Ant algorithm for the multidimensional knapsack problem

Ines Alaya, Christine Solnon, Khaled Ghedira

ANT ALGORITHM FOR THE MULTIDIMENSIONAL KNAPSACK PROBLEM

Inès Alaya
SOIE, Institut Supérieur de Gestion de Tunis
41 Rue de la Liberté- Cité Bouchoucha-2000 Le Bardo-Tunis
ines.alaya@isg.rnu.tn

Christine Solnon
LIRIS, CNRS FRE 2672, University Lyon 1
43 bd du 11 novembre, 69622 Villeurbanne cedex
christine.solnon@liris.cnrs.fr

Khaled Ghédira
SOIE, Institut Supérieur de Gestion de Tunis
41 Rue de la Liberté- Cité Bouchoucha-2000 Le Bardo-Tunis
khaled.ghedira@isg.rnu.tn

Abstract We propose a new algorithm based on the Ant Colony Optimization (ACO) meta-heuristic for the Multidimensional Knapsack Problem, the goal of which is to find a subset of objects that maximizes a given objective function while satisfying some resource constraints. We show that our new algorithm obtains better results than two other ACO algorithms on most instances.

Keywords: Ant Colony Optimization, Multidimensional Knapsack Problem

1. Introduction

The Multidimensional Knapsack Problem (MKP) is a NP-hard problem which has many practical applications, such as processor allocation in distributed systems, cargo loading, or capital budgeting. The goal of the MKP is to find a subset of objects that maximizes the total profit while satisfying some resource constraints. More formally, a MKP is

stated as follows:

$$\begin{array}{rl} \text{maximize} & \sum_{j=1}^{n} p_j.x_j \\ \text{subject to} & \sum_{j=1}^{n} r_{ij}.x_j \leq b_i, \forall i \in 1..m \\ & x_j \in \{0,1\}, \forall j \in 1..n \end{array}$$

where r_{ij} is the consumption of resource i for object j, b_i is the available quantity of resource i, p_j is the profit associated with object j, and x_j is the decision variable associated with object j and is set to 1 (resp. 0) if j is selected (resp. not selected).

In this paper, we describe a new algorithm for solving MKPs. This algorithm is based on Ant Colony Optimization (ACO) [4], a stochastic metaheuristic that has been applied to solve many combinatorial optimization problems such as traveling salesman problems [3], quadratic assignment problems [6], or vehicule routing problems [1]. The basic idea of ACO is to model the problem to solve as the search for a minimum cost path in a graph, and to use artificial ants to search for good paths. The behavior of artificial ants is inspired from real ants: they lay pheromone trails on components of the graph and they choose their path with respect to probabilities that depend on pheromone trails that have been previously laid; 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.

To solve MKPs 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 MKP is a set of selected objects $S = \{o_1, \ldots, o_k\}$ (we shall say that an object o_i is selected if the corresponding decision variable x_{o_i} has been set to 1). Given such a solution $S = \{o_1, \ldots, o_k\}$, one can consider three different ways of laying pheromone trails:

- A first possibility is to lay pheromone trails on each object selected in S. In this case, the idea is to increase the desirability of each object of S so that, when constructing a new solution, these objects will be more likely to be selected;
- A second possibility is to lay pheromone trails on each couple (o_i, o_{i+1}) of successively selected objects of S. In this case, the idea is to increase the desirability of choosing object o_{i+1} when the last selected object is o_i.
- A third possibility is to lay pheromone on all pairs (o_i, o_j) of different objects of S. In this case, the idea is to increase the desirability

Algorithm Ant-knapsack:

Initialize pheromone trails to τ_{max}
repeat the following cycle:
  for each ant k **in** 1..$nbAnts$, construct a solution $\mathcal{S}_k$ as follows:
    Randomly choose a first object $o_1 \in 1..n$
    $\mathcal{S}_k \leftarrow \{o_1\}$
    Candidates $\leftarrow \{o_i \in 1..n / o_i$ can be selected without violating resource constraints$\}$
    while *Candidates* $\neq \emptyset$ **do**
      Choose an object $o_i \in$ *Candidates* with probability $p_{S_k}(o_i)$
      $\mathcal{S}_k \leftarrow \mathcal{S}_k \cup \{o_i\}$
      remove from *Candidates* every object that violates some resource constraints
    end while
  end for
  Update pheromone trails w.r.t. $\{\mathcal{S}_1, \ldots, \mathcal{S}_{nbAnts}\}$
  if a pheromone trail is lower than τ_{min} **then** set it to τ_{min}
  if a pheromone trail is greater than τ_{max} **then** set it to τ_{max}
until maximum number of cycles reached **or** optimal solution found

Figure 1. ACO algorithm for solving MKPs

of choosing together two objects of S so that, when constructing a new solution S', the objects of S will be more likely to be selected if S' already contains some objects of S. More precisely, the more S' will contain objects of S, the more the other objects of S will be attractive.

To solve MKP with ACO, Leguizamon and Michalewizc [7] have proposed an algorithm based on the first possibility, whereas Fidanova [5] has proposed another algorithm based on the second possibility. In this paper, we propose a new ACO algorithm for solving MKPs that is based on the third possibility. Our intuition is that this strategy should attract ants in a more precise way as the desirability of an object depends on the objects that already belong to the partial solution under construction.

2. Ant-knapsack description

We define the construction graph, on which ants lay pheromone trails, as a complete graph that associates a node to each object of the MKP. The quantity of pheromone laying on an edge (o_i, o_j) is denoted by $\tau(o_i, o_j)$. Intuitively, this quantity represents the learnt desirability of selecting together objects o_i and o_j.

The proposed ACO algorithm for solving MKPs is described in figure 1 and more particularly follows the $\mathcal{MAX} - \mathcal{MIN}$ Ant System [8]: we explicitly impose lower and upper bounds τ_{min} and τ_{max} on pheromone trails (with $0 < \tau_{min} < \tau_{max}$), and pheromone trails are set to τ_{max} at the beginning of the search.

At each cycle of this algorithm, every ant constructs a solution. It first randomly chooses an initial object, and then iteratively adds objects that are chosen within a set *Candidates* that contains all the objects that can be selected without violating resource constraints. Once each ant has constructed a solution, pheromone trails are updated. 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.

2.1 Definition of transition probabilities

At each step of the construction of a solution, an ant k randomly selects the next object o_i within the set *Candidates* with respect to a probability $p_{S_k}(o_i)$. This probability is defined proportionally to a pheromone factor and a heuristic factor, i.e.,

$$p_{S_k}(o_i) = \frac{[\tau_{S_k}(o_i)]^{\alpha}.[\eta_{S_k}(o_i)]^{\beta}}{\sum_{o_j \in Candidates}[\tau_{S_k}(o_j)]^{\alpha}.[\eta_{S_k}(o_j)]^{\beta}}$$

where $\tau_{S_k}(o_i)$ is the pheromone factor of o_i, $\eta_{S_k}(o_i)$ is its heuristic factor, and α and β are two parameters that determine the relative importance of these two factors.

The pheromone factor $\tau_{S_k}(o_i)$ depends on the quantity of pheromone laid on edges connecting the objects that already are in the partial solution S_k and the candidate node o_i, i.e.,

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

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

The heuristic factor $\eta_{S_k}(o_i)$ also depends on the whole set S_k of selected objects. Let $c_{S_k}(i) = \sum_{g \in S_k} r_{ig}$ be the consumed quantity of the resource i when the ant k has selected the set of objects S_k. And let $d_{S_k}(i) = b_i - c_{S_k}(i)$ be the remaining capacity of the resource i. We

define the following ratio:

$$h_{S_k}(j) = \sum_{i=1}^{m} \frac{r_{ij}}{d_{S_k}(i)}$$

which represents the tightness of the object j on the constraints i 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:

$$\eta_{S_k}(j) = \frac{p_j}{h_{S_k}(j)}$$

2.2 Pheromone updating

Once each ant has constructed a solution, pheromone trails laying on the construction graph edges 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 edge of the construction graph by a pheromone persistence rate $(1 - \rho)$ such that $0 \leq \rho \leq 1$.

Then, the best ant of the cycle deposits pheromone. More precisely, let $\mathcal{S}_k \in \{\mathcal{S}_1, \ldots, \mathcal{S}_{nbAnts}\}$ be the best solution (with maximal profit) constructed during the cycle, and $\mathcal{S}_{best}$ be the best solution built since the beginning of the run. The quantity of pheromone laid by ant k is inversely proportional to the gap of profit between $\mathcal{S}_k$ and $\mathcal{S}_{best}$, i.e., it is equal to $1/(1 + \text{profit}(S_{\text{best}}) - \text{profit}(S_k))$. This quantity of pheromone is added on each edge connecting two different vertices of S_k.

3. Parameters setting

When solving a combinatorial optimization problem with a heuristic approach such as evolutionary computation or ACO, one usually has to find a compromise between two dual goals. On one hand, one has to intensify the search around the most "promising" areas, that are usually close to the best solutions found so far. On the other hand, one has to diversify the search and favor exploration in order to discover new, and hopefully more successful, areas of the search space. The behavior of ants with respect to this intensification/diversification duality can be influenced by modifying parameter values. In particular, diversification can be emphasized either by decreasing the value of the pheromone factor weight α —so that ants become less sensitive to pheromone trails— or by decreasing the value of the pheromone evaporation rate ρ —so that pheromone evaporates more slowly. When increasing the exploratory

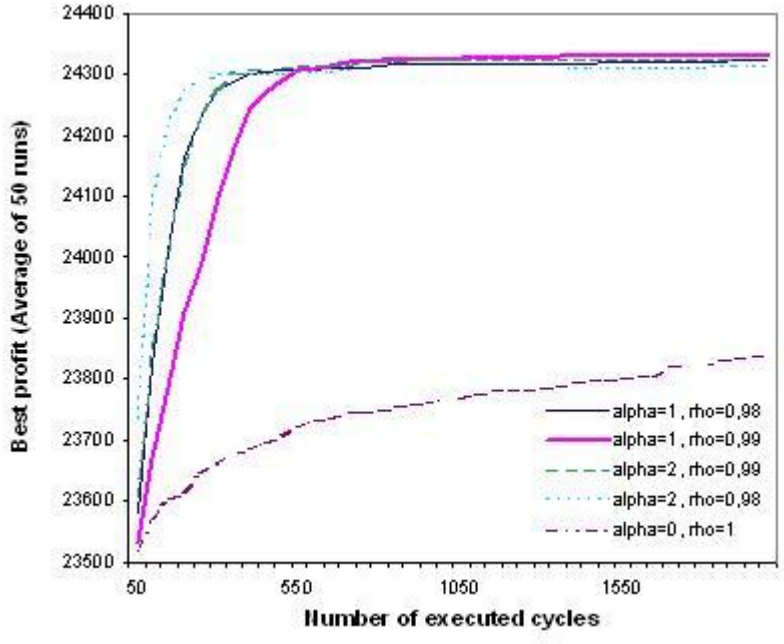

Figure 2. Influence of α and ρ on solution quality: each curve plots the evolution of the profit of the best solution when the number of cycles increases, for a given setting of α and ρ. The other parameters have been set to $\beta = 5$, nbAnts $= 30$, $\tau_{min} = 0.01$, and $\tau_{max} = 6$.

ability of ants in this way, one usually finds better solutions, but as a counterpart it takes longer time to find them.

This is illustrated in figure 2 on a MKP instance with 100 objects and 5 resource constraints. When emphasizing pheromone guidance, by choosing values such as $\alpha = 2$ and $\rho = 0.02$, Ant-knapsack quickly finds good solutions but it may fail in finding the optimal (or the best) solution. On the contrary, when choosing values for α and ρ that emphasize exploration, such as $\alpha = 1$ and $\rho = 0.01$, ants find better solutions, but they need more cycles to converge towards these solutions. A good compromise between solution quality and computation time is reached when α is set to 1 and ρ to 0.01.

For all experiments reported below, we have set α to 1, β to 5, ρ to 0.01, the number of ants nbAnts to 30, and the pheromone bounds τ_{min} and τ_{max} to 0.01 and 6. Finally, we limited the number of cycles to 2000.

4. Experiments and results

The Ant-knapsack has been tested on benchmarks of MKP from OR-Library [1]. We compare the results of Ant-knapsack with the two ACO algorithms of Leguizamon and Michalewicz [7] and Fidanova [5], and the genetic algorithm of Chu and Beasly [2].

Table 1. Results on 5.100 instances. For each instance, the table reports the best solutions found by Chu and Beasley as reported in [2](C. & B.), the best and average solutions found by Leguizamon and Michalewicz as reported in [7](L. & M.), and the best solutions found by Fidanova as reported in [5]. It then reports results obtained by Ant-knapsack: best and average solutions over 50 runs, followed by standard deviation in brackets, and the average number of cycles needed to find the best solution (C*).

N°	C. & B	L. & M.		Fidanova	Ant-knapsack		
	Best	Best	Avg	Best	Best	Avg (sdv)	C*
00	24381	24381	24331	23984	24381	24342 (29.3)	522
01	24274	24274	24245	24145	24274	24247 (38.5)	469
02	23551	23551	23527	23523	23551	23529 (8.0)	483
03	23534	23527	23463	22874	23534	23462 (32.6)	500
04	23991	23991	23949	23751	23991	23946 (31.8)	589
05	24613	24613	24563	24601	24613	24587 (31.3)	535
06	25591	25591	25504	25293	25591	25512 (43.8)	480
07	23410	23410	23361	23204	23410	23371 (30.3)	509
08	24216	24204	24173	23762	24216	24172 (32.9)	571
09	24411	24411	24326	24255	24411	24356 (44.3)	588
10	42757			42705	42757	42704 (14.3)	537
11	42545			42445	42510	42456 (15.8)	577
12	41968			41581	41967	41934 (22.3)	635
13	45090			44911	45071	45056 (24.0)	627
14	42218			42025	42218	42194 (33.2)	512
15	42927			42671	42927	42911 (33.3)	484
16	42009			41776	42009	41977 (45.2)	458
17	45020			44671	45010	44971 (32.5)	490
18	43441			43122	43441	43356 (38.5)	514
19	44554			44471	44554	44506 (25.2)	517
20	59822			59798	59822	59821 (3.2)	261
21	62081			61821	62081	62010 (47.1)	387
22	59802			59694	59802	59759 (21.7)	450
23	60479			60479	60479	60428 (21.8)	368
24	61091			60954	61091	61072 (20.0)	298
25	58959			58695	58959	58945 (14.5)	356
26	61538			61406	61538	61514 (24.0)	407
27	61520			61520	61520	61492 (25.6)	396
28	59453			59121	59453	59436 (40.5)	395
29	59965			59864	59965	59958 (8.4)	393

Table 2. Results on 10.100 instances. For each instance, the table reports the best solutions found by Chu and Beasley as reported in [2](C. & B), and by Leguizamon and Michalewicz as reported in [7](L. & M). It then reports results obtained by Ant-knapsack: best and average solutions over 50 runs, followed by standard deviation in brackets, and the average number of cycles needed to find the best solution (C*).

N°	**C. & B**	**L. & M.**		**Ant-knapsack**		
	Best	Best	Avg	Best	Avg (sdv)	C*
00	23064	23057	22996	23064	23016 (42.2)	538
01	22801	22801	22672	22801	22714 (67.2)	575
02	22131	22131	21980	22131	22034 (66.9)	598
03	22772	22772	22631	22717	22634 (60.6)	700
04	22751	22654	22578	22654	22547 (66.3)	640
05	22777	22652	22565	22716	22602 (63.3)	645
06	21875	21875	21758	21875	21777 (44.9)	552
07	22635	22551	22519	22551	22453 (89.2)	586
08	22511	22418	22292	22511	22351 (69.4)	534
09	22702	22702	22588	22702	22591 (88.5)	588
10	41395			41395	41329 (48.5)	501
11	42344			42344	42214 (49.5)	559
12	42401			42401	42300 (58.1)	584
13	45624			45624	45461 (73.6)	562
14	41884			41884	41739 (57.3)	536
15	42995			42995	42909 (76.3)	525
16	43559			43553	43464 (71.7)	597
17	42970			42970	42903 (47.7)	439
18	42212			42212	42146 (48.0)	598
19	41207			41207	41067 (89.7)	548
20	57375			57375	57318 (59.5)	330
21	58978			58978	58889 (40.2)	504
22	58391			58391	58333 (29.5)	513
23	61966			61966	61885 (42.4)	427
24	60803			60803	60798 (5.0)	316
25	61437			61437	61293 (52.7)	502
26	56377			56377	56324 (35.7)	453
27	59391			59391	59339 (53.3)	445
28	60205			60205	60146 (62.6)	360
29	60633			60633	60605 (36.1)	360

Table 3. Results on 5.500 instances. For each instance, the table reports the best solutions found by Vasquez and Hao as reported in [9](V. & H). It then reports results obtained by Ant-knapsack: best and average solutions over 50 runs, followed by standard deviation in brackets, and the average number of cycles needed to find the best solution (C*).

N°	V. & H.	Ant-knapsack		
	Best	Best	Avg (sdv)	C*
00	120134	119893	119658 (135.8)	885
01	117864	117604	117423 (130.4)	857
02	121112	120846	120622 (121.4)	860
03	120804	120534	120279 (152.3)	814
04	122319	122126	121829 (135.2)	826

Table 1 displays the results for 30 instances with 100 objects and 5 constraints (n=100 and m=5). On these instances, Ant-knapsack clearly outperforms Fidanova's algorithm. It also obtains better results than the algorithm of Leguizamon and Michalewicz: the best solutions found are always larger or equal, and the average solutions found are larger for 7 instances, and smaller for 3 instances. Ant-knapsack finds the best known results of Chu and Beasley for 26 instances over the 30 tested instances.

Table 2 displays the results for 30 instances with 100 objects and 10 constraints (n=100 and m=10). On these instances, Ant-knapsack also obtains better results than the algorithm of Leguizamon and Michalewicz: the best solutions found are larger or equal for 9 instances over 10, and the average solutions found are larger for 8 instances, and smaller for 2 instances. Ant-knapsack finds for this set also the best known results of Chu and Beasley for 25 instances over 30.

We also tested Ant-knapsack on larger MKP instances with 500 objects and 5 constraints (table 3). The best known results for this set are obtained by Vasquez and Hao [9]. They proposed an hybrid algorithm that combines tabu search and linear programming. On these difficult instances, we find worse results than those of Vasquez and Hao.

5. Conclusion

In this paper, we propose an ACO algorithm for the multidimensional knapsack problem. This algorithm differs from many ACO algorithms in the fact that pheromone trails are laid not only on the edges of the visited paths, but on all edges connecting any pair of nodes belonging to the solution. In addition, when adding a node to the solution under construction, the probability of choosing a node not only depends on the

pheromone trail between the last visited node and the candidate node but on the trails laying on all edges connecting the candidate node and all visited nodes in the solution. The proposed algorithm finds most of the best known results for the tested MKP benchmarks. This algorithm improves also many results found by other ACO algorithms.

Notes

1. available at `http://mscmga.ms.ic.ac.uk/`.

References

[1] B. Bullnheimer, R. F. Hartl, and C. Strauss. Applying the Ant System to the vehicle routing problem. In S. Voß S. Martello, I. H. Osman, and C. Roucairol, editors, Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, pp 285-296. Kluwer Academic Publishers, Dordrecht, 1999.

[2] P.C. Chu and J.E. Beasley. A genetic algorithm for the multidimentional knapsack problem. Journal of heuristic,4: 63-86, 1998.

[3] M. Dorigo, A. Colorni, and V. Maniezzo. The Ant System : Optimization By a Colony of Cooperating Agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B, Vol.26, No.1, pp.1-13, 1996.

[4] M. Dorigo and G. Di Caro. The Ant Colony Optimization Meta-Heuristic. in New Ideas in Optimization, D. Corne and M. Dorigo and F. Glover editors, pp11–32, 1999.

[5] S. Fidanova. Evolutionary Algorithm for Multidimensional Knapsack Problem. PPSNVII- Workshop 2002.

[6] L.M. Gambardella,É.D. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. Journal of the Operational Research Society , 50(2):167 176,1999.

[7] G. Leguizamon and Z. Michalewicz. A new version of Ant System for Subset Problem, Congress on Evolutionary Computation pp1459-1464, 1999.

[8] T. Stützle and H. H. Hoos. MAX-MIN Ant System. Future Generation Computer Systems, 16(8):889-914, 2000.

[9] M. Vasquez and J.K Hao, A Hybrid Approach for the 0-1 Multidimensional Knapsack Problem. IJCAI-01, Washington, 2001.