algorithm is basically a multi-agent system where low level interactions among the agents (i.e., artificial ants) result in a complex behavior of the whole ant colony. The basic idea of ACO is to model the problem under consideration as a searching problem, where a minimum cost path in a graph is searched; the artificial ants are employed to search for good paths. The pheromone trails are a kind of distributed information which is modified by the ants to reflect their experience accumulated during the problem solving. This substance influences the choices they make: the larger the amount of pheromone is on a particular path, the larger is the probability that an ant would select the path. Additionally these pheromone trails progressively decrease by evaporation. Intuitively, this indirect stigmergetic communication mean aims at giving information about the quality of path components in order to attract ants, in the following iterations, towards the corresponding areas of the search space.

MMKP has received significant amount of attention in the literature mostly motivated by capital budgeting, multimedia applications etc. There exist a number of heuristics in the literature for solving MMKP. Khan [1] proposed an algorithm named HEU, using the idea of aggregate resource consumption. In [11], a modified version of HEU named M-HEU was presented, which provides solutions with total value on average equal to 96% of the optimum. In [2] the authors presented a convex hull based heuristic called C-HEU, which is very fast and achieves optimality between 88% and 98%. Hifi et al. [12] proposed a guided local search-based heuristic and later improved upon it to achieve a "reactive" local search-based (RLS) algorithm [13]. Hernndez and Dimopoulos [14] also proposed a new heuristic for MMKP.

For solving MMKP with ACO, the most important design choice lies in deciding which component of the problem should be regarded as the pheromone depositing component. Here we have laid pheromone trails on each object selected in a solution. Essentially, the idea is to increase the desirability of each object selected in a feasible solution: during the constructing of a new solution, these objects will be more likely to be selected. The contributions of this paper are as follows.

We present a novel ACO based algorithm for solving MMKP. The algorithm produce comparable results with the current state-of-the-art heuristic algorithms. To the best of our knowledge, this work is the first attempt to solve MMKP using ACO. An interesting aspect of our algorithm is the introduction of a novel and unique random local search algorithm for improving the solutions generated by the ant colony. This process, coupled with the natural behavior of the artificial ants produces near-optimal solutions and greatly enhances convergence speed of the ant colony.

The rest of the paper is organized as follows. Section II gives a brief description of the ACO algorithm for solving the multi-dimensional knapsack problem (MKP), a related variant of KP. We present our main contribution in Section III, where we describe our new algorithm for solving MMKPs. Section IV presents the experimental results along with an insightful discussion on the experimental results. Finally we briefly conclude in Section V.

II. ACO AND MULTI-DIMENSIONAL KP

As has already been mentioned we did not find any ACO based algorithm to solve MMKP in the literature. However there exist a number of ACO based solution for a more restricted variant of KP, namely MKP [15], [16], [17]. In MKP resources have multiple dimensions as in MMKP; however there is no concept of group in MKP. As a result, MKP can be thought of as a restricted version of MMKP, which has all objects in a single group. The algorithms of [15], [16], [17] differ in deciding which component of the problem should be regarded as the pheromone depositing component and in the mechanisms of pheromone updating:

1. **Pheromone Trails on Each Object:** The first way is to lay pheromone trails on each object belonging to the current solution set [15]: the amount of pheromone represents the preference of the object.
2. **Pheromone Trails on Each Pair:** In this case, pheromone trails are laid on each pair (o_i, o_j) of successively selected objects of the solution set [16]: the idea is to increase the desirability of choosing object o_j when the last selected object is o_i.
3. **Pheromone Trails on All Pair:** The third one is to lay pheromone trails on all pairs of different objects of the solution set [17]. Here, the idea is to increase the desirability of choosing simultaneously two objects of S.
4. **Pheromone Diffusion Model:** The forth approach follows the same principle as the first one. Additionally it uses a pheromone diffusion scheme where pheromone trails are laid on objects that tend to occur together in previous solutions [18].

These approaches also differ in the way local heuristic information is defined. We are particularly interested in the dynamic local heuristic information used by [17], [15], [18] as defined below. Let S_k be the set of the selected objects at the k-th Iteration. For each candidate object j, the heuristic information $\eta_{S_k}(j)$ is given as follows:

$$_{S_k}(j) = \frac{v_j}{\sum_{i=1}^{m} r_{ij}/d_{S_k}(i)} \quad (1)$$

where,

$$d_{S_k}(i) = R_i - \sum_{t \in S_k} r_{it} \quad (2)$$

Since S_k will be changed from step to step, the heuristic information is dynamic. we will be using a variation of above heuristic.

III. Description of the proposed algorithm

We have proposed a variation of the ACO algorithm for solving MMKPs namely ***AntMMKP-GroupOrdering***. The algorithm select groups in the same way as it selects the objects. We need to maintain a separate pheromone trail for groups. Also it maintains a list of top k best solutions in order to direct the ants to a better area of the search space. They particularly follow the *MIN-MAX Ant System* [19], where explicit lower and upper bounds on pheromone values are imposed i.e. $\tau_{min} < \tau < \tau_{max}$, and all pheromone trails are initialized to τ_{max}. Below we describe the algorithm in greater details.

A. AntMMKP-GroupOrdering

This algorithm is described in Algorithm 1. At each cycle of this algorithm, k ants are used to build individual solutions. Each ant constructs a solution in a step by step manner. At first a group from the set of candidate groups is selected. All objects that violate resource constraints, are removed from this group. Then, the object with the highest probability (according to equation 5 below) is added to the solution. The probability of an object being selected depends on the amount of pheromone deposited on the object so far and its local heuristic value. The *candidategroups* data structure maintains a list of feasible candidate groups which can be considered next. After each ant has constructed a solution, the best solution of that iteration is identified and a random local search procedure and a random item swap procedure is applied to improve it. Then pheromone trail is updated according to the best solution. Also it maintains a database of top k solutions. After each iteration a small amount of pheromone is deposited in the pheromone trails of the objects belonging to the top k solutions. The motivation behind this strategy is to ensure quick convergence on good solutions and to explore better areas more thoroughly. The algorithm stops either when an ant has found an optimal solution (when the optimal bound is known), or when a maximum number of cycles has been performed.

A.1 Selection of next group: In our proposed algorithm we have unique idea of a separate pheromone trail for groups to save the ordering of groups that lead to a good solution. The group pheromone trail also follow a min max ant system approach and initialized to the max pheromone value. The algorithm chooses first group randomly but after that it chooses group which has the highest pheromone factor which can be defined as

$$\tau_{S_k}(g_i) = \sum_{g_j \in S_k} \tau(o_i, o_j)$$

Note that this pheromone factor can be computed in an incremental way: once the first group g_i has been randomly chosen, for each candidate group o_j , the pheromone factor $\tau_{S_k}(g_j)$ is initialized to $\tau(o_i, o_j)$; then, each time a new group g_l is added to the solution S_k, for each candidate group g_j , the pheromone factor $\tau_{S_k}(g_j)$ is incremented by $\tau(o_l, o_j)$.

Algorithm 1 Algorithm AntMMKP-GrpOrdering

```
Initialize pheromone trails to τ_max for both item and group
topkdb ⇐ ∅ {data structure that holds topmost k solutions}
repeat
  Solution S_globalbest ⇐ ∅
  for each ant k in 1 ... nants do
    Solution S_iterbest ⇐ ∅
    candidategroups ⇐ all the groups
    while candidategroups ≠ ∅ do
      C_g ⇐ Select a group from candidategroups according to group pheromone trail
      Candidates ⇐ {o_i ∈ objects in C_g that do not violate resource constraints}
      update local heuristic values
      Choose an object o_i ∈ Candidates with probability P_{S_k}(o_i)
      S_k ⇐ {S_k ∪ o_i}
      remove C_g from candidategroups
    end while
    if profit(S_k) > profit(S_iterbest) then
      S_iterbest ⇐ S_k
    end if
  end for
  S_iterbest ⇐ RandomLocalSearch(S_k)
  S_iterbest ⇐ RandomItemSwap(S_k)
  if profit(S_globalbest) < profit(S_iterbest) then
    S_globalbest ⇐ S_iterbest
  end if
  update top database
  Update pheromone trails w.r.t S_iterbest
  Update pheromone trails w.r.t topdatabase
  Update group pheromone trails w.r.t S_iterbest
  if pheromone value is lower than τ_min then
    set pheromone ⇐ τ_min
  end if
  if pheromone value is greater than τ_max then
    set pheromone ⇐ τ_max
  end if
until maximum number of cycles reached or optimal solution found
```

Before we run our actual algorithm a dummy run of the whole algorithm is performed a number of time to initialize the group pheromone trail hoping that when our algorithm starts, ants can perform search in a order of group that impose the lowest restriction on the next choices.

A.2 Pheromone trails: To solve MMKPs with ACO, the key point is to decide which components of the constructed solutions should be rewarded, and how to exploit these rewards when constructing new solutions.

A solution of a MMKP is a set of selected objects $S = \{o_{ij} | x_{o_{ij}} = 1\}$ (i.e., an object o_{ij} is selected if the corresponding decision variable x_{ij} has been set to 1). Given a constructed solution $S = \{o_{i_1 j_1}, \ldots, o_{i_n j_n}\}$, pheromone trails are laid on each objects selected in S. So pheromone trail τ_{ij} will be associated with object o_{ij}.

A.3 Pheromone updating: Once each ant has constructed a solution, pheromone trails laying on the solution objects are updated according to the ACO meta-heuristic. First, all amounts are decreased in order to simulate evaporation. This is done by multiplying the quantity of pheromone laying on each object by a pheromone persistence rate $(1-\rho)$ such that $0 \leq \rho \leq 1$.

Then, pheromone is increased for all the objects in the best solution of the iteration. More precisely, let $S_{iterbest}$ be the best solution constructed during the current cycle. Then the quantity of pheromone increased for each object is determined by the function $G(S_{iterbest}) = Q.profit(S_{iterbest})$, where $Q = \frac{1}{\sum_{j=1}^{n} P_j}$ and $profit(S_{iterbest}) = \sum_{o_{ij} \in S_{iterbest}} v_{ij}$.

At the end of each iteration group pheromone values decay like the original pheromone trail for items. then the best solution updates the group pheromone trail. All the adjacent groups get the highest amount of pheromone value that gradually diminishes as the distance between groups increases.

Algorithm 2 Algorithm for Random Local Search

procedure RANDOMLOCALSEARCH(S)
Input: a solution S_k
Output: an improved solution S_k or input if no improvemnt found
  for a prespecified number of times **do**
    $C_g \Leftarrow$ Randomly select a group
    for each object $o_i \in C_g$ other than the one in S_k **do**
      $S_{tmp} \Leftarrow$ include o_i removing the object selected in C_g
      if S_{tmp} not violates any resource constraints **then**
        if $profit(S_k) < profit(S_{tmp})$ **then**
          $S_k \Leftarrow S_{tmp}$
        end if
      end if
    end for
  end for
return S_k

A.4 Heuristic information: The heuristic factor $_{S_k}(O_{ij})$ also depends on the whole set S_k of selected objects. Let $c_{S_k}(l) = \sum_{O_{ij} \in S_k} r_{ijl}$ be the consumed quantity of the resource l when the ant k has selected the set of objects S_k. And let $d_{S_k}(l) = R_l - c_{S_k}(l)$ be the remaining capacity of the resource l. We define the following ratio:

$$h_{S_k}(O_{ij}) = \sum_{l\,=\,1}^{m} r_{ijl}/d_{S_k}(l) \tag{3}$$

which represents the tightness of the object O_{ij} on the constraints l relatively to the constructed solution S_k. Thus, the lower this ratio is, the more the object is profitable. We integrate the profit of the object in this ratio to obtain a pseudo-utility factor. We can now define the heuristic factor formula as follows:

$$_{S_k}(O_{ij}) \;=\; \frac{v_{ij}}{h_{S_k}(O_{ij})} \tag{4}$$

A.5 Constructing a solution: When constructing a solution, an ant starts with an empty knapsack. At the first construction step an ant selects a group randomly and at all the latter steps, groups are selected according to their associated pheromone value. After selecting a group, the algorithm removes all the bad *Candidates* that violates resource constraints. It then updates the local heuristic information of the remaining candidate objects of the group and selects an object according to the following probability equation:

$$\rho_{S_k(O_{ij})} = \frac{[\tau_{S_k}(O_{ij})]^{\alpha}.[\eta_{S_k}(O_{ij})]^{\beta}}{\sum_{O_{ij} \in \mathit{Candidates}} [\tau_{S_k}(O_{ij})]^{\alpha}.[\eta_{S_k}(Oij)]^{\beta}} \tag{5}$$

Here *Candidates* are all items from the currently selected group which do not violate any resource constraints. The parameters α and β control the relative importance of pheromone trail versus local heuristic value.

The construction process stops when exactly one item is chosen from each group.

Algorithm 3 Algorithm for Random Item Swap

procedure RANDOMITEMSWAP(S)
Input: a solution S_k
Output: an improved solution S_k or input if no improvemnt found
  for a prespecified number of times **do**
    for j = 1 to NUMBER-OF-ITEM-TO-FLIP **do**
      $C_g \Leftarrow$ Randomly select a group
      $O_i \Leftarrow$ Randomely select an item from C_g
      $S_{tmp} \Leftarrow$ include o_i removing the object selected in C_g
    end for
    if S_{tmp} not violates any resource constraints **then**
      if $profit(S_k) < profit(S_{tmp})$ **then**
        $S_k \Leftarrow S_{tmp}$
      end if
    end if
  end for
return S_k

A.6 Random local search: Random Local search described in Algorithm 2 is an exhaustive search within a group to improve the solution. It replaces current selected object of a group with every other object that do not violate resource constraints and checks if it is a better solution. The total procedure is repeated a number of times, each time for a random group.

A.7 Random Item Swap: Random Item Swap described in Algorithm 3 is an extended version of the random local search. In this case at a time, a specified number (> 1) of objects are swapped with other random objects from the same group without checking the resource constraints, then it checks if it is a valid solution and if it improves the solution.

IV. Experimental Results

In this section, we assess the performance of the our algorithm, and compare it to other heuristic algorithms available in the literature. The datasets we use are the benchmark data of MMKPs from OR-library [20]. The algorithms were coded in java and run on a PC with intel core 2 duo 2.8 Ghz CPU, 2GB memory running Windows XP. The parameters are set as follows:

Problem File	Exact	MOSER	HEU	CPCCP	RLS	FLTS	FanTabu	CCFT	Ant-G
I01	173	-	154	159	161	158	169	173	**173**
I02	364	294	354	312	354	351	354	352	**364**
I03	1602	1127	1518	1407	1496	1445	1557	1518	**1600**
I04	3597	2906	3297	3322	3435	3350	3473	3419	**3525**
I05	3905.7	1068.3	3894.5	3889.9	3847.3	3905.7	3905.7	3905.7	**3905.7**
I06	4799.3	1999.5	4788.2	4723.1	4680.6	4793.2	4799.3	4799.3	**4799.3**
I07	24587	20833	-	23237	23828	23547	23691	23739	**24115**
I08	36877	31643	34338	35403	35685	35487	35684	35698	**36085**
I09	49167	-	-	47154	47574	47107	47202	47491	**48306**
I10	61437	-	-	58990	59361	59108	58964	59549	**60191**
I11	73773	-	-	70685	71565	70549	70555	71651	**72322**
I12	86071	-	-	82754	83314	82114	81833	83358	**84118**
I13	98429	-	-	94465	95076	91551	94168	94874	**96126**

TABLE I
SOLUTION QUALITY COMPARISON

$nants =$ 50 (i.e., the number of ants is set to 50), $= 1$, $\beta = 5$, $\rho = 0.01$, $k = 10$ (for AntMMKP-TopDb), $\tau_{min} = 0.01$ and $\tau_{max} = 6$ times the amount each ant deposits if it selects an item. For Group Pheromone Trail, $\tau_{min} = 0.01$ and $\tau_{max} = 8$, amount of pheromone deposited = 0.1, diffusion rate = 0.5. For random item swap we used four flip and run 1000 times, also in random local search the loop runs $n * 5$ times, where n is the number of groups.

Table I gives the comparison results of the performance of different algorithms including our algorithm, namely, AntMMKP-GrpOrdering (Ant-G). For each instance, Table I reports the best solution found by MOSER [21], HEU [1], CPCCP [12], RLS [13], FLTS [22], FanTabu [22], CCFT [22] along with the exact solution reported in the data files and the best solutions of Ant-G found in 1000 runs. The results of the other algorithms were borrowed from [22]. Our algorithms clearly outperform all others on each file. Notably, for datasets I01, I02, I05 and I06 they found the exact solution.

Figure 2 reports the average time (milliseconds, over 20 runs) taken by our algorithm to reach within 3% of the known optimal solution for each of the instance file. Considering the solution quality, each of the algorithms run quite fast. The algorithm gives result before 1.5 seconds to reach within 3% of the optimal solution for data file I13 which is quite a large instance of MMKP consisting of 400 groups each having 10 objects and with number of resource dimension being 10. So our algorithms are attractive for large scale real time problems.

The random local search procedure presented in this paper improves the solution quality greatly in each iteration. In Figure 3 we have run three variation of Ant-G on instance file I07 with 100 groups, 10 items per group having resource dimension 10. At first we run the algorithm without the random local search. Then, we use a local search that we have developed earlier which tries to find a better object replacing the current selected object from all the groups in a order (not random). Finally the algorithm was executed with our random

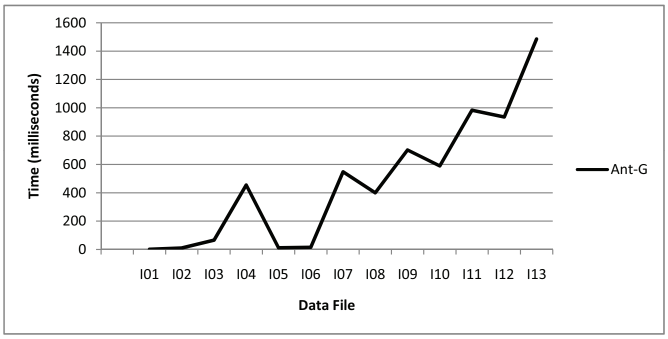

Fig. 2. Time taken by Ant-G to reach within 3% of the optimal solution.

local search. Figure 3 clearly shows that both versions of the local search strategy are quite good for improving the solution, random local search being the better. From this comparison we can understand that the order of the selection of group while generating partial solution is very important to find good solutions for MMKP.

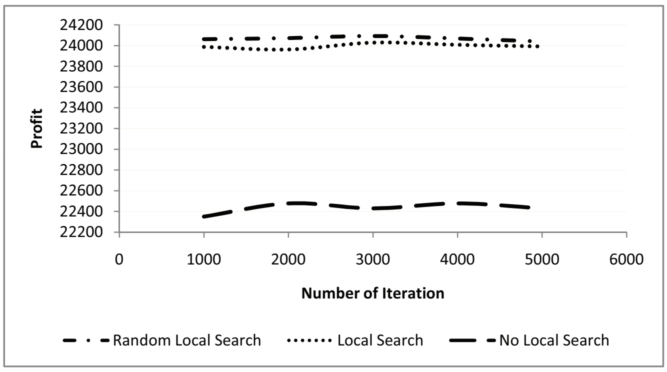

Fig. 3. Performance enhancement with our random local search.

V. CONCLUSIONS

This paper is a first attempt to solve MMKPs using ant colony optimization. Here, we have proposed a novel ACO algorithm for solving MMKPs along with a effective random local search strategy for performance improvement. We have presented simulation results, evaluating both runtime and solution quality of the proposed algorithm, and compared the solution quality of our algorithm with other existing state-of-the-art algorithms. From these simulation results it is clear that, our algorithm is better in terms of solution quality and can also provide very fast near optimal solutions. The random local search seems to have provided the boost needed for providing such good quality solutions.

REFERENCES

[1] Khan, S.: Quality Adaptation in a Multisession Multimedia System: Model, Algorithms and Architecture. PhD thesis, Department of Electrical and Computer Engineering, University of Victoria (1998) PhD dissertation.

[2] Akbar, M.M., Rahman, M.S., Kaykobad, M., Manning, E.G., Shoja, G.C.: Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. Comput. Oper. Res. **33**(5) (2006) 1259–1273

[3] Dorigo, M., Di Caro, G.: The ant colony optimization meta-heuristic: New ideas in optimization. McGraw-Hill Ltd., UK, Maidenhead, UK, England (1999)

[4] Dorigo, M., Di Caro, G., Gambardella, L.M.: Ant algorithms for discrete optimization. Artificial Life **5**(2) (1999) 137–172

[5] Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence: From natural to artificial systems. J. Artificial Societies and Social Simulation **4**(1) (2001)

[6] Parsons, S.: Ant colony optimization by marco dorigo and thomas stützle, mit press, 305 pp, isbn 0-262-04219-3. Knowledge Eng. Review **20**(1) (2005) 92–93

[7] Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans. Evolutionary Computation **1**(1) (1997) 53–66

[8] Gambardella, L.M., ric Taillard, Agazzi, G.: Macs-vrptw: A multiple colony system for vehicle routing problems with time windows. In: New Ideas in Optimization, McGraw-Hill (1999) 63–76

[9] Gambardella, L.M., ric Taillard, Dorigo, M.: Ant colonies for the quadratic assignment problem. Journal of the Operational Research Society **50** (1 February 1999) 167–176(10)

[10] Maniezzo, V., Colorni, A.: The ant system applied to the quadratic assignment problem. IEEE Trans. on Knowl. and Data Eng. **11**(5) (1999) 769–778

[11] Khan, S., Li, K.F., Manning, E.G., Akbar, M.M.: Solving the knapsack problem for adaptive multimedia systems. Studia Informatica Universalis **2** (2003) 157–178

[12] Hifi, M., Michrafy, M., Sbihi, A.: Heuristic algorithms for the multiple-choice multidimensional knapsack problem. Journal of the Operational Research Society **55** (December 2004) 1323–1332(10)

[13] Hifi, M., Michrafy, M., Sbihi, A.: A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem. Comput. Optim. Appl. **33**(2-3) (2006) 271–285

[14] Parra-Hernandez, R., Dimopoulos, N.J.: A new heuristic for solving the multichoice multidimensional knapsack problem. IEEE Transactions on Systems, Man, and Cybernetics, Part A **35**(5) (2005) 708–717

[15] Leguizamon, G., Michalewicz, Z.: A new version of ant system for subset problems. In: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. Volume 2. (1999)

[16] Fidanova, S.: Aco algorithm for mkp using different heuristic information. Lecture Notes in Computer Science **2542** (2003) 438–444

[17] Alaya, I., Solnon, C., Ghèdira, K.: Ant algorithm for the multi-dimensional knapsack problem. In: International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004. (2004) 63–72

[18] Ji, J., Huang, Z., Liu, C., Liu, X., Zhong, N.: An Ant Colony Optimization Algorithm for Solving the Multidimensional Knapsack Problems. In: Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IEEE Computer Society (2007) 10–16

[19] Stützle, T., Hoos, H.H.: Max–min ant system. Future Generation Computer Systems **16** (2000) 889–914

[20] Beasley, J.: OR-Library: Distributing test problems by electronic mail. The Journal of the Operational Research Society **41**(11) (1990) 1069–1072

[21] Moser, M., Jokanovic, D., Shiratori, N.: An algorithm for the multidimensional multiple-choice knapsack problem. IEICE transactions on fundamentals of electronics, communications and computer sciences **80**(3) (1997) 582–589

[22] Hiremath, C.: New heuristic and metaheuristic approaches applied to the multiple-choice multidimensional knapsack problem. PhD thesis, Wright State University (2008)