A Percentile Transition Ranking Algorithm Applied to Knapsack Problem

José García[1,2(✉)], Broderick Crawford[2], Ricardo Soto[2], and Gino Astorga[2,3]

[1] Centro de Investigación y Desarrollo Telefónica, 7500961 Santiago, Chile
joseantonio.garcia@telefonica.com

[2] Pontificia Universidad Católica de Valparaíso, Valparaíso, Chile
{broderick.crawford,ricardo.soto}@pucv.cl

[3] Universidad de Valparaíso, 2361864 Valparaíso, Chile
gino.astorga@uv.cl

Abstract. The binarization of Swarm Intelligence continuous metaheuristics is an area of great interest in operational research. This interest is mainly due to the application of binarized metaheuristics to combinatorial problems. In this article we propose a general binarization algorithm called Percentile Transition Ranking Algorithm (PTRA). PTRA uses the percentile concept as a binarization mechanism. In particular we will apply this mechanism to the Cuckoo Search metaheuristic to solve the set multidimensional Knapsack problem (MKP). We provide necessary experiments to investigate the role of key ingredients of the algorithm. Finally to demonstrate the efficiency of our proposal, we solve Knapsack benchmark instances of the literature. These instances show PTRA competes with the state-of-the-art algorithms.

Keywords: Combinatorial optimization · Multidimensional knapsack problem · Metaheuristics

1 Introduction

In recent years, the areas of physics and swarm intelligence have generated a large number of algorithms, many of which have been effective and efficient in solving complex optimization problems. Examples of these algorithms are Ant Colony Optimization [8], Firefly Algorithm [17], Gravitational Search Algorithm [14], Cuckoo Search Algorithm [18], Particle Swarm Optimization [9]. Many of these algorithms have the characteristic that the movement of the particles are performed in a continuous space. On the other hand, combinatorial problems arise in many areas of computer science and application domains. For example in protein structure prediction, grouping routing, planning, scheduling and timetabling problems. It is natural to try to apply algorithms inspired by physics and swarm intelligence in these combinatorial problems [4]. In the process of adaptation a series of difficulties arise when moving from continuous spaces to discrete spaces. Examples of these difficulties are spacial disconnect, hamming

R. Silhavy et al. (eds.), *Applied Computational Intelligence and Mathematical Methods*,
Advances in Intelligent Systems and Computing 662, DOI 10.1007/978-3-319-67621-0_11

cliffs, loss of precision and the curse of dimension [12]. This has the consequence that binarizations are not always effective and efficient [11].

In this paper, a general binarization technique called Percentile Transition Ranking Algorithm (PTRA) is proposed to binarize continuous swarm intelligence metaheuristics. The main operator corresponds to the percentile ranking transition operator. This operator performs the binarization using percentiles grouping process and it is complemented with local search and perturbation operators. The main goal of this work corresponds to evaluate our algorithm when dealing with an NP-hard combinatorial optimization problem such as the MKP. To develop the evaluation, we used the metaheuristic Cuckoo Search. The metaheuristic Cuckoo Search was chosen because it is a swarm intelligence continuous metaheuristic that has been widely used in combinatorial problems [5,15].

Experiments were developed that shed light on the contribution of the different operators to the effectiveness of the algorithm. Moreover, our algorithm was compared with recent algorithms that use transfer functions as binarization method. For this purpose we use tests problems from the OR-Library.[1] We compared our framework with the Binary Artificial Algae Algorithm (BAAA) published by [19]. The numerical results show that PTRA achieves highly competitive results.

The remainder of this paper is organized as follows. Section 2 briefly introduces the Knapsack problem. In Sect. 3 we explain the transition ranking binarization algorithm. The results of numerical experiments are presented in Sect. 4. Finally we provide the conclusions of our work.

2 KnapSack Problem

The MKP is a combinatorial problem that has multiple applications in science and engineering. For example capital budgeting and project selection applications. The MKP has also been introduced to model problems like cutting stock, loading problems, allocation of processors in a distributed data processing [7], and delivery in vehicles with multiple compartments [3].

Numerous methods have been developed to solve the MKP. The exact methods were applied in the 80's to solve MKP. They generate a variety of methods including dynamic programming, branch-and-bound, network approach and reduction schemes. The exact methods have made possible the solution of middle size MKP instances. The major drawback of these methods remains the temporal complexity when dealing with large instances. Therefore, many researchers focus on heuristic and meta-heuristic search methods which can produce solutions of good qualities in a reasonable amount of time. In recent years, many bio-inspired and physics based algorithms, such Swarm Optimization [2], Firefly algorithm [1], Binary Black Hole [6] and Binary Fruitfly [16] have been proposed to solve large instances of the MKP.

[1] OR-Library: http://www.brunel.ac.uk/mastjjb/jeb/orlib/mknapinfo.html.

The MKP problem belongs to the class of NP-hard problems. MKP corresponds to a model of resource allocation, whose objective is to select a subset of objects that produce the greatest benefit considering certain capacity constraints. Each object j consumes a different amount of resources in each dimension. Also each object has a profit associated. Formally the MKP can be set as:

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

$$\text{subjected to} \sum_{j=1}^{n} c_{ij} x_j \leq b_i \ , \ i \in \{1, ..., m\} \tag{2}$$

$$\text{with } x_j \in \{0, 1\} \ , \ j \in \{1, ..., n\} \tag{3}$$

where p_j is the profit for the item j, c_{ij} corresponds to the consumption of resources of item j in the dimension i, and b_i is the capacity constraint of each dimension i. The representation of a solution of the problem is modelled naturally in binary form where 0 in the j-th position means that the j item is not included in the Knapsack and 1 indicates that j is included.

3 Percentile Transition Ranking Algorithm

The first step of PTRA corresponds to the initialization of the feasible solutions Sect. 3.2. Once the initialization of the particles is performed, it is consulted if the detention criterion is satisfied. This criterion includes a maximum of iterations. Subsequently, if the criterion is not satisfied, the percentile transition ranking operator is executed (Sect. 3.2). This operator is responsible for performing the iteration of solutions. Once the transitions of the different solutions are made, we compare the resulting solutions with the best solution previously obtained. In the event that a superior solution is found, this replaces the previous one. When a replacement occurs, the new solution is subjected to a local search operator. Finally, having met a number of iterations where there has not been a replacement for the best solution, a perturbation operator is used. The general algorithm scheme is detailed in Fig. 1. In the following subsection we will explain in detail the initialization method, the percentile transition ranking operator and the repair operator. The explanation of the other operators will be left for an extended version.

3.1 Initialization and Element Weighting

PTRA uses a binarization of swarm-intelligence metaheuristics to try to find the optimum. Each of these possible solutions, is generated as follows: First we select an item randomly. Subsequently we consulted the constraints of our problem if there are other elements that can be incorporated. The list of possible elements to be incorporated is obtained, the weight for each of these elements is calculated

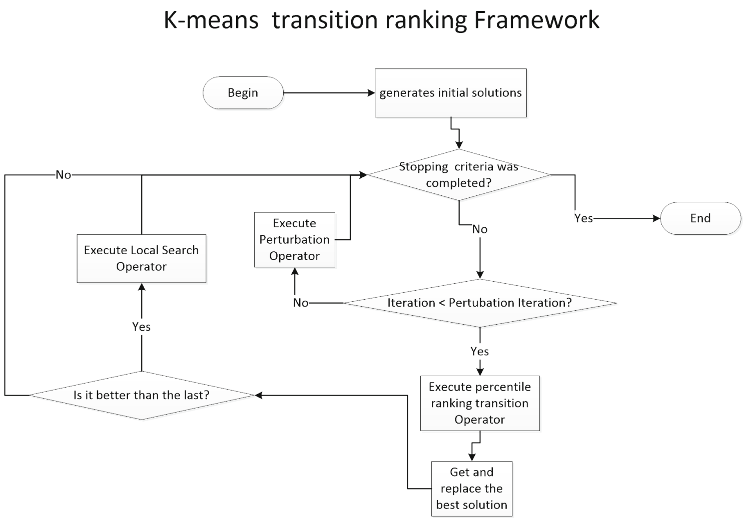

Fig. 1. Flowchart of the percentile transition ranking algorithm.

and the best element is selected. The procedure continues until no more elements can be incorporated. The initialization algorithm is detailed in Fig. 2.

Several techniques were proposed in the literature, to calculate the weight of each element. For example [13] introduced the pseudo-utility in the surrogate duality approach. The pseudo-utility of each variable was given in Eq. 4. The variable w_j is the surrogate multiplier between 0 and 1 which can be viewed as shadow prices of the j-th constraint in the linear programming (LP) relaxation of the original MKP

$$\delta_i = \frac{p_i}{\sum_{j=1}^{m} w_j c_{ij}} \tag{4}$$

Another more intuitive measure is proposed by [10]. This measure is focused on the average occupancy of resources. Its equation is shown in 5.

$$\delta_i = \frac{\sum_{j=1}^{m} \frac{c_{ij}}{m b_j}}{p_i} \tag{5}$$

In this paper, we propose a variation of this last measure focused on the average occupation. However this variation considers the elements that exist in backpacks to calculate the average occupancy. In each iteration depending on the selected items in the solution the measure is calculated again. The equation of this new measure is shown in Eq. 6.

$$\delta_i = \frac{\sum_{j=1}^{m} \frac{c_{ij}}{m(b_j - \sum_{i \in S} c_{ij})}}{p_i} \tag{6}$$

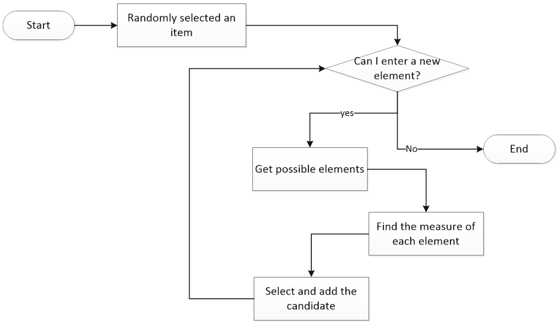

Fig. 2. Flowchart of generation of a new solution.

3.2 Percentile Transition Ranking Operator

Considering that our metaheuristic is a continuous and swarm intelligence. Due to its iterative nature, it needs to update the position of particles at each iteration. When the metaheuristic is continuous, this update is performed in $\mathbb{R}^n$ space. In Eq. 7, the position update is presented in a general form. The x_{t+1} variable represents the x position of the particle at time t+1. This position is obtained from the position x at time t plus a Δ function calculated at time t+1. The function Δ is proper to each metaheuristic and produces values in $\mathbb{R}^n$. For example in Cuckoo Search $\Delta(x) = \alpha \oplus Levy(\lambda)(x)$, in Black Hole $\Delta(x) = \text{rand} \ \times (x_{bh}(t) - x(t))$ and in the Firefly, Bat and PSO algorithms Δ can be written in simplified form as $\Delta(x) = v(x)$.

$$x_{t+1} = x_t + \Delta_{t+1}(x(t)) \tag{7}$$

In the percentile transition ranking operator, we considering the movements generated by the metaheuristic in each dimension for all particles. $\Delta^i(x)$ corresponds to the magnitude of the displacement $\Delta(x)$ in the i-th dimension for the particle x. Subsequently these displacement are grouped using $\Delta^i(x)$, the magnitude of the displacement. This grouping is done using the percentile list. In our case the percentile list used the values {20, 40, 60, 80, 100}.

The percentile operator has as entry the parameters percentile list (percentileList) and the list of values (valuesList). Given an iteration, the list of values corresponds to the magnitude Δ^i of all particles in all dimensions. As a first step the operator uses the valueList and obtains the values of the percentiles given in the percentileList. Later, each value in the valueList is assigned the group of the smallest percentile to which the value belongs. Finally, the list of the percentile to which each value belongs is returned (percentileGroupValue).

A transition probability through the function P_{tr} is assigned to each element of the valueList. This assignment is done using the percentile group assigned to each value (percentileGroupValue). For the case of this study, we particularly use the Step function given in rule 8.

$$P_{tr}(x^i) = \begin{cases} 0.1, & \text{if } x^i \in \text{group } \{0,1\} \\ 0.5, & \text{if } x^i \in \text{group } \{2,3,4\} \end{cases} \tag{8}$$

Afterwards the transition of each particle is performed. In the case of Cuckoo search the rule 9 is used to perform the transition, where $\hat{x}^i$ is the complement of x^i. Finally, each solution is repaired using the repair operator. The algorithm is shown in 1.

$$x^i(t+1) := \begin{cases} \hat{x}^i(t), & \text{if } rand < P_{tg}(x^i) \\ x^i(t), & \text{otherwise} \end{cases} \tag{9}$$

Algorithm 1. Percentile ranking operator

Function percentileRankingTransition(valueList, percentileList)
Input valueList, percentileList
Output percentileGroupValue
percentileValue = getPercentileValue(valueList, percentileList)
for each value in valueList **do**
  percetileGroupValue = getPercentileGroupValue(percentileValue,valueList)
end for
return percetileGroupValue

3.3 Repair Operator

In each movement performed by operators: transition ranking, local search and perturbation, it is possible to generate solutions that are infeasible. Therefore, each candidate solution must be checked and modified to meet every constraint. This verification and subsequent repairing is performed using the measure defined in Sect. 3.1 Eq. 6. The procedure is shown in Algorithm 2. As input the repair operator receives the solution S_{in} to repair, and the output of the repair operator gives the repaired solution S_{out}. As a first step, the repair algorithm asks whether the solution needs to be repaired. In the case that the solution needs repair, a weight is calculated for each element of the solution using the measure defined in Eq. 6. The element of the solution with the largest measure is returned and removed from the solution. This element is named s_{max}. This process is iterated until our solution does not require repair. The next step is to improve the solution. The Eq. 6 is again used for obtaining the element with the smallest measure that meets the constraints s_{min} and add s_{min} to the solution. In the case of absence of elements, empty is returned. The algorithm iterates until there are no elements that satisfy the constraints.

Algorithm 2. Repair Algorithm

Function Repair(S_{in})
Input Input solution S_{in}
Output The Repair solution S_{out}
$S \leftarrow S_{in}$
while needRepair(S) == True **do**
  $s_{max} \leftarrow$ getMaxWeight(S)
  $S \leftarrow$ removeElement(S, s_{max})
end while
state $\leftarrow$ False
while state == False **do**
  $s_{min} \leftarrow$ getMinWeight(S)
  if $s_{min} == \emptyset$ **then**
    state $\leftarrow$ True
  else
    $S \leftarrow$ addElement(S, s_{min})
  end if
end while
$S_{out} \leftarrow S$
return S_{out}

4 Results

4.1 Insight of PTRA Algorithm

In this section we investigated some important ingredients of PTRA to get insight into the behavior of the proposed algorithm. To carry out this comparison the first 10 problems of the set cb.5.250 of the OR library were chosen. The contribution of the percentile transition ranking operator on the final performance of the algorithm was studied. The contribution of the perturbation and local search operators will be developed in an extended version. To compare the distributions of the results of the different experiments we use violin Chart. The horizontal axis X corresponds to the problems, while Y axis uses the measure % - Gap defined in Eq. 10

$$\% - Gap = 100\frac{BestKnown - SolutionValue}{BestKnown} \tag{10}$$

Furthermore, a non-parametric test, Wilcoxon signed-rank test is carried out to determine if the results of PTRA with respect to other algorithms have significant difference or not. The parameter settings and browser ranges are shown in Table 1.

Evaluation of Percentile Transition Ranking Operator. To evaluate the contribution of the percentile transition ranking operator to the final result. We designed a random operator. This random operator executes the transition with

Table 1. Setting of parameters for Cuckoo Search Algorithm.

Parameters	Description	Value	Range
ν	Coefficient for the perturbation operator	3%	[2, 3, 4]
N	Number of Nest	20	[15, 20, 25]
G	Number of percentiles	5	[4, 5, 6]
γ	Step Length	0.01	[0.009,0.01,0.011]
κ	Levy distribution parameter	1.5	[1.4,1.5,1.6]
Iteration Number	Maximum iterations	1000	[1000]

a fixed probability (0.5) without considering the ranking of the particle in each dimension. Two scenarios were established. In the first one the perturbation and local search operators are included. In the second one these operators are excluded. PTRA corresponds to our standard algorithm. *05.pe* is the random variant that includes the perturbation and local search operators. *wpe* corresponds to the version with percentile transition operator without perturbation and local search operators. Finally *05.wpe* describes the random algorithm without perturbation and local search operators.

When we compared the Best Values between PTRA and *05.pe* which are shown in Table 2. PTRA outperforms to *05.pe*. However the Best Values between both algorithms are very close. In the Average comparison, PTRA outperforms *05.pe* in all problems. The comparison of distributions is shown in Fig. 3. We see the dispersion of the *05.pe* distributions are bigger than the dispersions of PTRA. In particular this can be appreciated in the problems 1, 4, 5, 6, and 9. Therefore, the percentile transition ranking operator together with perturbation

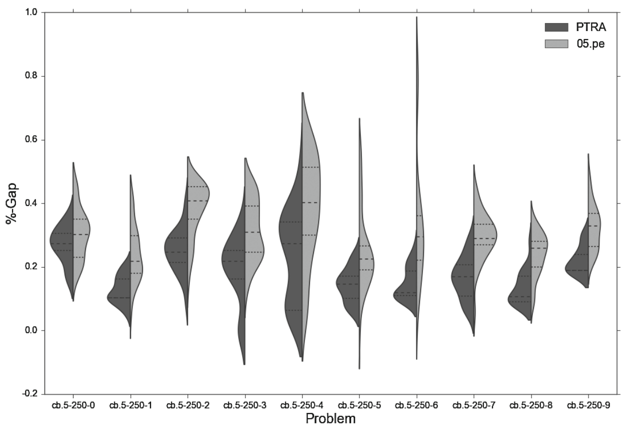

Fig. 3. Evaluation of percentile transition operator with perturbation and Local Search operators

Table 2. Evaluation of percentile transition ranking operator

Set	Best known	Best *05.pe*	Best PTRA	Best *05.wpe*	Best *wpe*	Avg *05.pe*	Avg PTRA	Avg *05.wpe*	Avg *wpe*
cb.5.250-0	59312	59211	59211	59158	59175	59132.1	59151.8	59071.8	59134.5
cb.5.250-1	61472	61435	61435	61409	61409	61324.6	61393.1	61288.3	61380.3
cb.5.250-2	62130	62036	62074	61969	61990	61894.4	61974.4	61801.6	61921.3
cb.5.250-3	59463	59367	59446	59365	59349	59257.8	59331.2	59136.1	59275.6
cb.5.250-4	58951	58914	58951	58883	58914	58725.6	58812.4	58693.6	58761.5
cb.5.250-5	60077	60015	60015	59990	60015	59904.6	59970.4	59837.8	59951.2
cb.5.250-6	60414	60355	60355	60348	60355	60208.2	60324.9	60230.6	60315.7
cb.5.250-7	61472	61436	61436	61407	61401	61290.8	61341.8	61233.9	61343.9
cb.5.250-8	61885	61829	61829	61790	61829	61737.1	61803.4	61644.9	61743.9
cb.5.250-9	58959	58832	58866	58822	58851	58769.1	58786.9	58653.7	58782.8
Average	60413.5	60343	60361.8	60314.1	60328.8	60224.4	60289.0	60159.2	60261.1
p-value							5.27 e-06		1.85 e-05

and local search operators, contribute to the precision of the results. Finally, the PTRA distributions are closer to zero than *05.pe* distributions, indicating that PTRA has consistently better results than *05.pe*. When we evaluate the behaviour of the algorithms through the Wilcoxon test, this indicates that there is a significant difference between the two algorithms.

Our next step is trying to separate the contribution of local search and perturbation operator from the percentile transition operator. For this, we compared the algorithms *wpe* and *05.wpe*.

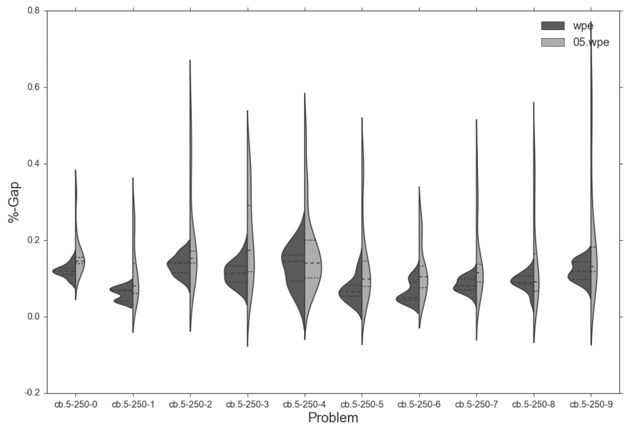

Fig. 4. Evaluation of percentile transition operator without perturbation and Local Search operators

When we check the Best Values shown in the Table 2, we note that *wpe* performs better than *05.wpe* in all problems except 3 and 7. However the results are quite close. In the case of the average indicator, *wpe* outperforms in all problems to *05.wpe*. The Wilcoxon test indicates that the difference is significant. This suggests that *wpe* is consistently better than *05.wpe*. In the violin chart

Table 3. OR-Library benchmarks MKP cb.5.500

Instance	Best known	BAAA best	Avg	PTRA best	Avg	Time(s)	Std
0	120148	120066	120013.7	**120070**	120022.6	343	26.8
1	117879	**117702**	117560.5	117690	**117609.5**	356	52.8
2	121131	120951	120782.9	**121011**	**120918.0**	354	39.5
3	120804	120572	120340.6	**120609**	**120525.7**	436	43.8
4	122319	122231	122101.8	**122280**	**122151.5**	418	48.5
5	122024	121957	121741.8	**121982**	**121874.4**	444	43.7
6	119127	**119070**	118913.4	119000	**118931.0**	449	30.1
7	120568	120472	120331.2	**120487**	**120342.6**	348	63.1
8	121586	121052	120683.6	**121295**	**121196.5**	326	63.5
9	120717	**120499**	120296.3	120485	**120387.0**	317	50.5
10	218428	218185	217984.7	**218251**	**218200.2**	339	32.6
11	221202	220852	220527.5	**220946**	**220863.2**	338	55.6
12	217542	217258	217056.7	**217388**	**217295.9**	315	43.5
13	223560	223510	223450.9	**223526**	**223459.2**	317	41.7
14	218966	218811	218634.3	**218890**	**218814.4**	318	36.9
15	220530	**220429**	**220375.9**	220410	220361.9	384	35.5
16	219989	219785	219619.3	**219885**	**219767.2**	369	60.0
17	218215	**218032**	217813.2	218027	**217956.6**	315	50.6
18	216976	**216940**	**216862.0**	216878	216840.0	354	23.1
19	219719	219602	219435.1	**219622**	**219572.6**	287	30.0
20	295828	295652	295505.0	**295722**	**295662.9**	270	32.2
21	308086	307783	307577.5	**307972**	**307918.0**	319	28.0
22	299796	**299727**	299664.1	299715	**299673.8**	246	22.4
23	306480	**306469**	306385.0	306439	**306393.8**	298	21.9
24	300342	300240	300136.7	**300291**	**300221.6**	273	29.5
25	302571	302492	302376.0	**302503**	**302459.8**	278	22.8
26	301339	301272	301158.0	**301284**	**301257.2**	267	21.5
27	306454	306290	306138.4	**306385**	**306311.2**	234	35.2
28	302828	302769	302690.1	**302771**	**302723.4**	245	29.6
29	299910	299757	299702.3	**299844**	**299773.7**	275	47.1
Average	214168.8	214014.2	213861.9	214055.3	213982.8	327.7	38.7

shown n the Fig. 4 it is further observed that the dispersion of the solutions for the case of *05.wpe* is much larger than in the case of *wpe*. This indicates that the operator percentile transition ranking plays an important role in the precision of the results.

4.2 PTRA Compared with BAAA

In this section we evaluate the performance of our PTRA with the algorithm BAAA developed in [19]. BAAA uses transfer functions as a general mechanism of binarization. In particular BAAA used the tanh $= \frac{e^{\tau|x|}-1}{e^{\tau|x|}+1}$ function to perform the transference. The parameter τ of the tanh function was set to a value 1.5. Additionally, a elite local search procedure was used by BAAA to improve solutions. As maximum number of iterations BAAA used 35000. The computer configuration used to run the BAAA algorithm was: PC Intel Core(TM) 2 dual CPU Q9300@2.5GHz, 4GB RAM and 64-bit Windows 7 operating system. In our PTRA algorithm, the configurations are the same used in the previous experiments. These are described in the Table 1.

The results are shown in Table 3. The comparison was performed for the set cb.5.500 of the OR-library. The results for PTRA were obtained from 30 executions for each problem. In black, the best results are marked for both indicators the Best Value and the Average. In the Best Value indicator, BAAA was higher in eight instances and PTRA in twenty two. In the averages indicator BAAA was higher in two instances, and PTRA in eighteen. We should also note that the standard deviation in most problems was quite low, indicating that PTRA has good accuracy.

5 Conclusions and Future Work

In this article, we proposed a algorithm whose main function is to binarize continuous swarm intelligence metaheuristics. To evaluate the performance of our algorithm, the multidimensional Knapsack problem was used together with the Cuckoo Search metaheuristic. The contribution of the main operator of the algorithm was evaluated, finding that the percentile transition ranking operator contributes significantly to improve the precision of the solutions. Finally, in comparison with state of the art algorithms our algorithm showed a good performance.

As a future works we want to investigate the behaviour of other metaheuristics. Furthermore, the algorithm must be verified with other NP-hard problems. Moreover to simplify the choice of the appropriate configuration, it is important to explore adaptive techniques. From an understanding point of view of how the framework performs binarization, it is interesting to understand how the algorithm alters the properties of exploration and exploitation. Also is interesting to study how the velocities and positions generated by continuous metaheuristics are mapped to positions in the discrete space.

Acknowledgments. Broderick Crawford is supported by grant CONICYT/ FONDECYT/REGULAR 1171243, Ricardo Soto is supported by Grant CONICYT /FONDECYT /REGULAR /1160455, and José García is supported by INF-PUCV 2016.

References

1. Baykasoğlu, A., Ozsoydan, F.B.: An improved firefly algorithm for solving dynamic multidimensional knapsack problems. Expert Syst. Appl. **41**(8), 3712–3725 (2014)
2. Bhattacharjee, K.K., Sarmah, S.P.: Modified swarm intelligence based techniques for the knapsack problem. Appl. Intell. **46**, 158–179 (2016)
3. Chajakis, E., Guignard, M.: A model for delivery of groceries in vehicle with multiple compartments and lagrangean approximation schemes. In: Proceedings of Congreso Latino Ibero-Americano de Investigación de Operaciones e Ingeniería de Sistemas (1992)
4. Crawford, B., Soto, R., Astorga, G., García, J., Castro, C., Paredes, F.: Putting continuous metaheuristics to work in binary search spaces. Complexity **2017**, 19 (2017)
5. García, J., Crawford, B., Soto, R., Carlos, C., Paredes, F.: A k-means binarization framework applied to multidimensional knapsack problem. Appl. Intell., 1–24 (2017)
6. García, J., Crawford, B., Soto, R., García, P.: A multi dynamic binary black hole algorithm applied to set covering problem. In: International Conference on Harmony Search Algorithm, pp. 42–51. Springer (2017)
7. Gavish, B., Pirkul, H.: Allocation of databases and processors in a distributed computing system. Manage. Distrib. Data Process. **31**, 215–231 (1982)
8. Glover, F., Kochenberger, G.A.: The ant colony optimization metaheuristic: Algorithms, applications, and advances. In: Handbook of Metaheuristics, pp. 250–285 (2003)
9. Kennedy, J.: Particle swarm optimization. In: Encyclopedia of Machine Learning, pp. 760–766. Springer (2011)
10. Kong, X., Gao, L., Ouyang, H., Li, S.: Solving large-scale multidimensional knapsack problems with a new binary harmony search algorithm. Comput. Oper. Res. **63**, 7–22 (2015)
11. Lanza-Gutierrez, J.M., Crawford, B., Soto, R., Berrios, N., Gomez-Pulido, J.A., Paredes, F.: Analyzing the effects of binarization techniques when solving the set covering problem through swarm optimization. Expert Syst. Appl. **70**, 67–82 (2017)
12. Leonard, B.J., Engelbrecht, A.P., Cleghorn, C.W.: Critical considerations on angle modulated particle swarm optimisers. Swarm Intell. **9**(4), 291–314 (2015)
13. Pirkul, H.: A heuristic solution procedure for the multiconstraint zero? one knapsack problem. Naval Res. Logist. **34**(2), 161–172 (1987)
14. Rashedi, E., Nezamabadi-Pour, H., Saryazdi, S.: Gsa: a gravitational search algorithm. Inf. Sci. **179**(13), 2232–2248 (2009)
15. Soto, R., Crawford, B., Olivares, R., Barraza, J., Figueroa, I., Johnson, F., Paredes, F., Olguín, E.: Solving the non-unicost set covering problem by using cuckoo search and black hole optimization. Natural Comput., January 2017
16. Wang, L., Zheng, X., Wang, S.: A novel binary fruit fly optimization algorithm for solving the multidimensional knapsack problem. Knowl.-Based Syst. **48**, 17–23 (2013)

17. Yang, X.-S.: Firefly algorithm, stochastic test functions and design optimisation. Int. J. Bio-Inspired Comput. **2**(2), 78–84 (2010)
18. Yang, X.-S., Deb, S.: Cuckoo search via lévy flights. In: 2009 World Congress on Nature & Biologically Inspired Computing, NaBIC 2009, pp. 210–214. IEEE (2009)
19. Zhang, X., Changzhi, W., Li, J., Wang, X., Yang, Z., Lee, J.-M., Jung, K.-H.: Binary artificial algae algorithm for multidimensional knapsack problems. Appl. Soft Comput. **43**, 583–595 (2016)