Efficiency groups and a randomized heuristic repair for the 0-1 multidimensional knapsack problem

Jean P. Martins*

Federal University of Technology Paraná, Academic Department of Informatics, Pato Branco–PR, Brazil

Abstract

The multidimensional knapsack problem (MKP) is an NP-hard combinatorial optimization problem whose solution consists of determining a subset of items of maximum total profit that does not violate capacity constraints. Due to its hardness, large-scale MKP instances are usually a target for metaheuristics, a context in which effective feasibility maintenance strategies are crucial. In 1998, Chu and Beasley proposed one of the most successful repair heuristics which is still employed in more recent metaheuristics. However, due to its deterministic nature, the diversity of solutions such heuristic provides is not sufficient for long runs. As a result, the search ceases to find new solutions after a while. This paper proposes an efficiency-based randomization strategy for the heuristic repair which increases the variability of the repaired solutions, without deteriorating quality and improves the overall results. We compared our randomized heuristic repair against the original one in 270 OR-LIBRARY instances, with better results found in runtime and solution quality.

Keywords: knapsack problem, heuristic repair, evolutionary algorithms, metaheuristics

1. Introduction

The *Multidimensional Knapsack Problem* (MKP) is a well-known strongly NP-Hard combinatorial optimization problem (Freville, 2004; Puchinger et al., 2010). To solve an instance of the MKP one must choose choose, from a set of n items, a subset that yields the maximum total profit. Every item chosen contributes with a profit $p_j > 0$ $(j = 1, \ldots, n)$ but also consumes w_{ij} from the amount of resources available $r_i > 0$ $(i = 1, \ldots, m)$. Therefore, a solution is *feasible* only if the total of resources it consumes do not surpass the amount available. By representing solutions as n-dimensional binary vectors, the MKP is formulated as:

$$\begin{aligned} \max \quad & f(\boldsymbol{x}) = \sum_{j=1}^{n} p_j x_j, \\ \text{subject to} \quad & \sum_{j=1}^{n} w_{ij} x_j \leq r_i \qquad i = 1, \ldots, m, \\ & x_j \in \{0, 1\}, \qquad\quad j = 1, \ldots, n. \end{aligned}$$

The MKP is considerably harder than its unidimensional counterpart, not admitting an efficient polynomial-time approximation scheme even for $m = 2$ (Kulik & Shachnai, 2010). Furthermore, MKP hardness increases with m, and larger instances still cannot be effectively solved to optimality (Mansini & Speranza, 2012). Due to these limitations, metaheuristics have been the most successful alternatives to solve the largest instances

*Corresponding author
Email address: jeanmartins@utfpr.edu.br (Jean P. Martins)

available at the OR-LIBRARY[1]. However, most of the best known solutions for such instances were attained by hybrid methods encompassing tabu search, evolutionary algorithms, linear programming and branch and bound (Puchinger et al., 2010).

Chu & Beasley (1998) results can be considered a landmark regarding metaheuristics for the MKP. The method they proposed, which we refer to as *Chu & Beasley Genetic Algorithm* (CBGA), was the first metaheuristic to deal with large-scale MKP instances and influenced many posterior analysis, see for example Gottlieb (2000, 2001) and Tavares et al. (2006, 2008). Additionally, the heuristic repair employed by CBGA has proved itself as one of the most effective alternatives for feasibility maintenance and it is commonly employed by other metaheuristics, see Kong et al. (2008),Wang et al. (2012b,a), Martins et al. (2013); Martins & Delbem (2013), Chih et al. (2014) and Azad et al. (2014).

Several of CBGA results have been improved or matched by many algorithms. Amongst them we mention Vasquez & Vimont (2005) with an hybrid tabu-search/linear programming, Mansini & Speranza (2012) with a core-based branch-and-bound and recently Lai et al. (2018) with an hybrid of tabu-search and evolutionary algorithm; all of which have improved the best known solutions for some instances of the OR-LIBRARY.

2. Heuristic repair and efficiencies

Many of the metaheuristics produce during their search process infeasible candidate solutions. In the context of the MKP, an infeasible solution is a subset of items whose resource consumption surpasses the available amount, i.e., they violate knapsack-capacity constraints. The only way to make an infeasible solution feasible consists of removing some of the items it contains. However, if an item is removed and the solution becomes feasible, it might be the case that another item would now fit in the knapsack.

Therefore, a heuristic repair for the MKP would consist of two phases: (1) DROP items, (2) ADD items. Naturally, if we remove items of high profit and low resource consumption and add items of low profit and high resource consumption, the resulting feasible solution will yield a low total profit. Therefore, a heuristic repair must consider the remotion and addition of items according to an ordering that favors highly profitable solutions. The algorithm 1 formalizes these ideas, considering a candidate solution $\boldsymbol{x}$ and an ordering $\mathcal{O}$, which orders the items from the best $\mathcal{O}_1$ to the worst $\mathcal{O}_n$ according to some criterion.

Algorithm 1 HEURISTIC-REPAIR($\boldsymbol{x}$, $\mathcal{O}$)

```
1: for all j = O_n, ..., O_1 do                          ▷ DROP items:
2:     if x is feasible then break
3:     Remove j from the knapsack
4: for all j = O_1, ..., O_n do                          ▷ ADD items:
5:     if j fits in the knapsack then
6:         Add j to the knapsack
```

In order to the Algorithm 1 produce high-quality solutions, the ordering employed must reflect the *efficiency* of the items. For example, in the unidimensional knapsack problem, every item j has a profit p_j and consumes w_j from the resource available. Therefore, by considering the items in non-increasing order of $e_j = p_j/w_j$, we would favor highly valuable items of small dimensions during the heuristic repair. Such efficiencies are also desirable for the multidimensional case. However, due to the multiple resources w_{ji} existent in this case, an aggregation must be defined to compose the denominator. The usual approach consists of a weighted-sum of the resources, as defined by Equation (1).

$$e_j = \frac{p_j}{\sum_{i=1}^{m} \lambda_i \cdot w_{ij}}, \forall j = 1, \ldots, n. \tag{1}$$

According to an extensive experimental analysis, Puchinger et al. (2010) showed that the most effective weights λ_i, $\forall i = 1, \ldots, m$, would be optimal solutions for the dual linear programming relaxation of the

[1] http://people.brunel.ac.uk/~mastjjb/jeb/info.html

MKP. Indeed, that was the weight-vector employed by Chu & Beasley (1998) in their experiments and many posterior studies corroborated their choice. From now on, we refer as e_j^{dual} to the efficiencies defined by (1), and as $\mathcal{O}^{\text{dual}}$ to the ordering of the knapsack items that follows from them.

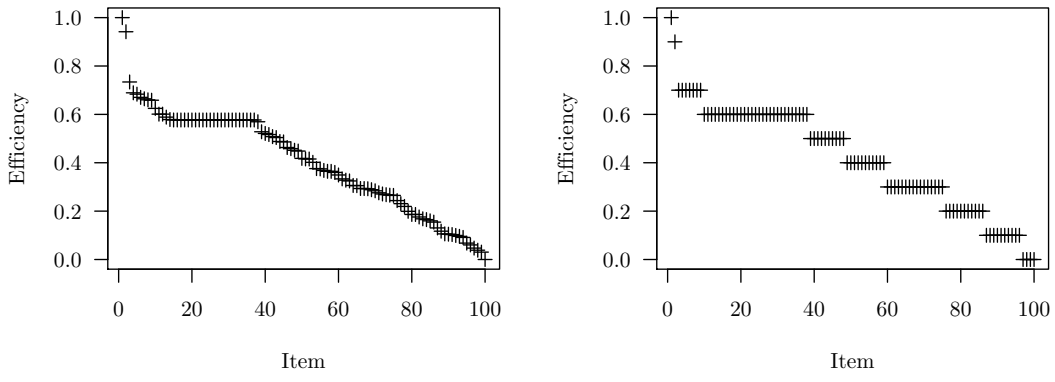
3. Efficiency groups and randomization

Efficiencies provide reasonable estimates of how probably the items belong to optimal solutions. In other words, if an item has a high efficiency value, it will probably be present in an optimal solution. On the other hand, if it has low efficiency value, it will most likely be rejected. Unfortunately, the number of items whose efficiencies are neither high or low enough grows with the number of knapsack constraints. As a result, for large-scale MKP instances, many of them will have similar efficiencies (*core items*), and it will be challenging to decide if they should, or not, compose an optimal solution.

Such limitation is not taken into account by the heuristic repair of CBGA, instead, it only relies on the initial ordering $\mathcal{O}^{\text{dual}}$. Therefore, the CBGA struggles to decide the *core items*, as shown by Martins et al. (2014). The idea we discuss insofar hypothesizes that by exploring the space of orderings along with the search for better solutions, we might increase the chances of finding better solutions. The question then, would be how to perform such exploration without completely destroying the valuable information provided by the efficiencies. To give a possible answer for this question, we introduce the notion of *efficiency groups*.

Assume, for the sake of argument, that we choose two items randomly and swap their positions in the ordering. The quality of the solutions produced by the heuristic repair should improve or deteriorate? Well, if one of the items has a high efficiency-value whereas the other has a low efficiency-value, by exchanging their positions the quality of the solutions produced are likely to decrease, since we would be destroying the information provided by the efficiencies. Therefore, to enable a non-disruptive modification of the ordering, only items of similar efficiencies should be allowed to swap positions.

To better understand this idea, observe the Figure 1(a), which shows us the efficiencies[2] for the instance `30.100-00` (from OR-LIBRARY, $n = 100$ items and $m = 30$ resources) in non-decreasing order. Aside from a few highly efficient items at the left-hand side of the figure, there are two more patterns. First, there is a plateau in which all the items share roughly the same efficiency (*core items*). Second, all the consecutive items share close but non-increasing efficiencies. Since these values are used to define an ordering for the items, it is reasonable to question the ordering of items whose efficiencies are too close.



(a) Original efficiencies

(b) Efficiencies rounded to one decimal case.

Figure 1: Efficiency groups produced by rounding the original efficiencies to one decimal case.

Figure 1(b) shows us the same efficiencies but now rounded to the first decimal case ($d = 1$). Rounding, as a pre-processing phase of the efficiencies, makes evident other plateaus, each of them indicating a group

[2]Normalized to the $[0, 1]$ interval.

of items whose efficiencies are approximately the same, i.e. an *efficiency group*. Therefore, the number of decimal cases kept during the rounding allows us to roughly control the sizes of the groups. In summary, an *efficiency group* is a subset of items of cardinality higher than one whose efficiencies are equal. Given that an efficiency group contains only items of similar efficiencies, the randomization of their order will not lead to massive deterioration of the heuristic information provided by the efficiencies. Therefore, efficiency groups enable an effective exploration of the space of orderings during the search for solutions.

4. Methodology

In this paper, we evaluate the possible benefits of two simple randomizing operators that modify the positioning of items in a efficiency group given an ordering $\mathcal{O}^{\text{dual}}$.

Random-group swap (RG-SWAP). Randomly chooses an efficiency group, swap the positions of two randomly chosen items in the group.

Random-group shuffle (RG-SHUFFLE). Randomly chooses an efficiency group, shuffles the positions of all its items.

To verify the effectiveness of these operators we must first decide how frequently randomization should take place. Since the CBGA is an effective algorithm to find high-quality solutions fast, the use of intensive randomization during the first generations would slow-down its progress. To avoid this drawback, we can associate the exploration of new orderings with the number of improving solutions produced by the heuristic repair during a generation. In other words, whenever an ordering ceases to produce improving solutions, randomization will be activated. Next section, describes a modified version of the CBGA that implements such strategy.

4.1. Chu & Beasley GA with efficiency groups

The CBGA uses a direct representation (binary strings), binary tournament selection, standard uniform crossover, random mutation of two bits and the heuristic repair (see Algorithm 3).

Algorithm 2 `CBGA-newsolution(`$\mathcal{P}$`)`

1: Select $\boldsymbol{x}^1 \in \mathcal{P}$ by a binary tournament
2: Select $\boldsymbol{x}^2 \in \mathcal{P}$ by a binary tournament
3: $\boldsymbol{x} \leftarrow$ `uniformCrossover(`$\boldsymbol{x}^1$`,` $\boldsymbol{x}^2$`)`
4: `flipTwoBits(`$\boldsymbol{x}^3$`)`
5: **return** $\boldsymbol{x}$.

CBGA is a steady-state algorithm, which means it generates and evaluates one solution at the time. However, to control the activation of the randomization of efficiency orderings, we must have access to the number of improving solutions produced. Therefore, we redesigned the CBGA regarding generations, every generation consisting of N attempts to produce improving solutions (where N is the population size). Each of such attempts produces one solution $\boldsymbol{x}$ and applies the heuristic repair to it. The resulting solution is an improvement if it is both unique and better than the worst solution in $\mathcal{P}$ (see Algorithm 3).

The modified version of the CBGA that supports the randomization of efficiency groups is defined by Algorithm 4, which we will refer to as CBGA^d_ϵ, where d refers to the number of decimal cases used to round the efficiencies. The first step is to sort the indexes of the items in non-increasing order according to e^{dual}. Next, the efficiency groups are identified and stored in g. A population $\mathcal{P}$ of N random candidate solutions is generated. All candidate solutions undergo heuristic repair and the search starts. At every generation, if no improving solutions are produced, the randomization of the ordering takes place (where `rand-ordering` is a placeholder for RG-SWAP or RG-SHUFFLE).

Algorithm 3 `CBGA-generation`$(\mathcal{P}, N, \mathcal{O}^{\text{dual}})$

$m \leftarrow 0$
improvements $\leftarrow 0$
while $m < N$ **do**
    $\boldsymbol{x} \leftarrow$ `CBGA-newsolution(`$\mathcal{P}$`)`
    `heuristic-repair(`$\boldsymbol{x}$, $\mathcal{O}^{\text{dual}}$`)`
    if $\boldsymbol{x}$ is unique and better than the worst solution in $\mathcal{P}$ **then**
        $\boldsymbol{x}$ substitutes the worst solution in $\mathcal{P}$
        improvements $\leftarrow$ improvements $+1$
    $m \leftarrow m + 1$
return improvements.

Algorithm 4 $\text{CBGA}_{\epsilon}^{d}$

Require: The population size N
$\mathcal{O}^{\text{dual}} \leftarrow$ `sort(`$\{1, \ldots, n\}$`,` $e^{\texttt{dual}}$`)`
$g \leftarrow$ `get-efficiency-groups(`$\mathcal{O}^{\text{dual}}, d$`)`
Generates N random candidate solutions in $\mathcal{P}$
`heuristic-repair(`$\boldsymbol{x}$, $\mathcal{O}^{\text{dual}}$`)`, $\forall \boldsymbol{x} \in \mathcal{P}$
while Stop criteria not met **do**
    $m \leftarrow$ `CBGA-generation(`$\mathcal{P}$, N, $\mathcal{O}^{\text{dual}}$`)`
    if $m \leq 0$ **then** `rand-ordering(`$\mathcal{O}^{\text{dual}}$, g`)`
return the best solution in $\mathcal{P}$.

4.2. Experiments

The instances provided by Chu & Beasley (1998) have been widely used in the literature and will be also the basis for our experiments. The set comprises instances of size $n \in \{100, 250, 500\}$, for each size there are instances of dimension $m \in \{5, 10, 30\}$ and tightness ratio $\alpha \in \{0.25, 0.5, 0.75\}$ (where $\alpha = c_i / \sum_{j=1}^{n} w_{ij}$). For each tuple (n, m, α) there are 10 instances, totalizing 270 different instances. All these instances are available in the OR-LIBRARY web page[3].

5. Results and Discussion

This section describes and compare the results of both scenarios previously defined. We first show the average performance of each algorithm in all 270 Chu and Beasley instances. The averages were computed considering the whole set of instances, instead of separating instances according to their α. This decision enables us to see the big picture of average performances of the algorithms and, since the same instances were used for all algorithms, the averages provide a relevant estimate for the performance measures used.

6. Conclusion

References

Azad, M. A. K., Rocha, A. M. A., & Fernandes, E. M. (2014). Improved binary artificial fish swarm algorithm for the 0–1 multidimensional knapsack problems. *Swarm and Evolutionary Computation*, *14*, 66–75. doi:10.1016/j.swevo.2013.09.002.

Chih, M., Lin, C.-J., Chern, M.-S., & Ou, T.-Y. (2014). Particle swarm optimization with time-varying acceleration coefficients for the multidimensional knapsack problem. *Applied Mathematical Modelling*, *38*, 1338–1350. doi:10.1016/j.apm.2013.08.009.

Chu, P., & Beasley, J. (1998). A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, *4*, 63–86. doi:10.1023/A:1009642405419.

[3] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/mknapinfo.html

		Gap					Time(s)				
Instance	Best	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$
5-100-00	24381	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.02	**0.01**	**0.01**	**0.01**	**0.01**
5-100-01	24274	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.01**	0.02	0.02	0.02	0.02
5-100-02	23551	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.52	0.51	**0.43**	0.53	0.62
5-100-03	23534	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	1.37	6.82	**0.93**	9.14	12.2
5-100-04	23991	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.49	**0.34**	0.38	0.43	0.35
5-100-05	24613	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.14	0.16	**0.1**	0.14	0.12
5-100-06	25591	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**
5-100-07	23410	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.44**	0.56	0.47	0.64	0.52
5-100-08	24216	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.13	0.09	**0.08**	0.09	0.12
5-100-09	24411	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.05	0.07	**0.04**	0.05	0.05
5-100-10	42757	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.06	0.06	**0.05**	0.06	0.06
5-100-11	42545	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	1.4	0.52	1.73	**0.34**	0.49
5-100-12	41968	✓**0**	0.03	✓**0**	✓**0**	✓**0**	4.32	93.96	**3.27**	107.61	91.44
5-100-13	45090	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	1.51	1.08	1.71	1.35	**0.96**
5-100-14	42218	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.02	**0.01**	0.02	**0.01**	**0.01**
5-100-15	42927	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.03	**0.02**	**0.02**	**0.02**	**0.02**
5-100-16	42009	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.01**	**0.01**	**0.01**	**0.01**	0.02
5-100-17	45020	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.04**	**0.04**	0.05	0.05	0.06
5-100-18	43441	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.32	0.19	0.23	**0.17**	0.34
5-100-19	44554	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.75	0.94	**0.52**	0.85	0.83
5-100-20	59822	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.03**	**0.03**	**0.03**	**0.03**	**0.03**
5-100-21	62081	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.04	**0.03**	0.04	**0.03**	**0.03**
5-100-22	59802	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.71	1.04	**0.53**	1.09	1.12
5-100-23	60479	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.18	0.1	0.12	**0.08**	0.36
5-100-24	61091	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.18	0.12	**0.1**	**0.1**	**0.1**
5-100-25	58959	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.26	0.49	**0.24**	0.62	0.49
5-100-26	61538	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.25	0.28	**0.19**	0.29	0.32
5-100-27	61520	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.12	0.17	**0.09**	0.14	0.15
5-100-28	59453	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.07	**0.06**	0.07	**0.06**	**0.06**
5-100-29	59965	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.61	2.34	**0.37**	2.85	2.41
5-100-*	-	**0**	0.03	**0**	**0**	**0**	14.1	110.09	**11.87**	126.83	113.33

Freville, A. (2004). The multidimensional 0–1 knapsack problem: An overview. *European Journal of Operational Research*, *155*, 1–21. doi:10.1016/S0377-2217(03)00274-1.

Gottlieb, J. (2000). Permutation-based Evolutionary Algorithms for Multidimensional Knapsack Problems. In *Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 1* SAC '00 (pp. 408–414). New York, NY, USA: ACM. doi:10.1145/335603.335866.

Gottlieb, J. (2001). On the Feasibility Problem of Penalty-Based Evolutionary Algorithms for Knapsack Problems. In E. Boers (Ed.), *Applications of Evolutionary Computing* (pp. 50–59). Springer Berlin Heidelberg volume 2037 of *Lecture Notes in Computer Science*. doi:10.1007/3-540-45365-2_6.

Kong, M., Tian, P., & Kao, Y. (2008). A new ant colony optimization algorithm for the multidimensional Knapsack problem. *Computers & Operations Research*, *35*, 2672–2683. doi:10.1016/j.cor.2006.12.029.

Kulik, A., & Shachnai, H. (2010). There is no EPTAS for two-dimensional knapsack. *Information Processing Letters*, *110*, 707–710. doi:10.1016/j.ipl.2010.05.031.

Lai, X., Hao, J.-K., Glover, F., & L, Z. (2018). A two-phase tabu-evolutionary algorithm for the 01 multidimensional knapsack problem. *Information Sciences*, *436-437*, 282 – 301. doi:10.1016/j.ins.2018.01.026.

Mansini, R., & Speranza, M. G. (2012). Coral: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, *24*, 399–415. doi:10.1287/ijoc.1110.0460.

Martins, J. P., Bringel Neto, C., Crocomo, M. K., Vittori, K., & Delbem, A. C. B. (2013). A Comparison of Linkage-learning-based Genetic Algorithms in Multidimensional knapsack Problems. In *Evolutionary Computation (CEC), 2013 IEEE Congress on* (pp. 502–509). volume 1. doi:10.1109/CEC.2013.6557610.

Martins, J. P., & Delbem, A. C. B. (2013). The influence of linkage-learning in the linkage-tree GA when solving multidimensional knapsack problems. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference* GECCO '13 (pp. 821–828). New York, NY, USA: ACM. doi:10.1145/2463372.2463476.

Martins, J. P., Longo, H., & Delbem, A. C. (2014). On the effectiveness of genetic algorithms for the multidimensional

		Gap					Time(s)				
Instance	Best	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$
5-250-00	59312	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**1.52**	3.8	1.65	4.13	4.48
5-250-01	61472	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	17.64	**8.16**	23.18	12.36	11.04
5-250-02	62130	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.77	0.13	1.78	**0.12**	**0.12**
5-250-03	59463	0.67	✓**0.33**	0.67	1.23	1.23	511.72	304.16	512.64	450.03	**257.93**
5-250-04	58951	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	8.47	3.58	6.83	**1.83**	12.11
5-250-05	60077	✓**0**	2.03	✓**0**	1.53	4.57	**40**	348.08	51.39	578.15	648.84
5-250-06	60414	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	4.12	**1.79**	2.39	2.59	2.29
5-250-07	61472	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	8.14	7.24	**6.63**	8.33	10.18
5-250-08	61885	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	1.74	4.2	**1.25**	7.14	4.74
5-250-09	58959	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	29.56	13.7	40.31	16.18	**12.45**
5-250-10	109109	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	112.25	5.06	121.42	13.44	**4.12**
5-250-11	109841	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	9.24	9.82	**7.01**	10.33	12.4
5-250-12	108508	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	10.08	38.87	**9.66**	30.76	49.93
5-250-13	109383	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	1.31	**0.33**	2.51	0.35	0.65
5-250-14	110720	✓**0**	✓**0**	0.07	✓**0**	✓**0**	150.87	68.98	204.93	**55.12**	165.01
5-250-15	110256	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**19.78**	29.65	29.53	33.89	20.45
5-250-16	109040	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	74.01	**63.1**	78.2	147.72	82.19
5-250-17	109042	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	175.43	52.12	200.21	**40.15**	108.13
5-250-18	109971	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	24.65	65.43	**20.12**	64.72	68.02
5-250-19	107058	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	2.98	6.77	**2.33**	4.27	10.83
5-250-20	149665	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	33.42	50.73	**20.06**	42.5	39.77
5-250-21	155944	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**38.82**	53.4	42.05	65.86	54.39
5-250-22	149334	0.07	✓**0**	✓**0**	✓**0**	✓**0**	457.37	269.84	447.76	222.61	**178.5**
5-250-23	152130	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**10.43**	25.57	11.25	22.25	16.23
5-250-24	150353	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	8.69	20.28	**3.33**	18.86	18.75
5-250-25	150045	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.88	0.68	**0.45**	0.55	0.97
5-250-26	148607	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	3.34	4.15	**1.5**	4.15	4.4
5-250-27	149782	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	118.66	**38.25**	131.4	44.9	56.67
5-250-28	155075	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	50.15	65.33	**43.38**	55.11	65.77
5-250-29	154668	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	9.35	**4.26**	19.23	5.98	9.58
5-250-*	-	**0.74**	2.36	**0.74**	2.76	5.8	1935.39	**1567.46**	2044.38	1964.38	1930.94

knapsack problem. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation* GECCO Comp '14 (pp. 73–74). New York, NY, USA: ACM. doi:10.1145/2598394.2598477.

Puchinger, J., Raidl, G. R., & Pferschy, U. (2010). The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, *22*, 250–265.

Tavares, J., Pereira, F., & Costa, E. (2006). The Role of Representation on the Multidimensional Knapsack Problem by means of Fitness Landscape Analysis. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on* (pp. 2307–2314). doi:10.1109/CEC.2006.1688593.

Tavares, J., Pereira, F., & Costa, E. (2008). Multidimensional Knapsack Problem: A Fitness Landscape Analysis. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, *38*, 604–616. doi:10.1109/TSMCB.2008.915539.

Vasquez, M., & Vimont, Y. (2005). Improved results on the 0–1 multidimensional knapsack problem. *European Journal of Operational Research*, *165*, 70–81. doi:10.1016/j.ejor.2004.01.024.

Wang, L., Fu, X., Mao, Y., Menhas, M. I., & Fei, M. (2012a). A novel modified binary differential evolution algorithm and its applications. *Neurocomputing*, *98*, 55–75. doi:10.1016/j.neucom.2011.11.033. Bio-inspired computing and applications (LSMS-ICSEE ' 2010).

Wang, L., Wang, S.-y., & Xu, Y. (2012b). An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem. *Expert Systems with Applications*, *39*, 5593–5599. doi:10.1016/j.eswa.2011.11.058.

		Gap					Time(s)				
Instance	Best	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$
5-500-00	120148	1.6	✓**0**	0.4	✓**0**	✓**0**	1022.82	164.54	649.72	**108.57**	180.56
5-500-01	117879	✓**8.87**	11.5	10.07	12.23	13	**2152.67**	2273.23	2216.56	2525.86	2593.87
5-500-02	121131	1.53	2.27	1.93	0.6	✓**0.47**	1644.38	1480.67	1427.08	**862.14**	883.66
5-500-03	120804	2.67	✓**0**	2	✓**0**	0.37	1558.2	**398.25**	1425.91	423.94	684.89
5-500-04	122319	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	14.15	7.61	26.17	5.89	**5.06**
5-500-05	122024	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	118.57	348.7	**105.36**	363.66	402.47
5-500-06	119127	1.93	✓**0**	✓**0**	✓**0**	0.33	1419.43	428.97	874.26	**347.41**	991.8
5-500-07	120568	✓**0**	✓**0**	✓**0**	✓**0**	0.57	301.58	**91.02**	189.63	91.19	444.71
5-500-08	121586	5.93	4.53	7.33	6.23	✓**3.73**	1846.37	1426.69	1831.65	1819.35	**1212.44**
5-500-09	120717	✓**0**	✓**0**	0.2	✓**0**	✓**0**	962.72	**245.85**	916.35	317	522.97
5-500-10	218428	0.27	✓**0**	0.13	✓**0**	0.07	1254.91	**244.71**	1190.4	332.63	513.41
5-500-11	221202	4.77	4.77	4.4	✓**2.57**	4.77	1702.38	1775.92	1862.68	**1284.08**	1803.55
5-500-12	217542	6.8	7	7.47	7.57	✓**6.33**	2429.02	**2414.8**	2514.43	2530.82	2416.09
5-500-13	223560	✓**0**	✓**0**	✓**0**	✓**0**	0.07	139.43	302.79	**91.18**	244.68	401.52
5-500-14	218966	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	79.46	54.37	73.44	**44.76**	78.71
5-500-15	220530	2.8	✓**0.53**	3.03	0.9	0.7	1694.41	736.45	1758.03	**699.34**	760.65
5-500-16	219989	0.13	✓**0**	0.07	✓**0**	✓**0**	815.75	603.38	956.39	395.04	**379.1**
5-500-17	218215	✓**8.33**	11.73	11.47	13.67	9.1	**1801.07**	2172.94	2092.93	2210.07	1893.02
5-500-18	216976	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	32.54	17.84	42.32	14.62	**12.12**
5-500-19	219719	1	0.93	1.13	✓**0.87**	1	1863.11	1802.32	1889	1848.69	**1760.82**
5-500-20	295828	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	208.8	25.5	211.2	**20.97**	118.34
5-500-21	308086	✓**1.23**	6.97	1.77	6.7	6.4	**954.6**	2355.3	1184.77	2188.48	2159.54
5-500-22	299796	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	136.44	67.62	100.19	**45.51**	61.35
5-500-23	306480	0.07	✓**0**	0.33	✓**0**	0.13	945.72	142.55	916.04	**124.93**	422.67
5-500-24	300342	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	23.57	6.63	15.39	**5.37**	6.56
5-500-25	302571	✓**0**	3.03	0.2	3.47	2.57	426.18	1244.02	**419.66**	1473.23	1070.49
5-500-26	301339	9.33	10.07	10	✓**8.67**	9	2371.41	2434.74	2441.5	**2235.44**	2282.43
5-500-27	306454	6.4	7.23	6.4	6.4	✓**6.3**	1643.46	1937.01	**1530.39**	1971.27	1560.59
5-500-28	302828	0.67	1.17	✓**0.33**	1	1.83	1114.23	937.05	1038.7	**936.25**	1530.57
5-500-29	299910	4.53	5.73	**4.47**	5.8	6	2441.76	**2405.4**	2447.25	2414.07	2465.47
5-500-*	-	**68.86**	77.46	73.13	76.68	72.74	33119.14	28546.87	32438.58	**27885.26**	29619.43

		Gap					Time(s)				
Instance	Best	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$
10-100-00	23064	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	4.99	5.34	**4.71**	6.85	5.5
10-100-01	22801	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.34	0.38	**0.16**	0.38	0.34
10-100-02	22131	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.22	0.12	0.22	0.18	**0.11**
10-100-03	22772	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.41	**0.24**	0.44	0.33	0.41
10-100-04	22751	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	4.82	11.28	**4.44**	11.25	9.88
10-100-05	22777	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.99	1.63	**0.43**	1.64	1.47
10-100-06	21875	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	2.45	7.25	**1.44**	6.67	4.7
10-100-07	22635	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.89	10.06	**0.47**	11.1	11.95
10-100-08	22511	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**1.2**	1.83	1.5	2.55	2.05
10-100-09	22702	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.03	**0.02**	**0.02**	0.03	0.03
10-100-10	41395	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	2.9	3.39	2.54	2.69	**2.31**
10-100-11	42344	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.17	0.08	0.17	**0.07**	0.11
10-100-12	42401	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.18**	0.49	0.21	0.93	0.34
10-100-13	45624	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	2.55	9.19	**2.14**	8.18	8.7
10-100-14	41884	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	1.53	3.02	**1.12**	2.26	3.76
10-100-15	42995	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.02**	**0.02**	0.03	**0.02**	0.03
10-100-16	43574	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**31.2**	81.08	48.28	68.61	81.22
10-100-17	42970	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.14	0.14	**0.1**	**0.1**	0.19
10-100-18	42212	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.08	**0.06**	0.07	0.07	0.1
10-100-19	41207	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**8.39**	28.51	9.77	31	39.79
10-100-20	57375	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.02**	**0.02**	**0.02**	**0.02**	**0.02**
10-100-21	58978	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.03	**0.02**	0.03	0.04	0.03
10-100-22	58391	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.74	0.67	**0.46**	0.59	1.4
10-100-23	61966	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.08	0.16	**0.05**	0.13	0.06
10-100-24	60803	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.14	0.15	**0.13**	**0.13**	0.15
10-100-25	61437	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	2.67	2.76	1.94	2.83	**1.81**
10-100-26	56377	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	1.03	1.25	**0.99**	1.23	1.32
10-100-27	59391	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.25	0.3	**0.13**	0.3	0.42
10-100-28	60205	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**0.08**	0.1	0.09	**0.08**	0.1
10-100-29	60633	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.13	0.16	**0.1**	0.15	0.15
10-100-*	-	**0**	**0**	**0**	**0**	**0**	**68.67**	169.72	82.2	160.41	178.45

		Gap					Time(s)				
Instance	Best	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$
10-250-00	59187	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**56.45**	82.45	57.85	86.92	112.49
10-250-01	58781	✓**4.87**	69.23	7.4	65.37	51.67	**522.46**	1580.5	641.22	1482.24	1415.37
10-250-02	58097	✓**0**	0.3	0.1	0.7	1.77	**381.7**	956.83	478.07	1043.67	1503.44
10-250-03	61000	9.53	✓**7.7**	8.8	8.8	✓**7.7**	1572.83	**1326.72**	1512.34	1505.98	1334.99
10-250-04	58092	20.6	✓**0**	16.53	✓**0**	3.2	1246.88	**24.15**	1303.63	45.93	300.24
10-250-05	58824	7.7	✓**0**	10.5	0.7	6.3	1000.23	**192.44**	1060.98	223.99	656.16
10-250-06	58704	2.03	✓**0**	6.47	✓**0**	✓**0**	241.46	143.45	353.05	**64.33**	261.3
10-250-07	58936	4.83	✓**0.2**	4.23	0.63	0.63	1303.05	452.14	1130.26	516.09	**433.75**
10-250-08	59387	✓**0**	2.4	✓**0**	2.5	2.6	433.8	1524.53	**398.87**	1624.06	1626.47
10-250-09	59208	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**38.48**	77.37	60.35	66.7	59.4
10-250-10	110913	3.83	11.1	✓**1.27**	9.7	14.63	573.25	1126.83	**437.14**	1029.21	1288.47
10-250-11	108717	2.37	4.53	**2**	7.5	6.5	1577.46	1567.97	1618.74	**1554.4**	1617.9
10-250-12	108932	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	164.77	28.57	223.84	67.54	**25.51**
10-250-13	110086	✓**6.5**	17.13	7.77	20.07	15.4	**1407.31**	1571.88	1410.44	1567.17	1599.6
10-250-14	108485	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	**42.74**	227.45	45.62	209.74	230.38
10-250-15	110845	✓**1.83**	3.2	2.33	3.2	3.6	1069.72	1433.41	**1046.86**	1464.57	1531.63
10-250-16	106077	✓**0**	✓**0**	✓**0**	✓**0**	0.07	73.91	434.18	**55.86**	376.99	522.77
10-250-17	106686	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	115.17	53.28	76.15	**20.84**	75.9
10-250-18	109829	3.6	3.87	✓**3.33**	4	4	1546.05	1583.13	**1479.61**	1618.2	1616.67
10-250-19	106723	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	10.55	6.62	6.62	**5.27**	14.87
10-250-20	151809	✓**3.73**	7.73	4	7.47	7.2	1121.62	1469.73	**1114.96**	1434.8	1432.33
10-250-21	148772	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	0.45	0.3	0.39	**0.25**	0.34
10-250-22	151909	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	97.66	281.73	**71.42**	269.81	162.94
10-250-23	151324	✓**38.7**	40.47	40.37	45.97	46.2	1477.53	**1405.66**	1494.01	1487.25	1512.86
10-250-24	151966	6	6.6	✓**3.6**	10.8	10.8	1109.78	1119.48	**910.34**	1216.81	1309.28
10-250-25	152109	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	3.74	1.6	3.27	**0.94**	2.34
10-250-26	153131	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	3.93	1.89	5.48	2.02	**1.38**
10-250-27	153578	✓**0**	✓**0**	0.4	✓**0**	✓**0**	188.91	113.61	293.8	**89**	248.44
10-250-28	149160	✓**0**	✓**0**	0.17	✓**0**	✓**0**	391.25	**243.64**	353.93	286.89	311.63
10-250-29	149704	✓**0**	✓**0**	✓**0**	✓**0**	✓**0**	97.1	52.09	90.3	32.38	**24.69**
10-250-*	-	**116.12**	174.46	119.27	187.41	182.27	17870.24	19083.63	**17735.4**	19393.99	21233.54

		Gap					Time(s)				
Instance	Best	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	$CBGA$	$Sw_{d=1}$	$Sw_{d=2}$	$Sh_{d=1}$	$Sh_{d=2}$	CBG
10-500-00	117821	36.63	34.23	**33.47**	38.6	42.03	3370.29	3362.17	3373.79	**3350.39**	3359.6
10-500-01	119249	**53.7**	55.93	55.83	56.7	59.83	3368.75	3357.26	3370.43	3356.65	**3342.4**
10-500-02	119215	23.6	23.67	**23.57**	24.5	23.73	**3332.13**	3348.1	3386.67	3350.91	3355.2
10-500-03	118829	**15.37**	19.2	16.3	18.47	18.67	3373.71	3347.63	3375.17	3342.33	**3328.9**
10-500-04	116530	51.27	**39.37**	51.83	39.43	41.27	3396.75	3328.75	3397.68	**3326.48**	3336.5
10-500-05	119504	40.3	44.83	37.97	42.47	✓**35.07**	3289.04	3355.78	3140.47	3169.98	**3088.**
10-500-06	119827	44.27	35.73	40.63	**35.03**	43.9	3374.59	3345.55	3314.62	3344.95	**3301.8**
10-500-07	118344	45.5	43.57	48.4	**42.33**	46.17	3386.23	**3384.08**	3392.51	3386.2	3393.0
10-500-08	117815	47.83	37.73	51.57	38.7	✓**35.33**	3258.75	3963.01	3302.76	**3090.74**	3142.5
10-500-09	119251	59.23	50.03	59.93	**48.47**	58.73	3369.21	3317.36	3376.94	**3303.68**	3315.
10-500-10	217377	34.37	✓**15.3**	34.5	19.83	28.67	3150.83	**2446.38**	3192	2539.38	3167.1
10-500-11	219077	29.63	31.53	✓**29.37**	33.57	31.4	3167.87	3148.79	2998.77	3151.81	**2958.8**
10-500-12	217847	71.67	65.77	69.8	**60.63**	67.8	3180.83	3154.29	3184.5	3153.93	**3149.0**
10-500-13	216868	20.37	✓**5.27**	19.53	7.03	15.67	2535.8	**1293.66**	2498.33	1405.6	1956.0
10-500-14	213873	43.97	29.67	45.93	**28.3**	33.33	3176.86	**3164.16**	3175.71	3168.79	3174.6
10-500-15	215086	22.87	25.37	✓**19.47**	28.77	21.77	2823.01	3004.39	**2493.7**	3104.47	2722.2
10-500-16	217940	40.47	43.87	✓**37.53**	38.27	41.03	3178.38	**3146.27**	3174.37	3146.4	3165.1
10-500-17	219990	33.37	**27.5**	30.63	31.6	28.1	3190.65	**3151.58**	3184.31	3154.28	3154.6
10-500-18	214382	32.63	**31.43**	32	31.83	35.23	3175.44	**3142.4**	3180.64	3152.99	3147.3
10-500-19	220899	35.57	39.93	✓**35.5**	38.87	36.97	3190.31	3159.66	**3144.66**	3161.57	3159.7
10-500-20	304387	**36.23**	41.1	36.5	38.47	38.73	3005	2989.04	3009.1	**2877.84**	2950.6
10-500-21	302379	✓**15.1**	33.2	15.77	32.33	32.1	2795.16	2897.9	**2710.2**	2952.9	2895.
10-500-22	302417	22.5	18.83	19.5	**16.9**	21.53	3012.4	2983.86	3003.82	2997.99	**2983.6**
10-500-23	300784	✓**38.5**	40.4	39.93	40.27	41	**2975.63**	2994.02	3003.55	2992.05	2985.5
10-500-24	304374	17.87	24.8	✓**17.33**	26.03	22.43	3003.91	2980.13	**2843.2**	2980.72	2973.8
10-500-25	301836	75.33	74.07	74.47	75.33	**70.93**	3016.38	2996.88	3011.28	2999.77	**2992.4**
10-500-26	304952	✓**1.8**	3	2.1	3	2.9	**2305.09**	2991.25	2467.58	2984	2969.
10-500-27	296478	✓**21.93**	25	23	24.13	25.9	**2955.7**	2987.81	3008.86	2991.65	2983.7
10-500-28	301359	17.2	**9.17**	17.07	11.3	13.5	**2909.84**	2988.32	3003.44	3003.78	2985.1
10-500-29	307089	31.13	32.3	33.77	✓**29.77**	32.6	2915.09	2860.43	**2858.77**	2912.34	2984.3
10-500-*	-	1060.21	1001.8	1053.2	**1000.93**	1046.32	93183.63	92590.91	92577.83	**91854.57**	92423.8