Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 35 (2008) 2672–2683
computers & operations research
www.elsevier.com/locate/cor

A new ant colony optimization algorithm for the multidimensional Knapsack problem

Min Kong[a,*], Peng Tian[a], Yucheng Kao[b]

[a] *Shanghai Jiaotong University, Shanghai, PR China*
[b] *Tatung University, Taipei, Taiwan, ROC*

Available online 8 January 2007

Abstract

The paper proposes a new ant colony optimization (ACO) approach, called binary ant system (BAS), to multidimensional Knapsack problem (MKP). Different from other ACO-based algorithms applied to MKP, BAS uses a pheromone laying method specially designed for the binary solution structure, and allows the generation of infeasible solutions in the solution construction procedure. A problem specific repair operator is incorporated to repair the infeasible solutions generated in every iteration. Pheromone update rule is designed in such a way that pheromone on the paths can be directly regarded as selecting probability. To avoid premature convergence, the pheromone re-initialization and different pheromone intensification strategy depending on the convergence status of the algorithm are incorporated. Experimental results show the advantages of BAS over other ACO-based approaches for the benchmark problems selected from OR library.

Keywords: Ant colony optimization; Binary ant system; Combinatorial optimization; Multidimensional Knapsack problem

1. Introduction

The multidimensional Knapsack problem (MKP) is one of the most intensively studied discrete programming problem [1]. Many practical problems can be formulated as a MKP, such as the capital budgeting problem, allocating processors and databases in a distributed computer system, project selection and cargo loading, and cutting stock problem. Formally, MKP can be formulated as

$$\text{maximize} \quad f(x) = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} r_{ij} x_j \leqslant b_i, \quad i = 1, \ldots, m, \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3}$$

* Corresponding author.
E-mail addresses: kongmin@sjtu.edu.cn (M. Kong), ptian@sjtu.edu.cn (P. Tian), ykao@ttu.edu.tw (Y. Kao).

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2006.12.029

Each of the m constraints described in condition (2) is called a Knapsack constraint, so the MKP is also called the m-dimensional Knapsack problem. Let $I = \{1, 2, \ldots, m\}$ and $J = \{1, 2, \ldots, n\}$, with $b_i > 0$ for all $i \in I$ and $r_{ij} \geqslant 0$ for all $i \in I$, $j \in J$, a well-stated MKP assumes that $p_j > 0$ and $r_{ij} \leqslant b_i < \sum_{j=1}^{n} r_{ij}$ for all $i \in I$, $j \in J$.

MKP can be regarded as a resource allocation problem, where we have m resources and n objects. Each resource $i \in I$ has a budget b_i, each object $j \in J$ has a profit p_j and consumes r_{ij} of resource i. The problem is to maximize the profit within a limited budget.

Ant colony optimization (ACO) is a recently developed, population-based meta-heuristic [2,3], which has been successfully applied to several NP-hard combinatorial optimization problems, such as traveling salesman problem [4,5], vehicle routing problem [6], and quadratic assignment problem [7,8]. The main idea of ACO algorithms is the indirect local communication among the individuals of a population of artificial ants. This is based on an analogy with trails of pheromone laid on the paths which real ants use for communication. 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 paper presents a binary ant system (BAS), a modification of the hyper-cube framework for ACO [9], applied to the constrained combinatorial optimization problem: MKP. A new pheromone laying method is designed for the binary solution structure of MKP. Instead of avoiding the constraint violation during solution construction, BAS allows the generation of infeasible solutions, and uses a problem specific repair operator to repair all the infeasible solutions generated in every iteration. Experimental results show that our proposed BAS approach can solve various MKP problems effectively and efficiently.

This paper is organized as follows. In the next section, we present the detail of BAS, together with a short discussion about its difference from other ACO based algorithms applied to MKP. Section 3 presents a theoretical analysis on why the pheromone can be directly regarded as selecting probabilities, and emphasizes the importance of incorporating methods to prevent premature convergence. Computational results in Section 4 show the performance of BAS in comparison with other ACO based algorithms. Finally, we end with some concluding remarks and discussions in Section 5.

2. BAS algorithm

The proposed BAS algorithm for MKP is briefly described in Fig. 1. It inherits the ACO meta-heuristic framework [3] and is basically modified from the HCF [9] in such a way that a repair operator is developed to extend the application of HCF to constrained problems. The remainder of this section will describes the algorithm in detail.

2.1. Solution representation and construction

BAS is designed specially for the problems with binary solution representation that is illustrated in Fig. 2. As for MKP, a bit string $x = \{x_1, \ldots, x_n\} \in \{0, 1\}^n$ represents a potential solution in BAS. Where n is the problem dimension, $x_j = 1$ means that the object j is selected, and $x_j = 0$ means that the object j is not selected. By this solution representation, we can see that a solution might not be feasible. An infeasible solution is one for which at least one of the Knapsack constraints is violated, i.e., $\sum_{j=1}^{n} r_{ij} x_j > b_i$ for some $i \in I$.

To apply BAS to MKP, we present MKP by a weighted graph $G = (C, L, \Gamma)$ as described in Fig. 3, where C is the set of nodes representing objects from 1 to n, together with an additional node $n + 1$, representing the end of the ants route. For any two consecutive nodes j and $j + 1$ ($j = 1, \ldots, n$), there are only two connections: the upper path $j0$ and the lower path $j1$. L is the set of all these connections. Γ is the set of pheromone on all the paths, pheromone are represented as τ_{j0} and τ_{j1}.

In BAS, a number of n_a ants cooperate together to search in the binary solution domain. At every iteration, each ant constructs its solution by walking sequentially from node 1 to node $n + 1$ on the routing graph. At each node j, ant either selects the upper path $j0$ or selects the lower path $j1$ to walk to the next node $j + 1$. Selecting $j0$ means that the ant does not select object j in its solution construction, such that $x_j = 0$; and selecting $j1$ means $x_j = 1$. The pheromone τ_{j0} and τ_{j1} cumulated on the paths determine the values of selecting probabilities:

$$p_{js}(t) = \frac{\tau_{js}(t)}{\tau_{j0}(t) + \tau_{j1}(t)}, \quad j = 1, \ldots, n, \quad s \in \{0, 1\}, \tag{4}$$

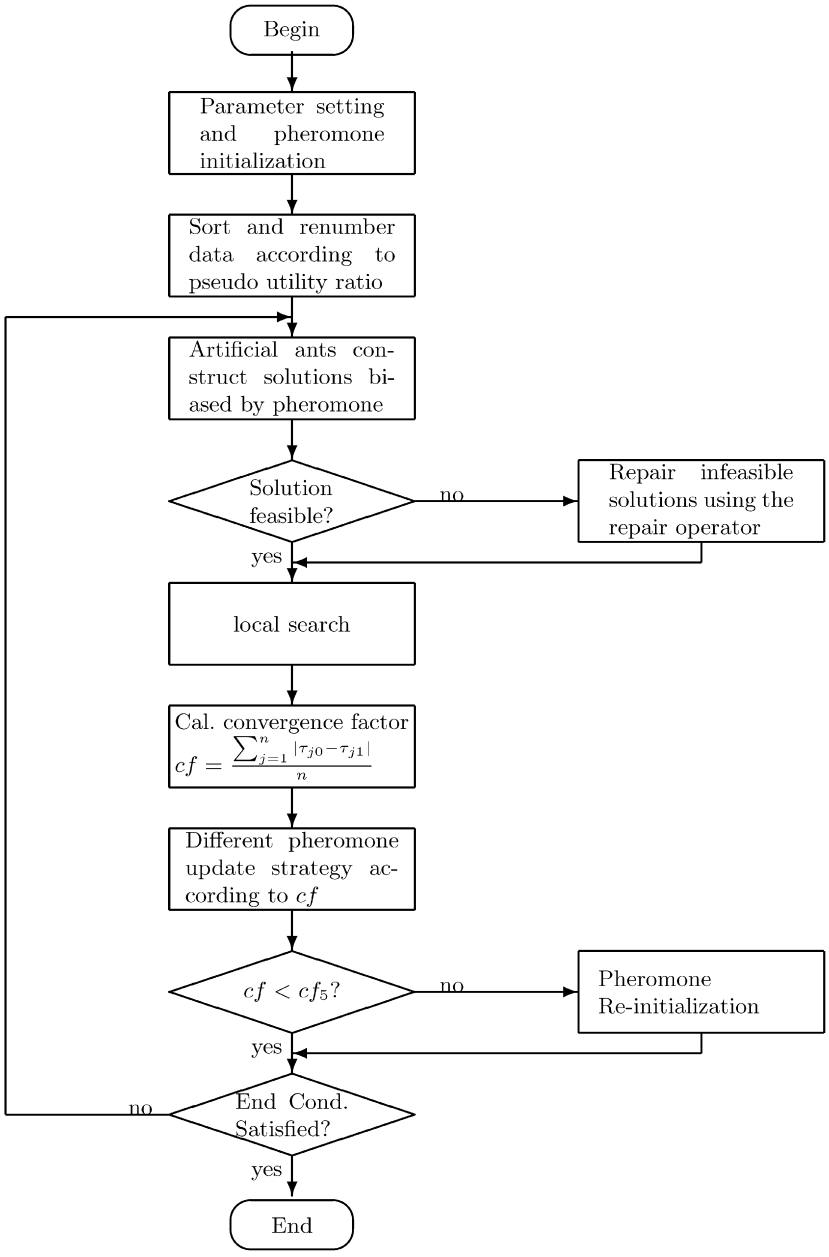

Fig. 1. Flowchart of BAS.

j	1	2	3	4	5	...	$n-1$	n
x_j	0	1	0	0	1	...	0	1

Fig. 2. Binary representation of a BAS_MKP solution.

where t is the number of iteration. BAS incorporates a special pheromone update rule in such a way that for every pair of the pheromone τ_{j0} and τ_{j1}, we have: $0 < \tau_{j0}, \tau_{j1} < 1$ and $\tau_{j0} + \tau_{j1} = 1$. So the pheromone can be directly regarded as selecting probability:

$$p_{js}(t) = \tau_{js}(t), \quad j = 1, \ldots, n, \quad s \in \{0, 1\}. \tag{5}$$

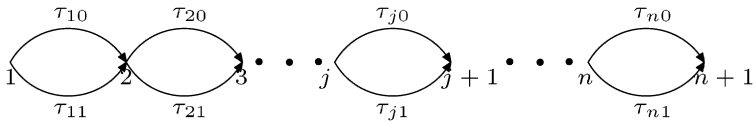

Fig. 3. Routing diagram for ants in BAS_MKP.

2.2. Pseudo-utility and repair operator

After all the ants have completed their tours, all the generated solutions are checked to see whether they are feasible. Infeasible solutions will be repaired by using a repair operator, which is a kind of greedy heuristic based on the pseudo-utility.

The pseudo-utility: At the initialization step, BAS sorts and renumbers variables according to the decreasing order of their pseudo-utility, which were calculated by the surrogate duality approach introduced by Pirkul [10]. The general idea of this approach is described very briefly as follows.

The surrogate relaxation problem of the MKP can be defined as

$$\text{maximize} \quad \sum_{j=1}^{n} p_j x_j \tag{6}$$

$$\text{subject to} \quad \sum_{j=1}^{n} \left(\sum_{i=1}^{m} \omega_i r_{ij} \right) x_j \leqslant \sum_{i=1}^{m} \omega_i b_i, \tag{7}$$

$$x_j \in \{0, 1\}, \quad j = 1, 2, \ldots, n, \tag{8}$$

where $\omega = \{\omega_1, \ldots, \omega_m\}$ is a set of surrogate multipliers (or weights) of some positive real numbers. We obtain these weights by using a simple method suggested by Pirkul [10], in which the LP relaxation of the original MKP is solved and the values of the dual variables are viewed as the weights. The weight ω_i can be seen as the shadow price of the ith constraint in the LP relaxation of the MKP.

The pseudo-utility ratio for each variable, based on the surrogate constraint coefficient, is defined as

$$u_j = \frac{p_j}{\sum_{i=1}^{m} \omega_i r_{ij}}. \tag{9}$$

The repair operator: The repair operator is inspired from the idea of Chu and Beasley [11], which consists of two phases. The first phase (called DROP) examines each bit of the solution string in increasing order of u_j and changes a bit from one to zero if feasibility is violated. The second phase (called ADD) reverses the process by examining each bit in decreasing order of u_j and changes a bit from zero to one as long as feasibility is not violated. The pseudo-code of the repair operator is given in Fig. 4.

2.3. Local search

Local search has been verified to be an effective approach to improve solutions generated by ACO [2,3]. In BAS_MKP, we design a simple random 4-flip method as the local search. That is, randomly select four variables from the solution, flip their values from 1 to 0 or 0 to 1, repair the new solution if necessary. If the new generated solution is better, replace the original solution with the new one. The local search method is performed 1000 times for each solution generated by ants. This method is somewhat ad hoc, however, the number of flips and the number of execution are selected through a set of massive experiments, which also proves the fact that incorporation of a local search in ACO is effective.

2.4. Pheromone update

Initially, pheromone on all the paths are set to an initial value: $\tau_0 = 0.5$. Later, as the algorithm goes on, pheromone are updated in such a way that those paths associated to better solutions receive more pheromone intensification, in order

Repair Operator for BAS_MKP
Let: R_i = the accumulated resources of constraint i in x
Initialize $R_i = \sum_{j=1}^{n} r_{ij}x_j, \forall i \in I$
$j = n$
While ($R_i > b_i$, for any $i \in I$) do /* DROP phase */
if $x_j = 1$ then
$x_j = 0; R_i = R_i - r_{ij}, \forall i \in I$
endif
$j = j - 1$;
endwhile
for $j = 1$ to n do /* ADD phase */
if $x_j = 0$ and $R_i + r_{ij} < b_i, \forall i \in I$ then
$x_j = 1; R_i = R_i + r_{ij}, \forall i \in I$;
endif
end for

Fig. 4. Pseudo-code of repair operator for BAS_MKP.

to guide ants in following iterations to search in a more promising area. BAS incorporates different pheromone update strategy depending on the convergence status of the algorithm. The convergence status of the algorithm is monitored through a convergence factor, which is defined as

$$cf = \frac{\sum_{j=1}^{n} |\tau_{j0} - \tau_{j1}|}{n}. \tag{10}$$

According to the definition of *cf*, at the beginning of the algorithm, all the pheromone have the value of $\tau_0 = 0.5$, such that $cf = 0$; as the algorithm goes on, the difference between τ_{j0} and τ_{j1} becomes larger and larger, and to be close to 1 when the algorithm falls into a local optima, such that $cf \to 1$. So, the convergence procedure of BAS to a local optima is equivalent to the procedure of *cf* changes from 0 to 1.

Pheromone update procedure in BAS consists of two parts: the first is the pheromone evaporation procedure, in which pheromone on all the paths evaporate according to the following equation:

$$\tau_{js}(t+1) \leftarrow (1-\rho)\tau_{js}(t), \quad j = 1, \ldots, n, \ \ s \in \{0, 1\}, \tag{11}$$

where ρ is the evaporation parameter, t is the iteration number. And the second part is the pheromone intensification procedure

$$\tau_{js}(t+1) \leftarrow \rho \sum_{x \in S_{\text{upd}} | js \in x} w_x, \quad j = 1, \ldots, n, \ \ s \in \{0, 1\}, \tag{12}$$

where S_{upd} is the set of solutions to be intensified in the pheromone update procedure, $w_x \in (0, 1)$ is the intensification weight for a solution x, which satisfies $\sum_{x \in S_{\text{upd}}} w_x = 1$, and $js \in x$ means that the bit x_{js} is set to 1 in the binary solution string x.

S_{upd} consists of three components, which are:

- the global best solution S^{gb}: the best solution generated since the start of the algorithm;
- the iteration best solution S^{ib}: the best solution generated in the current iteration by all the ants;
- the restart best solution S^{rb}: the best solution generated since the last re-initialization of the pheromone.

Depending on the value of *cf*, BAS uses different combinations of the pheromone intensification weight w_x, and decides whether a pheromone re-initialization procedure should be performed. The details of the pheromone intensification strategy is described in Table 1, in which five different pheromone update strategy are used in different status of *cf*. w_{ib}, w_{rb}, and w_{gb} are intensification weights for solution S^{ib}, S^{rb}, and S^{gb}, respectively. cf_i $(i = 1, \ldots, 5)$ are the threshold parameters for the division of the five stages. This method is first introduced by Blum and Dorigo [9], which is also proved to be effective in BAS.

Table 1
Pheromone intensification strategy for BAS

	$cf < cf_1$	$cf \in (cf_1, cf_2)$	$cf \in (cf_2, cf_3)$	$cf \in (cf_3, cf_4)$	$cf \in (cf_4, cf_5)$
w_{ib}	1	$\frac{2}{3}$	$\frac{1}{3}$	0	0
w_{rb}	0	$\frac{1}{3}$	$\frac{2}{3}$	1	0
w_{gb}	0	0	0	0	1

According to the value of cf, BAS uses different combination of S^{ib}, S^{rb} and S^{gb} to intensify the pheromone belonging to the solutions. While $cf \geqslant cf_5$, the pheromone re-initialization is performed.

When $cf \geqslant cf_5$, BAS performs the pheromone re-initialization procedure, in which all the pheromone are set to the initial value τ_0, and followed directly by a pheromone update procedure using S^{gb} as the only intensification solution in order to make the algorithm keep search around the most promising areas in the following iterations.

2.5. Termination condition

The iteration loop is repeated until some termination conditions are met, such as a maximum number of iterations has been performed or a satisfied solution is found. BAS stops as the optimum solution is found, or 3000 iterations are performed.

2.6. The difference to other ACO-based algorithms

There are several ACO algorithms available to solve MKP problems [12–14]. The main differences between BAS and these algorithms are summarized as follows:

- BAS uses a different way of pheromone laying, which is much simple and more general for the binary solution structure $\{0, 1\}^n$.
- Compared to other ACO-based approaches, BAS utilizes a repair operator with problem specific information, rather than complicated local heuristic methods.
- BAS allows to produce infeasible solutions and tries to repair them after solution construction, while other ACO-based approaches use the calculated constraint information to guide ants finding feasible solutions.

Since BAS bypasses the time-consuming calculation of the dynamic heuristic value and the constraint violation judgement during the tour construction procedure, it is much faster comparing to other ACO-based algorithms in each iteration. It can be seen from the algorithm we described previously in this section, the time complexity of the repair operator, as well as the complexity of the BAS per iteration, is approximately $O(mn)$, while the time complexity of other algorithms introduced before are at least $O(mn \log mn)$ per iteration.

3. Theoretical analysis

This section will present the theoretical analysis of using pheromone directly as selecting probability, and discuss the importance of incorporating methods to avoid premature convergence.

3.1. Pheromone as probability

Lemma 1. *For any pheromone value τ_{js} at any iteration step t in BAS,*

$$0 < \tau_{js}(t) < 1. \tag{13}$$

Proof. From the pheromone update procedure described in the previous section, obviously we have $\tau_{js}(t) > 0$. And because $\sum_{x \in S\text{upd}} w_x = 1$, we can calculate the upper limit for any particular path js according to

Eqs. (11) and (12):

$$\begin{aligned}\tau_{js}^{\max}(t) &\leqslant (1-\rho)\tau_{js}^{\max}(t-1)+\rho\\ &\leqslant (1-\rho)^t\tau_{js}(0)+\sum_{i=1}^{t}(1-\rho)^{i-1}\rho\\ &= (1-\rho)^t\tau_{js}(0)+1-(1-\rho)^t. \end{aligned} \tag{14}$$

Since BAS set the initial pheromone value as $\tau_{js}(0)=0.5$, we have $\tau_{js}^{\max}(t)<1$ according to the final sum of Eq. (14).

Theorem 1. *The pheromone values in BAS can be regarded as selecting probabilities throughout the iterations.*

Proof. Initially, since all the pheromone are set to $\tau_0=0.5$, obviously it meets the statement of the theorem. For the following iterations, what we need to do is to prove $\tau_{j0}(t)+\tau_{j1}(t)=1$, $0<\tau_{j0}(t),\tau_{j1}(t)<1$ holds for every variable x_i under the condition of $\tau_{j0}(t-1)+\tau_{j1}(t-1)=1$.

From Lemma 1, we can see that $0<\tau_{js}(t)<1$ holds for any path js. After the pheromone update procedure, all the pheromone values are partially evaporated, and there must be one and only one of τ_{j0} and τ_{j1} associated with any $x\in S_{\text{upd}}$ that receives pheromone intensification. Therefore, for any pheromone pair τ_{j0} and τ_{j1}, we have

$$\begin{aligned}\tau_{j0}(t)+\tau_{j1}(t) &= (1-\rho)\tau_{j0}(t-1)+\rho\sum_{x\in S_{\text{upd}}|j0\in x}w_x+(1-\rho)\tau_{j1}(t-1)+\rho\sum_{x\in S_{\text{upd}}|j1\in x}w_x\\ &= (1-\rho)(\tau_{j0}(t-1)+\tau_{j1}(t-1))+\rho\sum_{x\in S_{\text{upd}}}w_x\\ &= 1-\rho+\rho=1. \qquad \square\end{aligned}$$

3.2. Methods to avoid premature convergence

ACO meta-heuristic is able to find a satisfactory solution within a reasonable time for hard problems. The main convergence speed problem for an ACO algorithm is rather how to quickly jump out of a local optima than how to find it.

Proposition 1. *Once BAS falls into a local optima S^{local}, after a certain number of t iterations, all the pheromone associated to S^{local} are asymptotically within the range of $\tau_{\max}\pm\varepsilon$, while other pheromone values within the range of $\tau_{\min}\pm\varepsilon$, where $\tau_{\max}+\tau_{\min}=1$, $0<\tau_{\min}\ll\tau_{\max}<1$, and ε is a very small number.*

Proof. When BAS falls into a local optima S^{local}, we have $S^{\text{gb}}=S^{\text{ib}}=S^{\text{rb}}=S^{\text{local}}$. According to the pheromone update rule, for those τ_{js} associated to S^{local}, we have

$$\begin{aligned}\tau_{js}(t) &= (1-\rho)\tau_{js}(t-1)+\rho\\ &= (1-\rho)^t\tau_{js}(0)+\sum_{i=1}^{t}(1-\rho)^{i-1}\rho\\ &= 1-(1-\tau_{js}(0))(1-\rho)^t.\end{aligned}$$

Because $0<\tau_{js}(0)<1$ and $0<\rho<1$, we can see that $\tau_{js}(t)$ will increase quickly to be close to 1 as t increases. And the difference between any $\tau_{j_1s}(t)$ and $\tau_{j_2s}(t)$ associated to S^{local} is $(\tau_{j_1s}(0)-\tau_{j_2s}(0))(1-\rho)^t$, which decreases quickly to a very small number as t increases. Thus, any τ_{js} associated to S^{local} can be represented as $\tau_{\max}\pm\varepsilon$, where $\tau_{\max}$ is a number close to 1, and ε is a very small number.

For the same reason, for those τ_{js} not associated to S^{local}, we have

$$\begin{aligned}\tau_{js}(t) &= (1-\rho)\tau_{js}(t-1)\\ &= (1-\rho)^t\tau_{js}(0)\end{aligned}$$

which can be represented as $\tau_{\min} \pm \varepsilon$, where $\tau_{\min}$ is a small value close to 0.

Because for any pair of $\tau_{j0}(t)$ and $\tau_{j1}(t)$, we have $\tau_{j0}(t) + \tau_{j1}(t) = 1$, thus we can expect that $\tau_{\max} + \tau_{\min} = 1$.

Theorem 2. *Asymptotically, the probability to jump from a local optimal S^{local} to a better solution S^* is*

$$p^* \approx \tau_{\max}^{n-l}\tau_{\min}^{l}, \tag{15}$$

where n is the solution dimension. $l = |S^* - S^{\text{local}}|$ is the distance between S^* and S^{local}.

Proof. According to Proposition 1, when BAS falls into a local optima, after some certain number of iterations, those τ_{js} associated to S^{local} are asymptotically equal to $\tau_{\max}$, and other pheromone are asymptotically equal to $\tau_{\min}$. Since the distance between S^{local} and S^* is l, which means BAS needs a specific l flips to find S^* from S^{local}. According to the solution construction procedure of BAS, the probability to find S^* from S^{local} is

$$\begin{aligned}p^* &= \tau_{js|js\in S^{\text{local}}}^{n-l}\tau_{js|js\notin S^{\text{local}}}^{l}\\ &\approx \tau_{\max}^{n-l}\tau_{\min}^{l}.\end{aligned}$$

Theorem 2 gives the probability to jump out of a local optima, what interest us is how to control the pheromone to maximize this probability to speed the convergence. We get the maximum of p^* by solving the first partial differential equation of $\partial p^*/\partial\tau_{\max} = 0$, and get the results as

$$\begin{cases} p_{\max}^* = \frac{(n-l)^{n-l}l^l}{n^n},\\ \tau_{\max} = (n-l)/n,\\ \tau_{\min} = l/n. \end{cases} \tag{16}$$

Unfortunately, the value of $p_{\max}^*$ is too small, even is smaller than the probability of enumeration, which is $p_e = 1/C_n^l = (n-l)!\cdot l!/n!$. For example, in condition of $n = 100$ and $l = 1$, we get $p_{\max}^* = 0.0037$ and $p_e = 0.01$. The relative difference between $p_{\max}^*$ and p_e is even bigger with l increased, for example, $p_{\max}^*$ and p_e are 5.5235e − 5 and 2.0202e − 4, respectively, with $l = 2$.

This means that BAS is very difficult to jump out of a local optima once it gets in. That is the reason why some diversification methods are incorporated in BAS to avoid the low efficiency search around a local optima. These methods are: local search, pheromone re-initialization, and using different solutions for pheromone intensification.

Local search is an independent heuristic working in different neighborhood to improve the solutions generated by the ant system. The results of a local search is often a better solution found in the designed neighborhood of the current solution.

Pheromone re-initialization is a kind of restart of the algorithm. BAS performs the pheromone re-initialization once the algorithm gets into a local optima, this is achieved by monitoring the convergence factor *cf*. Pheromone re-initialization can avoid the algorithm searching around a local optima continuously with low effectiveness. Meanwhile, the pheromone re-initialization keeps the information of S^{gb} in the pheromone distribution which, on the one hand, keeps the previous search experience to guide the following search and, on the other, diversify the search in an enlarged search space.

The purpose of using a combination of different solutions for pheromone intensification is to enhance solution diversification in order to avoid falling into local optima. cf_i are the threshold parameters for determining the ways of combining different intensified solutions. To find a best solution combination in different stages of computational process, BAS initially emphasizes the importance of S^{ib} and then stresses the importance of S^{rb} and S^{gb} as iterations progress.

4. Experimental results

BAS has been tested on benchmarks of MKP selected from OR-Library. The tested instances are 5.100 with 100 objects and five constraints, and 10.100 with 100 objects and 10 constraints. If not other mentioned, the general parameter settings for all the tests are: $n_a = 30$, $\tau_0 = 0.5$, $cf = [0.3, 0.5, 0.7, 0.9, 0.95]$, and the maximum iteration number $t_{\max} = 3000$. The algorithm is programmed in ANSI C, based on ACOTSP1.0 developed by Stützle [15]. All the tests are running on a PC of Pentium(R) IV (2.80GHz) in MS Windows XP environment.

4.1. Parameter setting for ρ

Each set of benchmarks 5.100 and 10.100 contains 30 instances that is divided into three series with $\alpha = b_i / \sum_{j=1}^{n} r_{ij} = \frac{1}{4}$, $\alpha = \frac{1}{2}$ and $\alpha = \frac{3}{4}$. We select one instance from each series as the test instances for parameter setting experiment of ρ. The selected instances are: 5.100.01, 5.100.11, 5.100.21, 10.100.01, 10.100.11, and 10.100.21.

The range of ρ is set as $\rho \in [0.05, 0.75]$, step by 0.05. For each instance, every test is ran 30 times, where the test is stopped until a optimum is found.

Fig. 5 shows the average time to find the optimum solutions for each ρ. From Fig. 5 we can see that the setting of ρ does not affect a lot to the final results, but $\rho = 0.3$ consumes least average times in finding the optimums. So we will set $\rho = 0.3$ for the following experiments.

4.2. Comparison with other ACO approaches

We tested the results of BAS with those of other three ACO-based algorithms proposed by Leguizamon and Michalewicz [14], Fidanova [13], and Alaya et al. [12], respectively. Since none of these three papers provide any information about running times required and computer types used, we compared the performance of the four algorithms only in terms of solution quality under a maximum number of 3000 iterations.

Table 2 displays test results for 30 instances of benchmarks 5.100. For each instance, the table reports the best known solutions from OR-library, the best and average solutions found by Leguizamon and Michalewicz [14], the best solution found by Fidanova [13], the best and average solutions found by Alaya et al. [12], and the results from BAS_MKP, including the best, average solutions and the average time of finding the global best solutions over 30 runs for each instance. For these instances, BAS_MKP finds all the best solutions, which is not achieved by any other three

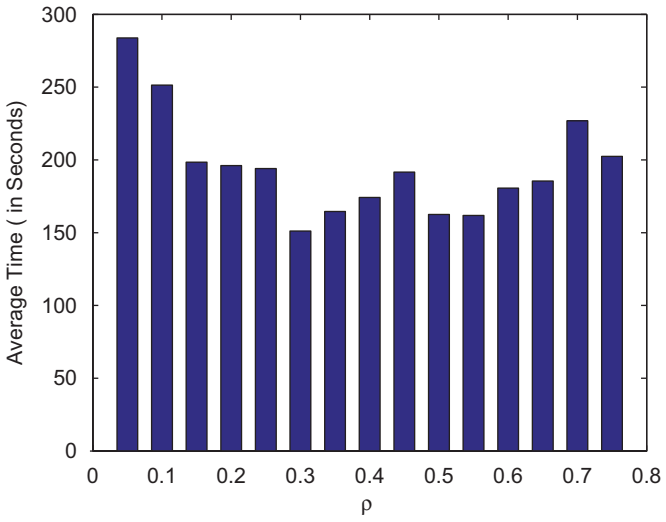

Fig. 5. Parameter setting for ρ. Test results are average time in finding the optimums of the six selected instances.

Table 2
The results of BAS_MKP on 5.100 instances

No	Best	L.&M.		Fidanova	Alaya et al.		BAS_MKP		
	known	Best	Avg.	Best	Best	Avg.	Best	Avg.	Avg. Time
00	24381	**24381**	24331	23984	**24381**	24342	**24381**	**24381.00**	1.52
01	24274	**24274**	24245	24145	**24274**	24247	**24274**	**24274.00**	22.63
02	23551	**23551**	23527	23523	**23551**	23529	**23551**	**23551.00**	82.40
03	23534	23527	23463	22874	**23534**	23462	**23534**	**23534.00**	141.45
04	23991	**23991**	23949	23751	**23991**	23946	**23991**	**23991.00**	12.87
05	24613	**24613**	24563	24601	**24613**	24587	**24613**	**24613.00**	3.20
06	25591	**25591**	25504	25293	**25591**	25512	**25591**	**25591.00**	0.25
07	23410	**23410**	23361	23204	**23410**	23371	**23410**	**23410.00**	0.95
08	24216	24204	24173	23762	**24216**	24172	**24216**	**24216.00**	135.60
09	24411	**24411**	24326	24255	**24411**	24356	**24411**	**24411.00**	11.50
10	42757			42705	**42757**	42704	**42757**	**42757.00**	38.96
11	42545			42445	42510	42456	**42545**	**42541.20**	203.08
12	41968			41581	41967	41934	**41968**	**41967.90**	173.23
13	45090			44911	45071	45056	**45090**	**45090.00**	74.74
14	42218			42025	**42218**	42194	**42218**	**42218.00**	4.18
15	42927			42671	**42927**	42911	**42927**	**42927.00**	1.63
16	42009			41776	**42009**	41977	**42009**	**42009.00**	1.28
17	45020			44671	45010	44971	**45020**	**45020.00**	14.26
18	43441			43122	**43441**	43356	**43441**	**43441.00**	60.58
19	44554			44471	**44554**	44506	**44554**	**44554.00**	11.22
20	59822			59798	**59822**	59821	**59822**	**59822.00**	6.61
21	62081			61821	**62081**	62010	**62081**	**62060.33**	139.41
22	59802			59694	**59802**	59759	**59802**	**59800.73**	113.73
23	60479			**60479**	**60479**	60428	**60479**	**60479.00**	69.74
24	61091			60954	**61091**	61072	**61091**	**61091.00**	59.42
25	58959			58695	**58959**	58945	**58959**	**58959.00**	3.77
26	61538			61406	**61538**	61514	**61538**	**61538.00**	35.20
27	61520			**61520**	**61520**	61492	**61520**	**61520.00**	52.15
28	59453			59121	**59453**	59436	**59453**	**59453.00**	2.03
29	59965			59864	**59965**	59958	**59965**	**59965.00**	50.87

algorithms. And BAS_MKP gets better average solutions for all the 30 instances tested. Actually BAS_MKP finds the optimal solutions at every trial for 26 instances, and is not able to find the optimal solution at some trials for the other four instances.

Table 3 displays the results for 30 instances of 10.100. For each instance, the table reports the best known solutions from OR-library, the best and average solutions found by Leguizamon and Michalewicz [14], the best and average solutions found by Alaya et al. [12], and the results from BAS_MKP, including the best, average solutions and the average times at finding the global best solutions over 30 runs for each instance. On these instances, BAS_MKP also obtains overall better results comparing to other two ACO algorithms. Except for instance 10.100.26, BAS_MKP finds optimal solutions for other 29 instances, and outperforms all the other algorithms in average solutions. More encouraging result is that BAS_MKP can find a better solution over the best known solution described in OR-library for instance 10.100.16.

From the viewpoint of computing time, for easier instances, the average computing time to find the optimal solutions is within one second; for some difficult instances, the average computing time is over 100 s; but the worst average computing time does not exceed 360 s. According to our experimental experiences, BAS can provide better solutions if the maximum iteration number is set to be larger. For example, if we set the maximum iteration number to 6000, BAS_MKP is able to find optimal solutions for most of the instances of 5.100; moreover, it can improve the overall average solutions for 10.100.

Table 3
The results of BAS_MKP on 10.100 instances

No	Best	L.&M.		Alaya et al.		BAS_MKP		
	known	Best	Avg.	Best	Avg.	Best	Avg.	Avg. Time
00	23064	23057	22996	**23064**	23016	**23064**	**23064.00**	28.89
01	22801	**22801**	22672	**22801**	22714	**22801**	**22801.00**	38.85
02	22131	**22131**	21980	**22131**	22034	**22131**	**22131.00**	26.61
03	22772	**22772**	22631	22717	22634	**22772**	**22771.70**	5.90
04	22751	22654	22578	22654	22547	**22751**	**22751.00**	130.27
05	22777	22652	22565	22716	22602	**22777**	**22771.93**	247.77
06	21875	**21875**	21758	**21875**	21777	**21875**	**21875.00**	31.81
07	22635	22551	22519	22551	22453	**22635**	**22635.00**	14.63
08	22511	22418	22292	**22511**	22351	**22511**	**22511.00**	91.87
09	22702	**22702**	22588	**22702**	22591	**22702**	**22702.00**	0.87
10	41395			**41395**	41329	**41395**	**41388.70**	38.53
11	42344			**42344**	42214	**42344**	**42344.00**	163.70
12	42401			**42401**	42300	**42401**	**42401.00**	57.83
13	45624			**45624**	45461	**45624**	**45606.67**	186.84
14	41884			**41884**	41739	**41884**	**41881.00**	359.18
15	42995			**42995**	42909	**42995**	**42995.00**	18.62
16	43559			43553	43464	**43574**	**43561.00**	67.44
17	42970			**42970**	42903	**42970**	**42970.00**	18.62
18	42212			**42212**	42146	**42212**	**42212.00**	0.58
19	41207			**41207**	41067	**41207**	**41198.60**	218.88
20	57375			**57375**	57318	**57375**	**57375.00**	25.59
21	58978			**58978**	58889	**58978**	**58978.00**	83.20
22	58391			**58391**	58333	**58391**	**58391.00**	34.22
23	61966			**61966**	61885	**61966**	**61919.20**	141.42
24	60803			**60803**	60798	**60803**	**60803.00**	3.24
25	61437			**61437**	61293	**61437**	**61418.60**	227.48
26	56377			**56377**	56324	56353	**56353.00**	133.99
27	59391			**59391**	59339	**59391**	**59391.00**	5.26
28	60205			**60205**	60146	**60205**	**60199.80**	194.22
29	60633			**60633**	60605	**60633**	**60633.00**	10.18

5. Conclusions

In this paper we proposed a new ACO-based algorithm, BAS, for solving the multidimensional Knapsack problem. BAS differs from previously proposed ACO-based algorithms in several ways. First, the pheromone laying method is specially designed, so pheromone levels can directly represent the probability of path selection when ants construct solutions. Second, in order to reduce the computational complexity of BAS, the local heuristic information is not considered. Third, BAS uses a problem-specific repair operator to guarantee the feasibility of solutions. Last, BAS effectively avoids premature convergence by means of combination of three types of solutions for pheromone updating. The computational results show that BAS outperforms other previously proposed ACO algorithms in solution quality. The results also reveal that BAS is able to effectively solve MKP problems with large size.

The binary solution structure and special pheromone laying method allow BAS to extend its applications to other 0–1 integer-programming problems and function optimization problems with binary solution coding.

References

[1] Martello S, Toth P. Knapsack problems: algorithms and computer implementations. New York: Wiley; 1990.
[2] Bonabeau E, Dorigo M, Theraulaz G. Swarm intelligence: from natural to artificial systems. New York, Oxford: Oxford University Press; 1999.
[3] Dorigo M, Stützle T. Ant colony optimization. Cambridge, Massachusetts, London, England: The MIT Press; 2004.
[4] Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolution and Computation 1997;1:53–66.

[5] Dorigo M, Di Caro G, Gambardella LM. Ant algorithms for discrete optimization. Artificial Life 1999;5(3):137–72.
[6] Gambardella LM, Taillard ED, Agazzi G. MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corne D, Dorigo M, Glover F, editors. New ideas in optimization. London, UK: McGraw-Hill; 1999. p. 63–76.
[7] Gambardella L, Taillard E, Dorigo M. Ant colonies for the quadratic assignment problem. Journal of Operations Research Society 1999;50(2): 167–76.
[8] Maniezzo V, Colorni A. The ant system applied to the quadratic assignment problem. IEEE Transations on Data Knowledge and Engineering 1999;11:769–78.
[9] Blum C, Dorigo M. The hyper-cube framework for ant colony optimization. IEEE Transactions on Man, Systems and Cybernetics—Part B 2004;34(2):1161–72.
[10] Pirkul H. A heuristic solution procedure for the multiconstraint zero–one knapsack problem. Naval Research Logistics 1987;34:161–72.
[11] Chu PC, Beasley JE. A genetic algorithm for the multidimentional knapsack problem. Journal of Heuristic 1998;4:63–86.
[12] Alaya I, Solnon C, Ghéira K. Ant algorithm for the multi-dimensional knapsack problem. International conference on bioinspired optimization methods and their applications. BIOMA 2004, October 2004, p. 63–72.
[13] Fidanova S. Evolutionary algorithm for multidimensional knapsack problem. PPSNVII-Workshop 2002.
[14] Leguizamon G, Michalewicz Z. A New version of ant system for subset problem. Congress on evolutionary computation, 1999; p. 1459–64.
[15] Stützle T. ACOTSP, Version 1.0. Available from ⟨http://www.aco-metaheuristic.org/aco-code⟩, 2004.