A Biased Random-Key Genetic Algorithm for the Set Orienteering Problem

Francesco Carrabs

Department of Mathematics, University of Salerno, 84084, Fisciano, Italy

Abstract

This paper addresses the Set Orienteering Problem which is a generalization of the Orienteering Problem where the customers are grouped in clusters, and the profit associated with each cluster is collected by visiting at least one of the customers in the respective cluster. The problem consists of finding a tour that maximizes the collected profit but, since the cost of the tour is limited by a threshold, only a subset of clusters can usually be visited. We propose a Biased Random-Key Genetic Algorithm for solving the Set Orienteering Problem in which three local search procedures are applied to improve the fitness of the chromosomes. In addition, we introduced three rules useful to reduce the size of the instances and to speed up the resolution of the problem. Finally, a hashtable is used to quickly retrieve the information that are required several times during the computation. The computational results, carried out on benchmark instances, show that our algorithm is significantly faster than the other algorithms, proposed in the literature, and it provides solutions very close to the best-known ones.

Keywords: Metaheuristics; Biased Random-Key Genetic Algorithm; Orienteering problem; Routing

1. Introduction

Routing problems with profits received significant attention in recent years, as witnessed by the large literature surveyed in recent papers [3, 20]. The most widely known problem in this class is the Orienteering Problem (OP) introduced in [37]. In the OP a profit is associated with each customer and the objective is to find a single vehicle tour maximizing the profit collected from visited customers and such that the duration of the tour does not exceed a maximum time limit.

Email address: fcarrabs@unisa.it (Francesco Carrabs)

The profit of each customer can be collected at most once.

In this paper, we face the Set Orienteering Problem (SOP) which is a generalization of the OP where customers are grouped in clusters and a profit is associated with each cluster. To gain the profit associated with a cluster it is necessary to visit at least a customer of that cluster. The problem consists of finding a tour over a subset of clusters such that *i*) the profit collected is maximized and *ii*) the tour length is within a given threshold T_{max}. The SOP is similar to another routing problem with profits, named Clustered Orienteering Problem (COP) [1]. The difference between these two problems lies in how the profit is gained because, in the COP, the profit of a cluster is collected if and only if all the customers of that cluster are visited.

The SOP was introduced the first time in [2]. In this paper, the authors proposed a formulation of the problem and a matheuristic algorithm named MASOP. MASOP is made by two phases: the first one builds an initial solution and the second one invokes a Tabu Search metaheuristic to improve this initial solution. The initial solution is found by a greedy algorithm that starts from a tour containing only the depot and, at each iteration, it adds the closest vertex belonging to the cluster with the highest profit and not yet visited. The greedy algorithm stops when no more vertices can be added due to the cost limit. The tabu search uses three operators: ExploreNeighborhood, MIPMove, and Shake. The first operator generates the neighborhood of the solution by using the insert and swap operators while MIPMove generates a different neighborhood obtained by solving a MILP model and neglecting the tabu list. Finally, the Shake procedure partially destroys the current solution by removing randomly some clusters (diversification phase). The authors defined two sets of benchmark instances, one by adapting the classical instances of the Generalized Traveling Salesman Problem (GTSP) and another one by generating random instances. In [27] the authors proposed an alternative formulation for the SOP and a Variable Neighborhood Search metaheuristic that is used also for other two variants of the OP: the Orienteering Problem with Neighborhoods [5, 15] and the Dubins Orienteering Problem [28, 29].

The SOP finds application in mass distribution products in which a different distribution plan is sought. For instance, let us consider the case when customers are clustered in areas and the service to each area is made by delivering the entire quantity required by all customers in that area to a single customer, the one that is visited. This happens also when private customers group together to reach large quantity orders, and thus hopefully a lower price. Typically, in this case, the delivery is made to a single location. There are even other applications of the SOP that are far from the ones originally outlined in [2]. Indeed, the SOP can be used for any application of the GTSP, discussed in [25], where the salesman has a limited budget, and cluster profits can be used for prioritization like the travel guide problem.

In this work, we propose a Biased Random-Key Genetic Algorithm (BRKGA) for the SOP. This method was chosen for its successful results obtained on a significant number of optimization problems [17], such as scheduling problems [8, 9, 30, 34], container loading problems [18, 32], and transportation problems [4, 22, 26, 33]. The main contributions of the paper are:

- the introduction of a BRKGA algorithm for the SOP,
- the introduction of three local search procedures used to improve the fitness of the chromosomes,
- the use of a hashtable and of a three-dimensional matrix to speed up the resolution of the problem by avoiding redundant computations and
- the introduction of a set of rules to reduce the size of the instances.

More in detail, the chromosome defines *i*) the clusters to be visited in the tour and *ii*) the visiting order of these clusters while the decoder function states the clusters that can be visited according to the cost limit imposed by the problem. The three local search procedures try to improve the fitness of the chromosomes either by introducing new clusters into the current solution or by swapping clusters. In the hashtable, we save the information, concerning the chromosomes, that is crucial to both reduce the number of invocations of the decoder function and to quickly provide the input data required by local search procedures. Finally, we prove that there are rules for the removal of useless vertices, arcs, and clusters, defined for the GTSP, that hold also for the SOP. These rules are applied in a preprocessing phase of the algorithm. The application of a hashtable, of the three-dimensional matrix and the reduction of the instance size significantly impact on the performance of our algorithm. The computational results, carried out on benchmark instances, show that the BRKGA is faster than the other algorithms, proposed in the literature, and the solutions it provides are very close to the best-known solutions.

The remainder of this paper is organized as follows. In Section 2, we introduce the terminology and the notation used throughout the paper. In Section 3, we describe the preprocessing procedure while, in Section 4, we present our new metaheuristic. Computational results as reported in Section 5. Finally, conclusions are provided in Section 6.

2. Definitions and notation

The Set Orienteering Problem is defined on a directed and complete arc weighted graph $G = (V, A)$, where $V = \{v_0\} \cup C$ is the set of vertices with $|V|$ and A is the set of arcs with $|A|$. The vertex v_0 represents the depot from which the vehicle starts and ends its tour while C is the set of customers. The vertices of G are grouped in clusters C_g with $g = 1, ..., l$, such that $\bigcup_{g=1}^{l} C_g = V$ and $C_g \cap C_h = \emptyset$, $\forall C_g, C_h \in \mathcal{P}$, where $\mathcal{P} = \{C_1, ..., C_l\}$ is the set of clusters. A profit p_g is associated with each cluster and it is collected if and only if at least a customer $i \in C_g$ is visited in the tour. The profit of each cluster can be collected at most once. The cluster C_1 contains only the depot v_0 and its profit is equal to 0. A cost c_{ij} is associated with each arc $(i, j) \in A$ and we assume that costs c_{ij} satisfy the triangle inequality. The SOP consists of finding a tour that maximizes the collected profit and such that its cost (or duration) does not exceed a maximum value T_{max}. In the following, we denote this last condition as *cost constraint*. Since the arc costs satisfy the triangle inequality, an optimal tour always includes at most one vertex per cluster [2]. Let $\mathcal{C} : V \rightarrow \mathcal{P}$ be a function that, given a vertex $v \in V$, returns the cluster containing v. For instance, $\mathcal{C}(v_0) = C_1$.

Any solution of the SOP can be described by a permutation $\sum_k = (\sigma_1, ..., \sigma_k)$ of the cluster indexes, with $1 \leq \sigma_i \leq l$, $\sigma_i \neq \sigma_j$ for $i \neq j$ and $\sigma_1 = 1$, defined according to the visiting sequence of the clusters in the tour.

3. Preprocessing phase

In this section, we prove that there are rules for the removal of useless vertices, arcs, and clusters, defined for the GTSP, that hold also for the SOP. These rules are useful to reduce the size of the instances and to speed up the resolution of the SOP on them. To this aim, we take advantage of the characteristics of the problem, as the grouping of the vertices in the clusters and the T_{max} cost limit of the tour. The main idea is to remove from G vertices, arcs, and clusters not necessary to build an optimal solution. In the following, we report how these removals are carried out.

- **Arcs and vertices removal**

  This procedure is based on the removal procedures proposed in [21] for the GTSP. However, it is worth noting that the correctness of the procedures given in [21] is based on the cost of the tour since the optimality of a solution for the GTSP depends only on its cost. For the SOP this is not true because the cost of the tour states if a solution is feasible or not but the optimality depends on the profit collected. For this reason, we provide a proof of the uselessness of the edges and vertices removed by taking into account both the cost of the tour and the profit collected. Finally, for the edge removal, we use the implementation proposed in [11] and successfully applied also in [10, 12]. The removal procedure, named *Graph Reduction Algorithm* (GRA), works as follows.

  Given a cluster C_g, let us consider two customers $u \in C_h$ and $v \in C_k$, such that $h \neq k \neq g$. The GRA computes the shortest path between u and v, passing through C_g. Since triangle inequality holds, this shortest path is composed of two arcs: (u, w) and (w, v), where $w \in C_g$. The GRA marks as needed these two arcs of the shortest path and the vertex w crossed in C_g. The algorithm repeats this operation for each cluster $C_g \in \mathcal{P}$ and for all possible couples of customers u and v, with $\mathcal{C}(u) \neq \mathcal{C}(v) \neq C_g$. At the end of the computation, the GRA removes all not-marked arcs and vertices from the graph. Proposition 1 ensures that there always exists an optimal solution without not marked arcs.

  Proposition 1. *Given a cluster $C_g \in \mathcal{P}$, let S be the set of the shortest paths from $u \in C_h$ to $v \in C_k$ passing through C_g, with $h \neq k \neq g$. Moreover, let $A_{\bar{S}}$ be the set of arcs incident to the vertices in C_g that do not belong to any shortest path of S. Then an optimal solution of the SOP, not containing arcs in $A_{\bar{S}}$, always exists.*

  Proof. W.l.o.g. let us suppose that T^* is an optimal solution containing the arcs (u, w) and (w, v), with $u \in C_h$, $w \in C_g$, $v \in C_k$ and $(u, w) \in A_{\bar{S}}$. Since (u, w) does not belong to any shortest path in S, then there exists another customer $w' \in C_g$ such that the path $\{u, w', v\}$ is shorter than $\{u, w, v\}$. By replacing $\{u, w, v\}$ with $\{u, w', v\}$ in T^*, we obtain a new tour T' that is feasible and optimum because $c(T') < c(T^*)$ and $p(T') = p(T^*)$. $\square$

Proposition 2. *Given a cluster $C_g \in \mathcal{P}$, let S be the set of the shortest paths from $u \in C_h$ to $v \in C_k$ passing through C_g, for each $C_h, C_k \in \mathcal{P} \backslash \{C_g\}$ with $h \neq k$. Moreover, let $V_{\bar{S}}$ be the set of vertices in C_g that do not belong to any shortest path of S. Then an optimal solution of the SOP, not containing vertices in $V_{\bar{S}}$, always exists.*

Proof. Similar to Proposition 1. $\square$

- **Clusters removal**

  Since the feasibility of a tour depends on the T_{max} value, lower is this value lower is the number of feasible solutions in G. To state if a cluster C_k is useless, it is sufficient to check the distance between the depot v_0 and each vertex $v_i \in C_k$. More in detail, if the distance between v_0 and a vertex $v_i \in C_k$ is greater than $\frac{T_{max}}{2}$ then any tour starting from v_0 and visiting v_i is infeasible because violated the cost constraint. As a consequence, the vertex v_i is useless and then it is removed. This check is carried out for all the vertices in C_k and if, at the end, the cluster is empty then it is removed from G. The procedure to find useless clusters requires $O(|E|)$ time, and as expected, its effectiveness increases as T_{max} decreases. A similar strategy was proposed in [13] for the Orienteering Problem where the aim was to find a path from a starting vertex to an ending vertex maximizing the profit and satisfying a maximum length T_{max}. The authors used the starting and ending vertices as foci of an ellipse having the length of the major axis equal to T_{max} and they removed all the vertices outside the ellipse because useless.

The application of the previous strategies improves the performance of our BRKGA algorithm because the subgraph G', obtained by applying the preprocessing phase on G, contains fewer vertices, arcs, and clusters of G.

3.1 Insertion cost matrix

One of the most used operations in our algorithm is the insertion of a new cluster in the current tour. To carry out this operation, it is necessary to choose the vertex w of the cluster C_k to insert and the two consecutive vertices u and v, of the current tour, between which w will be inserted. Because of the cost constraint, it is important to know how much the cost of the tour increases, to state if the new tour is feasible or not. Instead of computing this information every time the algorithm performs an insertion operation, we compute it just one time, during the preprocessing phase, and we save it in a three-dimensional matrix, named *ICM*, having size $|V| \times |V| \times l$. More in detail, given a couple of vertices u and v and a cluster C_k, with $\mathcal{C}(u) \neq \mathcal{C}(v) \neq C_k$, we compute the shortest path from u to v crossing C_k and we save the cost of this path and the vertex $w \in C_k$, crossed by this shortest path, in the position [u,v,k] of the matrix. In this way, it is possible to retrieve in constant time both the minimum insertion cost of a cluster C_k, between two vertices of the tour, and the vertex of C_k to insert to have that insertion cost.

It is worth noting that the costs of *ICM* are locally optimal because they are computed according to the vertices visited in the current tour T. As a consequence, if *ICM* states that the insertion of a cluster C_k, in any position of T, violates the cost constraint, this is true only for the current visited vertices in T. However, it could be possible to obtain a new feasible solution, by inserting C_k in T, provided that some vertices of T are replaced by other ones of the same clusters.

4. Biased Random-Key Genetic Algorithm

In this section, we present the BRKGA concept, including a detailed description of the solution encoding and decoding, the evolutionary process and the fitness function. We also describe three operators used to intensify the search in promising communities and the hashing strategy used to improve the performance of the algorithm.

4.1 The BRKGA framework

The BRKGA is a metaheuristic proposed in [17], in which chromosomes (solutions) are encoded as vectors with n elements of real numbers in the interval [0,1]. These numbers are called *random-keys* or *allele*. A decoding function associates to each chromosome a solution of the underlying optimization problem, from which the objective function value or fitness can be computed. This function is named *decoder*. The decoder function and the chromosome definition represent the main aspects which define the BRKGA, since the other aspects are essentially problem-independent. The BRKGA starts with and then evolves a population containing exactly p chromosomes, each having n allele, for a number of generations until a stopping criterion is met. The population is partitioned in two sets of chromosomes: the *elite* containing p_e individuals with the best fitness values and a *non-elite* set with the remaining individuals. The evolutionary process at generation $i + 1$ is carried out as follows. The p_e elite chromosomes of generation i are copied in the new population.

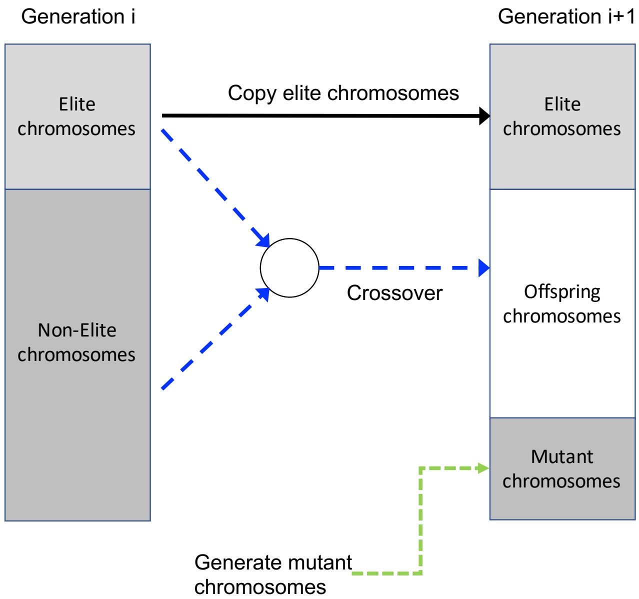


Figure 1: Generation of a new population in the BRKGA. The p_e elite chromosomes are directly copied in the new population in which p_m mutant chromosomes are added. To complete the population, $p - p_e - p_m$ offspring chromosomes are generated by randomly selecting one elite and one non-elite chromosome and by applying on them the parametrized uniform crossover operator.

Moreover, p_m chromosomes are randomly generated and introduced in the new population. These p_m chromosomes are named *mutants*, and they are used in place of the mutation operator usually found in evolutionary algorithms [6, 23]. The remaining $p - p_e - p_m$ chromosomes (offspring) are generated by randomly selecting one chromosome from the elite set and one chromosome from the not-elite set and carrying out the parametrized uniform crossover [35]. More in detail, fixed a probability ρ_e, this crossover defines the value of the j-th allele of offspring i by generating a random number r in the interval [0,1); if $r > \rho_e$ then the offspring i inherits the j-th allele of its elite parent otherwise it inherits the j-th allele of the non-elite parent. Figure 1 shows how a new generation is created from the previous one in BRKGA. In the following subsections, we describe in detail how we designed BRKGA to solve SOP.

4.2 Chromosome representation and decoding

We defined the chromosome representation for the SOP by taking into account the following proposition.

Proposition 3. *[7] Given a clustered graph $G = (V, E)$ and a visiting sequence $\sum_k$ of clusters,*

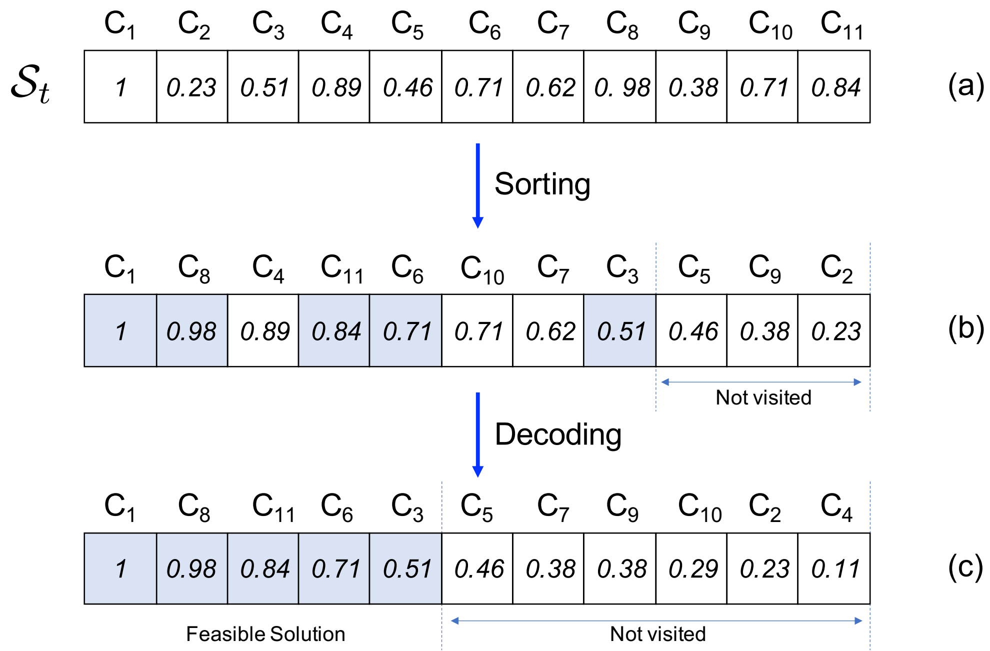


Figure 2: (a) Chromosome representation. (b) The sorting operation carried out to state the (possible) clusters to visit and in which order. (c) Decoding operation.

the problem of finding the shortest tour, that visits the clusters according to this sequence, can be solved in polynomial time by solving a shortest path problem.

The computation of the shortest tour of Proposition 3 is carried out on a graph $G' = (V', E')$ built as follows. The vertices in G' are the vertices belonging to the clusters in $\sum_k$ plus a dummy vertex named v'_0, which is $V' = \bigcup_{i=1}^{k} C_{\sigma_i} \cup v'_0$. The arcs in G' connect vertices belonging to adjacent clusters of $\sum_k$. More in detail, $(u, v) \in E'$ if $u \in C_{\sigma_i}$, $v \in C_{\sigma_{i+1}}$, and $(u, v) \in E$. Moreover, any vertex $v \in C_{\sigma_k}$ is connected to the dummy vertex v'_0 with an arc having cost equal to c_{v,v_0}. The shortest path from v_0 to v'_0 is the shortest tour we are looking for.

From Proposition 3, we derive that each chromosome has to report two information: *i)* the visited clusters in the tour and *ii)* the visiting order of these clusters. For this reason, the chromosome in our BRKGA is a vector of random keys (real numbers between 0 and 1) having a number of alleles equal to the number of clusters in G. Figure 2(a) shows a chromosome $\mathcal{S}_t$, having eleven clusters, and its alleles. In the following, we denote by a_i the allele associated with the cluster C_i.

To state what clusters should be visited and their visiting order, the clusters are sorted according to their alleles, in non-increasing order. In case of ties, the cluster with lower id is selected first

(Figure 2(b)). The clusters candidate to be visited on the tour are the ones having the allele greater than 0.5. In our example, clusters C_2, C_5, and C_9 are discarded from the construction of the tour because of their alleles (Figure 2(b)). The sorted list of clusters with allele greater than 0.5 represents the visiting sequence $\sum_k$ of the clusters. In our example this sequence is $\sum_8 = (1, 8, 4, 11, 6, 10, 7, 3)$. Since any feasible tour has to start from the depot, the allele of C_1 is set to 1 to assure that C_1 is always the first visited cluster. Note that, the shortest tour, visiting the clusters according to $\sum_k$, can be infeasible for the SOP if its cost is greater than T_{max}. For this reason, a deterministic procedure, named *Decoder*, has to be invoked on the sequence $\sum_k$ to produce a feasible solution.

The *Decoder* procedure starts with a visiting sequence $\sum'$ containing only C_1. At iteration i, the *Decoder* adds to $\sum'$ the i-th cluster of $\sum_k$ and it finds the shortest tour T' visiting the clusters according to $\sum'$. If $c(T') \leq T_{max}$ then the i-th cluster of $\sum_k$ is left in $\sum'$ otherwise it is removed. The procedure proceeds with the next cluster in $\sum_k$ and so on until no more clusters are available.

In Figure 2(b) the clusters selected by the *Decoder* to build the tour are highlighted. Since clusters C_4, C_{10} and C_7 are not selected by the *Decoder*, their alleles have to be decreased to a value lower than or equal to 0.5. To this end, the *Decoder* sets the alleles $a_{\sigma_4} = 1 - a_{\sigma_4}$, $a_{\sigma_{10}} = 1 - a_{\sigma_{10}}$ and $a_{\sigma_7} = 1 - a_{\sigma_7}$ and it sorts again the clusters according to the new allele values (Figure 2(c)). The final sequence produced by the *Decoder* is $\sum_5 = (1, 8, 11, 6, 3)$ and the fitness associated to this chromosome is given by the sum of the profits of clusters in $\sum_5$.

4.2.1 Hashtable

The decoding function should be invoked on all the chromosomes, generated during the evolutionary process, to define the feasible solution associated with the chromosome and to compute its fitness. However, since this operation is computationally expensive, and the decoder is a deterministic procedure, the idea is to avoid its invocation on the chromosomes on which it has been already invoked in the previous evolutionary steps. To this end, we use a hashtable in which, after the invocation of the decoder on a chromosome, we save three information: *i*) the cost of the tour found by *Decoder*, *ii*) the vertices of the tour and *iii*) the fitness of the chromosome. The hash key associated with each chromosome is generated according to the sorted sequence of its alleles greater than 0.5. For instance, given the chromosome in Figure 2(a), after the sorting of the alleles (Figure 2(b)), the hash key of this chromosome is $< 1, 8, 4, 11, 6, 10, 7, 3 >$. Before invoking the decoder on a chromosome, we verify if its hash key is already present into the hashtable. If not, *Decoder* is invoked on the chromosome otherwise all the information about the tour are directly obtained from the hashtable.

4.3 Initial population

Each chromosome of the initial population is generated by assigning to its alleles a random number chosen in the real interval [0,1]. The only exception is for the allele associated with cluster C_1 that is always set to 1 because any feasible tour has to start from the depot v_0. The number of chromosomes in each population is equal to p. We use two populations, not evolved in parallel, that exchange their best chromosome every *swap_best* iterations [18]. Finally, we apply the idea proposed in [19] consisting of resetting the populations after pop_{reset} iterations without improvement. This is to avoid the BRKGA staying trapped in local optimum regions.

4.4 Improvement heuristics

In this section, we describe three local search operators used to improve the fitness of a given chromosome $\mathcal{S}_t$. In the following, we suppose that $\sum_k$ is the visiting sequence associated with $\mathcal{S}_t$ by *Decoder*. Since we do not allow operators to change a visited vertex in $\sum_k$ with another one of the same cluster, then the insertion cost of a new cluster in $\sum_k$ can be computed in constant time thanks to *ICM* (see Section 3.1).

- **Insert Operator**

  The *Insert* operator tries to insert new clusters in the current sequence $\sum_k$ to increase the fitness of the chromosome. First, the operator checks if $\mathcal{S}_t$ is in the hashtable and, if this is the case, the operator stops because it has been already invoked on this chromosome. Otherwise, *Insert* proceeds as follows.

  The operator builds a list $\ell_{\bar{v}}$ of the not visited clusters that is sorted according to their allele, in non-increasing order. *Insert* goes through the sorted list $\ell_{\bar{v}}$ by selecting the clusters, one by one, as a possible candidate for the insertion in $\sum_k$. More in detail, given a $C_h \in \ell_{\bar{v}}$, the operator finds the cheapest position j in $\sum_k$ where to insert a vertex of C_h. If the cost of this new tour is lower than or equal to T_{max}, then C_h is inserted in position j of $\sum_k$ and its allele is set to $(a_{\sigma_{j-1}} + a_{\sigma_j})/2$. Otherwise, C_h is rejected, and *Insert* selects the next cluster in $\ell_{\bar{v}}$. The operator stops when all the clusters in $\ell_{\bar{v}}$ are checked for the insertion. Finally, the operator generates the hash key of this new chromosome and inserts it into the hashtable, if not yet present, with the other information.

Notice that, the insertion cost of C_h, in any position of $\sum_k$, is computed in constant time thanks to *ICM*. This makes much faster the *Insert* operator.

- **Swap Operator**

  The *Swap* operator tries to improve the fitness of $\mathcal{S}_t$ by replacing some visited clusters with some other not visited ones. First, the operator checks the presence of $\mathcal{S}_t$ in the hashtable and, if it is present, the operator stops because it has been already invoked on this chromosome. Otherwise, *Swap* proceeds as follows.

  According to the alleles of $\mathcal{S}_t$, the operator builds a list ℓ_v of the visited clusters, and it sorts this list according to the clusters' profit, in non-decreasing order. Moreover, *Swap* builds a list $\ell_{\bar{v}}$ of the not visited clusters, and it sorts this list according to the clusters' profit, in non-increasing order.

  Swap goes through the sorted list ℓ_v by selecting a cluster in this list as a possible candidate for the swap operation. Once selected a cluster C_i from ℓ_v, *Swap* goes through the sorted list $\ell_{\bar{v}}$ to find a cluster C_j that could replace C_i in $\sum_k$. We state that a swap operation between two clusters C_i and C_j can be done if and only if the following two conditions are satisfied: *i)* $p_i < p_j$ and *ii)* the cost of the new tour, crossing C_j instead of C_i, does not exceed T_{max}. Notice that the second condition can be quickly verified thanks to *ICM* that returns, in constant time, the insertion cost of cluster C_j between the two clusters adjacent to C_i in $\sum_k$. After the swap operation, the cluster C_i and C_j are removed from their respective sorted lists and $\sum_k$ is updated accordingly. The allele of C_i is set to $1 - a_{\sigma_i}$ while for the allele of C_j it is used the same policy applied by the *Insert* operator.

  Swap starts a new iteration by selecting the next cluster in ℓ_v. The operator stops when the profit p_i of cluster $C_i \in \ell_v$ is greater than or equal to the profit p_j of the first cluster $C_j \in \ell_{\bar{v}}$. Indeed, if $p_i \geq p_j$, the profit of any cluster in $\ell_{\bar{v}}$ is not greater than p_i. This statement holds because of the sorting, we carried out on ℓ_v and $\ell_{\bar{v}}$. Since the first condition for the swap operation cannot be satisfied anymore, the operator stops. Finally, the operator generates the hash key of this new chromosome and inserts it into the hashtable, if not yet present, with the other information.

Notice that, the Insert and Swap operators sort the clusters by using a different criterion: the allele and the profit value, respectively. There is a reason behind this choice. After the decoding operation, generally, the cost of the chromosomes is far enough away from the T_{max} threshold to allow the insertion of different clusters inside them. Since the profit of the clusters never changes, using this criterion to establish the insertion order leds to the risk to insert always the same clusters in the chromosomes. For this reason, we prefer to use the allele values to ensure that it is the evolutionary process to establish this order of insertion according to the allele values assigned to the chromosomes. The situation is different for the swap operator. This operator is invoked after the insert operator and therefore it works on a chromosome whose cost is already close to the T_{max} threshold. Moreover, since the main aim of this operator is to increase the fitness of the chromosome, all the swap operations that do not increase the profit are rejected. As a consequence, the number of possible operations that the Swap operator can carry out is lower with respect to the Insert operator. Having fewer operations available, we try to maximize the profit, gained by each swap operation, by sorting the lists ℓ_v and $\ell_{\bar{v}}$, according to the profit, in non-decreasing order and in non-increasing order, respectively.

- ***Mck* operator**

  Given a sequence $\sum_k$, let $\ell_{\bar{v}}$ be the set of not visited clusters. Since the insertion of a new cluster in $\sum_k$ is carried out between two consecutive clusters of $\sum_k$, then there are exactly k available positions to perform this insertion. Moreover, thanks to *ICM*, for each cluster $C_h \in \ell_{\bar{v}}$ and for each position k in $\sum_k$, we already know the vertex $v \in C_h$ which insertion cost in position k is minimum.

  By examining how our insertion operation works, we found out that we are facing a variant of the Multiple-Choice Knapsack Problem (MCKP) [24, 31]. This problem is formally defined as follows. Let $N_1, ..., N_m$ be a set of mutually disjoint classes of items to be packed into a knapsack of capacity W. To each item $i \in N_j$ a profit p_{ij} and a weight w_{ij} are associated. MCKP consists of choosing one item from each class such that the profit sum is maximized without exceeding the capacity W in the corresponding weight sum.

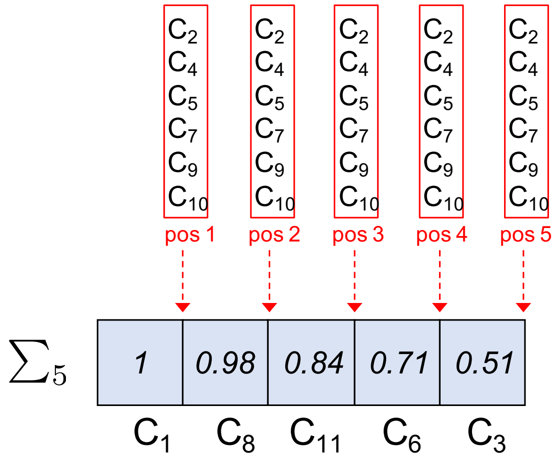


Figure 3: The *Mck* operator framework.

For our insertion operation, the classes of the MCKP correspond with the k positions available in $\sum_k$. Moreover, each class j has exactly one vertex $v_i \in C_h, \forall C_h \in \ell_{\bar{v}}$, where v_i is the vertex that minimizes the insertion cost of cluster C_h in position j. The profit associated with v_i is equal to the profit of its cluster C_h while its weight is given by the insertion cost of v_i in position j of $\sum_k$. Since the vertex $v_i \in C_h$ to insert in j is unique, in the following we talk about the insertion in position j of cluster C_h or of vertex v_i interchangeably. By using the information saved in *ICM*, we can quickly build the k classes with the appropriate vertices and the right weights. Finally, the capacity W of the knapsack is equal to the difference between T_{max} and the cost of $\sum_k$.

Figure 3 shows how *Mck* works. We use the same chromosome and the same sequence reported in Figure 2. This means that $\ell_{\bar{v}} = \{C_2, C_4, C_5, C_7, C_9, C_{10}\}$ and there are 5 positions of $\sum_5$ where *Mck* can introduce these clusters. Therefore, we have 5 classes, each one associated with a different position j in $\sum_5$. For each $C_h \in \ell_{\bar{v}}$, the class associated with the position j contains the vertex of C_h that minimizes the insertion cost of this cluster in position j.

There are three differences between the MCKP and the version we use for the SOP. The first difference is that the classes are not mutually disjoint because the same vertex can belong to several classes. The second one is that, when a vertex $v \in C_h$ is inserted into a position j, then no other vertex of C_h can be inserted in the other positions of $\sum_k$. The third difference is that we can select at most one vertex for each class rather than exactly one as in the original problem. The formulation of our modified version of the MCKP is the following. The decision variables are x_{ij} equal to 1 if cluster i is inserted in position j and 0 otherwise.

$$\max \sum_{i \in \ell_{\bar{v}}} \sum_{j=1}^{k} p_i x_{ij} \tag{1}$$

$$\sum_{i \in \ell_{\bar{v}}} \sum_{j=1}^{k} w_{ij} x_{ij} \leq W \tag{2}$$

$$\sum_{j=1}^{k} x_{ij} \leq 1 \qquad \forall i \in \ell_{\bar{v}} \tag{3}$$

$$\sum_{i \in \ell_{\bar{v}}} x_{ij} \leq 1 \qquad j = 1, ..., k \tag{4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i \in \ell_{\bar{v}},\ j = 1, ..., k \tag{5}$$

The objective function (1) maximizes the profit collected by the insertion of new clusters in $\sum_k$. Constraint (2) ensures that the cost of the final tour does not exceed the limit T_{max}. Constraints (3) impose that each cluster can be inserted in at most one position of the sequence $\sum_k$ while Constraints (4) ensure that at most one cluster is inserted in each position. Finally, (5) are variable definitions.

Our *Mck* operator solves the previous model to obtain the best possible insertion of the clusters in the current sequence $\sum_k$. However, since this operator solves a MIP model, it is much more expensive than the *Insert* operator. For this reason, we invoke it only on the chromosomes of the last population generated by the BRKGA.

4.5 Termination criteria

The algorithm ends when max_{it} iterations have been executed, or when $max_{it}/3$ consecutive iterations have failed to improve the incumbent solution and populations have been reset at least once. The second criterion is often satisfied with the small instances where the algorithm usually finds the best solution during the first iterations.

4.6 Pseudocode

The pseudocode of the BRGKA is listed in Algorithm 1. In this algorithm, the procedures having the suffix POP are applied on all the chromosomes of the population. For instance, the

Algorithm 1: The BRKGA pseudocode

Input: $G, \mathcal{P}, T_{max}$
Output: A feasible tour for the SOP

1 $i \leftarrow 1, \quad resetDone \leftarrow false$;
2 $P_i \leftarrow$ `genPopulation`$(\mathcal{P})$, $\quad P''_i \leftarrow$ `genPopulation`$(\mathcal{P})$;
3 $P_i \leftarrow$ `decodePOP`(P_i), $\quad P''_i \leftarrow$ `decodePOP`(P''_i);
4 $P_i \leftarrow$ `insertPOP`(P_i), $\quad P''_i \leftarrow$ `insertPOP`(P''_i);
5 $P_i \leftarrow$ `swapPOP`(P_i), $\quad P''_i \leftarrow$ `swapPOP`(P''_i);
6 **while** $i \leq max_{it}$ **do**
7 $\quad P_{i+1} \leftarrow$ `evolve`(P_i), $\quad P''_{i+1} \leftarrow$ `evolve`(P''_i);
$\quad$ *// if the current iteration is not the last one*
8 $\quad$ **if** $i < max_{it}$ **then**
9 $\quad\quad P_{i+1} \leftarrow$ `insertPOP`(P_{i+1}), $\quad P''_{i+1} \leftarrow$ `insertPOP`(P''_{i+1});
10 $\quad\quad P_{i+1} \leftarrow$ `swapPOP`(P_{i+1}), $\quad P''_{i+1} \leftarrow$ `swapPOP`(P''_{i+1});
11 $\quad$ **else**
12 $\quad\quad P_{i+1} \leftarrow$ `mckPOP`(P_{i+1}), $\quad P''_{i+1} \leftarrow$ `mckPOP`(P''_{i+1});
$\quad$ *// aspiration criterion*
13 $\quad$ **if** $((i - iter_{Best}) \geq max_{it}/3$ *&& resetDone = true)* **then**
14 $\quad\quad$ break;
$\quad$ *// reset the populations after pop_{reset} iterations without improvements*
15 $\quad$ **if** $((i - iter_{Best})\ \%\ (pop_{reset}) = 0)$ **then**
16 $\quad\quad P_i \leftarrow$ `genPopulation`$(\mathcal{P})$, $\quad P''_i \leftarrow$ `genPopulation`$(\mathcal{P})$;
17 $\quad\quad resetDone \leftarrow true$;
$\quad$ *// best individual exchange every $swap_{best}$ iterations*
18 $\quad$ **if** $(i\ \%\ (swap_{best}) = 0)$ **then**
19 $\quad\quad exchange\_best(P_{i+1}, P''_{i+1})$;
20 $\quad i \leftarrow i + 1$;
21 **return** *the tour associated to the best individual found*;

procedure $insertPOP(P_i)$ applies the *insert* operator on all the chromosomes of the population P_i and it returns the population with the updated chromosomes.

The algorithm takes as input a graph G, a set of clusters $\mathcal{P}$, and a threshold T_{max}. To make the pseudocode more readable, we avoid passing these three parameters to the procedures but all of them, except *genPopulation*, use these parameters. The first step of the BRKGA is the initialization of the iteration counter i to 1 and of the flag *resetDone* to false. The next step is the generation of the starting populations P_i and P''_i (line 2) according to the rules described in Section 4.3. Then, the $decodePOP$, $insertPOP$ and $swapPOP$ procedures are invoked on P_i and on P''_i (line 3-5). The while loop (line 6) iterates until max_{it} iterations are carried out. The first step of the loop is the generation of the next populations P_{i+1} and P''_{i+1} by invoking the *evolve* procedure on the current populations P_i and P''_i (line 7), respectively. To this end, the *evolve* procedure executes the steps reported in Sections 4.1 and summarized in Figure 1. If the current iteration i is not the last one, then the procedures $insertPOP$ and $swapPOP$ are invoked on P_{i+1} and P''_{i+1} (lines 9-10), otherwise, the procedure $mckPOP$ is invoked on these two populations (line 12). The aspiration criterion states that if $max_{it}/3$ iterations are carried out without improving the incumbent solution, found at iteration $iter_{best}$, and if at least one reset of the population has been carried out, then the algorithm stops (line 13-14). Every pop_{reset} iterations, without improvements of the incumbent solution, a reset of the populations is carried out and the flag $resetDone$ is set to true (line15-17). Finally, every $swap_{best}$ iterations the procedure $exchange\_best$ is invoked to exchange the best individuals between the populations (line 18-19). The last step of the while loop increases by one the iteration counter i (line 20). The algorithm returns the best individual found (line 21).

5. Computational Tests

In this section, we describe the results of BRKGA obtained during our computational test phase carried out on the SOP benchmark instances. Our algorithm was coded in C++ using the LEMON graph library [14] and the brkgaAPI [36]. All tests have been performed on an OSX platform (iMac late 2012), running on an Intel Core i7 2.8 GHz processor with 16 GB of RAM. The mathematical formulation was solved using the ILOG Concert Technology library and CPLEX 12.8.

The computational tests are carried out on 306 instances proposed in [2] and named $Set1$. This set of 306 instances was obtained by adapting 51 instances for the Generalized Traveling Salesman Problem proposed in [16]. These instances have a number of vertices ranging from 52 to 1084 and a number of clusters equal to $\sim 20\%$ of the number of vertices. T_{max} is set to $\omega \times GTSP^*$, where $GTSP^*$ is the best-known solution value of the GTSP (taken from [16]) and ω has been set to 0.4, 0.6, and 0.8. Finally, the profit associated with each cluster is assigned by using two different rules. The first rule sets the profit of each cluster C_g equal to $|C_g|$. The second rule sets the profit of a vertex j equal to $1 + (7141j + 73)mod(100)$ in order to obtain pseudo-random profits. The profit of a cluster is then obtained by summing up the profit of all the vertices belonging to it. In the following we call g_1 and g_2 the first and the second rule, respectively.

In [2] the authors generated a new set of instances named $Set2$. This new set of instances is equal to $Set1$ except for the generation of clusters. In particular, the number of clusters remains the same of $Set1$ but the vertices are randomly assigned to these clusters. The computational

Parameter	Value	Description
n	$\|\mathcal{P}\|$	Number of alleles per chromosome
p	$100 + 25\alpha$	Number of chromosomes in the population ($\alpha = 1$ if $\|V\| \geq 200$ and zero otherwise)
p_e	20%	Size of the elite set in the population
p_m	25%	Number of mutants to be introduced in the population at each generation
ρ_e	0.8	Probability that an allele is inherited from the elite parent
max_{it}	$150 + 0.3\|V\|$	Maximum number of iterations
pop_{reset}	$\frac{max_{it}}{3}$	number of iterations without improvement before resetting the populations
$swap_{best}$	$\frac{max_{it}}{6}$	number of iterations after which an exchange of best individuals among the populations is carried out

Table 1: The BRKGA parameters.

results of the three algorithms on $Set2$ are reported in Appendix A.

The values of the BRKGA parameters were chosen after a preliminary tuning phase carried out on Set1 and they are reported in Table 1. According to the literature [9, 17, 18, 19], the main parameters of BRKGA are usually chosen in the following sets: $p_e \in \{15\%, 20\%, 25\%\}$ of p, $p_m \in \{15\%, 20\%, 25\%\}$ of p, and $\rho_e \in \{0.6, 0.7, 0.8\}$. Among all the possible combinations of these values, we have chosen the ones that provided us the best results in terms of performance and effectiveness.

Finally, before starting the description of the results, there are some observations, concerning the instances with $\omega = 1$, that have to be reported. As described above, $Set1$ was obtained by adapting instances proposed in [16] for the GTSP. If an optimal solution T^* is known for GTSP, then it is possible to visit all the clusters paying a cost equal to $c(T^*)$. However, in the GTSP addressed in [16], the problem is solved without specifying a starting depot. On the contrary, for the SOP instances, the first vertex of the instance is always selected as depot v_0. This means that if v_0 does not belong to any optimal solution of GTSP, then the shortest tour visiting all the clusters and containing v_0 has a cost greater than $c(T^*)$. As a consequence, by setting $T_{max} = c(T^*)$ (i.e. $\omega = 1$), it is not guarantee that all the clusters can be visited by a tour, containing v_0, and then $\sum_{g=1}^{l} p_g$ is not necessary the optimal solution value for SOP (as erroneously reported in [2]) but it is surely an upper bound.

Moreover, for $Set2$ with $\omega = 0.8$, the T_{max} value is enough large to guarantee that, for all the instances but one, all the clusters can be visited. For this reason, it is useless to further increase to 1 the ω value and then no results will be reported in the following for this value.

Hash Hits						
	$\omega = 0.4$		$\omega = 0.6$		$\omega = 0.8$	
Instance	g_1	g_2	g_1	g_2	g_1	g_2
11berlin52	82.59%	82.93%	79.48%	79.84%	54.39%	89.09%
11eil51	81.94%	73.02%	62.50%	73.93%	56.02%	65.65%
14st70	65.60%	53.49%	53.18%	54.23%	55.48%	44.43%
16eil76	52.02%	84.47%	42.98%	44.30%	26.85%	39.28%
16pr76	52.51%	58.54%	41.24%	40.46%	34.18%	37.80%
20kroA100	46.95%	54.78%	37.20%	39.32%	27.75%	32.63%
20kroB100	49.70%	52.11%	38.52%	38.94%	32.28%	30.81%
20kroC100	59.26%	63.04%	37.36%	38.46%	32.29%	30.77%
20kroD100	51.76%	58.84%	21.44%	37.51%	31.04%	31.99%
20kroE100	65.75%	85.81%	39.19%	41.34%	27.73%	31.52%
20rat99	69.73%	65.94%	50.80%	57.83%	28.84%	36.95%
20rd100	48.62%	59.43%	36.56%	37.28%	32.23%	29.66%
21eil101	48.04%	53.23%	32.64%	33.98%	32.25%	22.93%
21lin105	67.68%	77.57%	45.90%	48.24%	30.52%	32.81%
22pr107	40.67%	53.17%	56.02%	74.44%	24.22%	39.21%
25pr124	39.72%	45.97%	34.29%	35.02%	26.67%	30.29%
26bier127	33.58%	37.24%	24.59%	24.56%	23.16%	19.95%
26ch130	43.78%	34.70%	29.73%	30.30%	23.54%	20.92%
28pr136	42.08%	47.36%	31.48%	29.94%	20.79%	23.08%
29pr144	41.31%	43.86%	26.45%	31.48%	23.97%	21.83%
30ch150	41.34%	41.59%	27.91%	21.99%	25.57%	22.83%
30kroA150	37.47%	38.93%	30.12%	23.31%	23.01%	19.57%
30kroB150	32.86%	36.98%	23.41%	29.70%	26.07%	21.80%
31pr152	47.60%	51.33%	27.85%	27.02%	27.10%	26.01%
32u159	30.63%	38.02%	27.03%	27.43%	19.09%	17.07%
39rat195	35.64%	34.82%	20.46%	25.84%	22.29%	23.48%
40d198	39.79%	44.08%	20.11%	24.09%	13.43%	17.81%
40kroa200	24.58%	24.73%	16.82%	16.31%	13.31%	13.62%
40krob200	21.18%	25.53%	18.54%	19.36%	15.73%	18.93%
45ts225	25.34%	23.91%	17.38%	16.19%	14.86%	15.94%
45tsp225	26.12%	26.90%	21.36%	25.30%	18.13%	17.71%
46pr226	24.65%	25.18%	30.03%	25.63%	18.21%	21.29%
53gil262	24.92%	28.89%	22.49%	18.50%	14.08%	15.58%
53pr264	18.12%	18.04%	25.59%	24.26%	16.84%	21.43%
56a280	19.72%	23.52%	21.88%	20.74%	16.16%	15.23%
60pr299	19.52%	19.80%	20.06%	23.06%	19.33%	16.49%
64lin318	22.55%	21.81%	18.68%	15.48%	12.17%	14.49%
80rd400	18.46%	18.12%	17.10%	16.60%	15.61%	14.86%
84fl417	13.93%	17.79%	16.36%	14.07%	9.37%	12.13%
88pr439	15.93%	18.04%	12.09%	14.33%	9.90%	13.43%
89pcb442	16.30%	17.69%	15.59%	14.69%	11.91%	13.80%
99d493	22.48%	24.54%	14.35%	16.77%	13.54%	13.05%
115rat575	17.97%	20.38%	14.33%	14.59%	12.06%	13.00%
115u574	20.14%	15.61%	12.58%	14.25%	10.79%	12.02%
131p654	10.31%	13.26%	13.23%	12.85%	6.51%	10.11%
132d657	20.71%	18.82%	14.10%	14.57%	11.01%	11.17%
145u724	15.73%	18.54%	11.81%	12.66%	11.50%	11.91%
157rat783	14.46%	15.70%	12.33%	12.14%	10.51%	10.92%
201pr1002	13.18%	14.06%	11.23%	11.40%	8.82%	9.35%
212u1060	12.04%	13.33%	10.02%	10.89%	8.58%	8.35%
217vm1084	12.27%	13.73%	8.81%	9.28%	7.74%	7.22%
Avg	**35.28%**	**38.22%**	**27.36%**	**28.72%**	**21.71%**	**23.38%**

Table 2: Number of times, in percentage, that a chromosome is found into the hashtable before invoking the decoder functions.

5.1 Hashtable, graph reduction algorithm and ICM effectiveness

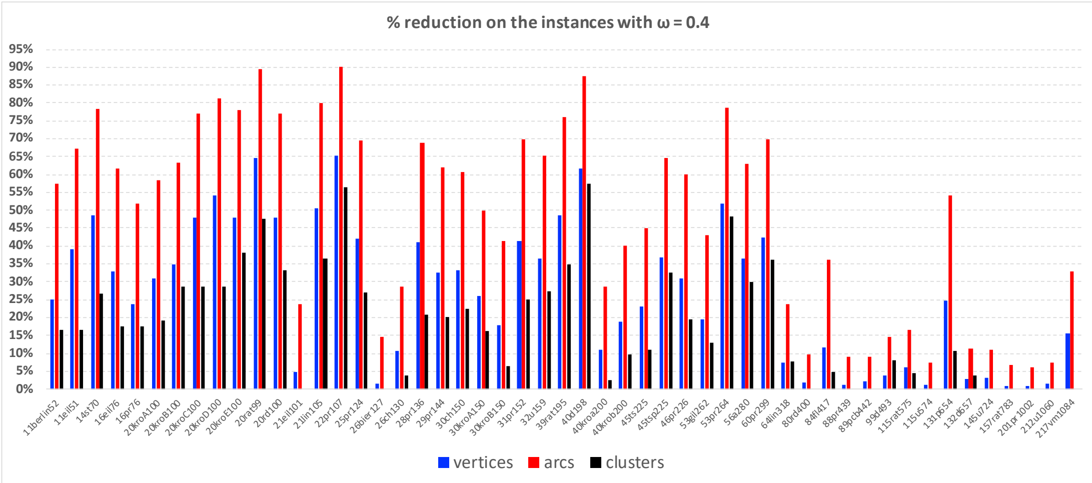
In this section, we evaluate the effectiveness of the hashtable, the reduction procedures, and the *ICM* strategy introduced in Section 3.

As reported in Section 4.2.1, the hashtable avoids invoking the decoder procedure for the chromosomes that have been already met during the evolutionary process. It is worth noting that the invocation of *Decoder* on a chromosome requires i) to build a number of graphs equal to the number of alleles greater than 0.5 and ii) to solve a shortest path problem on each of these graphs. Taking into account the number of chromosomes generated during the evolutionary process, it is easy to see that *Decoder* is one of the most expensive operations carried out by BRKGA. For this reason, the use of the hashtable is crucial to reduce as much as possible the number of invocations of this procedure.

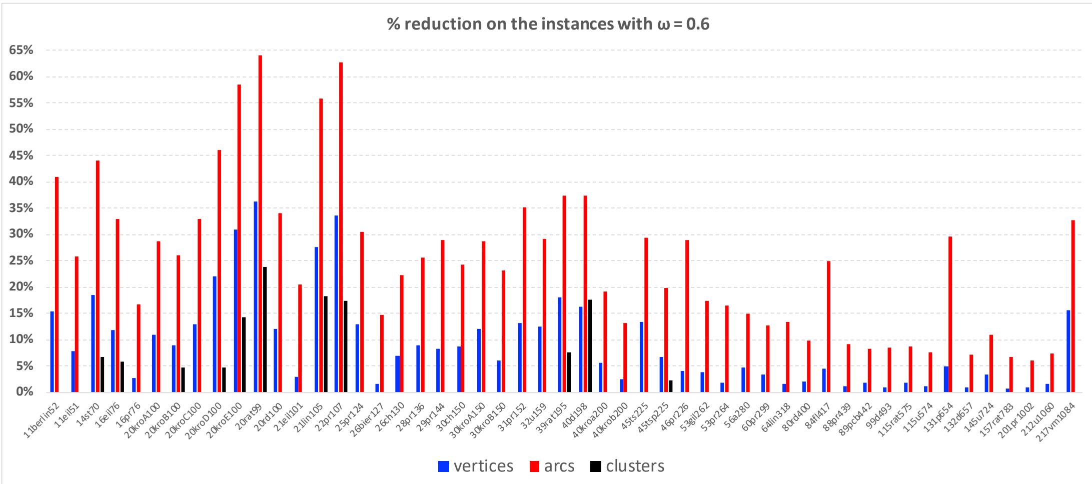
In Table 2 the number of avoided invocations of *Decoder*, because the chromosome is already into the hashtable, is reported, in percentage. The results on each row of the table are the average values obtained by performing 10 independent runs of the algorithm on each instance. This last statement holds for the next tables too. The table is vertically divided according to the T_{max} value (ω) and the type of profit used (g_1 and g_2). Under the *Instance* heading, we report the instance name. The percentage values are computed by using the formula: $100 \times \frac{numCrom - hits}{numCrom}$, where $numCrom$ is the number of chromosomes on which the decoder should be invoked and $hits$ is the number of times these chromosomes are already found into the hashtable. At the bottom, *Avg* reports the average percentage.
From the *Avg* values the effectiveness of the hashtable is evident because it reduces the number of invocations of *Decoder* from 20% to 38%. In particular, we observe that the lower is the ω value, the higher are the percentages. This occurs because the number of feasible solutions depends on the ω value and the lower is the number of feasible solutions the higher will be the number of hits obtained. By analyzing the single results, we observe that, generally, the percentage value decreases as the instance size increases because of a greater number of feasible solutions available. It is worth noting that there are instances where the percentage is greater than 80% and that only on 11 out of 306 instances this percentage is lower than 10%.

Regarding the graph reduction algorithm, in Figure 4 the percentage of vertices, arcs, and clusters removed during the preprocessing phase is shown for the instances with $\omega = 0.4$, $\omega = 0.6$ and $\omega = 0.8$, respectively.

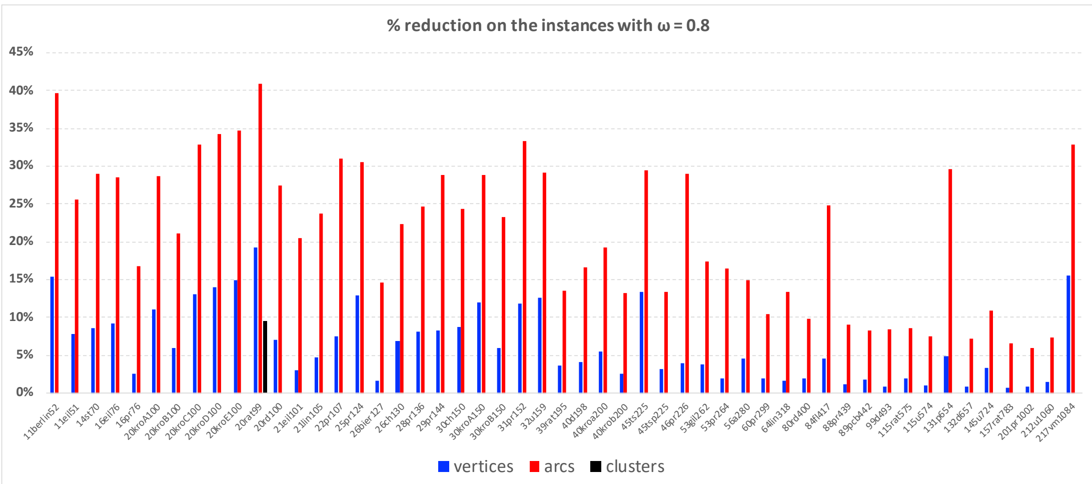
The instance name is reported on the x-axis while on the y-axis the percentage of reduction is shown. Three bars are associated with each instance representing the percentage of vertices (blue),



(a)



(b)



(c)

Figure 4: Percentage of vertices, arcs and clusters removed by the preprocessing phase on the instances with (a) $\omega = 0.4$, (b) $\omega = 0.6$ and (c) $\omega = 0.8$, respectively.

arcs (red) and clusters (black) removed for that instance.

On the instances with $\omega = 0.4$, Figure 4(a), the number of vertices removed is greater than 20% on 30 out of 51 instances and this percentage of reduction is, in particular, observed on the instances with up to 300 vertices (i.e. 60pr299). By considering all the instances, the average percentage of vertices removed is around 27%. Even more interesting are the results obtained on the removal of the arcs where an average of 48% is observed. On 39 out of 51 instances, this percentage is greater than 20% while on 11 instances this percentage exceeds the 75% with a peak equal to 90% on the instance 22pr107. Finally, the percentage of clusters removed is significantly lower than the other two parameters but remains relevant with an average equal to 18% and 20 instances where this percentage is greater than 20%.

The bars in Figure 4(b) reveal that the percentages of removal on the instances with $\omega = 0.6$ are lower with respect to the instances with $\omega = 0.4$. This behaviour was expected because lower is the T_{max} value greater is the chance of the preprocessing phase to find vertices, arcs and clusters useless. Anyway, on average, we observe a percentage of vertices, arcs and clusters removed equal to 9.1%, 25.3% and 2.4%, respectively. The threshold of 20% is reached on 5 instances for the vertex removal, and on 30 instances for the arc removal. Only in one case (20rat99) the percentage of clusters removed is greater than 20%.

The results in Figure 4(c) certify the trend observed in the previous charts with a further reduction of the removal percentages. On average, the percentage of vertices, arcs, and clusters removed is equal to 6.3%, 21.1%, and 0.19%. There are 28 instances where the percentage of arcs removed is greater than 20% while the percentage of vertices and clusters removed is always lower than this threshold.

Summarizing, the results of Figure 4 highlights the effectiveness of the reduction procedures, in particular, when $\omega = 0.4$. It is worth noting that, whatever is the ω value, the percentage of arcs removed remains relevant.

In order to evaluate the impact of the graph reduction procedures and of *ICM* strategy, on the performance of BRKGA, we implemented a version of BRKGA without the reduction procedures (*noRed*) and another version without the *ICM* matrix ($noICM$). The comparison of the computational time of these three algorithms is reported in Table 3.

	BRKGA without reductions or without *ICM*																	
	$\omega = 0.4$						$\omega = 0.6$						$\omega = 0.8$					
		g_1			g_2			g_1			g_2			g_1			g_2	
Instance	**BRKGA**	**noRed**	**noICM**	**BRKGA**	**noRed**	**noICM**	**BRKGA**	**noRed**	**noICM**	**BRKGA**	**noRed**	**noICM**	**BRKGA**	**noRed**	**noICM**	**BRKGA**	**noRed**	**noICM**
11berlin52	1.24	12.53%	7.19%	1.23	16.53%	8.55%	1.42	8.15%	10.68%	1.44	5.14%	5.14%	1.67	3.42%	4.98%	1.41	1.91%	3.68%
11eil51	1.26	5.38%	4.83%	1.34	6.92%	5.29%	1.51	1.92%	6.35%	1.45	1.73%	6.02%	1.65	1.94%	4.24%	1.58	1.27%	4.24%
14st70	1.40	9.05%	5.34%	1.48	9.78%	5.26%	1.72	3.14%	7.10%	1.70	3.24%	7.30%	2.02	3.47%	7.04%	1.99	2.01%	11.19%
16eil76	1.56	7.65%	12.60%	1.38	8.04%	11.15%	1.92	3.81%	7.99%	1.91	3.77%	8.49%	2.42	2.85%	6.87%	2.23	6.06%	9.16%
16pr76	1.62	7.24%	7.92%	1.58	5.52%	8.05%	2.04	1.18%	9.27%	2.05	1.07%	9.21%	2.49	1.20%	4.69%	2.42	6.57%	9.30%
20kroA100	1.74	14.04%	10.41%	1.68	14.42%	10.55%	2.12	3.54%	14.98%	2.11	3.46%	14.22%	2.62	6.87%	14.66%	2.45	2.21%	14.68%
20kroB100	1.70	10.21%	8.39%	1.68	10.46%	8.68%	2.11	3.41%	14.82%	2.11	3.23%	14.85%	2.50	1.64%	15.81%	2.53	1.54%	18.76%
20kroC100	1.59	13.77%	12.77%	1.57	13.00%	9.62%	2.14	3.60%	15.04%	2.12	2.79%	13.70%	2.46	3.90%	13.79%	2.47	2.99%	13.26%
20kroD100	1.61	14.13%	8.55%	1.58	12.74%	8.11%	2.29	5.60%	14.74%	2.11	4.88%	13.35%	2.43	4.60%	13.23%	2.45	9.02%	13.31%
20kroE100	1.52	20.68%	7.62%	1.42	15.88%	2.96%	2.03	11.08%	9.31%	2.01	8.81%	9.46%	2.45	5.02%	11.84%	2.51	4.38%	10.76%
20rat99	1.42	17.65%	4.15%	1.47	16.77%	4.28%	1.81	12.18%	10.31%	1.83	8.57%	5.78%	2.62	2.94%	8.93%	2.35	9.64%	20.43%
20rd100	1.70	14.20%	13.26%	1.63	13.29%	8.37%	2.23	3.46%	13.52%	2.23	1.75%	12.00%	2.86	0.00%	14.04%	2.79	0.00%	13.93%
21eil101	1.93	1.51%	14.13%	1.87	2.19%	15.31%	2.41	3.16%	12.29%	2.46	4.46%	14.29%	3.01	-1.46%	9.51%	2.83	11.11%	11.92%
21lin105	1.56	20.13%	5.83%	1.49	17.02%	5.16%	2.03	9.56%	9.41%	2.01	9.50%	9.25%	2.61	4.18%	13.72%	2.59	0.81%	12.40%
22pr107	1.76	28.47%	5.87%	1.65	27.87%	4.55%	1.87	8.67%	8.24%	1.72	7.69%	12.24%	2.41	2.16%	20.90%	2.24	1.88%	18.36%
25pr124	1.95	14.03%	11.62%	1.88	14.17%	11.20%	2.48	4.77%	16.97%	2.46	5.32%	17.80%	3.02	7.68%	17.72%	3.04	13.12%	19.92%
26bier127	2.95	4.33%	17.16%	3.15	0.35%	15.51%	3.70	6.93%	10.39%	3.95	0.68%	7.63%	4.32	-1.39%	8.68%	4.08	-2.63%	10.26%
26ch130	2.36	2.76%	26.41%	2.41	8.83%	21.96%	2.81	1.78%	28.61%	2.80	0.50%	25.70%	3.41	3.79%	21.65%	3.64	4.07%	20.71%
28pr136	1.96	12.23%	14.37%	1.91	12.17%	14.94%	2.71	4.14%	25.50%	2.76	3.27%	22.72%	3.56	9.94%	29.13%	3.43	6.45%	21.35%
29pr144	2.10	12.29%	17.62%	2.08	12.19%	16.32%	3.06	8.68%	35.46%	3.09	10.73%	31.48%	3.50	12.83%	21.21%	3.48	8.71%	22.26%
30ch150	2.05	12.73%	23.50%	2.11	13.06%	28.22%	2.94	3.37%	29.85%	3.32	1.87%	20.27%	3.76	7.68%	25.88%	3.54	8.34%	27.64%
30kroA150	2.23	9.37%	20.04%	2.27	9.36%	19.16%	2.90	4.55%	31.53%	2.97	2.60%	23.20%	3.69	0.35%	24.12%	3.71	10.96%	21.36%
30kroB150	2.43	6.67%	23.52%	2.38	6.02%	22.60%	3.49	0.77%	28.17%	3.58	6.36%	27.21%	3.91	2.84%	24.53%	3.92	6.60%	21.95%
31pr152	1.97	14.61%	16.34%	1.94	14.99%	15.40%	3.14	2.01%	22.18%	3.12	2.78%	24.01%	3.76	1.01%	20.55%	3.52	6.20%	26.67%
32u159	2.37	19.45%	19.11%	2.28	14.10%	17.14%	3.10	8.55%	25.97%	3.12	7.50%	23.72%	4.10	10.20%	20.36%	4.15	7.94%	19.16%
39rat195	2.45	26.15%	22.03%	2.42	27.91%	24.77%	3.71	8.17%	23.02%	3.95	8.19%	30.56%	5.00	0.98%	26.26%	5.08	0.67%	26.93%
40d198	1.94	31.56%	12.17%	1.90	30.98%	12.33%	4.46	23.75%	41.98%	4.92	10.84%	21.61%	5.24	3.49%	28.78%	5.03	4.51%	28.62%
40kroa200	3.12	29.53%	44.15%	3.11	31.28%	38.79%	4.72	40.40%	32.65%	4.83	27.62%	33.06%	6.16	20.30%	32.79%	6.39	32.15%	30.18%
40krob200	3.09	36.23%	32.57%	3.00	34.72%	31.29%	5.04	19.80%	35.47%	5.11	19.16%	37.39%	6.34	24.79%	40.01%	6.00	38.83%	32.91%
45ts225	3.15	37.54%	39.64%	3.49	42.41%	47.05%	5.09	39.34%	39.65%	5.28	36.74%	34.32%	6.44	39.03%	34.42%	6.78	40.21%	32.33%
45tsp225	2.71	50.13%	36.85%	2.72	50.33%	32.23%	5.95	6.74%	36.94%	5.49	3.72%	38.13%	8.15	0.42%	35.10%	8.59	3.35%	38.08%
46pr226	2.87	44.54%	29.05%	2.85	43.94%	28.94%	5.16	9.65%	52.86%	5.25	5.43%	45.30%	7.68	16.62%	36.23%	7.64	19.06%	39.63%
53gil262	5.06	8.48%	34.00%	5.03	10.04%	44.65%	8.01	2.73%	38.29%	8.15	8.41%	47.12%	10.80	11.82%	42.63%	11.59	6.28%	39.74%
53pr264	3.56	83.94%	18.48%	3.55	77.15%	18.54%	7.05	2.54%	47.03%	6.59	4.23%	51.34%	11.96	9.89%	33.19%	11.38	6.25%	43.48%
56a280	3.64	50.69%	44.42%	4.00	76.95%	41.21%	8.24	-8.52%	46.26%	8.01	2.56%	50.18%	12.48	-3.29%	33.90%	12.52	-0.68%	34.01%
60pr299	4.02	65.83%	36.11%	4.26	65.23%	33.40%	8.35	5.75%	70.05%	8.22	1.34%	48.10%	14.44	-0.88%	42.00%	14.27	2.40%	44.88%
64lin318	7.49	7.80%	60.06%	7.18	10.20%	55.59%	12.70	4.99%	52.40%	12.46	-2.15%	50.87%	16.95	2.77%	43.06%	16.23	10.79%	49.85%
80rd400	11.41	8.73%	96.31%	12.76	0.36%	79.84%	18.44	1.93%	63.61%	18.79	8.91%	48.92%	24.71	2.46%	47.86%	25.19	3.90%	47.24%
84fl417	11.10	21.54%	78.61%	12.43	12.00%	68.60%	15.60	12.25%	61.44%	16.42	17.38%	65.67%	24.36	16.77%	42.85%	26.48	17.62%	52.47%
88pr439	20.61	-0.81%	60.45%	22.85	-0.48%	45.49%	28.59	7.45%	51.50%	29.84	-0.12%	45.86%	33.66	3.30%	49.71%	34.51	-3.30%	44.93%
89pcb442	14.24	2.01%	88.53%	14.91	2.23%	79.51%	21.98	2.56%	66.28%	23.47	0.83%	53.57%	30.06	-3.43%	51.10%	29.84	6.25%	55.28%
99d493	16.12	4.12%	78.64%	15.21	12.20%	72.66%	36.15	-2.98%	50.09%	35.20	4.18%	49.53%	45.72	3.86%	48.27%	49.19	6.88%	42.43%
115rat575	20.56	7.31%	99.96%	21.88	5.48%	89.76%	34.82	3.68%	77.08%	36.29	3.01%	69.51%	47.94	4.44%	59.99%	48.88	2.90%	60.01%
115u574	22.66	2.51%	99.11%	23.32	-1.92%	110.97%	40.98	1.31%	69.41%	40.54	2.22%	67.64%	55.65	1.21%	54.79%	56.14	2.83%	51.51%
131p654	25.53	35.87%	78.31%	25.67	32.98%	76.62%	48.26	16.53%	65.81%	46.93	11.15%	56.75%	79.01	22.15%	61.67%	79.63	22.43%	60.25%
132d657	27.65	-0.33%	94.26%	27.51	2.56%	100.02%	51.98	2.45%	61.71%	50.31	4.61%	79.62%	72.99	3.13%	57.68%	72.91	5.36%	55.33%
145u724	35.69	2.79%	113.96%	36.05	-0.76%	108.15%	59.95	2.67%	80.33%	60.15	5.32%	79.86%	86.01	2.06%	57.49%	83.79	4.47%	61.19%
157rat783	42.93	-2.41%	121.86%	44.28	-1.77%	116.81%	69.64	0.33%	88.87%	70.60	0.06%	86.20%	100.67	-0.63%	61.21%	102.78	-2.06%	60.29%
201pr1002	88.97	-2.54%	122.66%	89.40	2.41%	125.17%	156.37	-0.09%	71.79%	155.51	4.33%	75.29%	213.96	1.56%	57.68%	216.35	0.68%	57.35%
212u1060	108.07	3.97%	114.95%	110.62	-0.01%	118.25%	185.13	4.23%	72.73%	184.72	5.39%	77.55%	247.62	2.99%	62.18%	248.01	5.83%	62.00%
217vm1084	97.41	35.01%	91.73%	95.07	42.02%	106.53%	144.19	51.32%	75.67%	146.62	57.00%	74.10%	195.57	53.25%	57.28%	186.99	50.76%	67.06%
Avg		**17.55%**	**40.54%**		**17.73%**	**39.32%**		**7.59%**	**37.09%**		**7.10%**	**35.24%**		**6.60%**	**30.91%**		**8.23%**	**31.66%**

Table 3: Impact on the BRKGA performance of the reduction procedures and *ICM* strategy. The gap percentage reports how much slower is BRKGA without these components.

The table is vertically divided according to the T_{max} value (ω) and the profit used (g_1 and g_2). Under the *Instance*, we report the instance name while under the heading *BRKGA* it is reported the computational time, in seconds, of this algorithm. The other two columns, *noRed* and *noICM*, report the percentage gap of their computational time with respect to the computational time of BRKGA, respectively. The percentage gap is positive when BRKGA is faster and negative otherwise. At the bottom, *Avg* reports the average percentage. The values of the *Avg* line show that, with $\omega = 0.4$, the version of BRKGA without the reduction procedures is 17.5% slower with the g_1 profit and 17.7% slower with g_2 profit. There are instances where this gap is greater than 50% but there are even instances where *noRed* results faster. This can happen because of the overhead generated by the cost of the reduction procedures but even because of the aspiration criterion that can significantly reduce the total number of iterations carried out by the algorithms if the

best solution is found soon. Usually, the effectiveness of the reduction procedures decreases as the ω value increases (Figure 4), and, indeed, the average percentage gaps are lower when $\omega = 0.6$ and $\omega = 0.8$. In particular, with $\omega = 0.6$ the average percentage gap is equal to 7.5% and 7.1%, respectively while, with $\omega = 0.8$, the values are 6.6% and 8.2%.
Much more relevant is the impact of *ICM* strategy on the performance of BRKGA with respect to the reduction procedures. Indeed, from the *Avg* line, we can see that whatever are the values of ω and the type of profits considered, *noICM* is at least 30% slower than BRKGA and, in the worst case, the average gap is 40.5%. The detailed results show that it is never faster than BRKGA while on 192 out of 306 instances BRKGA results at least 20% faster than *noICM* and, in eleven instances, the gap is greater than 100%. These results highlight that the idea to save information rather than recompute them several times is effective and significantly reduces the computational time of BRKGA.

The results of Table 3 have shown the effectiveness of the reduction procedures and *ICM* strategy. However, it is interesting to know also the cost that we pay, in terms of computational time, to apply these strategies. This information is reported in Table 4 where the percentage computational time used for the reduction procedures and for the *ICM* strategy is shown. For readability reasons, the table reports only the results for g_1 profit because the results for g_2 are very similar. The table is vertically divided according to the T_{max} value (ω). Under the *Instance* heading, we report the instance name while under the *BRKGA* heading the computational time, in seconds, of this algorithm is reported. The other two columns, *Reductions* and *ICM*, show the percentage of time spent on the reduction procedures and to build the *ICM* matrix, respectively. Finally, the last line of the table reports the average percentage (*Avg*). The values in this last line show that, on average, the computational time required by the reduction procedures is lower than 1%, with respect to the total computational time, while the time spent for building the *ICM* matrix is lower than 2.14%. It is worth noting that, even analyzing the single results, the reduction procedures appear cheap because they never exceed the 4.45% of the total computational time. Much more time is, usually, required by *ICM* strategy that, in three cases, has required over 10% of the total time. Anyway, on 140 out of 153 instances this percentage is lower than or equal to 5%. Summarizing, according to the results shown in Table 3, the computational cost paid for these

Computational time of the reduction procedures and the *ICM* strategy									
	$\omega = 0.4$			$\omega = 0.6$			$\omega = 0.8$		
Instance	**BRKGA**	**Reductions**	**ICM**	**BRKGA**	**Reductions**	**ICM**	**BRKGA**	**Reductions**	**ICM**
11berlin52	1.24	0.00%	0.00%	1.42	0.00%	0.00%	1.67	0.00%	0.00%
11eil51	1.26	0.00%	0.00%	1.51	0.00%	0.00%	1.65	0.00%	0.00%
14st70	1.40	0.00%	0.00%	1.72	0.00%	0.00%	2.02	0.00%	0.00%
16eil76	1.56	0.00%	0.00%	1.92	0.00%	0.00%	2.42	0.00%	0.00%
16pr76	1.62	0.00%	0.00%	2.04	0.00%	0.00%	2.49	0.00%	0.00%
20kroA100	1.74	0.00%	0.00%	2.12	0.00%	0.00%	2.62	0.00%	0.00%
20kroB100	1.70	0.00%	0.00%	2.11	0.00%	0.00%	2.50	0.00%	0.24%
20kroC100	1.59	0.00%	0.00%	2.14	0.00%	0.00%	2.46	0.00%	0.00%
20kroD100	1.61	0.00%	0.00%	2.29	0.00%	0.00%	2.43	0.00%	0.00%
20kroE100	1.52	0.00%	0.00%	2.03	0.00%	0.00%	2.45	0.00%	0.00%
20rat99	1.42	0.00%	0.00%	1.81	0.00%	0.00%	2.62	0.00%	0.00%
20rd100	1.70	0.00%	0.00%	2.23	0.00%	0.00%	2.86	0.00%	0.00%
21eil101	1.93	0.00%	0.10%	2.41	0.00%	0.25%	3.01	0.00%	0.03%
21lin105	1.56	0.00%	0.00%	2.03	0.00%	0.00%	2.61	0.00%	0.27%
22pr107	1.76	0.00%	0.00%	1.87	0.00%	0.00%	2.41	0.00%	0.00%
25pr124	1.95	0.00%	0.00%	2.48	0.00%	0.40%	3.02	0.33%	0.33%
26bier127	2.95	0.34%	0.34%	3.70	0.27%	0.27%	4.32	0.23%	0.23%
26ch130	2.36	0.42%	0.42%	2.81	0.36%	0.36%	3.41	0.29%	0.29%
28pr136	1.96	0.00%	0.51%	2.71	0.37%	0.37%	3.56	0.28%	0.28%
29pr144	2.10	0.00%	0.48%	3.06	0.33%	0.33%	3.50	0.29%	0.29%
30ch150	2.05	0.39%	0.49%	2.94	0.34%	0.68%	3.76	0.27%	0.45%
30kroA150	2.23	0.45%	0.45%	2.90	0.34%	0.65%	3.69	0.27%	0.52%
30kroB150	2.43	0.41%	0.41%	3.49	0.29%	0.54%	3.91	0.26%	0.51%
31pr152	1.97	0.00%	0.51%	3.14	0.32%	0.32%	3.76	0.27%	0.27%
32u159	2.37	0.00%	0.42%	3.10	0.32%	0.32%	4.10	0.24%	0.27%
39rat195	2.45	0.00%	0.41%	3.71	0.35%	0.57%	5.00	0.40%	0.60%
40d198	1.94	0.00%	0.52%	4.46	0.22%	0.45%	5.24	0.38%	0.57%
40kroa200	3.12	0.74%	0.99%	4.72	0.64%	0.85%	6.16	0.49%	0.65%
40krob200	3.09	0.65%	0.97%	5.04	0.60%	0.79%	6.34	0.47%	0.63%
45ts225	3.15	0.63%	0.98%	5.09	0.59%	0.79%	6.44	0.47%	0.62%
45tsp225	2.71	0.37%	0.74%	5.95	0.50%	0.84%	8.15	0.38%	0.61%
46pr226	2.87	0.35%	1.01%	5.16	0.58%	0.77%	7.68	0.39%	0.52%
53gil262	5.06	0.79%	1.27%	8.01	0.85%	1.19%	10.80	0.59%	0.87%
53pr264	3.56	0.28%	0.56%	7.05	0.71%	1.14%	11.96	0.42%	0.67%
56a280	3.64	0.55%	1.10%	8.24	0.73%	1.21%	12.48	0.52%	0.83%
60pr299	4.02	0.50%	1.02%	8.35	0.90%	1.54%	14.44	0.53%	0.90%
64lin318	7.49	1.07%	1.75%	12.70	0.72%	1.42%	16.95	0.54%	1.03%
80rd400	11.41	2.12%	3.87%	18.44	1.29%	2.37%	24.71	0.98%	1.76%
84fl417	11.10	1.54%	2.63%	15.60	1.35%	2.28%	24.36	0.84%	1.43%
88pr439	20.61	1.19%	2.61%	28.59	0.85%	1.84%	33.66	0.72%	1.57%
89pcb442	14.24	1.90%	4.03%	21.98	1.21%	2.66%	30.06	0.89%	1.98%
99d493	16.12	2.10%	4.67%	36.15	1.02%	2.53%	45.72	0.81%	2.01%
115rat575	20.56	2.38%	6.50%	34.82	1.63%	4.22%	47.94	1.18%	3.07%
115u574	22.66	2.40%	6.41%	40.98	1.33%	3.55%	55.65	0.99%	2.65%
131p654	25.53	1.59%	4.12%	48.26	1.60%	3.45%	79.01	0.97%	2.10%
132d657	27.65	3.00%	7.84%	51.98	1.67%	4.49%	72.99	1.19%	3.20%
145u724	35.69	3.27%	8.37%	59.95	1.95%	4.98%	86.01	1.35%	3.47%
157rat783	42.93	3.76%	9.48%	69.64	2.31%	5.84%	100.67	1.60%	4.05%
201pr1002	88.97	3.31%	11.77%	156.37	1.87%	6.67%	213.96	1.37%	4.87%
212u1060	108.07	3.14%	11.20%	185.13	1.83%	6.54%	247.62	1.37%	4.89%
217vm1084	97.41	4.45%	10.20%	144.19	2.97%	6.78%	195.57	2.19%	5.00%
Avg		**0.86%**	**2.14%**		**0.65%**	**1.46%**		**0.49%**	**1.07%**

Table 4: Percentage of computational time dedicated by BRKGA to the reduction procedures and the building of *ICM* matrix.

reduction procedures and *ICM* strategy is widely repaid by the speed up gained by BRKGA.

5.2 *The BRKGA effectiveness and performance*

In this section, we verify the effectiveness and the performance of BRKGA by comparing it with the matheuristics MASOP proposed in [2], the VNS-SOP metaheuristic proposed in [27] and the best-known solutions, available in the literature. All the algorithms run on the same machine so that their CPU times are directly comparable. Obviously, the computational time of BRKGA includes the time required by the removal procedures and the time spent to build the *ICM* matrix. The results of this comparison, when $\omega = 0.4$ and the profit is g_1, are shown in Table 5.

In order to verify the stability of the algorithms, the results on each row of the table are obtained by performing 10 independent runs of the algorithms on each instance. We report the instance name under the *Instance* heading and the best solution value, obtained by selecting the maximum value between the best value provided by the three algorithms and the best-known solution value, available in the literature, under the *Best* heading. The Best value is reported in bold whenever the best-known solution value is improved by one of the three algorithms. The next twelve headings report the best solution value (*Sol*), the average solution value (Sol_{avg}), the average computational time ($Time$), in seconds, and the percentage gap (*Gap*) between the *Best* value and *Sol* value of MASOP, VNS-SOP and BRKGA, respectively. More in detail, this gap is computed by using the formula: $100 \times \frac{Best-Sol}{Best}$. The Sol values are in bold whenever they coincide with the Best values. At the bottom, *Avg* reports the average computational time and the average gap of the algorithms. *#Best* reports the number of times that an algorithm finds the *Best* solution value and *Dev.st%* reports the percentage standard deviation computed on the percentage gap between Sol and Sol_{avg}.

From the values of #Best line, we note that MASOP finds 40 times the best solution and one of these times it provides a new best solution for the instance 115u574. Instead, the best solution is found 41 times by VNS-SOP that, in three cases (40d198, 80rd400, and 115rat575), improves the best-known solution. The best results are obtained by BRKGA with 42 best-known solutions found, and two of them (115rat575 and 157rat783) are new ones. Moreover, BRKGA shows the lowest average gap value equal to 0.19% but MASOP is very close with a value equal to 0.20%. These gap values show that these two algorithms provide solutions very close to the best-known ones. VNS-SOP is less effective with respect to the other algorithms because its average gap value is equal to 0.65%. Regarding the performance, we observe a significant difference between the

Set1						**$\omega = 0.4$ and g_1**								
		MASOP				**VNS-SOP**				**BRKGA**				
Instance	**Best**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**
11berlin52	37	**37**	37.0	1.86	0.00%	**37**	37.0	0.28	0.00%	**37**	37.0	1.24	0.00%
11eil51	24	**24**	24.0	1.98	0.00%	**24**	24.0	0.26	0.00%	**24**	24.0	1.26	0.00%
14st70	33	**33**	33.0	4.46	0.00%	**33**	33.0	0.36	0.00%	**33**	33.0	1.40	0.00%
16eil76	40	**40**	40.0	3.48	0.00%	**40**	40.0	0.49	0.00%	**40**	40.0	1.56	0.00%
16pr76	47	**47**	47.0	6.06	0.00%	**47**	47.0	0.55	0.00%	**47**	47.0	1.62	0.00%
20kroA100	42	**42**	42.0	6.29	0.00%	**42**	41.6	0.81	0.00%	**42**	42.0	1.74	0.00%
20kroB100	49	**49**	49.0	5.42	0.00%	**49**	49.0	0.68	0.00%	**49**	49.0	1.70	0.00%
20kroC100	42	**42**	42.0	5.80	0.00%	**42**	42.0	0.68	0.00%	**42**	42.0	1.59	0.00%
20kroD100	39	**39**	39.0	6.42	0.00%	**39**	39.0	0.63	0.00%	**39**	39.0	1.61	0.00%
20kroE100	52	**52**	52.0	7.77	0.00%	**52**	52.0	0.69	0.00%	**52**	52.0	1.52	0.00%
20rat99	37	**37**	37.0	4.70	0.00%	**37**	37.0	0.57	0.00%	**37**	37.0	1.42	0.00%
20rd100	45	**45**	45.0	6.82	0.00%	**45**	45.0	0.68	0.00%	**45**	45.0	1.70	0.00%
21eil101	67	**67**	67.0	6.41	0.00%	**67**	67.0	0.95	0.00%	**67**	67.0	1.93	0.00%
21lin105	50	**50**	50.0	7.62	0.00%	**50**	50.0	0.71	0.00%	**50**	50.0	1.56	0.00%
22pr107	41	**41**	41.0	8.73	0.00%	**41**	41.0	0.61	0.00%	**41**	41.0	1.76	0.00%
25pr124	46	**46**	46.0	9.00	0.00%	**46**	46.0	1.00	0.00%	**46**	46.0	1.95	0.00%
26bier127	110	**110**	110.0	8.53	0.00%	**110**	110.0	1.93	0.00%	**110**	110.0	2.95	0.00%
26ch130	70	**70**	69.8	9.76	0.00%	**70**	70.0	1.22	0.00%	**70**	70.0	2.36	0.00%
28pr136	53	**53**	53.0	7.61	0.00%	**53**	53.0	1.34	0.00%	**53**	53.0	1.96	0.00%
29pr144	60	**60**	60.0	9.47	0.00%	**60**	60.0	1.20	0.00%	**60**	60.0	2.10	0.00%
30ch150	61	**61**	59.8	2.35	0.00%	**61**	61.0	1.60	0.00%	**61**	61.0	2.05	0.00%
30kroA150	58	**58**	58.0	9.02	0.00%	**58**	58.0	1.67	0.00%	**58**	58.0	2.23	0.00%
30kroB150	66	**66**	66.0	7.88	0.00%	**66**	65.9	1.58	0.00%	**66**	66.0	2.43	0.00%
31pr152	57	**57**	57.0	8.95	0.00%	**57**	57.0	1.11	0.00%	**57**	57.0	1.97	0.00%
32u159	76	**76**	76.0	11.93	0.00%	**76**	76.0	2.07	0.00%	**76**	76.0	2.37	0.00%
39rat195	71	**71**	71.0	13.53	0.00%	**71**	71.0	2.32	0.00%	**71**	71.0	2.45	0.00%
40d198	**70**	67	67.0	9.99	4.29%	**70**	70.0	1.88	0.00%	67	67.0	1.94	4.29%
40kroa200	92	**92**	92.0	14.36	0.00%	**92**	92.0	2.93	0.00%	**92**	92.0	3.12	0.00%
40krob200	87	**87**	87.0	10.79	0.00%	**87**	87.0	3.00	0.00%	**87**	87.0	3.09	0.00%
45ts225	101	**101**	101.0	11.76	0.00%	**101**	101.0	4.20	0.00%	**101**	101.0	3.15	0.00%
45tsp225	80	**80**	80.0	12.86	0.00%	**80**	80.0	3.25	0.00%	**80**	80.0	2.71	0.00%
46pr226	86	**86**	85.4	15.04	0.00%	**86**	86.0	2.82	0.00%	**86**	86.0	2.87	0.00%
53gil262	105	**105**	104.8	19.42	0.00%	**105**	104.2	5.54	0.00%	**105**	104.7	5.06	0.00%
53pr264	128	**128**	128.0	36.52	0.00%	**128**	128.0	5.21	0.00%	127	127.0	3.56	0.78%
56a280	107	**107**	107.0	19.05	0.00%	106	105.1	5.59	0.93%	**107**	107.0	3.64	0.00%
60pr299	131	**131**	131.0	29.18	0.00%	**131**	130.6	6.72	0.00%	**131**	131.0	4.02	0.00%
64lin318	169	**169**	169.0	28.91	0.00%	**169**	168.8	9.28	0.00%	**169**	169.0	7.49	0.00%
80rd400	**209**	208	206.5	65.05	0.48%	**209**	208.2	16.41	0.00%	208	207.6	11.41	0.48%
84fl417	201	**201**	175.5	48.04	0.00%	**201**	201.0	14.48	0.00%	**201**	200.6	11.10	0.00%
88pr439	335	334	333.9	72.11	0.30%	333	332.4	33.97	0.60%	334	329.0	20.61	0.30%
89pcb442	224	223	222.2	65.88	0.45%	223	220.9	24.85	0.45%	**224**	221.9	14.24	0.00%
99d493	278	**278**	277.7	92.11	0.00%	**278**	275.9	25.43	0.00%	**278**	278.0	16.12	0.00%
115rat575	**264**	262	259.5	191.49	0.76%	**264**	258.9	36.52	0.00%	**264**	260.9	20.56	0.00%
115u574	**275**	**275**	270.2	122.31	0.00%	270	265.1	39.34	1.82%	274	269.7	22.66	0.36%
131p654	315	**315**	315.0	193.36	0.00%	**315**	315.0	39.78	0.00%	**315**	313.0	25.53	0.00%
132d657	308	304	303.0	246.39	1.30%	300	295.0	46.75	2.60%	305	299.3	27.65	0.97%
145u724	326	324	321.8	173.22	0.61%	315	303.1	64.94	3.37%	323	317.0	35.69	0.92%
157rat783	**349**	348	344.9	225.44	0.29%	346	331.2	80.92	0.86%	**349**	342.8	42.93	0.00%
201pr1002	536	535	532.6	652.11	0.19%	519	491.4	149.37	3.17%	535	507.2	88.97	0.19%
212u1060	563	559	532.8	723.77	0.71%	535	521.2	187.21	4.97%	**563**	538.0	108.07	0.00%
217vm1084	672	666	656.9	463.80	0.89%	574	551.9	227.34	14.58%	663	655.6	97.41	1.34%
Avg				**73.04**	**0.20%**			**20.86**	**0.65%**			**12.39**	**0.19%**
#Best					**40**				**41**				**42**
Dev.st%			**1.90%**				**1.25%**				**1.05%**		

Table 5: Comparison among MASOP, VNS-SOP and BRKGA on $Set1$ instances with $\omega = 0.4$ and g_1 profit.

average computational time of MASOP and of BRKGA: the former algorithm requires 73 seconds, while the latter only 12 seconds. Moreover, in the worst case, MASOP needs 723 seconds while

Set1				**$\omega = 0.4$** and **g_2**									
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	**Sol_{avg}**	**Time**	**Gap%**	**Sol**	**Sol_{avg}**	**Time**	**Gap%**	**Sol**	**Sol_{avg}**	**Time**	**Gap%**
11berlin52	1829	**1829**	1829.0	1.87	0.00%	**1829**	1829.0	0.29	0.00%	**1829**	1829.0	1.23	0.00%
11eil51	1279	**1279**	1279.0	2.09	0.00%	**1279**	1279.0	0.26	0.00%	**1279**	1279.0	1.34	0.00%
14st70	1672	**1672**	1672.0	4.39	0.00%	**1672**	1672.0	0.40	0.00%	**1672**	1672.0	1.48	0.00%
16eil76	2223	**2223**	2223.0	4.92	0.00%	**2223**	2223.0	0.51	0.00%	**2223**	2223.0	1.38	0.00%
16pr76	2449	**2449**	2449.0	5.96	0.00%	**2449**	2449.0	0.56	0.00%	**2449**	2449.0	1.58	0.00%
20kroA100	2151	**2151**	2151.0	6.42	0.00%	**2151**	2151.0	0.66	0.00%	**2151**	2151.0	1.68	0.00%
20kroB100	2431	2202	2202.0	2.14	9.42%	**2431**	2431.0	0.66	0.00%	**2431**	2431.0	1.68	0.00%
20kroC100	2174	**2174**	2174.0	5.57	0.00%	**2174**	2174.0	0.67	0.00%	**2174**	2174.0	1.57	0.00%
20kroD100	1740	**1740**	1740.0	6.07	0.00%	**1740**	1740.0	0.63	0.00%	**1740**	1740.0	1.58	0.00%
20kroE100	2415	**2415**	2415.0	7.62	0.00%	**2415**	2415.0	0.69	0.00%	**2415**	2415.0	1.42	0.00%
20rat99	1905	**1905**	1905.0	4.76	0.00%	**1905**	1905.0	0.56	0.00%	**1905**	1905.0	1.47	0.00%
20rd100	2228	**2228**	2228.0	6.54	0.00%	**2228**	2228.0	0.69	0.00%	**2228**	2228.0	1.63	0.00%
21eil101	3365	**3365**	3365.0	7.10	0.00%	**3365**	3365.0	0.99	0.00%	**3365**	3365.0	1.87	0.00%
21lin105	2489	**2489**	2489.0	7.94	0.00%	**2489**	2489.0	0.72	0.00%	**2489**	2489.0	1.49	0.00%
22pr107	2123	**2123**	2123.0	8.42	0.00%	**2123**	2123.0	0.63	0.00%	**2123**	2123.0	1.65	0.00%
25pr124	2302	**2302**	2302.0	7.29	0.00%	**2302**	2302.0	1.08	0.00%	**2302**	2302.0	1.88	0.00%
26bier127	5420	**5420**	5420.0	10.35	0.00%	**5420**	5420.0	1.94	0.00%	**5420**	5420.0	3.15	0.00%
26ch130	3423	**3423**	3412.2	6.84	0.00%	**3423**	3423.0	1.36	0.00%	**3423**	3419.4	2.41	0.00%
28pr136	2699	**2699**	2699.0	7.80	0.00%	**2699**	2691.6	1.38	0.00%	**2699**	2699.0	1.91	0.00%
29pr144	3055	**3055**	3055.0	10.13	0.00%	**3055**	3055.0	1.22	0.00%	**3055**	3055.0	2.08	0.00%
30ch150	**3131**	3078	3076.1	9.89	1.69%	**3131**	3131.0	1.54	0.00%	3078	3078.0	2.11	1.69%
30kroA150	3039	**3039**	3039.0	10.01	0.00%	**3039**	3039.0	1.61	0.00%	**3039**	3039.0	2.27	0.00%
30kroB150	3172	**3172**	3172.0	9.02	0.00%	**3172**	3172.0	1.67	0.00%	**3172**	3172.0	2.38	0.00%
31pr152	2915	**2915**	2915.0	9.15	0.00%	**2915**	2915.0	1.12	0.00%	**2915**	2915.0	1.94	0.00%
32u159	4002	**4002**	4002.0	10.41	0.00%	**4002**	4000.8	1.95	0.00%	**4002**	4002.0	2.28	0.00%
39rat195	3656	**3656**	3656.0	14.73	0.00%	**3656**	3656.0	2.33	0.00%	**3656**	3656.0	2.42	0.00%
40d198	**3595**	3400	3400.0	10.30	5.42%	**3595**	3595.0	1.96	0.00%	3400	3400.0	1.90	5.42%
40kroa200	4550	**4550**	4550.0	15.76	0.00%	**4550**	4550.0	3.05	0.00%	**4550**	4550.0	3.11	0.00%
40krob200	4348	**4348**	4348.0	12.58	0.00%	**4348**	4348.0	3.07	0.00%	**4348**	4348.0	3.00	0.00%
45ts225	5037	**5037**	5037.0	18.16	0.00%	**5037**	4945.0	4.63	0.00%	**5037**	5037.0	3.49	0.00%
45tsp225	4297	**4297**	4297.0	16.11	0.00%	**4297**	4297.0	3.16	0.00%	**4297**	4297.0	2.72	0.00%
46pr226	4403	**4403**	4394.6	14.74	0.00%	**4403**	4403.0	2.68	0.00%	**4403**	4403.0	2.85	0.00%
53gil262	5330	**5330**	5330.0	20.92	0.00%	**5330**	5284.9	5.76	0.00%	**5330**	5330.0	5.03	0.00%
53pr264	6423	**6423**	6423.0	38.43	0.00%	**6423**	6423.0	5.35	0.00%	6397	6383.5	3.55	0.40%
56a280	5630	**5630**	5611.3	23.15	0.00%	**5630**	5605.0	6.30	0.00%	**5630**	5606.3	4.00	0.00%
60pr299	6600	**6600**	6600.0	31.91	0.00%	**6600**	6600.0	6.91	0.00%	**6600**	6600.0	4.26	0.00%
64lin318	8600	**8600**	8600.0	35.88	0.00%	**8600**	8593.1	7.19	0.00%	**8600**	8600.0	7.18	0.00%
80rd400	**10966**	10825	10752.3	59.59	1.29%	**10966**	10858.8	16.91	0.00%	10874	10816.6	12.76	0.84%
84fl417	**10163**	10133	9375.4	46.73	0.30%	**10163**	10119.6	14.93	0.00%	10133	10123.4	12.43	0.30%
88pr439	**17082**	17037	17013.5	121.16	0.26%	17022	16804.6	31.13	0.35%	**17082**	16944.6	22.85	0.00%
89pcb442	11627	11565	11546.3	79.99	0.53%	**11627**	11502.1	22.27	0.00%	**11627**	11587.6	14.91	0.00%
99d493	14131	**14131**	14083.3	83.23	0.00%	13992	13992.0	25.16	0.98%	**14131**	14131.0	15.21	0.00%
115rat575	**13367**	13300	13203.0	296.66	0.50%	13203	12950.8	41.04	1.23%	**13367**	13293.0	21.88	0.00%
115u574	**14046**	13969	13739.6	124.07	0.55%	13782	13407.2	37.37	1.88%	**14046**	13823.3	23.32	0.00%
131p654	16003	**16003**	16003.0	260.42	0.00%	15983	15958.9	35.53	0.12%	**16003**	15902.5	25.67	0.00%
132d657	**15903**	15855	15790.2	311.43	0.30%	15141	14633.2	46.81	4.79%	**15903**	15738.4	27.51	0.00%
145u724	**17230**	**17230**	17127.2	216.07	0.00%	16616	15998.6	69.03	3.56%	17094	16974.3	36.05	0.79%
157rat783	**18216**	18018	17720.8	315.26	1.09%	17889	17088.8	73.92	1.80%	**18216**	17838.8	44.28	0.00%
201pr1002	27275	26239	26086.2	766.51	3.80%	25357	24632.4	149.33	7.03%	27189	26369.8	89.40	0.32%
212u1060	**28221**	27348	26638.9	771.67	3.09%	26463	25952.5	181.95	6.23%	**28221**	27458.3	110.62	0.00%
217vm1084	**34398**	34291	33818.5	710.76	0.31%	31473	27875.1	217.37	8.50%	**34398**	33799.6	95.07	0.00%
Avg				**89.98**	**0.56%**			**20.37**	**0.72%**			**12.57**	**0.19%**
#Best					**37**				**40**				**44**
Dev.st%			**1.14%**				**1.88%**				**0.70%**		

Table 6: Comparison among MASOP, VNS-SOP and BRKGA on $Set1$ instances with $\omega = 0.4$ and g_2 profit.

BRKGA never requires more than 108 seconds. With an average time of 20 seconds, also VNS-SOP results much faster than MASOP but slower than BRKGA. Finally, BRKGA is the more stable algorithm on these instances with a Dev.st% value equal to 1.05%, followed by VNS-SOP with 1.25%, and MASOP with 1.90%.

In Table 6 the results with $\omega = 0.4$ and g_2 profit are reported. In these instances, we observe a better behaviour of BRKGA with respect to the other two algorithms. The #Best value of BRKGA is equal to 44, whereas for VNS-SOP and MASOP it is equal to 40 and 37. Moreover, the best-known solutions are improved 8, 5, and 4 times by BRKGA, MASOP and VNS-SOP, respectively. It is worth noting that, in these instances, the average gap value of BRKGA is equal to 0.19% and that only in two cases its percentage gap is over 1%. Less effective are MASOP and VNS-SOP that show an average gap value equal to 0.56% and 0.72%, respectively. Moreover, the Gap% value is greater than 1% seven times for MASOP and 8 times for VNS-SOP. Regarding the performance, it is interesting to observe that the computational time of BRKGA and VNS-SOP are very similar to the values reported in Table 5. This means that the performance of these two algorithms is not affected by profit considered. We will see that this trend will be confirmed even in the other tables. On the contrary, with g_2 profit, the computational time of MASOP significantly increases, passing from 73 to 89.9 seconds with a peak of 771 seconds. The most stable algorithm results again BRKGA with a Dev.st% value equal to 0.70% while for MASOP and VNS-SOP this value is equal to 1.14% and 1.88%, respectively.

From the results of Tables 5 and 6, we can conclude that BRKGA is the best algorithm in terms of solution quality, performance and stability when $\omega = 0.4$. Regarding the other two algorithms, VNS-SOP is much faster than MASOP, but less effective.

Table 7 shows the results of the three algorithms with $\omega = 0.6$ and g_1 profit. Here, the most effective algorithm is MASOP with a #Best value equal to 45 and an average gap equal to 0.09%. Moreover, MASOP finds six times a solution better than the best-known one, and it is the most stable algorithm. The second place is obtained by BRKGA with an average gap equal to 0.28% and a #Best value equal to 38. The #Best value of VNS-SOP is similar to the one of BRKGA, however, its average gap is higher with a value equal to 0.87%. Moreover, VNS-SOP appears less stable on these instances with a Dev.st% value close to 2%.

Regarding the performance, BRKGA is the fastest algorithm with an average time of 20 seconds and a peak of 185 seconds. The average time of MASOP and VNS-SOP is equal to 45 and 35 seconds, with a peak equal to 285 and 410 seconds, respectively. These values show that BRKGA is 120% faster than MASOP and 70% faster than VNS-SOP. Moreover, from the computational

Set1				$\omega = \mathbf{0.6}$ and g_1									
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**
11berlin52	43	**43**	43.0	2.40	0.00%	**43**	43.0	0.37	0.00%	**43**	43.0	1.42	0.00%
11eil51	39	**39**	39.0	5.41	0.00%	**39**	39.0	0.34	0.00%	**39**	39.0	1.51	0.00%
14st70	50	**50**	50.0	4.68	0.00%	**50**	50.0	0.53	0.00%	**50**	50.0	1.72	0.00%
16eil76	59	**59**	58.8	3.60	0.00%	**59**	59.0	0.67	0.00%	**59**	59.0	1.92	0.00%
16pr76	65	**65**	65.0	7.89	0.00%	**65**	65.0	0.78	0.00%	**65**	65.0	2.04	0.00%
20kroA100	65	**65**	65.0	7.87	0.00%	**65**	65.0	0.90	0.00%	**65**	65.0	2.12	0.00%
20kroB100	66	**66**	66.0	8.51	0.00%	**66**	66.0	0.90	0.00%	**66**	66.0	2.11	0.00%
20kroC100	62	**62**	62.0	6.63	0.00%	**62**	62.0	1.01	0.00%	**62**	62.0	2.14	0.00%
20kroD100	64	**64**	64.0	10.15	0.00%	**64**	64.0	0.89	0.00%	**64**	64.0	2.29	0.00%
20kroE100	63	**63**	63.0	7.36	0.00%	**63**	63.0	0.95	0.00%	**63**	63.0	2.03	0.00%
20rat99	52	**52**	52.0	8.71	0.00%	**52**	52.0	0.84	0.00%	**52**	52.0	1.81	0.00%
20rd100	72	**72**	72.0	2.49	0.00%	**72**	72.0	1.05	0.00%	**72**	72.0	2.23	0.00%
21eil101	82	**82**	82.0	7.62	0.00%	**82**	82.0	1.29	0.00%	**82**	82.0	2.41	0.00%
21lin105	78	**78**	78.0	9.84	0.00%	**78**	78.0	1.04	0.00%	**78**	78.0	2.03	0.00%
22pr107	53	**53**	53.0	12.74	0.00%	**53**	53.0	0.94	0.00%	**53**	53.0	1.87	0.00%
25pr124	75	**75**	75.0	12.14	0.00%	**75**	75.0	1.43	0.00%	**75**	75.0	2.48	0.00%
26bier127	118	**118**	117.8	5.30	0.00%	**118**	118.0	2.90	0.00%	**118**	117.8	3.70	0.00%
26ch130	**99**	98	98.0	5.95	1.01%	**99**	99.0	2.60	0.00%	98	98.0	2.81	1.01%
28pr136	89	**89**	89.0	8.51	0.00%	**89**	89.0	1.93	0.00%	**89**	89.0	2.71	0.00%
29pr144	98	**98**	98.0	7.31	0.00%	**98**	97.9	1.70	0.00%	**98**	98.0	3.06	0.00%
30ch150	96	**96**	96.0	8.96	0.00%	**96**	95.4	2.28	0.00%	**96**	96.0	2.94	0.00%
30kroA150	90	**90**	90.0	10.45	0.00%	**90**	89.0	2.19	0.00%	**90**	90.0	2.90	0.00%
30kroB150	105	**105**	105.0	8.31	0.00%	**105**	103.7	2.89	0.00%	**105**	103.9	3.49	0.00%
31pr152	105	**105**	105.0	12.17	0.00%	**105**	105.0	1.44	0.00%	**105**	105.0	3.14	0.00%
32u159	114	**114**	114.0	10.97	0.00%	**114**	114.0	2.48	0.00%	**114**	114.0	3.10	0.00%
39rat195	118	**118**	117.8	18.24	0.00%	**118**	117.3	3.57	0.00%	**118**	117.4	3.71	0.00%
40d198	160	**160**	160.0	17.41	0.00%	**160**	160.0	5.23	0.00%	**160**	159.6	4.46	0.00%
40kroa200	142	**142**	141.7	18.08	0.00%	**142**	141.1	5.06	0.00%	**142**	141.4	4.72	0.00%
40krob200	138	**138**	137.3	15.13	0.00%	**138**	137.1	6.03	0.00%	**138**	137.4	5.04	0.00%
45ts225	152	**152**	151.1	6.40	0.00%	**152**	151.0	7.82	0.00%	**152**	151.2	5.09	0.00%
45tsp225	**147**	146	146.0	21.47	0.68%	**147**	146.3	4.90	0.00%	146	145.7	5.95	0.68%
46pr226	162	**162**	162.0	10.20	0.00%	**162**	161.4	5.69	0.00%	**162**	162.0	5.16	0.00%
53gil262	**177**	176	176.0	17.56	0.56%	176	173.3	8.63	0.56%	**177**	175.9	8.01	0.00%
53pr264	162	**162**	161.5	29.49	0.00%	**162**	160.4	7.40	0.00%	**162**	161.4	7.05	0.00%
56a280	164	**164**	163.5	23.80	0.00%	**164**	163.8	9.33	0.00%	**164**	161.9	8.24	0.00%
60pr299	187	**187**	187.0	25.48	0.00%	177	175.0	9.39	5.35%	**187**	187.0	8.35	0.00%
64lin318	256	**256**	255.1	21.96	0.00%	**256**	249.4	15.80	0.00%	**256**	255.1	12.70	0.00%
80rd400	**295**	291	288.5	41.04	1.36%	294	287.5	24.83	0.34%	**295**	286.4	18.44	0.00%
84fl417	288	**288**	285.9	35.30	0.00%	286	254.3	17.73	0.69%	**288**	284.9	15.60	0.00%
88pr439	385	**385**	384.1	47.22	0.00%	**385**	383.5	36.33	0.00%	384	382.7	28.59	0.26%
89pcb442	327	325	322.8	58.24	0.61%	326	314.1	34.08	0.31%	323	319.0	21.98	1.22%
99d493	406	**406**	404.1	82.79	0.00%	402	399.1	67.96	0.99%	405	399.6	36.15	0.25%
115rat575	**396**	**396**	391.6	102.67	0.00%	392	377.9	59.25	1.01%	394	383.2	34.82	0.51%
115u574	444	442	441.0	77.04	0.45%	435	427.2	69.96	2.03%	440	434.1	40.98	0.90%
131p654	529	**529**	528.9	118.69	0.00%	476	470.4	40.17	10.02%	**529**	518.1	48.26	0.00%
132d657	**478**	**478**	474.1	145.54	0.00%	463	456.6	91.08	3.14%	468	459.3	51.98	2.09%
145u724	**514**	**514**	510.9	120.72	0.00%	494	476.6	114.56	3.89%	508	502.2	59.95	1.17%
157rat783	**524**	**524**	517.4	257.50	0.00%	515	495.4	130.95	1.72%	521	503.5	69.64	0.57%
201pr1002	**770**	**770**	762.6	249.39	0.00%	736	716.3	268.70	4.42%	751	740.5	156.37	2.47%
212u1060	**827**	**827**	821.8	285.05	0.00%	790	753.6	305.65	4.47%	812	790.9	185.13	1.81%
217vm1084	871	**871**	867.3	269.00	0.00%	822	779.0	410.45	5.63%	860	834.8	144.19	1.26%
Avg				**45.32**	**0.09%**			**35.19**	**0.87%**			**20.60**	**0.28%**
#Best					**45**				**36**				**38**
Dev.st %			**0.34%**				**1.96%**				**0.93%**		

Table 7: Comparison among MASOP, VNS-SOP and BRKGA on $Set1$ instances with $\omega = 0.6$ and g_1 profit.

times of VNS-SOP, it is evident that the performance of this algorithm is affected by the increment of the ω value. In fact, its average time is almost doubled with respect to the previous tables.

Set1				$\omega = 0.6$ and g_2									
		MASOP				VNS-SOP				BRKGA			
Instance	Best	Sol	Sol_{avg}	Time	Gap%	Sol	Sol_{avg}	Time	Gap%	Sol	Sol_{avg}	Time	Gap%
11berlin52	2190	**2190**	2190.0	3.03	0.00%	**2190**	2190.0	0.37	0.00%	**2190**	2190.0	1.44	0.00%
11eil51	1911	**1911**	1911.0	5.14	0.00%	**1911**	1911.0	0.35	0.00%	**1911**	1911.0	1.45	0.00%
14st70	2589	**2589**	2589.0	7.02	0.00%	**2589**	2589.0	0.53	0.00%	**2589**	2589.0	1.70	0.00%
16eil76	3119	**3119**	3119.0	5.30	0.00%	**3119**	3119.0	0.68	0.00%	**3119**	3119.0	1.91	0.00%
16pr76	3275	**3275**	3275.0	8.22	0.00%	**3275**	3275.0	0.80	0.00%	**3275**	3275.0	2.05	0.00%
20kroA100	3192	**3192**	3192.0	6.36	0.00%	**3192**	3192.0	0.94	0.00%	**3192**	3192.0	2.11	0.00%
20kroB100	3203	**3203**	3203.0	7.80	0.00%	**3203**	3203.0	0.92	0.00%	**3203**	3203.0	2.11	0.00%
20kroC100	3110	**3110**	3110.0	6.20	0.00%	**3110**	3110.0	0.96	0.00%	**3110**	3110.0	2.12	0.00%
20kroD100	3133	**3133**	3133.0	10.74	0.00%	**3133**	3133.0	0.90	0.00%	**3133**	3133.0	2.11	0.00%
20kroE100	2950	**2950**	2950.0	8.08	0.00%	**2950**	2950.0	0.97	0.00%	**2950**	2950.0	2.01	0.00%
20rat99	2643	**2643**	2643.0	8.01	0.00%	**2643**	2643.0	0.87	0.00%	**2643**	2643.0	1.83	0.00%
20rd100	**3591**	3585	3585.0	5.22	0.17%	**3591**	3588.0	1.33	0.00%	3585	3585.0	2.23	0.17%
21eil101	4187	**4187**	4186.2	8.81	0.00%	**4187**	4185.0	1.50	0.00%	**4187**	4187.0	2.46	0.00%
21lin105	3955	**3955**	3955.0	9.33	0.00%	**3955**	3955.0	1.05	0.00%	**3955**	3955.0	2.01	0.00%
22pr107	2697	**2697**	2697.0	12.51	0.00%	**2697**	2697.0	0.96	0.00%	**2697**	2697.0	1.72	0.00%
25pr124	3763	**3763**	3763.0	11.44	0.00%	**3763**	3763.0	1.49	0.00%	**3763**	3763.0	2.46	0.00%
26bier127	5882	**5882**	5882.0	9.54	0.00%	**5882**	5882.0	2.82	0.00%	**5882**	5882.0	3.95	0.00%
26ch130	5123	**5123**	5123.0	4.98	0.00%	**5123**	5123.0	1.62	0.00%	**5123**	5123.0	2.80	0.00%
28pr136	4579	**4579**	4579.0	9.18	0.00%	**4579**	4579.0	1.85	0.00%	**4579**	4579.0	2.76	0.00%
29pr144	4947	**4947**	4947.0	7.57	0.00%	**4947**	4916.4	1.65	0.00%	**4947**	4947.0	3.09	0.00%
30ch150	4825	**4825**	4825.0	9.17	0.00%	**4825**	4819.0	2.40	0.00%	**4825**	4825.0	3.32	0.00%
30kroA150	4542	**4542**	4542.0	10.79	0.00%	**4542**	4469.4	2.20	0.00%	**4542**	4542.0	2.97	0.00%
30kroB150	5123	**5123**	5123.0	9.58	0.00%	5120	5079.6	3.14	0.06%	**5123**	5121.5	3.58	0.00%
31pr152	5235	**5235**	5235.0	11.98	0.00%	**5235**	5235.0	1.49	0.00%	**5235**	5235.0	3.12	0.00%
32u159	5906	**5906**	5906.0	11.04	0.00%	**5906**	5906.0	2.54	0.00%	**5906**	5906.0	3.12	0.00%
39rat195	5846	**5846**	5839.1	18.88	0.00%	**5846**	5829.2	4.60	0.00%	**5846**	5846.0	3.95	0.00%
40d198	8102	**8102**	8102.0	17.41	0.00%	**8102**	8102.0	5.84	0.00%	**8102**	8102.0	4.92	0.00%
40kroa200	7105	**7105**	7104.6	17.36	0.00%	**7105**	7062.4	4.90	0.00%	**7105**	7104.6	4.83	0.00%
40krob200	6943	**6943**	6943.0	14.57	0.00%	**6943**	6884.8	6.01	0.00%	**6943**	6940.0	5.11	0.00%
45ts225	**7948**	**7948**	7896.7	8.89	0.00%	**7948**	7853.5	7.26	0.00%	**7948**	7918.2	5.28	0.00%
45tsp225	7603	**7603**	7603.0	19.03	0.00%	**7603**	7584.0	4.94	0.00%	**7603**	7603.0	5.49	0.00%
46pr226	8280	**8280**	8280.0	12.60	0.00%	**8280**	8233.0	4.82	0.00%	**8280**	8280.0	5.25	0.00%
53gil262	8864	**8864**	8854.6	34.77	0.00%	**8864**	8862.4	8.60	0.00%	**8864**	8863.4	8.15	0.00%
53pr264	8404	**8404**	8404.0	30.08	0.00%	**8404**	8397.0	7.24	0.00%	**8404**	8404.0	6.59	0.00%
56a280	8470	**8470**	8468.1	25.79	0.00%	**8470**	8330.7	9.65	0.00%	**8470**	8468.1	8.01	0.00%
60pr299	9544	**9544**	9544.0	23.69	0.00%	9495	8939.9	10.58	0.51%	**9544**	9496.3	8.22	0.00%
64lin318	13188	**13188**	13188.0	21.22	0.00%	13185	13038.3	14.97	0.02%	**13188**	13188.0	12.46	0.00%
80rd400	**15147**	15051	14983.9	38.14	0.63%	**15147**	15009.8	21.94	0.00%	15118	14901.9	18.79	0.19%
84fl417	14937	**14937**	14937.0	43.20	0.00%	14812	12890.8	16.79	0.84%	**14937**	14826.8	16.42	0.00%
88pr439	**19642**	19640	19613.2	82.12	0.01%	19601	19489.7	40.23	0.21%	**19642**	19516.9	29.84	0.00%
89pcb442	**16644**	16523	16455.4	61.87	0.73%	16341	16099.6	35.83	1.82%	**16644**	16463.7	23.47	0.00%
99d493	20584	20574	20481.5	74.17	0.05%	20348	20060.5	61.42	1.15%	20376	20216.1	35.20	1.01%
115rat575	20062	19962	19817.6	103.39	0.50%	19811	18977.7	63.52	1.25%	19933	19530.3	36.29	0.64%
115u574	22685	22606	22149.3	75.50	0.35%	21688	20976.8	63.78	4.39%	22542	22032.8	40.54	0.63%
131p654	26905	**26905**	26904.4	134.53	0.00%	26636	24064.8	49.55	1.00%	**26905**	25828.1	46.93	0.00%
132d657	24186	24132	24013.7	146.78	0.22%	23867	23225.9	91.32	1.32%	24135	23644.3	50.31	0.21%
145u724	**26564**	26510	26302.3	173.86	0.20%	25459	24562.6	111.10	4.16%	**26564**	25945.0	60.15	0.00%
157rat783	26883	26740	26516.8	210.35	0.53%	25512	25076.3	138.19	5.10%	26739	25794.7	70.60	0.54%
201pr1002	39262	39123	38537.7	231.66	0.35%	37133	36207.1	264.15	5.42%	38336	37611.0	155.51	2.36%
212u1060	41533	41362	41042.7	434.21	0.41%	38921	37879.5	348.27	6.29%	40311	39577.9	184.72	2.94%
217vm1084	44641	44573	44325.9	341.70	0.15%	41730	40052.2	365.50	6.52%	43938	42925.3	146.62	1.57%
Avg				**51.03**	**0.08%**			**35.20**	**0.79%**			**20.67**	**0.20%**
#Best					**38**				**35**				**41**
Dev.st %			**0.41%**				**2.46%**				**0.99%**		

Table 8: Comparison among MASOP, VNS-SOP and BRKGA on $Set1$ instances with $\omega = 0.6$ and g_2 profit.

The results of the three algorithms with profit type g_2 are reported in Table 8. This time BRKGA reports the highest #Best value equal to 41, whereas for MASOP and VNS-SOP this value is equal to 38 and 35. Moreover, four times BRKGA improves the best-known solution while MASOP does it one time and VNS-SOP three times. The best average gap is shown by MASOP that remains even the most stable algorithm. With a value equal to 0.20% the average gap of BRKGA is close to the one of MASOP while for VNS-SOP this value is around 0.80%. Finally, the computational time of BRKGA and VNS-SOP are very similar to the ones shown in the previous table, certifying that the profit type does not affect their performance. On the contrary, we again observe that, with g_2 profit, the computational time of MASOP increases. In particular, its peak here is equal to 434 seconds, while in Table 7 it was equal to 285. For BRKGA this peak remains around 185 seconds while for VNS-SOP is equal to 365 seconds.

Summarizing the results of Tables 7 and 8, we can conclude that MASOP shows the best average gaps and stability and the highest #Best value with g_1 profit. BRKGA has a better #Best value with g_2 profit and its average gap from MASOP is always lower than 0.20%. Moreover, BRKGA remains the fastest algorithm. The results of VNS-SOP show that, with $\omega = 0.6$, the algorithm is slower and less effective than the other two algorithms and it Dev.st% is close to 2.5% on the instances with g_2 profit.

The results of the three algorithms on the instances with $\omega = 0.8$ and g_1 profit are shown in Table 9. The best results are obtained by MASOP that is surely the best algorithm in terms of the quality of the solution and stability. The results of BRKGA and VNS-SOP are very similar but worse than the ones of MASOP. However, it is worth noting that VNS-SOP improves three times the best-known solution while the other two algorithms just one time. It is interesting to notice that the computational time of MASOP is decreased on these instances with respect to Table 7 but it remains slower than BRKGA even if it has a lower peak equal to 228 seconds. Moreover, it is worth noting the increment of VNS-SOP computational time: it requires now 52 seconds, on average, and 688 seconds, in the worst case, because of the increment of ω value.

Finally, the results of the algorithms when $\omega = 0.8$ and g_2 profit are shown in Table 10. The values of Avg, #Best and Dev.st% lines confirm that, for $\omega = 0.8$, the most effective and stable algorithm is MASOP that reaches to improve the best solution six times. The average gap of BRKGA remains lower than 0.55% and a similar result is obtained by VNS-SOP too. The computational times certify that BRKGA remains the fastest algorithm and that MASOP is the only algorithm which computational time is affected by the type of profit used.

Set1				**$\omega = 0.8$** and g_1									
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**
11berlin52	47	**47**	47.0	7.76	0.00%	**47**	47.0	0.42	0.00%	**47**	47.0	1.67	0.00%
11eil51	43	**43**	43.0	2.93	0.00%	**43**	43.0	0.42	0.00%	**43**	43.0	1.65	0.00%
14st70	65	**65**	65.0	9.24	0.00%	**65**	65.0	0.64	0.00%	**65**	64.8	2.02	0.00%
16eil76	69	**69**	69.0	4.03	0.00%	**69**	69.0	0.81	0.00%	**69**	69.0	2.42	0.00%
16pr76	72	71	71.0	3.71	1.39%	**72**	71.7	1.12	0.00%	**72**	72.0	2.49	0.00%
20kroA100	79	**79**	79.0	7.81	0.00%	**79**	78.6	1.50	0.00%	**79**	78.9	2.62	0.00%
20kroB100	86	**86**	86.0	7.72	0.00%	**86**	86.0	1.10	0.00%	**86**	86.0	2.50	0.00%
20kroC100	83	**83**	83.0	8.58	0.00%	**83**	83.0	1.25	0.00%	**83**	83.0	2.46	0.00%
20kroD100	85	**85**	85.0	9.56	0.00%	**85**	85.0	1.28	0.00%	**85**	85.0	2.43	0.00%
20kroE100	80	**80**	80.0	3.32	0.00%	**80**	80.0	1.38	0.00%	**80**	80.0	2.45	0.00%
20rat99	79	**79**	79.0	9.88	0.00%	**79**	79.0	1.19	0.00%	**79**	78.4	2.62	0.00%
20rd100	91	**91**	91.0	10.74	0.00%	**91**	91.0	1.42	0.00%	**91**	91.0	2.86	0.00%
21eil101	91	**91**	91.0	7.71	0.00%	**91**	91.0	1.44	0.00%	**91**	91.0	3.01	0.00%
21lin105	90	**90**	90.0	9.88	0.00%	**90**	90.0	1.19	0.00%	**90**	90.0	2.61	0.00%
22pr107	65	**65**	65.0	4.04	0.00%	**65**	65.0	0.86	0.00%	**65**	65.0	2.41	0.00%
25pr124	99	**99**	99.0	10.63	0.00%	**99**	99.0	1.66	0.00%	**99**	99.0	3.02	0.00%
26bier127	123	**123**	123.0	7.64	0.00%	**123**	123.0	3.05	0.00%	**123**	122.8	4.32	0.00%
26ch130	**117**	115	114.7	4.87	1.71%	**117**	116.8	2.21	0.00%	115	115.0	3.41	1.71%
28pr136	123	**123**	122.7	7.03	0.00%	**123**	123.0	2.77	0.00%	**123**	123.0	3.56	0.00%
29pr144	125	**125**	125.0	4.86	0.00%	**125**	124.1	2.29	0.00%	**125**	125.0	3.50	0.00%
30ch150	127	**127**	126.8	7.85	0.00%	**127**	126.9	3.58	0.00%	**127**	127.0	3.76	0.00%
30kroA150	120	**120**	120.0	9.44	0.00%	**120**	119.6	2.99	0.00%	**120**	120.0	3.69	0.00%
30kroB150	138	**138**	138.0	6.69	0.00%	**138**	137.8	3.48	0.00%	**138**	138.0	3.91	0.00%
31pr152	118	**118**	118.0	10.44	0.00%	115	115.0	1.83	2.54%	**118**	116.5	3.76	0.00%
32u159	140	**140**	140.0	9.90	0.00%	**140**	140.0	3.76	0.00%	**140**	140.0	4.10	0.00%
39rat195	167	**167**	167.0	12.44	0.00%	**167**	167.0	6.40	0.00%	**167**	167.0	5.00	0.00%
40d198	171	**171**	171.0	32.96	0.00%	**171**	171.0	7.84	0.00%	**171**	171.0	5.24	0.00%
40kroa200	184	**184**	184.0	13.24	0.00%	**184**	183.7	5.14	0.00%	**184**	184.0	6.16	0.00%
40krob200	179	**179**	178.1	16.74	0.00%	**179**	177.1	7.45	0.00%	**179**	177.3	6.34	0.00%
45ts225	**193**	188	187.1	4.80	2.59%	192	188.5	9.21	0.52%	**193**	188.5	6.44	0.00%
45tsp225	**198**	196	195.9	19.04	1.01%	**198**	197.9	7.48	0.00%	196	195.8	8.15	1.01%
46pr226	213	**213**	213.0	12.86	0.00%	**213**	213.0	5.13	0.00%	**213**	213.0	7.68	0.00%
53gil262	226	**226**	225.0	17.39	0.00%	**226**	225.0	10.69	0.00%	**226**	224.5	10.80	0.00%
53pr264	238	**238**	238.0	23.72	0.00%	233	233.0	5.96	2.10%	**238**	237.5	11.96	0.00%
56a280	232	**232**	231.8	23.05	0.00%	**232**	228.7	12.27	0.00%	**232**	231.3	12.48	0.00%
60pr299	270	**270**	269.5	26.29	0.00%	269	267.3	15.83	0.37%	**270**	269.2	14.44	0.00%
64lin318	295	**295**	294.1	22.17	0.00%	**295**	291.5	17.79	0.00%	**295**	291.8	16.95	0.00%
80rd400	364	363	360.7	37.22	0.27%	359	354.3	36.63	1.37%	356	350.6	24.71	2.20%
84fl417	399	**399**	398.8	67.36	0.00%	**399**	395.1	20.87	0.00%	398	397.4	24.36	0.25%
88pr439	419	**419**	418.9	80.23	0.00%	415	412.5	38.09	0.95%	414	412.1	33.66	1.19%
89pcb442	402	401	399.2	53.78	0.25%	394	385.4	44.48	1.99%	399	389.5	30.06	0.75%
99d493	468	**468**	467.0	44.04	0.00%	467	458.6	74.60	0.21%	466	462.1	45.72	0.43%
115rat575	498	495	493.0	79.36	0.60%	487	478.1	80.53	2.21%	486	478.2	47.94	2.41%
115u574	540	539	537.4	87.25	0.19%	533	527.2	102.86	1.30%	532	521.5	55.65	1.48%
131p654	637	636	636.0	95.31	0.16%	636	632.0	45.60	0.16%	636	633.0	79.01	0.16%
132d657	594	592	589.3	97.33	0.34%	575	561.1	122.89	3.20%	575	565.1	72.99	3.20%
145u724	649	645	639.4	84.98	0.62%	626	603.1	170.54	3.54%	634	621.1	86.01	2.31%
157rat783	663	661	653.8	149.52	0.30%	652	643.3	188.87	1.66%	641	632.1	100.67	3.32%
201pr1002	**928**	**928**	925.1	184.13	0.00%	896	884.4	403.31	3.45%	909	874.5	213.96	2.05%
212u1060	1001	**1001**	994.5	228.23	0.00%	967	958.3	476.64	3.40%	960	945.0	247.62	4.10%
217vm1084	1006	1004	1001.8	119.15	0.20%	979	968.4	688.76	2.68%	979	965.3	195.57	2.68%
Avg				**35.66**	**0.19%**			**51.95**	**0.62%**			**28.29**	**0.57%**
#Best					**38**				**34**				**35**
Dev.st %			**0.25%**				**0.80%**				**0.85%**		

Table 9: Comparison among MASOP, VNS-SOP and BRKGA on $Set1$ instances with $\omega = 0.8$ and g_1 profit.

6. Conclusion

In this paper, we developed a Biased Random-Key Genetic Algorithm to solve the Set Orienteering Problem. This algorithm uses the alleles of the chromosomes to state the clusters to

Set1				**$\omega = 0.8$** and g_2									
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**
11berlin52	2384	**2384**	2384.0	7.16	0.00%	**2384**	2384.0	0.42	0.00%	**2384**	2384.0	1.41	0.00%
11eil51	2114	**2114**	2114.0	3.26	0.00%	**2114**	2114.0	0.44	0.00%	**2114**	2114.0	1.58	0.00%
14st70	3355	**3355**	3355.0	8.49	0.00%	**3355**	3355.0	0.68	0.00%	**3355**	3355.0	1.99	0.00%
16eil76	3573	**3573**	3568.2	6.03	0.00%	**3573**	3573.0	0.82	0.00%	**3573**	3573.0	2.23	0.00%
16pr76	3611	**3611**	3608.0	3.25	0.00%	**3611**	3607.0	1.10	0.00%	**3611**	3611.0	2.42	0.00%
20kroA100	4115	**4115**	4115.0	7.87	0.00%	**4115**	4115.0	1.21	0.00%	**4115**	4115.0	2.45	0.00%
20kroB100	4188	**4188**	4188.0	5.49	0.00%	**4188**	4169.2	1.34	0.00%	**4188**	4188.0	2.53	0.00%
20kroC100	3999	**3999**	3999.0	7.56	0.00%	**3999**	3999.0	1.14	0.00%	**3999**	3999.0	2.47	0.00%
20kroD100	4267	**4267**	4267.0	9.31	0.00%	**4267**	4267.0	1.25	0.00%	**4267**	4267.0	2.45	0.00%
20kroE100	4002	**4002**	4002.0	4.83	0.00%	**4002**	3974.4	1.37	0.00%	**4002**	4002.0	2.51	0.00%
20rat99	3992	**3992**	3992.0	9.85	0.00%	**3992**	3992.0	1.21	0.00%	**3992**	3992.0	2.35	0.00%
20rd100	4640	**4640**	4640.0	10.30	0.00%	**4640**	4640.0	1.27	0.00%	**4640**	4640.0	2.79	0.00%
21eil101	4717	**4717**	4705.3	9.04	0.00%	**4717**	4717.0	1.45	0.00%	**4717**	4707.9	2.83	0.00%
21lin105	4561	**4561**	4561.0	9.92	0.00%	**4561**	4561.0	1.21	0.00%	**4561**	4561.0	2.59	0.00%
22pr107	3275	**3275**	3275.0	7.59	0.00%	**3275**	3275.0	0.99	0.00%	**3275**	3275.0	2.24	0.00%
25pr124	4947	**4947**	4932.2	4.88	0.00%	**4947**	4947.0	1.63	0.00%	**4947**	4947.0	3.04	0.00%
26bier127	6218	**6218**	6218.0	8.64	0.00%	**6218**	6218.0	2.98	0.00%	**6218**	6211.5	4.08	0.00%
26ch130	**5967**	5895	5882.1	7.43	1.21%	**5967**	5920.2	3.09	0.00%	5895	5895.0	3.64	1.21%
28pr136	6330	**6330**	6330.0	8.21	0.00%	**6330**	6330.0	2.53	0.00%	**6330**	6330.0	3.43	0.00%
29pr144	6356	**6356**	6356.0	6.38	0.00%	**6356**	6356.0	2.30	0.00%	**6356**	6356.0	3.48	0.00%
30ch150	**6382**	6331	6331.0	8.84	0.80%	**6382**	6373.5	3.10	0.00%	6331	6331.0	3.54	0.80%
30kroA150	6081	**6081**	6081.0	10.70	0.00%	**6081**	5974.8	3.15	0.00%	**6081**	6081.0	3.71	0.00%
30kroB150	6880	**6880**	6880.0	7.67	0.00%	**6880**	6880.0	3.49	0.00%	**6880**	6880.0	3.92	0.00%
31pr152	5928	**5928**	5916.8	13.17	0.00%	5800	5800.0	1.84	2.16%	**5928**	5825.6	3.52	0.00%
32u159	7164	**7164**	7164.0	10.19	0.00%	**7164**	7164.0	4.35	0.00%	**7164**	7164.0	4.15	0.00%
39rat195	8522	**8522**	8522.0	11.59	0.00%	**8522**	8522.0	5.87	0.00%	**8522**	8522.0	5.08	0.00%
40d198	8628	**8628**	8628.0	32.84	0.00%	**8628**	8628.0	8.52	0.00%	**8628**	8628.0	5.03	0.00%
40kroa200	9338	**9338**	9338.0	15.44	0.00%	**9338**	9331.8	5.53	0.00%	**9338**	9338.0	6.39	0.00%
40krob200	9077	**9077**	9077.0	15.18	0.00%	**9077**	8876.9	5.42	0.00%	**9077**	9066.4	6.00	0.00%
45ts225	9888	**9888**	9752.4	6.72	0.00%	**9888**	9785.2	10.11	0.00%	**9888**	9748.4	6.78	0.00%
45tsp225	**10030**	9934	9921.4	20.41	0.96%	**10030**	10024.4	6.94	0.00%	9934	9927.4	8.59	0.96%
46pr226	10770	**10770**	10770.0	10.03	0.00%	**10770**	10734.6	5.57	0.00%	**10770**	10770.0	7.64	0.00%
53gil262	11606	**11606**	11586.6	20.67	0.00%	**11606**	11439.0	11.43	0.00%	**11606**	11586.6	11.59	0.00%
53pr264	12048	**12048**	12048.0	30.97	0.00%	**12048**	11824.5	6.55	0.00%	**12048**	12022.7	11.38	0.00%
56a280	**11984**	**11984**	11898.5	21.60	0.00%	**11984**	11911.4	12.68	0.00%	11952	11925.6	12.52	0.27%
60pr299	13653	**13653**	13619.2	30.09	0.00%	**13653**	13533.8	16.36	0.00%	**13653**	13649.5	14.27	0.00%
64lin318	15103	**15103**	15055.9	32.56	0.00%	**15103**	14981.5	19.13	0.00%	**15103**	15012.9	16.23	0.00%
80rd400	**18529**	**18529**	18438.3	40.62	0.00%	18239	18040.7	37.51	1.57%	18519	18225.9	25.19	0.05%
84fl417	20248	**20248**	20245.2	54.45	0.00%	20220	19885.6	21.94	0.14%	20220	20152.2	26.48	0.14%
88pr439	21134	**21134**	21094.3	77.32	0.00%	21068	21017.6	43.80	0.31%	21068	20951.4	34.51	0.31%
89pcb442	20243	20214	20159.4	55.84	0.14%	20099	19693.5	42.32	0.71%	20120	19688.9	29.84	0.61%
99d493	**23726**	**23726**	23645.9	51.99	0.00%	23646	23233.3	78.87	0.34%	23511	23361.6	49.19	0.91%
115rat575	25145	25072	24891.3	95.12	0.29%	24875	24434.6	76.20	1.07%	24779	24498.0	48.88	1.46%
115u574	**27389**	**27389**	27293.9	112.68	0.00%	26828	26308.3	97.57	2.05%	27077	26586.6	56.14	1.14%
131p654	32335	**32335**	32331.6	107.37	0.00%	32216	32030.2	49.79	0.37%	32227	32074.9	79.63	0.33%
132d657	30642	30497	30372.7	110.93	0.47%	30123	29311.8	110.47	1.69%	29962	29437.6	72.91	2.22%
145u724	33050	32935	32768.9	116.67	0.35%	31863	31633.2	165.53	3.59%	32357	31830.0	83.79	2.10%
157rat783	**34081**	**34081**	33731.1	170.03	0.00%	32613	31793.5	202.87	4.31%	32972	32383.5	102.78	3.25%
201pr1002	46972	46800	46719.9	146.77	0.37%	45394	44801.6	420.85	3.36%	45404	44720.2	216.35	3.34%
212u1060	**50704**	**50704**	50152.4	233.12	0.00%	48879	47680.8	566.02	3.60%	48602	47784.1	248.01	4.15%
217vm1084	51539	51485	51323.3	185.85	0.10%	49635	48736.8	605.05	3.69%	49233	48523.5	186.99	4.47%
Avg				**39.27**	**0.09%**			**52.50**	**0.57%**			**28.30**	**0.54%**
#Best					**42**				**36**				**33**
Dev.st %			**0.31%**				**0.85%**				**0.69%**		

Table 10: Comparison among MASOP, VNS-SOP and BRKGA on $Set1$ instances with $\omega = 0.8$ and g_2 profit.

visit and visiting sequence of these clusters. Three local search procedures are applied during the evolutionary process to improve the fitness of the chromosomes. To improve the performance of the algorithm, we proposed a preprocessing phase that removes from the instances the useless vertices, arcs, and clusters. Moreover, we implemented a hashtable to avoid the invocation of the decoder function on the chromosomes already decoded in a previous step of the evolutionary process. Finally, we used a three-dimensional matrix to save and to quickly retrieve information required several times during the computation. Thanks to this idea, the same information is not re-computed more and more times avoiding waste of the computational time and, significantly, improving the performance of the algorithm. The computational results show that BRKGA is the fastest algorithm, on average, on all the tested instances. Moreover, it is the best algorithm in terms of solution quality and stability on the $Set1$ with $\omega = 0.4$. Finally, with an average gap from the best-known solution always lower than 0.65% on all tested instances, it results able to quickly produce high-quality solutions.

Acknowledgement

The author would like to thank Robert Pěnička for his support by providing the source code of VNS-SOP adapted to run on the OSX system.

References

[1] E. Angelelli, C. Archetti, and M. Vindigni. The clustered orienteering problem. *European Journal of Operational Research*, 238:404–414, 2014.

[2] C. Archetti, F. Carrabs, and R. Cerulli. The set orienteering problem. *European Journal of Operational Research*, 267(1):264–272, 2018.

[3] C. Archetti, M.G. Speranza, and D. Vigo. Vehicle routing problems with profits. In P. Toth and D. Vigo, editors, *Vehicle Routing: Problems, Methods, and Applications, Second Edition, MOS-SIAM Series on Optimization 18*, pages 273–298. SIAM, 2014.

[4] R. Bernardino and A. Paias. Metaheuristics based on decision hierarchies for the traveling purchaser problem. *International Transactions in Operational Research*, 25(4):1269–1295, 2018.

[5] G. Best and G.A. Hollinger. Decentralised self-organising maps for the online orienteering problem with neighbourhoods. In *International Symposium on Multi-Robot and Multi-Agent Systems*, pages 139–141, 2019.

[6] H.G. Beyer and H.P. Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[7] H. Bin and Günther R.R. Effective neighborhood structures for the generalized traveling salesman problem. In *Evolutionary Computation in Combinatorial Optimization: 8th European Conference, EvoCOP 2008, Naples, Italy, March 26-28, 2008. Proceedings*, pages 36–47, 2008.

[8] J.S. Brandão, T.F. Noronha, M.G.C. Resende, and C.C. Ribeiro. A biased random-key genetic algorithm for single-round divisible load scheduling. *International Transactions in Operational Research*, 22(5):823–839, 2015.

[9] J.S. Brandão, T.F. Noronha, M.G.C. Resende, and C.C. Ribeiro. A biased random-key genetic algorithm for scheduling heterogeneous multi-round systems. *International Transactions in Operational Research*, 24(5):1061–1077, 2017.

[10] F. Carrabs, C. Cerrone, R. Cerulli, and C. D'Ambrosio. Improved upper and lower bounds for the close enough traveling salesman problem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10232 LNCS:165–177, 2017.

[11] F. Carrabs, C. Cerrone, R. Cerulli, and M. Gaudioso. A novel discretization scheme for the close enough traveling salesman problem. *Computers and Operations Research*, 78:163–171, 2017.

[12] F. Carrabs, C. Cerrone, R. Cerulli, and B. L. Golden. An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, to appear. doi:10.1287/ijoc.2020.0962.

[13] I.-M. Chao, B.L. Golden, and E.A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3):475–489, 1996.

[14] B. Dezső, A. Jüttner, and P. Kovács. LEMON – an Open Source C++ Graph Template Library. *Electronic Notes in Theoretical Computer Science*, 264:23–45, 2011.

[15] J. Faigl, R. Pěnička, and G. Best. Self-organizing map-based solution for the orienteering problem with neighborhoods. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 1315–1321, 2017.

[16] M. Fischetti, J. J. Salazar Gonzàlez, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[17] J.F. Gonçalves and M.G.C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525, 2011.

[18] J.F. Gonçalves and M.G.C. Resende. A parallel multi-population biased random-key genetic algorithm for a container loading problem. *Computers and Operations Research*, 39(2):179–190, 2012.

[19] J.F. Gonçalves, M.G.C. Resende, and R.F. Toso. An experimental comparison of biased and unbiased random-key genetic algorithms. *Pesquisa Operacional*, 34(2):143–164, 2014.

[20] A. Gunawan, H.C. Lau, and P. Vansteenwegen. Orienteering problem: A survey of recent variants, solution approaches and applications. *European Journal of Operational Research*, 255(2):315–332, 2016.

[21] G. Gutin and D. Karapetyan. Generalized traveling salesman problem reduction algorithms. *arXiv preprint :0804.0735*, 2009.

[22] Y. Huang, A.C. Santos, and C. Duhamel. Model and methods to address urban road network problems with disruptions. *International Transactions in Operational Research*, 2019.

[23] H. Jürgen and M. Reinhard. Towards an optimal mutation probability for genetic algorithms. In *Parallel Problem Solving from Nature*, pages 23–32, 1991.

[24] H. Kellerer, U. Pferschy, and D. Pisinger. *The Multiple-Choice Knapsack Problem*, pages 317–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[25] G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah. Some applications of the generalized travelling salesman problem. *The Journal of the Operational Research Society*, 47(12):1461–1467, 1996.

[26] L.F. Morán-Mirabal, J.L. González-Velarde, and M.G.C. Resende. Randomized heuristics for the family traveling

salesperson problem. *International Transactions in Operational Research*, 21(1):41–57, 2014.

[27] R. Pěnička, J. Faigl, and M. Saska. Variable neighborhood search for the set orienteering problem and its application to other orienteering problem variants. *European Journal of Operational Research*, 276(3):816–825, 2019.

[28] R. Pěnička, J. Faigl, P. Váňa, and M. Saska. Dubins orienteering problem. *IEEE Robotics and Automation Letters*, 2(2):1210–1217, 2017.

[29] R. Pěnička, J. Faigl, P. Váňa, and M. Saska. Dubins orienteering problem with neighborhoods. In *Proceedings of the International conference on unmanned aircraft systems*, pages 1555–1562, 2017.

[30] L.S. Pessoa and C.E. Andrade. Heuristics for a flowshop scheduling problem with stepwise job objective function. *European Journal of Operational Research*, 266(3):950–962, 2018.

[31] S. Prabhakant and A.A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27(3):503–515, 1979.

[32] A.G. Ramos, E. Silva, and J.F. Oliveira. A new load balance methodology for container loading problem in road transportation. *European Journal of Operational Research*, 266(3):1140–1152, 2018.

[33] R. Reis, M. Ritt, L.S. Buriol, and M.G.C. Resende. A biased random-key genetic algorithm for ospf and deft routing to minimize network congestion. *International Transactions in Operational Research*, 18(3):401–423, 2011.

[34] L.C.R. Soares and M.A.M. Carvalho. Biased random-key genetic algorithm for scheduling identical parallel machines with tooling constraints. *European Journal of Operational Research*, 285(3):955–964, 2020.

[35] V.M. Spears and K.A. De Jong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

[36] R.F. Toso and M.G.C. Resende. A c++application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, 30(1):81–93, 2015.

[37] T. Tsiligirides. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35:797–809, 1984.

Appendix A. Computational Results on $Set2$ instances

In this section, we report the computational results of MASOP, VNS-SOP and BRKGA on $Set2$ instances proposed in [2]. The only difference between $Set1$ and $Set2$ instances concerns the clusters generation. More in detail, the number of clusters in each instance remains the same of $Set1$ but the vertices are randomly assigned to these clusters.

From the results of Tables A.11 and A.12 we observe that the best average gap is obtained by VNS-SOP, with g_1 profit, and BRKGA, with g_2 profit. MASOP shows the highest #Best value whatever is the profit used. BRKGA and VNS-SOP obtain the same #Best values for both types

Set2				$\boldsymbol{\omega = 0.4}$ and $\boldsymbol{g_1}$									
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**
11berlin52	50	**50**	50.0	8.17	0.00%	**50**	50.0	0.63	0.00%	**50**	50.0	1.99	0.00%
11eil51	**37**	34	34.0	1.01	8.11%	**37**	37.0	0.44	0.00%	**37**	37.0	1.60	0.00%
14st70	56	**56**	56.0	6.81	0.00%	**56**	56.0	0.76	0.00%	**56**	56.0	1.72	0.00%
16eil76	51	**51**	51.0	4.11	0.00%	**51**	51.0	0.88	0.00%	**51**	51.0	1.91	0.00%
16pr76	70	**70**	69.7	4.68	0.00%	**70**	70.0	1.17	0.00%	**70**	70.0	2.24	0.00%
20kroA100	80	74	74.0	1.12	7.50%	**80**	80.0	1.74	0.00%	**80**	80.0	2.51	0.00%
20kroB100	86	**86**	85.7	2.85	0.00%	**86**	86.0	2.28	0.00%	**86**	86.0	2.51	0.00%
20kroC100	**72**	**72**	71.3	6.12	0.00%	**72**	71.2	1.69	0.00%	**72**	72.0	2.47	0.00%
20kroD100	78	**78**	78.0	7.95	0.00%	**78**	76.4	2.45	0.00%	**78**	78.0	2.44	0.00%
20kroE100	90	**90**	90.0	9.95	0.00%	**90**	90.0	2.37	0.00%	**90**	90.0	2.48	0.00%
20rat99	73	**73**	73.0	6.58	0.00%	**73**	73.0	1.46	0.00%	**73**	73.0	2.06	0.00%
20rd100	**82**	80	80.0	7.37	2.44%	**82**	81.6	2.18	0.00%	80	80.0	2.21	2.44%
21eil101	83	76	76.0	4.62	8.43%	**83**	83.0	1.71	0.00%	**83**	83.0	2.70	0.00%
21lin105	95	**95**	95.0	9.20	0.00%	**95**	95.0	2.17	0.00%	**95**	95.0	2.40	0.00%
22pr107	94	**94**	94.0	13.77	0.00%	**94**	94.0	3.21	0.00%	**94**	94.0	2.36	0.00%
25pr124	101	**101**	101.0	6.59	0.00%	**101**	101.0	3.27	0.00%	**101**	101.0	2.90	0.00%
26bier127	125	**125**	125.0	15.17	0.00%	**125**	125.0	5.24	0.00%	**125**	125.0	3.44	0.00%
26ch130	111	**111**	107.2	6.46	0.00%	**111**	110.3	3.98	0.00%	**111**	111.0	3.65	0.00%
28pr136	120	**120**	119.1	5.52	0.00%	**120**	120.0	4.44	0.00%	**120**	120.0	3.10	0.00%
29pr144	137	**137**	137.0	16.08	0.00%	**137**	137.0	5.28	0.00%	**137**	137.0	3.89	0.00%
30ch150	**114**	111	111.0	5.50	2.63%	**114**	112.7	5.15	0.00%	111	111.0	3.58	2.63%
30kroA150	**110**	104	103.0	2.89	5.45%	109	106.4	4.17	0.91%	**110**	109.8	3.75	0.00%
30kroB150	**120**	**120**	119.4	8.20	0.00%	**120**	118.9	6.16	0.00%	**120**	119.9	4.22	0.00%
31pr152	136	**136**	136.0	14.90	0.00%	**136**	136.0	7.11	0.00%	**136**	136.0	3.61	0.00%
32u159	143	**143**	143.0	11.10	0.00%	**143**	143.0	5.21	0.00%	**143**	143.0	4.08	0.00%
39rat195	135	**135**	135.0	7.19	0.00%	**135**	133.8	7.70	0.00%	**135**	135.0	4.03	0.00%
40d198	**149**	148	148.0	7.57	0.67%	**149**	148.1	6.61	0.00%	148	148.0	3.21	0.67%
40kroa200	173	**173**	173.0	13.06	0.00%	**173**	173.0	8.55	0.00%	**173**	173.0	6.31	0.00%
40krob200	162	**162**	161.3	15.78	0.00%	161	160.4	10.99	0.62%	**162**	160.7	5.76	0.00%
45ts225	198	**198**	193.1	11.18	0.00%	**198**	195.2	13.42	0.00%	**198**	197.2	7.71	0.00%
45tsp225	**167**	**167**	148.6	16.33	0.00%	**167**	166.1	10.31	0.00%	**167**	166.5	5.61	0.00%
46pr226	207	**207**	207.0	18.34	0.00%	**207**	207.0	14.76	0.00%	**207**	207.0	5.88	0.00%
53gil262	204	**204**	203.3	20.99	0.00%	**204**	203.8	18.25	0.00%	**204**	202.9	11.76	0.00%
53pr264	245	**245**	244.8	28.76	0.00%	**245**	242.9	35.03	0.00%	242	242.0	6.19	1.22%
56a280	**204**	**204**	203.3	19.16	0.00%	203	201.9	15.14	0.49%	**204**	202.9	7.65	0.00%
60pr299	**254**	**254**	242.9	25.43	0.00%	253	247.2	22.91	0.39%	**254**	248.4	8.83	0.00%
64lin318	289	**289**	288.9	40.11	0.00%	285	284.5	42.49	1.38%	**289**	286.4	24.58	0.00%
80rd400	**351**	**351**	345.8	42.29	0.00%	348	344.2	59.68	0.85%	346	340.8	34.11	1.42%
84fl417	375	**375**	375.0	114.40	0.00%	371	371.0	119.42	1.07%	371	371.0	27.32	1.07%
88pr439	**431**	**431**	430.7	59.92	0.00%	**431**	427.2	120.91	0.00%	427	423.4	61.74	0.93%
89pcb442	376	364	354.5	19.33	3.19%	372	365.1	69.50	1.06%	**376**	365.7	46.41	0.00%
99d493	**412**	**412**	408.3	53.68	0.00%	409	402.2	93.55	0.73%	**412**	405.1	51.05	0.00%
115rat575	**420**	**420**	410.7	76.79	0.00%	413	397.4	96.66	1.67%	401	394.3	57.47	4.52%
115u574	**502**	499	496.8	70.05	0.60%	**502**	485.1	143.24	0.00%	491	477.6	85.67	2.19%
131p654	606	**606**	606.0	2453.02	0.00%	**606**	606.0	435.87	0.00%	**606**	606.0	63.66	0.00%
132d657	**518**	513	506.0	119.41	0.97%	504	497.5	139.65	2.70%	**518**	501.9	103.09	0.00%
145u724	**592**	579	572.5	136.48	2.20%	**592**	572.7	226.57	0.00%	577	567.8	141.35	2.53%
157rat783	**571**	**571**	553.6	159.83	0.00%	563	543.0	187.23	1.40%	551	538.6	134.35	3.50%
201pr1002	**856**	**856**	838.3	336.22	0.00%	849	815.9	591.27	0.82%	839	809.5	353.71	1.99%
212u1060	947	940	919.3	530.15	0.74%	944	923.1	826.72	0.32%	887	876.3	431.72	6.34%
217vm1084	**1045**	1020	1016.8	407.08	2.39%	**1045**	1017.1	983.04	0.00%	1033	1020.2	589.06	1.15%
Avg				**97.83**	**0.89%**			**85.78**	**0.28%**			**46.04**	**0.64%**
#Best					**38**				**37**				**37**
Dev.st%			**1.79%**				**1.17%**				**0.96%**		

Table A.11: Comparison among MASOP, VNS-SOP and BRKGA on *Set*2 instances with $\omega = 0.4$ and g_1 profit.

of profit, and this value is only by one lower than the #Best value of MASOP, when it is considered the g_1 profit. On the instances with g_2 profit the gap is more relevant. The most stable algorithm is BRKGA, with g_1 profit and MASOP, with g_2 profit. Finally, regarding the performance, BRKGA

Set2				**$\omega = 0.4$** and g_2									
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	**Sol_{avg}**	**Time**	**Gap%**	**Sol**	**Sol_{avg}**	**Time**	**Gap%**	**Sol**	**Sol_{avg}**	**Time**	**Gap%**
11berlin52	2584	**2584**	2584.0	7.61	0.00%	**2584**	2584.0	0.63	0.00%	**2584**	2584.0	1.95	0.00%
11eil51	1929	**1929**	1929.0	1.78	0.00%	**1929**	1929.0	0.48	0.00%	**1929**	1929.0	1.53	0.00%
14st70	2736	**2736**	2736.0	6.76	0.00%	**2736**	2736.0	0.85	0.00%	**2736**	2736.0	1.58	0.00%
16eil76	2518	**2518**	2518.0	2.44	0.00%	**2518**	2518.0	0.98	0.00%	**2518**	2518.0	1.90	0.00%
16pr76	3550	**3550**	3550.0	5.21	0.00%	**3550**	3550.0	1.30	0.00%	**3550**	3550.0	2.17	0.00%
20kroA100	**3894**	3434	3434.0	1.60	11.81%	**3894**	3894.0	1.69	0.00%	**3894**	3894.0	2.50	0.00%
20kroB100	4357	**4357**	4357.0	2.41	0.00%	**4357**	4357.0	2.14	0.00%	**4357**	4357.0	2.51	0.00%
20kroC100	3586	**3586**	3570.7	7.45	0.00%	**3586**	3560.8	1.74	0.00%	**3586**	3586.0	2.29	0.00%
20kroD100	3799	**3799**	3799.0	7.43	0.00%	**3799**	3749.0	2.51	0.00%	**3799**	3799.0	2.40	0.00%
20kroE100	4614	**4614**	4614.0	11.12	0.00%	**4614**	4614.0	1.92	0.00%	**4614**	4614.0	2.32	0.00%
20rat99	3624	**3624**	3624.0	5.60	0.00%	**3624**	3624.0	1.43	0.00%	**3624**	3624.0	2.08	0.00%
20rd100	**4181**	4038	4038.0	5.96	3.42%	**4181**	4152.4	2.51	0.00%	4038	4038.0	2.06	3.42%
21eil101	**4264**	**4264**	4132.6	3.50	0.00%	**4264**	4264.0	1.84	0.00%	**4264**	4264.0	2.60	0.00%
21lin105	4814	**4814**	4814.0	9.69	0.00%	**4814**	4814.0	2.23	0.00%	**4814**	4814.0	2.35	0.00%
22pr107	4740	**4740**	4740.0	12.99	0.00%	**4740**	4740.0	2.92	0.00%	**4740**	4740.0	2.39	0.00%
25pr124	5035	**5035**	5028.0	8.33	0.00%	**5035**	5035.0	3.93	0.00%	**5035**	5031.5	3.12	0.00%
26bier127	6329	**6329**	6329.0	15.93	0.00%	**6329**	6329.0	6.46	0.00%	**6329**	6327.2	4.58	0.00%
26ch130	**5630**	**5630**	5297.9	4.18	0.00%	**5630**	5630.0	3.95	0.00%	**5630**	5630.0	3.64	0.00%
28pr136	6106	**6106**	6106.0	4.96	0.00%	**6106**	6106.0	4.42	0.00%	**6106**	6106.0	3.29	0.00%
29pr144	6848	**6848**	6848.0	15.34	0.00%	**6848**	6848.0	5.31	0.00%	**6848**	6848.0	3.82	0.00%
30ch150	**6025**	5896	5792.0	6.70	2.14%	**6025**	5765.8	4.73	0.00%	5896	5886.0	3.88	2.14%
30kroA150	**5450**	5399	5336.1	10.48	0.94%	**5450**	5413.3	3.69	0.00%	**5450**	5450.0	3.68	0.00%
30kroB150	6255	**6255**	6198.3	7.48	0.00%	**6255**	6128.4	4.42	0.00%	**6255**	6235.7	4.30	0.00%
31pr152	6928	**6928**	6928.0	14.55	0.00%	**6928**	6928.0	6.84	0.00%	**6928**	6928.0	3.65	0.00%
32u159	7507	**7507**	7507.0	13.62	0.00%	**7507**	7507.0	5.25	0.00%	**7507**	7507.0	3.84	0.00%
39rat195	6813	**6813**	6813.0	6.29	0.00%	**6813**	6803.1	7.13	0.00%	**6813**	6813.0	3.83	0.00%
40d198	7412	**7412**	7412.0	7.30	0.00%	**7412**	7412.0	6.15	0.00%	**7412**	7412.0	3.33	0.00%
40kroa200	9014	**9014**	9014.0	11.96	0.00%	**9014**	9014.0	9.37	0.00%	**9014**	9014.0	6.47	0.00%
40krob200	8315	**8315**	8224.6	12.05	0.00%	**8315**	8132.1	9.82	0.00%	**8315**	8186.3	5.81	0.00%
45ts225	**9835**	**9835**	9569.2	11.02	0.00%	**9835**	9519.4	12.47	0.00%	**9835**	9835.0	7.50	0.00%
45tsp225	8373	**8373**	8373.0	10.93	0.00%	**8373**	8370.1	12.46	0.00%	**8373**	8373.0	5.44	0.00%
46pr226	10322	**10322**	10322.0	25.97	0.00%	10312	10312.0	16.54	0.10%	10312	10308.6	7.09	0.10%
53gil262	10309	**10309**	10274.4	23.65	0.00%	**10309**	10309.0	11.69	0.00%	**10309**	10282.6	11.93	0.00%
53pr264	**12304**	**12304**	12297.4	36.70	0.00%	**12304**	12198.4	41.15	0.00%	12106	12106.0	6.19	1.61%
56a280	10285	**10285**	10275.5	15.99	0.00%	10274	10191.3	17.82	0.11%	**10285**	10168.6	8.03	0.00%
60pr299	**12995**	**12995**	12648.2	28.14	0.00%	**12995**	12744.5	24.63	0.00%	**12995**	12735.7	8.76	0.00%
64lin318	**14743**	**14743**	14721.1	38.27	0.00%	14704	14604.8	48.71	0.26%	14719	14587.0	24.14	0.16%
80rd400	**17917**	**17917**	17772.4	36.47	0.00%	17676	17245.5	61.83	1.35%	17521	17238.6	34.67	2.21%
84fl417	19107	**19107**	19107.0	281.52	0.00%	**19107**	19101.6	128.36	0.00%	19101	19101.0	26.79	0.03%
88pr439	**21815**	**21815**	21765.3	58.43	0.00%	21738	21598.9	117.49	0.35%	21565	21433.9	58.84	1.15%
89pcb442	**18908**	18449	18244.4	32.77	2.43%	18702	18270.0	65.96	1.09%	**18908**	18579.5	45.69	0.00%
99d493	**21030**	20864	20610.8	58.98	0.79%	20722	20410.7	87.41	1.46%	**21030**	20548.9	51.27	0.00%
115rat575	**21167**	**21167**	20671.2	53.04	0.00%	20740	20145.5	82.91	2.02%	20972	20534.5	58.80	0.92%
115u574	**25493**	**25493**	25333.7	80.04	0.00%	25016	24365.2	151.24	1.87%	25451	24860.1	86.23	0.16%
131p654	30212	**30212**	30212.0	3208.31	0.00%	**30212**	30212.0	456.05	0.00%	**30212**	30212.0	60.46	0.00%
132d657	**26772**	26031	25657.6	113.91	2.77%	25859	25530.8	139.22	3.41%	**26772**	25825.5	102.82	0.00%
145u724	29997	29526	28946.1	179.53	1.57%	29888	29040.2	229.45	0.36%	29952	28776.4	139.95	0.15%
157rat783	**29807**	**29807**	29473.0	182.79	0.00%	28314	27533.1	183.40	5.01%	28992	28033.5	134.63	2.73%
201pr1002	**44229**	**44229**	43322.8	369.69	0.00%	43615	42392.9	616.96	1.39%	43320	42283.2	364.83	2.06%
212u1060	**48481**	**48481**	48016.7	387.87	0.00%	47330	46540.3	841.93	2.37%	46705	45073.5	437.69	3.66%
217vm1084	**52987**	**52987**	52831.5	749.58	0.00%	52335	51977.2	1001.42	1.23%	52534	51640.4	563.33	0.85%
Avg				**121.85**	**0.51%**			**87.37**	**0.44%**			**45.78**	**0.42%**
#Best					**43**				**36**				**36**
Dev.st%			**1.13%**				**1.15%**				**1.14%**		

Table A.12: Comparison among MASOP, VNS-SOP and BRKGA on $Set2$ instances with $\omega = 0.4$ and g_2 profit.

remains the fastest algorithm even on $Set2$ instances with a computational time that is almost half the time of the other two algorithms. As for $Set1$ instances, the computational time of MASOP increases on the instances with g_2 profit and, in fact, BRKGA is 168% faster than MASOP on these instances. It is worth noting that the effectiveness of VNS-SOP is increased on $Set2$ instances with

Set2						$\omega = 0.6$ and g_1							
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**
11berlin52	51	**51**	51.0	8.56	0.00%	**51**	51.0	0.78	0.00%	**51**	51.0	2.28	0.00%
11eil51	50	**50**	50.0	6.58	0.00%	**50**	50.0	0.66	0.00%	**50**	50.0	1.91	0.00%
14st70	64	**64**	64.0	8.14	0.00%	**64**	64.0	1.08	0.00%	**64**	64.0	1.87	0.00%
16eil76	**74**	**74**	73.9	5.32	0.00%	**74**	74.0	1.52	0.00%	**74**	73.9	2.69	0.00%
16pr76	74	**74**	74.0	11.00	0.00%	**74**	74.0	1.78	0.00%	**74**	74.0	2.42	0.00%
20kroA100	98	**98**	97.3	10.53	0.00%	96	95.9	3.17	2.04%	96	96.0	3.38	2.04%
20kroB100	98	**98**	96.8	10.52	0.00%	**98**	93.5	2.81	0.00%	**98**	96.5	3.38	0.00%
20kroC100	93	**93**	92.7	9.53	0.00%	**93**	90.8	2.83	0.00%	**93**	92.6	3.54	0.00%
20kroD100	93	**93**	93.0	9.21	0.00%	**93**	93.0	2.58	0.00%	**93**	93.0	3.03	0.00%
20kroE100	97	**97**	97.0	13.22	0.00%	**97**	97.0	3.07	0.00%	**97**	97.0	2.90	0.00%
20rat99	87	**87**	87.0	7.47	0.00%	**87**	87.0	2.19	0.00%	**87**	87.0	2.71	0.00%
20rd100	99	**99**	98.5	11.69	0.00%	**99**	99.0	2.71	0.00%	**99**	99.0	3.27	0.00%
21eil101	**97**	**97**	96.8	5.98	0.00%	**97**	97.0	3.24	0.00%	**97**	96.7	3.62	0.00%
21lin105	104	**104**	104.0	13.84	0.00%	**104**	104.0	2.94	0.00%	**104**	104.0	2.30	0.00%
22pr107	106	**106**	106.0	14.34	0.00%	**106**	106.0	3.57	0.00%	**106**	106.0	2.30	0.00%
25pr124	119	**119**	119.0	11.65	0.00%	**119**	119.0	4.42	0.00%	**119**	118.6	4.23	0.00%
26bier127	126	**126**	126.0	20.85	0.00%	**126**	126.0	5.98	0.00%	**126**	126.0	3.14	0.00%
26ch130	127	**127**	125.9	10.40	0.00%	**127**	127.0	5.18	0.00%	**127**	126.8	4.60	0.00%
28pr136	132	**132**	131.7	12.94	0.00%	129	129.0	5.43	2.27%	129	129.0	4.22	2.27%
29pr144	141	**141**	141.0	17.22	0.00%	**141**	141.0	7.45	0.00%	**141**	141.0	4.79	0.00%
30ch150	145	**145**	144.7	11.69	0.00%	**145**	145.0	7.03	0.00%	**145**	144.5	5.85	0.00%
30kroA150	145	**145**	144.6	11.65	0.00%	143	142.7	6.81	1.38%	**145**	143.6	5.39	0.00%
30kroB150	147	**147**	147.0	17.82	0.00%	**147**	147.0	7.60	0.00%	**147**	147.0	5.78	0.00%
31pr152	151	**151**	150.7	22.73	0.00%	**151**	151.0	7.85	0.00%	150	150.0	5.76	0.66%
32u159	156	**156**	156.0	15.30	0.00%	**156**	156.0	9.55	0.00%	**156**	155.2	6.83	0.00%
39rat195	**178**	**178**	175.6	10.90	0.00%	175	175.0	9.85	1.69%	**178**	174.6	7.01	0.00%
40d198	196	**196**	196.0	22.10	0.00%	**196**	196.0	17.07	0.00%	**196**	194.2	8.23	0.00%
40kroa200	198	**198**	198.0	22.94	0.00%	**198**	196.7	18.71	0.00%	**198**	195.2	11.76	0.00%
40krob200	198	**198**	197.5	25.28	0.00%	**198**	194.1	17.35	0.00%	197	194.9	12.31	0.51%
45ts225	224	**224**	224.0	28.02	0.00%	**224**	224.0	19.27	0.00%	**224**	223.7	13.41	0.00%
45tsp225	**212**	**212**	209.1	15.12	0.00%	**212**	209.6	21.52	0.00%	209	207.5	14.10	1.42%
46pr226	221	**221**	221.0	22.66	0.00%	**221**	221.0	20.23	0.00%	**221**	221.0	12.38	0.00%
53gil262	**253**	**253**	251.5	28.01	0.00%	**253**	246.4	28.56	0.00%	**253**	247.7	19.07	0.00%
53pr264	252	**252**	252.0	43.01	0.00%	**252**	252.0	52.00	0.00%	**252**	252.0	14.44	0.00%
56a280	256	**256**	255.3	21.09	0.00%	254	250.4	29.42	0.78%	253	248.8	20.12	1.17%
60pr299	286	**286**	286.0	28.40	0.00%	**286**	286.0	46.36	0.00%	**286**	285.7	25.25	0.00%
64lin318	316	**316**	315.5	45.22	0.00%	**316**	314.4	70.06	0.00%	313	311.3	30.94	0.95%
80rd400	**396**	**396**	394.4	60.50	0.00%	395	394.9	98.64	0.25%	392	386.6	40.21	1.01%
84fl417	407	**407**	407.0	239.85	0.00%	**407**	406.1	122.49	0.00%	**407**	406.4	51.70	0.00%
88pr439	438	**438**	438.0	144.75	0.00%	**438**	438.0	186.07	0.00%	436	435.6	57.88	0.46%
89pcb442	439	438	436.7	66.11	0.23%	433	428.8	107.14	1.37%	433	426.9	62.49	1.37%
99d493	**488**	**488**	485.7	128.56	0.00%	485	483.2	175.24	0.61%	482	479.3	88.33	1.23%
115rat575	**542**	**542**	534.7	71.61	0.00%	533	523.0	172.76	1.66%	520	506.6	94.99	4.06%
115u574	**571**	**571**	570.0	91.35	0.00%	**571**	566.1	266.10	0.00%	566	557.8	118.21	0.88%
131p654	**642**	**642**	639.1	499.22	0.00%	**642**	639.1	484.25	0.00%	638	630.5	142.12	0.62%
132d657	**646**	**646**	644.9	147.60	0.00%	636	631.2	346.31	1.55%	628	622.9	162.75	2.79%
145u724	**707**	706	704.2	135.62	0.14%	**707**	695.3	415.46	0.00%	697	677.4	200.15	1.41%
157rat783	**744**	**744**	740.6	198.45	0.00%	722	711.8	371.05	2.96%	721	696.5	203.43	3.09%
201pr1002	**995**	**995**	990.5	422.60	0.00%	979	969.0	1200.03	1.61%	965	951.0	505.41	3.02%
212u1060	1058	**1058**	1058.0	721.93	0.00%	1055	1050.4	1589.21	0.28%	1025	1015.7	582.64	3.12%
217vm1084	1083	**1083**	1083.0	769.31	0.00%	**1083**	1079.4	1538.52	0.00%	1079	1070.0	721.55	0.37%
Avg				**84.48**	**0.01%**			**147.57**	**0.36%**			**64.96**	**0.64%**
#Best					**49**				**38**				**31**
Dev.st%			**0.38%**				**0.90%**				**0.83%**		

Table A.13: Comparison among MASOP, VNS-SOP and BRKGA on $Set2$ instances with $\omega = 0.6$ and g_1 profit.

respect of $Set1$ instances. This trend will be confirmed in the next tables.

In the instances with $\omega = 0.6$ the most effective algorithm is MASOP whatever is the profit

Set2							**$\omega = 0.6$** and g_2						
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**
11berlin52	2608	**2608**	2608.0	8.36	0.00%	**2608**	2608.0	0.78	0.00%	**2608**	2608.0	2.36	0.00%
11eil51	2575	**2575**	2575.0	6.76	0.00%	**2575**	2575.0	0.62	0.00%	**2575**	2575.0	1.83	0.00%
14st70	3218	**3218**	3218.0	7.01	0.00%	**3218**	3218.0	1.14	0.00%	**3218**	3218.0	2.08	0.00%
16eil76	**3728**	**3728**	3712.1	5.84	0.00%	**3728**	3717.5	1.62	0.00%	**3728**	3721.1	3.07	0.00%
16pr76	3729	**3729**	3729.0	11.40	0.00%	**3729**	3729.0	1.73	0.00%	**3729**	3729.0	1.92	0.00%
20kroA100	**4920**	**4920**	4717.1	7.41	0.00%	**4920**	4808.6	3.44	0.00%	**4920**	4847.2	3.44	0.00%
20kroB100	4925	**4925**	4925.0	15.25	0.00%	**4925**	4925.0	3.83	0.00%	**4925**	4910.6	3.56	0.00%
20kroC100	4717	**4717**	4651.0	7.21	0.00%	4619	4619.0	2.89	2.08%	**4717**	4668.0	3.27	0.00%
20kroD100	4695	**4695**	4695.0	7.11	0.00%	**4695**	4695.0	2.62	0.00%	**4695**	4695.0	2.93	0.00%
20kroE100	4910	**4910**	4910.0	13.90	0.00%	**4910**	4910.0	2.86	0.00%	**4910**	4910.0	2.68	0.00%
20rat99	4516	**4516**	4516.0	6.97	0.00%	**4516**	4516.0	2.29	0.00%	**4516**	4516.0	2.51	0.00%
20rd100	5008	**5008**	5008.0	12.51	0.00%	**5008**	5008.0	2.84	0.00%	**5008**	5008.0	3.23	0.00%
21eil101	4933	**4933**	4932.2	5.82	0.00%	**4933**	4933.0	3.27	0.00%	**4933**	4929.8	3.48	0.00%
21lin105	5228	**5228**	5228.0	14.51	0.00%	**5228**	5228.0	3.13	0.00%	**5228**	5228.0	2.89	0.00%
22pr107	5363	**5363**	5363.0	13.29	0.00%	**5363**	5363.0	3.57	0.00%	**5363**	5363.0	2.40	0.00%
25pr124	5947	**5947**	5947.0	11.14	0.00%	**5947**	5947.0	4.81	0.00%	**5947**	5947.0	4.21	0.00%
26bier127	6333	**6333**	6333.0	20.92	0.00%	**6333**	6333.0	6.73	0.00%	**6333**	6333.0	2.45	0.00%
26ch130	**6472**	**6472**	6422.5	9.79	0.00%	**6472**	6408.8	5.51	0.00%	6393	6389.9	4.38	1.22%
28pr136	**6692**	**6692**	6592.6	7.04	0.00%	**6692**	6590.3	7.67	0.00%	**6692**	6530.8	4.74	0.00%
29pr144	7151	**7151**	7151.0	16.16	0.00%	**7151**	7151.0	8.24	0.00%	**7151**	7151.0	4.63	0.00%
30ch150	7382	**7382**	7305.2	14.51	0.00%	7286	7273.6	6.34	1.30%	**7382**	7334.0	6.14	0.00%
30kroA150	7315	**7315**	7291.3	13.72	0.00%	7236	7205.4	6.98	1.08%	**7315**	7275.5	5.39	0.00%
30kroB150	**7476**	**7476**	7471.6	16.49	0.00%	7454	7454.0	8.08	0.29%	**7476**	7456.2	5.86	0.00%
31pr152	7658	**7658**	7644.5	21.74	0.00%	**7658**	7658.0	8.11	0.00%	7613	7613.0	5.32	0.59%
32u159	7942	**7942**	7942.0	11.80	0.00%	**7942**	7942.0	10.06	0.00%	**7942**	7866.4	6.56	0.00%
39rat195	**9022**	**9022**	8816.6	10.68	0.00%	8824	8816.8	11.32	2.19%	**9022**	8848.2	7.20	0.00%
40d198	9952	**9952**	9952.0	22.04	0.00%	**9952**	9952.0	18.40	0.00%	**9952**	9952.0	8.19	0.00%
40kroa200	10010	**10010**	10010.0	21.21	0.00%	**10010**	9965.4	19.35	0.00%	**10010**	9924.7	11.65	0.00%
40krob200	10011	**10011**	10004.5	25.18	0.00%	9946	9836.2	18.99	0.65%	**10011**	9913.3	12.57	0.00%
45ts225	11308	**11308**	11308.0	32.68	0.00%	**11308**	11308.0	20.68	0.00%	**11308**	11308.0	14.91	0.00%
45tsp225	**10786**	10715	10667.4	11.68	0.66%	**10786**	10650.6	21.87	0.00%	10742	10626.2	14.30	0.41%
46pr226	11063	**11063**	11063.0	24.76	0.00%	**11063**	11063.0	22.62	0.00%	**11063**	11063.0	11.64	0.00%
53gil262	12839	**12839**	12829.0	38.29	0.00%	12722	12470.8	29.65	0.91%	12780	12605.5	19.36	0.46%
53pr264	12658	**12658**	12658.0	50.94	0.00%	**12658**	12658.0	50.45	0.00%	**12658**	12658.0	14.68	0.00%
56a280	**13252**	13229	13080.7	23.25	0.17%	**13252**	12739.1	33.26	0.00%	13029	12816.6	20.01	1.68%
60pr299	14688	**14688**	14688.0	31.43	0.00%	**14688**	14688.0	49.92	0.00%	**14688**	14660.9	24.43	0.00%
64lin318	15993	**15993**	15993.0	55.08	0.00%	15955	15887.3	66.36	0.24%	15877	15844.3	30.84	0.73%
80rd400	**20090**	**20090**	20056.9	61.10	0.00%	20043	19938.9	96.18	0.23%	20015	19804.1	42.72	0.37%
84fl417	**20642**	20635	20628.2	249.82	0.03%	**20642**	20570.2	133.04	0.00%	**20642**	20532.2	50.25	0.00%
88pr439	22177	**22177**	22177.0	133.66	0.00%	**22177**	22177.0	180.12	0.00%	22046	22024.4	53.35	0.59%
89pcb442	**22194**	**22194**	22108.3	71.26	0.00%	22072	21723.6	120.17	0.55%	22064	21558.4	61.78	0.59%
99d493	**24679**	**24679**	24650.3	109.50	0.00%	24617	24444.8	164.74	0.25%	24551	24307.9	84.79	0.52%
115rat575	**27170**	**27170**	26959.0	85.82	0.00%	26958	26124.6	167.53	0.78%	26824	26203.3	95.75	1.27%
115u574	**28937**	**28937**	28880.6	96.28	0.00%	28829	28726.0	263.55	0.37%	28799	28595.9	117.83	0.48%
131p654	**32442**	**32442**	32340.4	459.26	0.00%	**32442**	32346.9	432.71	0.00%	32315	32069.2	155.08	0.39%
132d657	**32805**	**32805**	32786.3	179.19	0.00%	32205	32028.1	328.56	1.83%	32120	31759.9	156.55	2.09%
145u724	**35888**	35614	35467.9	163.81	0.76%	**35888**	35187.1	395.80	0.00%	35267	34609.8	200.40	1.73%
157rat783	**37357**	**37357**	37010.8	111.70	0.00%	36610	36017.3	377.99	2.00%	36518	35703.6	206.81	2.25%
201pr1002	**50412**	**50412**	50247.7	403.50	0.00%	49941	49135.5	1130.68	0.93%	49552	48575.7	496.69	1.71%
212u1060	53468	**53468**	53468.0	776.23	0.00%	53442	53126.9	1577.57	0.05%	52364	51371.1	564.76	2.06%
217vm1084	54712	**54712**	54712.0	797.88	0.00%	**54712**	54480.9	1569.24	0.00%	54551	53752.0	660.58	0.29%
Avg				**84.96**	**0.03%**			**145.29**	**0.31%**			**63.30**	**0.38%**
#Best					**47**				**34**				**32**
Dev.st%			**0.72%**				**0.87%**				**0.78%**		

Table A.14: Comparison among MASOP, VNS-SOP and BRKGA on *Set*2 instances with $\omega = 0.6$ and g_2 profit.

used. In fact, its average gap is close to 0%, its #Best value is the highest one and it results the more stable algorithm. Moreover, it improves the best-known solution 14 times on the instances with g_1 profit and 17 times, on the instances with g_2 profit. Taking into account the average gap and #Best values, VNS-SOP is classified as second best algorithm. However, it results the less stable algorithm and the slowest one, whatever is the profit used. In particular, its peak time on these instances is over 1500 seconds. BRKGA results less effective, in these instances, but its average gap is lower than 0.65%, in the worst case. Moreover, it remains the fastest algorithm. It is interesting to observe that, the performance gap between BRKGA and VNS-SOP increases from 84% to 130% changing the ω value from 0.4 to 0.6. This trend will be confirmed in the instances with $\omega = 0.8$.

The performance comparison between BRKGA and MASOP shows that, with $\omega = 0.6$, the gap is reduced with respect to the results with $\omega = 0.4$. In fact, with $\omega = 0.4$ BRKGA is around two or three times faster than MASOP whereas, with $\omega = 0.6$, the gap is much lower.

In the instances with $\omega = 0.8$, we observed that the T_{max} value is enough large to guarantee that all the clusters can be visited without violating this threshold. This is true for all the instances except one (40d198). This means that the optimal solution value for SOP on these instances is equal to the sum of the profit of all the clusters. Ruling out the instance 40d198, we verified that the values reported under the Best heading in Tables A.15 and A.16 coincide with $\sum_{g=1}^{l} p_g$ and then they are the optimal values for these instances. We can see that both MASOP and VNS-SOP obtain a #Best value equal to 51 while for BRKGA this value is equal to 43 and 44 on the instances with g_1 and g_2 profit, respectively. However, the average gap value of BRKGA is very low because it is at most equal to 0.07%. Even if BRKGA results lightly less effective than the other two algorithms, it is much faster. In particular, it is around two times faster than MASOP and almost four times faster than VNS-SOP. It is worth noting that, the effectiveness of MASOP and VNS-SOP on these instances is paid in terms of computational time since the highest time is equal to 677 seconds for BRKGA and it increases to 1327 and to 2827 seconds for MASOP and VNS-SOP, respectively. According to the performance of BRKGA and its very low percentage gap, it may be preferable to use this algorithm in contexts where it is necessary to have high-quality solutions in a shorten time.

Set2				$\omega = \mathbf{0.8}$	and	g_1							
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**	**Sol**	$\boldsymbol{Sol_{avg}}$	**Time**	**Gap%**
11berlin52	51	**51**	51.0	8.50	0.00%	**51**	51.0	0.83	0.00%	**51**	51.0	2.36	0.00%
11eil51	50	**50**	50.0	7.24	0.00%	**50**	50.0	0.78	0.00%	**50**	50.0	1.49	0.00%
14st70	69	**69**	69.0	10.79	0.00%	**69**	69.0	1.31	0.00%	**69**	69.0	2.16	0.00%
16eil76	75	**75**	75.0	10.72	0.00%	**75**	75.0	1.68	0.00%	**75**	75.0	2.76	0.00%
16pr76	75	**75**	75.0	12.08	0.00%	**75**	75.0	1.80	0.00%	**75**	75.0	1.96	0.00%
20kroA100	99	**99**	99.0	13.24	0.00%	**99**	99.0	3.25	0.00%	**99**	99.0	3.41	0.00%
20kroB100	99	**99**	99.0	14.52	0.00%	**99**	99.0	3.34	0.00%	**99**	99.0	3.27	0.00%
20kroC100	99	**99**	99.0	15.10	0.00%	**99**	99.0	2.77	0.00%	**99**	99.0	3.35	0.00%
20kroD100	99	**99**	99.0	13.13	0.00%	**99**	99.0	3.71	0.00%	**99**	99.0	3.50	0.00%
20kroE100	99	**99**	99.0	13.73	0.00%	**99**	99.0	2.95	0.00%	**99**	99.0	3.12	0.00%
20rat99	98	**98**	98.0	14.05	0.00%	**98**	98.0	3.47	0.00%	**98**	98.0	3.15	0.00%
20rd100	99	**99**	99.0	13.60	0.00%	**99**	99.0	3.19	0.00%	**99**	99.0	2.57	0.00%
21eil101	100	**100**	100.0	14.59	0.00%	**100**	100.0	3.85	0.00%	**100**	100.0	3.31	0.00%
21lin105	104	**104**	104.0	13.57	0.00%	**104**	104.0	3.42	0.00%	**104**	104.0	3.28	0.00%
22pr107	106	**106**	106.0	14.53	0.00%	**106**	106.0	4.17	0.00%	**106**	106.0	3.26	0.00%
25pr124	123	**123**	123.0	14.85	0.00%	**123**	123.0	5.50	0.00%	**123**	122.6	4.21	0.00%
26bier127	126	**126**	126.0	20.75	0.00%	**126**	126.0	7.33	0.00%	**126**	126.0	5.91	0.00%
26ch130	129	**129**	129.0	17.96	0.00%	**129**	129.0	6.27	0.00%	**129**	129.0	4.50	0.00%
28pr136	135	**135**	135.0	16.04	0.00%	**135**	135.0	6.86	0.00%	**135**	134.9	5.66	0.00%
29pr144	143	**143**	143.0	21.09	0.00%	**143**	143.0	7.60	0.00%	**143**	143.0	4.72	0.00%
30ch150	149	**149**	149.0	20.14	0.00%	**149**	149.0	8.72	0.00%	**149**	149.0	5.73	0.00%
30kroA150	149	**149**	149.0	18.38	0.00%	**149**	149.0	8.39	0.00%	**149**	149.0	5.60	0.00%
30kroB150	149	**149**	149.0	19.66	0.00%	**149**	149.0	9.83	0.00%	**149**	149.0	6.29	0.00%
31pr152	151	**151**	151.0	23.03	0.00%	**151**	151.0	9.03	0.00%	**151**	151.0	5.08	0.00%
32u159	158	**158**	158.0	21.14	0.00%	**158**	158.0	9.47	0.00%	**158**	158.0	6.10	0.00%
39rat195	194	**194**	194.0	23.86	0.00%	**194**	193.0	15.90	0.00%	**194**	193.0	9.32	0.00%
40d198	196	**196**	196.0	32.73	0.00%	**196**	196.0	18.76	0.00%	**196**	196.0	6.55	0.00%
40kroa200	199	**199**	199.0	32.63	0.00%	**199**	199.0	22.97	0.00%	**199**	199.0	10.69	0.00%
40krob200	199	**199**	199.0	32.61	0.00%	**199**	199.0	21.24	0.00%	**199**	199.0	10.74	0.00%
45ts225	224	**224**	224.0	34.58	0.00%	**224**	224.0	24.55	0.00%	**224**	224.0	11.94	0.00%
45tsp225	224	**224**	224.0	36.78	0.00%	**224**	224.0	24.78	0.00%	**224**	224.0	14.22	0.00%
46pr226	225	**225**	225.0	31.27	0.00%	**225**	225.0	21.91	0.00%	**225**	225.0	12.69	0.00%
53gil262	261	**261**	261.0	51.16	0.00%	**261**	261.0	51.35	0.00%	**261**	260.6	20.19	0.00%
53pr264	263	**263**	263.0	89.95	0.00%	**263**	262.6	57.10	0.00%	262	262.0	16.55	0.38%
56a280	279	**279**	279.0	35.49	0.00%	**279**	279.0	38.54	0.00%	**279**	276.6	22.93	0.00%
60pr299	298	**298**	298.0	55.32	0.00%	**298**	298.0	64.30	0.00%	**298**	295.8	27.52	0.00%
64lin318	317	**317**	317.0	57.59	0.00%	**317**	317.0	83.70	0.00%	**317**	316.8	31.85	0.00%
80rd400	399	**399**	399.0	102.15	0.00%	**399**	399.0	152.72	0.00%	**399**	398.1	40.54	0.00%
84fl417	416	**416**	416.0	436.19	0.00%	**416**	416.0	157.09	0.00%	**416**	415.4	51.57	0.00%
88pr439	438	**438**	438.0	145.14	0.00%	**438**	438.0	254.24	0.00%	**438**	438.0	48.15	0.00%
89pcb442	441	**441**	441.0	117.60	0.00%	**441**	441.0	187.48	0.00%	**441**	440.3	66.32	0.00%
99d493	492	**492**	492.0	216.12	0.00%	**492**	492.0	280.00	0.00%	491	491.0	84.30	0.20%
115rat575	574	**574**	574.0	177.18	0.00%	**574**	572.0	285.25	0.00%	568	564.7	118.68	1.05%
115u574	573	**573**	573.0	204.19	0.00%	**573**	573.0	436.19	0.00%	**573**	572.5	118.14	0.00%
131p654	653	**653**	653.0	930.05	0.00%	**653**	653.0	557.37	0.00%	**653**	653.0	139.04	0.00%
132d657	656	**656**	656.0	271.87	0.00%	**656**	655.6	501.77	0.00%	655	652.4	182.63	0.15%
145u724	723	**723**	723.0	300.99	0.00%	**723**	723.0	658.26	0.00%	721	712.2	228.41	0.28%
157rat783	782	**782**	782.0	459.64	0.00%	**782**	781.2	684.39	0.00%	773	761.0	261.05	1.15%
201pr1002	1001	**1001**	1001.0	836.03	0.00%	**1001**	1001.0	1858.81	0.00%	996	992.2	547.36	0.50%
212u1060	1059	**1059**	1059.0	907.05	0.00%	**1059**	1059.0	2496.95	0.00%	1058	1057.3	675.71	0.09%
217vm1084	1083	**1083**	1083.0	946.07	0.00%	**1083**	1083.0	2693.35	0.00%	**1083**	1082.2	677.32	0.00%
Avg				**136.21**	**0.00%**			**230.75**	**0.00%**			**69.24**	**0.07%**
#Best					**51**				**51**				**43**
Dev.st%			**0.00%**				**0.09%**				**0.32%**		

Table A.15: Comparison among MASOP, VNS-SOP and BRKGA on $Set2$ instances with $\omega = 0.8$ and g_1 profit.

Set2				$\omega = 0.8$	and	g_2							
		MASOP				**VNS-SOP**				**BRKGA**			
Instance	**Best**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**	**Sol**	Sol_{avg}	**Time**	**Gap%**
11berlin52	2608	**2608**	2608.0	8.38	0.00%	**2608**	2608.0	0.82	0.00%	**2608**	2608.0	2.35	0.00%
11eil51	2575	**2575**	2575.0	7.35	0.00%	**2575**	2575.0	0.77	0.00%	**2575**	2575.0	1.90	0.00%
14st70	3513	**3513**	3513.0	10.68	0.00%	**3513**	3513.0	1.41	0.00%	**3513**	3513.0	1.64	0.00%
16eil76	3800	**3800**	3800.0	10.53	0.00%	**3800**	3800.0	1.72	0.00%	**3800**	3800.0	2.33	0.00%
16pr76	3800	**3800**	3800.0	12.10	0.00%	**3800**	3800.0	1.85	0.00%	**3800**	3800.0	2.01	0.00%
20kroA100	5008	**5008**	5008.0	13.84	0.00%	**5008**	5008.0	3.35	0.00%	**5008**	5008.0	3.15	0.00%
20kroB100	5008	**5008**	5008.0	14.61	0.00%	**5008**	5008.0	3.58	0.00%	**5008**	5008.0	3.33	0.00%
20kroC100	5008	**5008**	5008.0	14.54	0.00%	**5008**	5008.0	2.95	0.00%	**5008**	5008.0	3.24	0.00%
20kroD100	5008	**5008**	5008.0	13.90	0.00%	**5008**	5008.0	3.50	0.00%	**5008**	5008.0	3.18	0.00%
20kroE100	5008	**5008**	5008.0	14.84	0.00%	**5008**	5008.0	3.15	0.00%	**5008**	5008.0	2.85	0.00%
20rat99	5007	**5007**	5007.0	14.50	0.00%	**5007**	5007.0	3.23	0.00%	**5007**	5007.0	3.12	0.00%
20rd100	5008	**5008**	5008.0	13.12	0.00%	**5008**	5008.0	3.25	0.00%	**5008**	5008.0	2.44	0.00%
21eil101	5050	**5050**	5050.0	13.86	0.00%	**5050**	5050.0	3.70	0.00%	**5050**	5050.0	3.24	0.00%
21lin105	5228	**5228**	5228.0	14.40	0.00%	**5228**	5228.0	3.52	0.00%	**5228**	5228.0	1.92	0.00%
22pr107	5363	**5363**	5363.0	13.44	0.00%	**5363**	5363.0	4.03	0.00%	**5363**	5363.0	2.82	0.00%
25pr124	6232	**6232**	6232.0	15.44	0.00%	**6232**	6232.0	5.52	0.00%	**6232**	6232.0	4.18	0.00%
26bier127	6333	**6333**	6333.0	21.26	0.00%	**6333**	6333.0	7.61	0.00%	**6333**	6333.0	5.19	0.00%
26ch130	6503	**6503**	6503.0	18.55	0.00%	**6503**	6503.0	6.55	0.00%	**6503**	6503.0	4.28	0.00%
28pr136	6850	**6850**	6850.0	16.64	0.00%	**6850**	6850.0	7.45	0.00%	**6850**	6850.0	5.79	0.00%
29pr144	7242	**7242**	7242.0	20.51	0.00%	**7242**	7242.0	9.49	0.00%	**7242**	7242.0	4.18	0.00%
30ch150	7533	**7533**	7533.0	20.71	0.00%	**7533**	7533.0	10.34	0.00%	**7533**	7533.0	5.42	0.00%
30kroA150	7533	**7533**	7533.0	19.23	0.00%	**7533**	7533.0	8.13	0.00%	**7533**	7533.0	5.30	0.00%
30kroB150	7533	**7533**	7533.0	19.51	0.00%	**7533**	7533.0	9.72	0.00%	**7533**	7533.0	5.35	0.00%
31pr152	7658	**7658**	7658.0	22.62	0.00%	**7658**	7658.0	9.83	0.00%	**7658**	7658.0	4.84	0.00%
32u159	8037	**8037**	8037.0	21.55	0.00%	**8037**	8037.0	10.31	0.00%	**8037**	8037.0	5.82	0.00%
39rat195	9863	**9863**	9863.0	23.17	0.00%	**9863**	9854.6	18.56	0.00%	**9863**	9837.8	9.29	0.00%
40d198	9952	**9952**	9952.0	33.04	0.00%	**9952**	9952.0	20.71	0.00%	**9952**	9952.0	5.90	0.00%
40kroa200	10058	**10058**	10058.0	32.32	0.00%	**10058**	10058.0	22.24	0.00%	**10058**	10058.0	10.19	0.00%
40krob200	10058	**10058**	10058.0	31.94	0.00%	**10058**	10058.0	17.47	0.00%	**10058**	10058.0	10.64	0.00%
45ts225	11308	**11308**	11308.0	39.18	0.00%	**11308**	11308.0	26.65	0.00%	**11308**	11308.0	12.70	0.00%
45tsp225	11308	**11308**	11308.0	38.65	0.00%	**11308**	11308.0	28.39	0.00%	**11308**	11308.0	13.56	0.00%
46pr226	11375	**11375**	11375.0	32.05	0.00%	**11375**	11375.0	22.06	0.00%	**11375**	11375.0	12.12	0.00%
53gil262	13193	**13193**	13193.0	57.50	0.00%	**13193**	13193.0	45.55	0.00%	**13193**	13179.2	19.44	0.00%
53pr264	13302	**13302**	13302.0	87.98	0.00%	**13302**	13274.4	59.36	0.00%	13210	13210.0	15.57	0.69%
56a280	14178	**14178**	14178.0	32.50	0.00%	**14178**	14162.1	47.97	0.00%	**14178**	14079.7	23.83	0.00%
60pr299	15107	**15107**	15107.0	55.19	0.00%	**15107**	15107.0	63.51	0.00%	**15107**	15092.5	27.95	0.00%
64lin318	16037	**16037**	16037.0	61.30	0.00%	**16037**	16037.0	80.12	0.00%	**16037**	16026.8	30.42	0.00%
80rd400	20158	**20158**	20158.0	107.21	0.00%	**20158**	20158.0	139.52	0.00%	**20158**	20135.5	38.28	0.00%
84fl417	21048	**21048**	21048.0	454.93	0.00%	**21048**	21048.0	182.68	0.00%	**21048**	21044.7	54.20	0.00%
88pr439	22177	**22177**	22177.0	137.14	0.00%	**22177**	22177.0	233.59	0.00%	**22177**	22177.0	47.51	0.00%
89pcb442	22323	**22323**	22323.0	117.49	0.00%	**22323**	22323.0	183.29	0.00%	**22323**	22294.5	64.08	0.00%
99d493	24862	**24862**	24862.0	232.25	0.00%	**24862**	24862.0	268.64	0.00%	**24862**	24832.5	84.35	0.00%
115rat575	29033	**29033**	29033.0	154.57	0.00%	**29033**	28942.5	270.31	0.00%	28856	28545.3	115.64	0.61%
115u574	28957	**28957**	28957.0	211.46	0.00%	**28957**	28957.0	425.84	0.00%	**28957**	28950.8	122.48	0.00%
131p654	32997	**32997**	32997.0	1327.59	0.00%	**32997**	32997.0	534.84	0.00%	**32997**	32997.0	141.35	0.00%
132d657	33188	**33188**	33188.0	263.79	0.00%	**33188**	33181.3	519.49	0.00%	33121	33005.2	173.52	0.20%
145u724	36532	**36532**	36532.0	325.98	0.00%	**36532**	36532.0	690.59	0.00%	36444	36274.9	220.06	0.24%
157rat783	39517	**39517**	39517.0	463.61	0.00%	**39517**	39493.9	647.08	0.00%	39113	38656.5	264.39	1.02%
201pr1002	50583	**50583**	50583.0	1010.01	0.00%	**50583**	50583.0	1850.51	0.00%	50496	50148.9	547.91	0.17%
212u1060	53548	**53548**	53548.0	839.06	0.00%	**53548**	53548.0	2552.23	0.00%	53468	53430.1	648.67	0.15%
217vm1084	54712	**54712**	54712.0	1038.93	0.00%	**54712**	54712.0	2827.31	0.00%	**54712**	54712.0	653.10	0.00%
Avg				**148.92**	**0.00%**			**233.42**	**0.00%**			**67.78**	**0.06%**
#Best					**51**				**51**				**44**
Dev.st%			**0.00%**				**0.06%**				**0.26%**		

Table A.16: Comparison among MASOP, VNS-SOP and BRKGA on *Set*2 instances with $\omega = 0.8$ and g_2 profit.