TECHNICAL TRANSACTIONS | CZASOPISMO TECHNICZNE
AUTOMATIC CONTROL | AUTOMATYKA
3-AC/2013

KRZYSZTOF SCHIFF*

ANT COLONY OPTIMIZATION ALGORITHM FOR THE 0-1 KNAPSACK PROBLEM

ALGORYTM MRÓWKOWY DLA 0-1 PROBLEMU PLECAKOWEGO

A b s t r a c t

This article describes a new ant colony optimisation algorithm for the discrete knapsack problem with a new heuristic pattern, based on the ratio of the square of the profit coefficient to the square of the weight coefficient of the original problem. This new heuristic is used in order to choose objects that should be packed into the knapsack. This pattern was compared with two used in ant algorithms and which have been presented in the literature on the subject of ant colony optimisation algorithms for the 0-1 Knapsack Problem. The two other patterns are based on the ratio of the profit coefficient to the weight coefficient multiplied respectively by the total and the current knapsack load capacity. Results of tests under a width range of ant algorithm parameters such as the number of cycles, the number of ants, the evaporation rate, and the load knapsack capacity are shown and discussed.

Keywords: *knapsack problem, ant colony optimisation, heuristic algorithm*

S t r e s z c z e n i e

W artykule przedstawiono algorytm mrówkowy dla dyskretnego problemu plecakowego z nową heurystyką wyboru obiektów i został on porównany z dwoma innymi algorytmami spotkanymi w literaturze przedmiotu pod względem uzyskiwanego całkowitego zysku z załadowanych do plecaka przedmiotów. Nowa heurystyka wyboru została wyrażona poprzez stosunek kwadratu zysku do kwadratu wagi wybranego przedmiotu, gdy dwie znane już heurystyki to stosunek zysku do wagi odpowiednio pomnożony przez całkowitą i bieżącą ładowność plecaka. W artykule przedstawiono wyniki przeprowadzonych testów dla szerokiego zakresu parametrów algorytmów mrówkowych takich jak: współczynnik parowania, liczba cykli, liczba mrówek, ładowności plecaka jak i dla różnej liczby dostępnych przedmiotów do załadunku.

Słowa kluczowe: *problem plecakowy, algorytm mrówkowy, heurystyka*

* Ph.D. Krzysztof Schiff, e-mail: kschiff@pk.edu.pl, Department of Automatic Control and Information Technology, Faculty of Electrical and Computer Engineering, Cracow University of Technology.

1. Introduction

Many optimisation problems in decision-making can be presented as the 0-1 Knapsack Problem (KP) [1]. The 0-1 Knapsack Problem consists of loading objects in to a knapsack in such a way that the obtained total profit of all objects included in the knapsack is maximum and the sum of the weights of all packed objects does not exceed the total knapsack load capacity. Each object can be loaded or not loaded into the knapsack; this is the 0-1 decision concerning object loading. There are also other versions of this problem such as the Multi-dimensional 0-1 Knapsack Problem [10, 13–15] or the Multiple 0-1 Knapsack Problem [7–9, 11, 12].

The 0-1 Knapsack Problem is an NP-difficult (NP: non-polynomial) problem [2]. The exact solution to an NP problem is not obtained in a short period of time, computer algorithms take a great deal of time to arrive at a solution. The 0-1 Knapsack Problem can be solved by using the exact methods [4–6]. In order to obtain the solution for the 0-1 KP in a short period of time, heuristic algorithms are used. Such algorithms, which are based on the behaviour of ants, are taken into consideration in this paper.

Ant algorithms are very suitable for NP-complete problems [17]. Ants construct solutions to the problem and the best solution from their work is remembered in each algorithm cycle. Ants construct their solution using a pheromone, which is a chemical signal. The quantity of the pheromone connected with the objects, which constitute a solution to the problem, varies during the algorithm action. The quantity of the pheromone decreases for all objects and is shown as evaporation. The quantity of the pheromone rises when an additional quantity is added to all objects, which constitutes the best solution. A greater quantity of the pheromone means a greater probability of the object being selected by ants during their search for the optimal solution to the problem. A decision on element selecting depends not only on the quantity of the pheromone, but also on heuristic information, which can be expressed with a different kind of pattern. The heuristic pattern is additional information for ants about the problem, which helps them to find a better solution in comparison to a situation when this heuristic information is not used during optimal construction of a solution by all ants.

A mathematical model of the 0-1 Knapsack Problem is presented in section 2, a general pseudo-code of the Ant Colony Optimisation algorithm is discussed, a proposed heuristic pattern and two other patterns which have been used in ant algorithms, are formulated in section 3. The results of the conducted tests are shown and discussed in section 4. These experiments compare the action of algorithms with all three heuristic patterns – the new version presented in this paper, the static version used in papers [7–9, 11] and the dynamic version used in papers [10, 12–16].

In the static heuristic pattern, the total load knapsack capacity does not change at all during the algorithm's operation, whereas in the dynamic heuristic pattern, the current load knapsack capacity changes constantly since objects are constantly added to the knapsack in each cycle of the ant algorithm.

2. Mathematical formalisation

The mathematical model of the 0-1 Knapsack Problem can be stated in the same way as in paper [3]:

$$\max \sum_{i=1}^{n} z_i x_i \tag{1}$$

with constraints:

$$\sum_{i=1}^{n} w_i x_i \leq C \tag{2}$$

where:

C – is the total knapsack load capacity,
z_i – is the profit on an object i,
w_i – is the weight of an object i,
C, z_i, w_i – are all integers and positive numbers,
and $x_i = 0$ – when an object i has not been loaded into a knapsack,
or $x_i = 1$ – when an object i has been loaded into a knapsack.

The knapsack has its own capacity C. Each object has its own weight w_i. The total weight of all objects which have been packed into the knapsack should not exceed the total knapsack load capacity. All objects should be selected to be packed so that the total profit Z of all objects which have been packed into the knapsack should be maximal. An object o_i has been loaded into a knapsack or it has not been loaded; thus a variable x_i corresponding to this object has two states or values $\{0, 1\}$. There are n objects to be loaded into the knapsack $\{o_1, o_2, ..., o_n\}$. Each object o_i has its own profit z_i and its own weight w_i. The set $N = \{o_1, o_2, ..., o_n\}$ is the set of all available objects which can be loaded into the knapsack.

When an object o_i goes into the knapsack, the latter's capacity is decreased by the weight of this loaded object o_i. This new value for the available knapsack capacity is called the current knapsack capacity V_c. If the weight of any object is greater than the current knapsack capacity V_c, then that object cannot be loaded into the knapsack and is removed from the list N. After an object o_i is loaded into the knapsack, a new list N_j has to be compiled. This new list N_j is obtained from the preceding list N_i, which was stated before the object o_i was loaded into the knapsack. The solution to the problem constitutes the objects loaded into the knapsack, i.e. objects which have been included in the set S.

The total weight of all packed objects in the knapsack should not be larger than the knapsack load capacity and the total profit of all packed objects in the knapsack should be maximal. At the beginning of the ants' work, the knapsack is empty; thus the set S is also empty ($S = \{\}$). Next, all objects from the set N are verified in terms of their weights and the knapsack load capacity. A new object o_i can be selected to be packed into the knapsack only from those objects whose weight is less than the current knapsack load capacity. After a selected object o_i has been packed into the knapsack, the current knapsack capacity is reduced. There is an object o_i inside the knapsack $S = \{o_{i1}\}$.

In some steps j, other objects are loaded into the knapsack, so the contents of the set S_j are $\{o_{i1}, o_{i2}, ..., o_{ij}\}$. If the set $S_k = \{o_{i1}, o_{i2}, ..., o_{ik}\}$ is the solution to the problem, then the set $S_j = \{o_{i1}, o_{i2}, ..., o_{ij}\}$ is a partial solution to the problem or a solution under construction, $j < k$. The final contents of the set S_k are obtained as a result of the selection of objects from available objects N. When the next object o_{im} is selected, then state S_j changes to another state

$S_m = \{o_{i1}, o_{i2}, ..., o_{ij}, o_{im}\}, j \ m < k$. The total current profit of objects is Z_j and Z_m according to the states j and m. After an object o_j is packed into the knapsack, the current knapsack load capacity is less than it was before.

As a result of the reduced current knapsack load capacity, not all objects from the set N_i can now be loaded since their weights are too great in comparison to the current knapsack load capacity. Some objects from the set N_i are removed, since their weights are too great to be packed into the knapsack, and thus a new set N_j of available objects is obtained. The set N_j, from which a new object o_j can be selected in order to be packed into the knapsack, is called the neighbourhood of state S_i. Objects which constitute this neighbourhood N_j come from neighbourhood N_i; these objects came in turn from neighbourhood N_i, with a weight less than or equal to the current knapsack load capacity V_c.

3. Structure of the ACO algorithm

In ant algorithms a colony of artificial ants is looking for a good solution to the investigated problem. The pseudo-code of the ACO algorithm is presented as procedure 1. Each artificial ant constructs an entire solution to the problem in a certain number of steps; at each step there is an intermediate solution, a partial solution or a state. In each step, each ant k goes from one state i to another state j and thus constructs a new intermediate solution. At the end, the entire solution will have been obtained in a certain number of steps. At each step, each ant k takes into consideration a set of feasible expansions to its current state and moves to one of these in probability. This set of feasible expansions is called a neighbourhood.

In the presented algorithm for the 0-1 Knapsack Problem, at each state i there is a partial solution S_i of the Knapsack Problem; each ant selects the next object o_i from the set N_i of available objects, goes to the next state j and adds this selected object to a partial solution S_j in order to construct, at the end of the algorithm operation, the entire solution S to the 0-1 Knapsack Problem. At the end of the algorithm operation, the set of objects S constitutes a solution to the 0-1 Knapsack Problem. Each ant k starts with an empty set S and successively adds to this set objects selected one after the other with probability p_j moving from one state i to another state j. At each state i there are certain objects in set S_i which constitute a partial solution. Each ant, in order to construct a solution, uses common information which is encoded in pheromone trails τ_j. Each ant also deposits pheromones on all objects included in the knapsack when a solution has been found. The quantity of the pheromones $\Delta\tau$ deposited depends on the quality of this solution Q. Each ant's move also depends on the so-called attractiveness of the move μ_j. In order to avoid a very rapid convergence to a locally optimal solution, the evaporation mechanism $\tau = \rho\tau$ is used. Over time, the pheromone trail evaporates, thus reducing its attractive strength. Each ant k moves from one state i to another state j according to a transition probability rule p_j:

$$p_j = \begin{cases} \dfrac{\tau_j^{\alpha}\mu_j^{\beta}}{\sum_{j \in N_i} \tau_j^{\alpha}\mu_j^{\beta}}, & \text{for } \ j \in N_i \\ 0, & \text{for } \ j \notin N_i \end{cases} \tag{3}$$

Procedure 1

ACO procedure for the 0-1 Knapsack Problem

begin
  while (a cycle exists) **do**
    while (an ant k, which has not yet worked, exists) **do**
      while ($V_c \geq 0$) **do**
        select a next object o_j from N_i with probability $p_j = \begin{cases} \frac{\tau_j^{\alpha}\mu_j^{\beta}}{\sum_{j \in N_i} \tau_j^{\alpha}\mu_j^{\beta}}, & \text{for } j \in N_i \\ 0, & \text{for } j \notin N_i \end{cases}$
        add a selected object to a partial solution $S = S + \{o_j\}$
        update the current knapsack load capacity $V_C = V_C - w_j$
        update the profit $Z = Z + z_j$
        update the neighbourhood of the current state $N_i = \{o_i : w_i \leq V_C\}$
      end
      remember the best solution if a better solution has been found
    end
    remember a global best solution if a better solution has been found
    use an evaporation mechanism $\tau = \rho\tau$
    update pheromone trails $\tau = \tau + \Delta\tau$
  end
end.

using the pheromone trail τ_j and the attractiveness μ_j of the move. The pheromone trail τ_j is useful information, deposited by other ants, about their usage of object j in the past. The attractiveness μ_j is the desire to select an object j from the neighbourhood N_i of the current state. The attractiveness μ_j enables better selection of an object from all available objects which constitute the neighbourhood N_i of the current state and which can be added to the solution under construction. The neighbourhood N_i of state i is composed of objects which can be added to a constructed partial solution. At the start of the ants' work, all objects can be added to a partial solution of the problem, i.e. to a solution of a problem under construction. The number of these objects is reduced afterwards not only because of their inclusion in the partial solution S, but also because some of these objects cannot be added to a solution which is under construction, since these objects already fail to satisfy solution constraints. Only those objects which still satisfy solution constraints can be added to a constructed partial solution. The partial solution of the problem is a part of a solution or a solution under construction. The partial solution is a subset of the objects which constitute an entire solution to the problem. Parameters α and β, which are used in the transition probability rule p_j expressed by formula (3), indicate how important the pheromone trail τ_j and the attractiveness μ_j are during transitions from one state to another. After a solution has been found, each ant deposits some quantity $\Delta\tau$ of pheromones on all objects which constitute the solution S, in accordance with the pattern:

$$\tau = \tau + \Delta\tau \tag{4}$$

A quantity of deposited pheromones $\Delta\tau$ is expressed as:

$$\Delta\tau = f(Q) = \frac{1}{1 + \frac{z_{best} - z}{z_{best}}} \tag{5}$$

Thus those objects which were included into a solution have received an additional quantity of pheromones and can be selected afterwards with a higher probability than other objects.

An evaporation mechanism is incorporated into ant algorithms in order to avoid too rapid a convergence to a suboptimal solution. The intensity of evaporation is controlled by the parameter ρ. The quantity of the pheromone on each object is updated at the end of each cycle in accordance with the pattern:

$$\tau = \rho\tau, \quad \rho \in (0, 1] \tag{6}$$

Three ant colony optimisation algorithms were implemented. They are called:
1) AKA1, – with the attractiveness μ_j of the move expressed as:

$$\mu_j = \frac{z_j}{\frac{w_j}{V_C}} \tag{7}$$

2) AKA2, – with the attractiveness μ_j of the move expressed as:

$$\mu_j = \frac{z_j}{w_j^2} \tag{8}$$

3) AKA3, – with the attractiveness μ_j of the move expressed as:

$$\mu_j = \frac{z_j}{\frac{w_j}{C}} \tag{9}$$

where:

C	–	is the total knapsack load capacity,
V_c	–	is the current knapsack load capacity, $V_C = C - \sum_{g \in S_i}(w_g)$,
S_i	–	is a partial solution,
$\sum_{g \in S_i}(w_g)$	–	is the weight of all objects which were included in the partial solution S_i,
w_j	–	is the weight of selected object j,
z_j	–	is the profit of selected object j,
μ_j	–	is the attractiveness of selecting an object j.

AKA1 and AKA3 are algorithms with static and dynamic heuristic patterns. The heuristic patterns for algorithms AKA1 and AKA3 are the ones used in ant algorithms for multi-knapsack systems and which have been adopted to the one-knapsack system in this paper. The heuristic pattern of the AKA2 algorithm is the one proposed in this paper. All tests were conducted on a computer with an Intel Celeron CPU, 1.7 GHz and 256 MB RAM.

4. Results of experiments

The first experiment was conducted for a different number of algorithm cycles {100, 200, 300, 400, 500, 600} for 300 objects and for a knapsack load capacity equal to 3000, for an evaporation rate set at 0.95 and for a number of ants equal to 120. A profit z_i and a weight w_i were randomly generated for each object o_i in the range <1, 10> and <1, 100> adequately. Thus, it was possible to generate 1000 different objects (w_i, z_i). Average values were obtained from 10 measurements for each number of cycles – these values are shown in Table 4.1 and in Fig. 4.1. The results show that the AKA2 algorithm yields a higher profit than the other two algorithms. These three values of profit converge when the number of cycles rises, but if the result must be obtained as fast as possible, the AKA2 algorithm is the most suitable. The AKA2 algorithm yields a higher profit than the other two algorithms when the number of cycles is equal to or less than 200.

Table 1

Profit in dependence on number of cycles

n. of cycles	100	200	300	400	500	600
AKA1	815.0	820.0	820.2	820.2	820.4	820.8
AKA2	819.9	820.9	821.1	821.2	820.9	820.9
AKA3	817.3	819.5	819.9	820.8	820.4	820.9

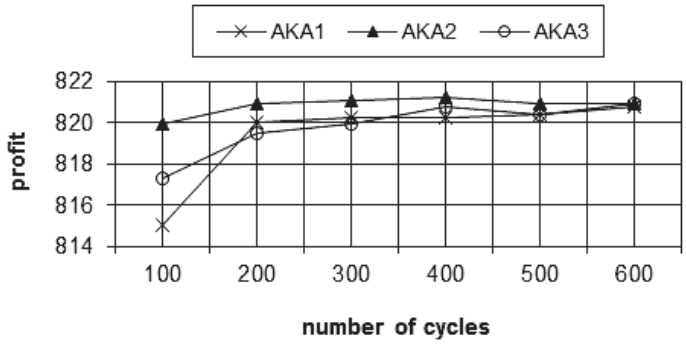

Fig. 1. Profit in dependence on number of cycles

Table 2

Profit in dependence on number of ants

n. of ants	20	40	60	80	100	120	140
AKA1	799.8	800.6	800.4	801.2	801.5	801.9	801.5
AKA2	801.2	801.6	802.1	801.8	802.1	802.3	802.2
AKA3	799.7	800.6	800.6	800.8	801.0	801.7	801.5

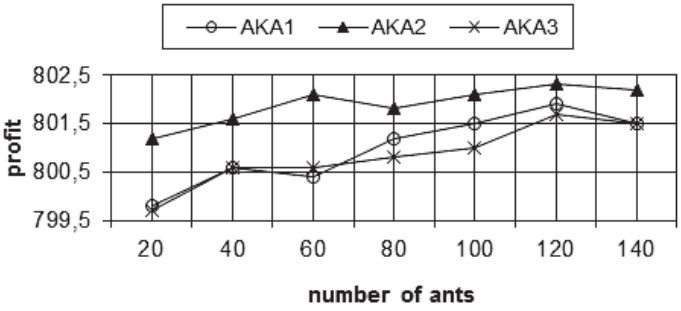

Fig. 2. Profit in dependence on number of ants

The second experiment was conducted for a constant number of cycles equal to 500 and for a different number of ants {20, 40, 60, 80, 100, 120, 140}, for a load knapsack capacity equal to 3000, an evaporation rate equal to 0.95 and a constant number of available objects set at 300. A profit z_i and a weight w_i were randomly generated for each object o_i in the range <1, 10> and <1, 100> adequately. Thus, it was possible to generate 1000 different objects (w_i, z_i). Average values were obtained from 10 measurements for each different number of cycles; these values are shown in Table 2 and in Fig. 2. The results show that the AKA2 algorithm yields a higher profit than the other two algorithms for all numbers of ants. All profit values of three algorithms rise when the number of ants rises. There is also some degree of saturation – if the number of ants is higher than 60, there is no significant improvement in obtained profit.

Table 3

Profit in dependence of evaporation rate

ρ	0.91	0.92	0.93	0.94	0.95	0.96	0.97	0.98	0.99
AKA1	827.7	827.6	828.3	828.3	828.5	828.9	828.2	828.1	827.3
AKA2	829.4	829.5	829.8	829.5	829.7	829.5	829.5	829.6	828.9
AKA3	828.6	828.6	828.7	829.4	828.6	828.8	828.6	828.2	827.1

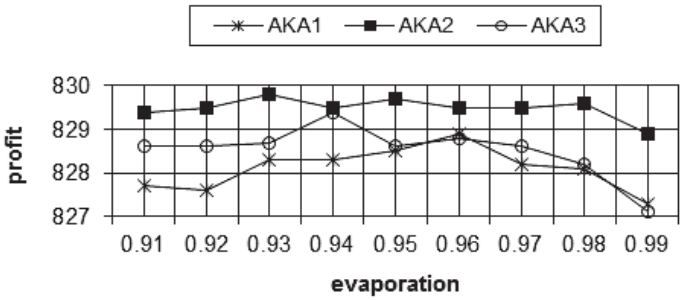

Fig. 3. Profit in dependence of evaporation rate

The third experiment was conducted for a constant number of cycles equal to 500 and for a constant number of ants equal to 100, for a constant load knapsack capacity set at 3000 and for a constant number of available objects set at 300, but for different evaporation rates (0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99). A profit z_i and a weight w_i were randomly generated for each object o_i in the range <1, 10> and <1, 100>, respectively. Thus, it was possible to generate 1000 different objects (w_i, z_i). Average values were obtained from 10 measurements and shown in Table 3 and in Fig. 3. The results of the experiment show that the AKA2 algorithm yields a higher value of profit than the other two algorithms. The best range for the AKA2 algorithm is between 0.93 and 0.98.

Table 4

Profit in dependence on the number of ants for a number of cycles equal to 200

n. of ants	40	60	80	100	120
AKA1	833.8	833.8	834.3	834.0	835.0
AKA2	835.7	836.2	836.4	836.5	836.8
AKA3	833.9	833.4	833.9	834.7	834.5
p. of reference	837.2	837.2	837.2	837.2	837.2

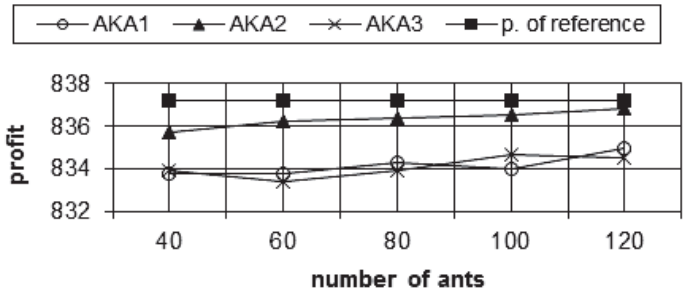

Fig. 4. Profit in dependence on the number of ants for a number of cycles equal to 200

The first experiment has shown that there is no need to set the number of cycles higher than 200 when the number of ants equals 120. Therefore, the number of cycles can be set lower without lowering the quality of the solution. In Tables 4 and 5 and in Fig. 4 and 5, the results for the numbers of cycles equal to 200 and 300 and for different numbers of ants have been shown. The point of reference is the profit obtained when the number of cycles and ants were set at 500 and 120, respectively. This point of reference is achieved when the number of cycles and the number of ants are set at 300 and 60, respectively. Thus, there is no need to set the number of cycles at 500 and the number of ants at 120 and wait longer in order to find a solution to the problem.

Table 5

Profit in dependence on number of ants for a number of cycles equal to 300

n. of ants	40	60	80	100	120
AKA1	835.0	834.8	834.9	835.0	835.7
AKA2	836.5	837.2	837.0	836.9	836.9
AKA3	834.6	835.2	835.7	834.8	835.7
p. of reference	837.2	837.2	837.2	837.2	837.2

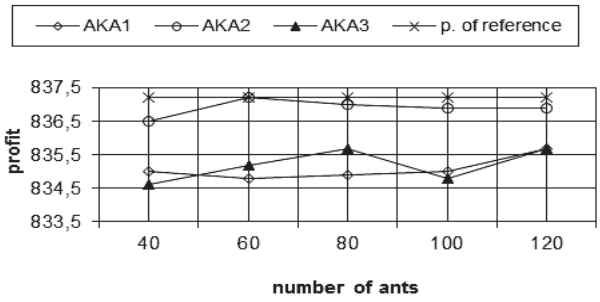

Fig. 5. Profit in dependence on number of ants for a number of cycles equal to 300

The next experiment concerns a profit in dependence on the knapsack load capacity. The number of cycles was set at 300, the number of ants to 80, the evaporation rate to 0.95, the number of objects to 300, the knapsack load capacity consecutively to (1000, 2000, 3000, 4000, 5000). A profit z_i and a weight w_i were randomly generated for each object o_i in the range <1, 10> and <1, 100>, respectively. Thus it was possible to generate 1000 different objects (w_i, z_i). Average values were obtained from 10 measurements and are shown in Table 6 and in Fig. 6. The AKA2 algorithm yields higher profits than the other two algorithms and its superiority is applicable for all knapsack load capacity values.

Table 6

Profit and differences in profit for variable load knapsack capacity

Load capacity	1000	2000	3000	4000	5000
AKA1	466.7	664.1	817.2	946.4	1058.3
AKA2	467.7	665.2	817.7	947.0	1059.7
AKA3	466.6	664.1	816.6	946.3	1058.4
AKA1-AKA3	0.1	0.0	0.6	0.1	–0.1
AKA2-AKA3	1.1	1.1	1.1	0.7	1.3
AKA3-AKA3	0.0	0.0	0.0	0.0	0.0

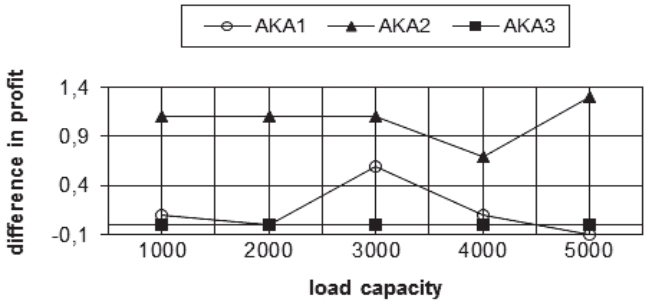

Fig. 6. Differences in profit for variable knapsack load capacity

The following experiment concerns profit and the number of objects. The number of cycles was set at 300, the number of ants at 80, the evaporation rate at 0.95, the knapsack load capacity at 3000 and the number of objects consecutively at (100, 200, 300, 400, 500). A profit z_i and the weight w_i were randomly generated for each object o_i in the range <1, 10> and <1, 100>, respectively. It was possible to generate 1000 different objects (w_i, z_i). Average values were obtained from 10 measurements – these are shown in Table 7 and in Fig. 7. The AKA2 algorithm yields higher profits than the two other algorithms and rises when the number of objects rises.

Table 7

Profits for different numbers of objects

n. of objects	100	200	300	400	500
AKA1	482.5	672.8	817.2	936.3	1048.5
AKA2	482.5	673.0	817.7	938.6	1051.7
AKA3	482.6	672.6	816.6	937.1	1048.6
AKA1-AKA1	0.0	0.0	0.0	0.0	0.0
AKA2-AKA1	0.0	0.2	0.5	2.5	3.2
AKA1-AKA1	0.1	–0.2	–0.6	0.8	0.1

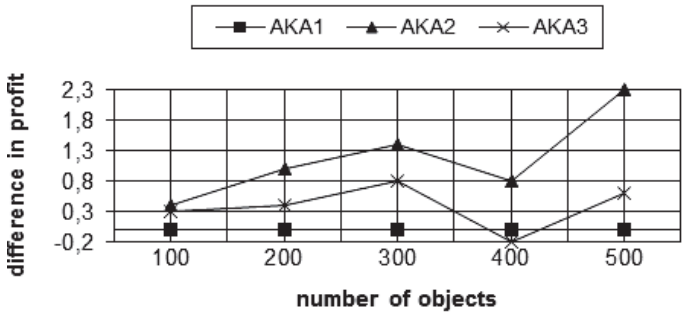

Fig. 7. Differences in profit for different entry numbers of objects

The last (but not least) experiment concerns profit and the number of objects. The number of cycles was set at 300, the number of ants at 80, the evaporation rate at 0.95, the knapsack load capacity at 3000 and the number of objects consecutively at (100, 200, 300, 400, 500). A profit z_i and the weight w_i were randomly generated for each object o_i in the range <1, 10> and <1, 10>, respectively. Thus, it was possible to generate 100 different objects (w_i, z_i), so that objects were generated with the same characteristics (w_i, z_i). Average values were obtained from 10 measurements for each different number of cycles – these values are shown in Table 8 and in Fig. 8. The AKA2 algorithm yields higher profits than the other two algorithms and rises when the number of objects rises. Objects with repeated characteristics (w_i, z_i) do not influence the result of the experiment.

Table 8

Profits for different entry numbers of objects

n. of objects	100	200	300	400	500
AKA1	458.6	635.4	750.0	842.9	925.2
AKA2	459.0	636.4	751.4	843.7	927.5
AKA3	458.9	635.8	750.8	842.7	925.8
AKA1-AKA1	0.0	0.0	0.0	0.0	0.0
AKA2-AKA1	0.4	1.0	1.4	0.8	2.3
AKA3-AKA1	0.3	0.4	0.8	–0.2	0.6

The last experiment concerns a profit and a number of objects. The number of cycles was set at 300, the number of ants at 80, the evaporation rate at 0.95, the knapsack load capacity at 3000 and the number of objects consecutively at (100, 200, 300, 400, 500). A profit z_i and the weight w_i were randomly generated for each object o_i in the range <1, 100> and <1, 100> respectively. Thus, it was possible to generate 10,000 different objects (w_i, z_i) and there was almost no chance to generate objects with the same characteristic (w_i, z_i). Average values were obtained from 10 measurements and are shown in Table 9 and in Fig. 9. The AKA2 algorithm yields higher profits than the other two algorithms and rises when the number of objects rises.

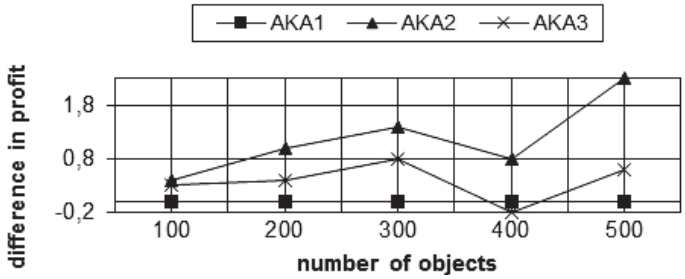

Fig. 8. Differences in profit for different entry numbers of objects

Table 9

Profit for the different entry numbers of objects

n. of objects	100	200	300	400	500
AKA1	4400.8	6465.7	7826.7	8921.9	9890.5
AKA2	4399.3	6466.3	7835.3	8943.5	9914.1
AKA3	4396.9	6465.7	7833.1	8922.9	9898.4
AKA1-AKA1	0.0	0.0	0.0	0.0	0.0
AKA2-AKA1	–1.5	0.6	8.6	21.6	23.6
AKA3-AKA1	–3.9	0.0	6.4	1.0	7.9

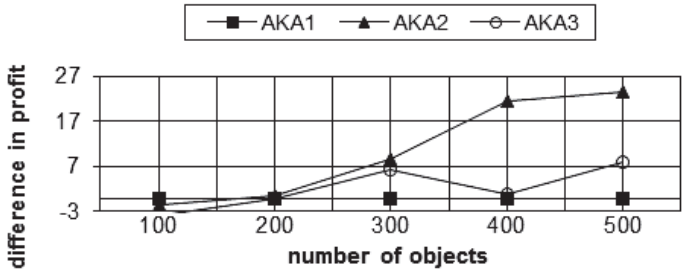

Fig. 9. Differences in profit for different entry numbers of objects

5. Conclusion

The experiments have shown that the AKA2 algorithm can find a solution with a higher total profit than the other two algorithms. Moreover, it can find this solution more rapidly, as it appears from the first experiment, in which the AKA2 algorithm yields a maximal profit after 200 cycles as compared to 600 cycles for the other two algorithms.

References

[1] Kolhe P., Christensen H., *Planning in Logistics: A survey*, PerMIS'10 September 28–30, Baltimore 2010.

[2] Garey M.R., Johnson D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco 1979.

[3] Sysło M., Deo N., Kowalik J., *Algorytmy optymalizacji dyskretnej*, PWN, 1993.

[4] Toth P., *Dynamic programming algorithms for the 0-1 knapsack problem*, Computing 25, 1980, 29-45.

[5] Fayard D., Plateau G., *Algorithm 47: An algorithm for the solution of the 0-1 knapsack problem*, Computing 28, 1982, 269-287.

[6] Koleasr P., *A branch and bound algorithm for the knapsack problem*, Management Science 13, 1967, 723-735.

[7] Fidanova S., *Ant Colony Optimization for Multiple Knapsack Problem and Heuristic Model*, Kluwer Academic Publishers, 2004.

[8] Fidanova S., *Ant Colony Optimization for Multiple Knapsack Problem and Model Bias*, [in:] Margenov S., Vulkov L.G., Wasniewski J. (Eds.), *Numerical Analysis and Its Applications*, LNCS, Vol. 3401, Springer, Berlin Heidelberg, 2005, 280-287.

[9] Fidanova S., *Probabilistic Model of Ant Colony Optimization for Multiple Knapsack Problem*, In Lirkov I., Margenov S., Wasniewski J. (Eds.), LSSC 2007, LNCS 4818, Berlin 2008, 545-552.

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

[11] Fidanova S., *Heuristic for the multiple knapsack problem*, IADIS International Conference on Applied Computing, 2005, 255-260.

[12] Boryczka U., *Ants and Multiple Knapsack Problem*, 6th International Conference on Computer Information Systems and Industrial Management Applications (CISIM'07), 2007, IEEE Computer Society, No. P2894, 2007, 149-154.

[13] Ke L., Feng Z., Ren Z., Wei X., *An ant colony optimization approach for the multi-dimensional knapsack problem*, Journal of Heuristics, Vol. 16, No. 1, 2010, 65-83.

[14] Shahrear I., Faizul B., Sohel R., *Solving the Multidimensional Multi-choice Knapsack Problem with the Help of Ants*, M. Dorigo et al. (Eds.), ANTS 2010, LNCS 6234, Berlin 2010, 312-323.

[15] 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, Los Alamitos, 2007, 10-16.

[16] Leguizamon G., Michalewicz Z., *A new version of ant system for subset problems*; [in:] Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999, Vol. 2, 1999, 1458-1464.

[17] Dorigo M., Caro D.G., Gambardella L.M., *Ant algorithms for discrete optimization*, Artificial Life, Vol. 5, No. 2, 1999, 137-172.