Biased Random-Key Genetic Algorithms for the Winner Determination Problem in Combinatorial Auctions

Carlos Eduardo de Andrade andrade@ic.unicamp.br / ce.andrade@gmail.com
Institute of Computing, University of Campinas, Avenida Albert Einstein 1251, Campinas, SP 13083-852, Brazil

Rodrigo Franco Toso rtoso@cs.rutgers.edu
Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854, USA

Mauricio G. C. Resende mgcr@research.att.com
Network Evolution Research Department, AT&T Labs Research, 200 S. Laurel Avenue, Middletown, NJ 07748, USA

Flávio Keidi Miyazawa fkm@ic.unicamp.br
Institute of Computing, University of Campinas, Avenida Albert Einstein 1251, Campinas, SP 13083-852, Brazil

doi:10.1162/EVCO_a_00138

Abstract

In this paper we address the problem of picking a subset of bids in a general combinatorial auction so as to maximize the overall profit using the first-price model. This winner determination problem assumes that a single bidding round is held to determine both the winners and prices to be paid. We introduce six variants of biased random-key genetic algorithms for this problem. Three of them use a novel initialization technique that makes use of solutions of intermediate linear programming relaxations of an exact mixed integer linear programming model as initial chromosomes of the population. An experimental evaluation compares the effectiveness of the proposed algorithms with the standard mixed linear integer programming formulation, a specialized exact algorithm, and the best-performing heuristics proposed for this problem. The proposed algorithms are competitive and offer strong results, mainly for large-scale auctions.

Keywords

Combinatorial auctions, winner determination problem, genetic algorithms, biased random-key genetic algorithms.

1 Introduction

An auction is a mechanism or negotiation protocol for exchanging goods and services. In general, such goods are offered for bid, followed by a predetermined round of bids, after which the highest bidder is pronounced the winner and pays for the negotiated item. Procurement auctions, on the other hand, are defined as follows. The auctioneer requests a set of goods, and each bidder can submit bids for this set. The lowest bidder is pronounced the winner and the auctioneer is paid for the goods. Today, auctions are widespread and, more important, distributed, thanks mainly to the Internet. Examples can be found in advertisement and position auctions in search engines such as Google

Manuscript received: February 2, 2012; revised: September 12, 2012, June 6, 2013, March 26, 2014, and June 27, 2014; accepted: August 19, 2014.

and Yahoo! as well as those coordinated by governments to negotiate radio spectrum, offshore oil and gas exploration, general goods, and services, among others.

In this paper we are interested in general combinatorial auctions where bidders place bids (usually sealed) on subsets of goods, known as *bundles*. Each bidder has access to a finite set of goods and is asked to come up with a list of bids, where each bid is an offer for a subset of goods. The objective of a bidder is to win the bid at an acceptable price. The greatest advantage of this type of auction is that it generates high economic efficiency because it allows the bidders to express both complementarity and substitutability of their preferences within bids. More formally, let M be a set of goods, $g_1, g_2 \in M$ be two goods, and let $f : 2^M \rightarrow \mathbb{R}$ be a valuation function for sets of these goods. Goods g_1 and g_2 are said to be *complementary* if and only if $f(\{g_1\}) + f(\{g_2\}) \leq f(\{g_1, g_2\})$, where $\{g_1, g_2\}$ denotes a bundle of goods g_1 and g_2. They are said to be *substitutes* if and only if $f(\{g_1\}) + f(\{g_2\}) \geq f(\{g_1, g_2\})$. For other variations, see Parsons et al. (2011). Since we allow bids for any subset of goods, there could be as many as $n(2^m - 1)$ bids, where n is the number of bidders and m is the number of goods. Hence, one of the key problems arising in auction mechanisms is to determine the winners of the auction, that is, selecting pairwise disjoint bids to maximize the sum of the values of the selected bids. For other associated problems, see Cramton et al. (2006).

We focus on the *winner determination problem* (WDP). In general, the WDP is equivalent to the weighted set packing problem, a well-known $\mathcal{NP}$-hard problem (Garey and Johnson, 1979). Notice that solving the WDP in auctions with no additional constraints and where only simple bids are allowed (i.e., bids for a single good) can be easily done in $O(nm)$ time. The seminal work on the WDP is credited to Rothkopf et al. (1998), who identified several special cases that can be solved in polynomial time. Such cases involve special bid structures like bid trees, geometrical regions, and cardinal restricted bids. However, these structures limit the expressiveness of the bids, potentially leading to an inefficient economy (Bichler et al., 2009).

Refined exact approaches to solve the WDP were proposed by Sandholm (2002, 2006) and Escudero et al. (2009), who also presented a polyhedral study applying cuts in an exact algorithm that scaled well in auctions with up to 300 bids. With regard to approximation algorithms, the general case cannot be approximated by a factor of $O(m^{1/2+\epsilon})$ of the optimal total value of the selected bids (unless $\mathcal{P} = \mathcal{NP}$), a bound inherited from the set packing problem [see Halldórsson (2000), who also describes an algorithm with $O(\ell/(\log \ell)^2)$ approximation that runs in $O(\max(\ell^c, m^2\ell^2))$ time, where ℓ is the number of bids and c is a constant]. An approximation algorithm with factor $O(\sqrt{m})$ is described in Lehmann et al. (2002) for the case in which each bidder has interest in only one particular bundle. For more approximation algorithms for special formulations, see Dobzinski et al. (2005) and Feige and Vondrák (2010). Several such algorithms and special cases of the winner determination problem are revisited in Blumrosen and Nisan (2007).

The first heuristic addressing the WDP specifically is the Casanova algorithm (Hoos and Boutilier, 2000), a multistart stochastic local search algorithm that runs on top of greedy randomized initial solutions. A hill-climbing procedure can be found in Holte (2001). The first metaheuristic-based heuristics addressing the problem were a genetic algorithm and a simulated annealing heuristic (Schwind et al., 2003). A hybrid simulated annealing with local search called SAGII was proposed by Guo et al. (2006). To date, the strongest results come from a memetic algorithm by Boughaci et al. (2009) and Boughaci (2013).

It is interesting to observe that the WDP can also be modeled as the *multidimensional knapsack problem* (MDKP), enabling the utilization of the algorithms developed to tackle the latter. As with the weighted set packing, the MDKP is a well-studied $\mathcal{NP}$-hard problem frequently used to evaluate new algorithms because of its intrinsic difficulty and its well-established benchmark test sets. One of the best heuristics to deal with the MDKP was developed by Raidl and Gottlieb (2005) and consists of a genetic algorithm with weight-biased representation using surrogate duality to modify item weights. Mansini and Speranza (2012) presented an exact algorithm based on the idea of restricted core problems where a recursive variable-fixing step is done until a given threshold is reached. The remaining subproblems are explored by a branch-and-bound approach. Several other approaches can be found, among them genetic algorithms (Chu and Beasley, 1998), tabu search (Vasquez and Vimont, 2005), ant-based optimization (Alaya et al., 2004), GRASP (Chardaire et al., 2001), and other hybridization techniques, e.g., Puchinger et al. (2010) and Boyer et al. (2010).

In this paper we address the winner determination problem of general combinatorial auctions using the first-price model for single goods. We restrict ourselves to sealed auctions that use a single round to determine winners and prices to be paid, where the bids can be placed with no constraints other than their non-negativity. We also consider that the bids are anonymous and that bidder identity is not used to model or solve the underlying problem. Six variants of biased random-key genetic algorithms (BRKGAs) are implemented to address this problem, three of them adopting a novel scheme that employs linear programming (LP) relaxations to initialize the population when the underlying problem can be modeled as a 0-1 integer linear program. In such problems an LP relaxation directly serves as a chromosome of the BRKGA heuristic, since both are defined over the interval $[0, 1]$. Experiments comparing the BRKGAs with a standard mixed integer linear programming formulation of the WDP solved with a commercial solver (as well as the best performing heuristics proposed for the problem, the best performing exact algorithm, and the best performing heuristic for the MDKP) are carried out to identify the strengths and drawbacks of each approach.

The paper is organized as follows. In Section 2 we formalize the winner determination problem in combinatorial auctions. We then address biased random-key genetic algorithms in Section 3 and follow up with a description of our heuristics in Section 4 and how to initialize the BRKGA population in Section 5. Experimental results are provided and discussed in Sections 6 and 7, respectively. Concluding remarks are made in Section 8.

2 General Combinatorial Auctions and Their Formulations

Several models for combinatorial auctions have been proposed in the literature, but most of them introduce additional constraints to limit the context of the auction so as to expose special properties that make the problem computationally easier. We present a general description. Let $N = \{1, 2, \ldots, n\}$ be a set of bidders and $M = \{1, 2, \ldots, m\}$ be a set of goods. A collection of bids is represented by a tuple $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_n)$ such that $\mathcal{B}_i$ is the bid set of bidder i. Each bid $B \in \mathcal{B}_i$, for $i = 1, \ldots, n$, is a list of desired goods (bundle), that is, $B \subseteq M$ such that bidder i provides the function $b_i : 2^M \to \mathbb{R}^+$ that measures how much bidder i is willing to pay for a bundle. An allocation of goods is represented by a tuple $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_n)$ where $\mathcal{S}_i$ is the set of winner bids of bidder $i = 1, \ldots, n$. Note that

$$\left(\bigcup_{S \in \mathcal{S}_i} S\right) \cap \left(\bigcup_{R \in \mathcal{S}_j} R\right) = \emptyset, \text{ for all } i, j \in N. \tag{1}$$

We consider the *private information model* in which the auctioneer only knows the set of bids $\mathcal{B}$ and the functions b_i, for all $i = 1, 2, \ldots, n$. We restrict ourselves to first-price sealed auctions where the bidders submit one bid per desired bundle. This contrasts with *iterative auctions*, where the bidders may submit multiple bids to the same bundle in different rounds. Only the auctioneer can handle the bids and the winning bidders pay what they offered in their winning bids, namely, $b_i(\mathcal{S}_i)$. Sealed auctions are used mainly in government and industry procurements. For more on the theory of auctions and its variants, see Krishna (2010).

The major work done in the literature has related the WDP with both the weight set packing problem and the multidimensional knapsack problem (see Bikhchandani and Ostroy [2006] for other models). In the set packing problem, we want to select weighted pairwise disjoint sets from a collection of items while maximizing the sum of the weights of the selected sets. A special case of this problem is the *weighted stable set problem*, where a graph's nodes have weights and one must choose a subset of nodes with no common incident edge and maximize the sum of weights. The WDP can be reduced to the weighted stable set problem in the following way. Consider the intersection graph $G = (V, E)$, where each $s \in V$ represents a bid. An edge $(s, s') \in E$ exists if and only if $B_s \cap B_{s'} \neq \emptyset$, such that $B_s \in \mathcal{B}_i$, $B_{s'} \in \mathcal{B}_j$, $i, j \in N$ and $i \neq j$, i.e., the edge exists if and only if two bids from different bidders request a common good. A stable set on this intersection graph corresponds to a set of pairwise-disjoint bids from different bidders. Mathematically, the stable set problem can be written as the standard integer programming model

$$\begin{aligned} \max \quad & \sum_{s \in V} b_s x_s \\ \text{s.t.} \quad & x_s + x_{s'} \leq 1 \quad \forall (s, s') \in E \\ & x_s \in \{0, 1\} \quad \forall s \in V, \end{aligned} \tag{1}$$

where $b_s = b(B_s)$, namely, b_s is the value offered for bundle s. Let the binary variable $x_s = 1$ if and only if bid s is a winner. This formulation enables overlapping among bids of a same bidder. Its main advantage is that the bidder need not present a bid for each subset of desired goods although the bidder may possibly overpay for some of them. In fact, this formulation is appropriate for superadditive valuations. The number of variables and constraints of this formulation are, respectively, ℓ and $O(\ell^2)$, where ℓ is the total number of bids.

Another very common way to deal with WDP is to model it as a multidimensional knapsack problem. We consider that each bid is an item to be packed in the dimensions induced by the goods. Let $\hat{\mathcal{B}} = \bigcup_{i=1}^{n} \mathcal{B}_i$ be the set of all bids. In case two or more bids contain the same goods, we add a dummy good to each bid such that the new good uniquely identifies the bundle (Nisan, 2000). Let $w_{jk} = 1$ if good $j \in M$ is considered in bid $k \in \hat{\mathcal{B}}$, and $w_{jk} = 0$ otherwise. The MDKP can be modeled as

$$\begin{aligned} \max \quad & \sum_{k \in \hat{\mathcal{B}}} b_k x_k \\ \text{s.t.} \quad & \sum_{k \in \hat{\mathcal{B}}} w_{jk} x_k \leq c_j \quad \forall j \in M \\ & x_k \in \{0, 1\} \quad \forall k \in \hat{\mathcal{B}}. \end{aligned} \tag{2}$$

Again, we abuse the notation of b_k as the value offered for bundle k, and we consider as winning bids all k such that $x_k = 1$. For combinatorial auctions with single goods, we have that $c_j = 1$ for all $j \in M$. A first observation is that this formulation can deal with multiunit auctions where we can have multiple copies of good j (by allowing $c_j \in \mathbb{N}$ for all $j \in M$) and a bid can request a certain number of copies of the good (by allowing $w_{jk} \in \mathbb{N}$, for $j \in M$ and $k \in \hat{\mathcal{B}}$). The number of variables and constraints of this formulation are, respectively, ℓ and $O(m)$, where ℓ is the number of bids and m is the number of goods.

The choice between the stable set and MDKP models can be very tricky in the case of single-unit combinatorial auctions. There are two important aspects to analyze: the tightness of the formulations and their sizes. With respect to tightness, it is well known that the MDKP model (for the single-unit case) generates tighter formulations than the stable set model, since the former contains more clique inequalities than the latter and therefore results in better linear programming relaxations (Padberg, 1973). In fact, it is known that the stable set model, even with clique inequalities, leads to poor relaxations (Carr and Lancia, 2014).

With respect to size, although the MDKP formulation has the size $O(\ell m)$, it can be much larger than the stable set formulation. The problem does not lie in the formulation itself but in the input that can potentially be exponential for the MDKP in the number of goods. To illustrate this, suppose that a bidder has the following bids: $B_1 = (\{1, 2\}, \$10)$, $B_2 = (\{2, 3, 4\}, \$10)$, and $B_3 = (\{4, 5\}, \$10)$. If we consider only these bids, the stable set formulation will have three variables and no constraint, since the overlapping bids belong to the same bidder. In the MDKP the bidder must generate, besides the given bids, the bids $B'_{12} = (\{1, 2, 3, 4\}, \$20)$, $B'_{13} = (\{1, 2, 4, 5\}, \$20)$, $B'_{23} = (\{2, 3, 4, 5\}, \$20)$, and $B'_{123} = (\{1, 2, 3, 4, 5\}, \$30)$, since the bidder cannot win overlapping bids in this model. Although we have only one constraint, the number of variables (bids) is exponential with respect to those in the stable set model. Note that if bidder identities are unknown, we must consider each bid individually and the MDKP becomes the best choice. In this paper we do not consider bidder identities and therefore adopt the MDKP model.

3 Biased Random-Key Genetic Algorithms

To search for good solutions for the winner determination problem, we implemented a biased random-key genetic algorithm (Gonçalves and Resende, 2011a). Our choice was mainly grounded on recent successes with classical hard combinatorial optimization problems such as routing (Andrade et al., 2013), packing (Gonçalves and Resende, 2011b), clustering (Andrade et al., 2014), and others. Although the BRKGA is relatively new, the random-key idea has been used by several authors since the seminal work of Bean (1994). Norman and Bean (1999) used a random-key genetic algorithm to solve complex scheduling problems; Raidl (1999) applied this technique to solve the multiple container packing problem; and Rothlauf et al. (2002) studied several representations of network trees using simple and random-key encoded solutions.

Two key features distinguish BRKGAs from traditional genetic algorithms (Goldberg, 1989):

- A standardized chromosome encoding that uses a vector with t uniformly drawn random keys (alleles) over the interval $[0, 1]$ (Bean, 1994)
- A well-defined evolutionary process that uses parameterized uniform crossover (Spears and DeJong, 1991) for exploitation and substitutes the application of the mutation operator on existing chromosomes with newly

Algorithm 1 BRKGA scheme

```
1 Generate the initial population $P$;
2 while a stopping criteria is not reached do
3     Decode each chromosome of $P$ and extract its solution and fitness;
4     Sort the population $P$ in nonincreasing order of fitness. Consider the top $p_e$
      individuals as the elite group $E$;
5     Copy $E$ to the next generation $Q$, unaltered;
6     Add $p_\mu$ randomly generated new chromosomes (mutants) to $Q$;
7     Generate $p - p_e - p_\mu$ chromosomes (offspring) by parameterized crossover,
      selecting a random parent from $E$ and another from $P \setminus E$. Add them to $Q$;
8     $P \leftarrow Q$;
9 return best individual found.
```

introduced mutants, defined as t-long vectors of (uniformly drawn) random keys, for exploration

Notice that no task depends on the optimization problem for which a solution is being sought. In fact, the only connection between this metaheuristic and the underlying problem occurs when a chromosome is decoded, that is, when a solution to the problem is constructed from a chromosome from which the objective function value or fitness can be extracted for the sake of comparing distinct chromosomes. Analogously, decoders indirectly map the chromosome space $[0, 1]^t$ into the set of feasible solutions to the optimization problem. The pair formed by a chromosome and its fitness is called an individual.

Algorithm 1 summarizes a typical BRKGA framework. Basically, we generate p chromosomes as initial individuals using vectors with t uniformly drawn random keys over the interval $[0, 1]$. In each iteration a problem-specific decoder extracts the fitness of the chromosomes. To build a new population, we copy the p_e best individuals (the elite set), add p_μ random chromosomes (the mutants), and generate $p - p_e - p_\mu$ offspring by applying the crossover operator. Crossover is done between a random individual from the elite set and an individual from the remainder of the population: an offspring is generated by mating, where each allele is taken from the elite parent with probability ρ_e or from the other parent with probability $1 - \rho_e$. With $\rho_e = .5$, the standard uniform crossover occurs. With $\rho_e > .5$ by definition, exploitation happens at two levels: when parents are selected, because one is drawn from the elite set, and when offspring are conceived, because their alleles are inherited from the elite parent with greater probability. Exploration happens with the introduction of mutants at each generation, since they are vectors with t uniformly drawn random keys. The usual mutation operators on individual genes are not employed by the BRKGA. Observe that this scheme prevents infeasibility, since by definition the resulting chromosomes, both offspring and mutants, are always vectors of random keys over $[0, 1]$.

A common approach to genetic algorithms is the island model (Whitley et al., 1998), where several populations are evolved independently and exchange their best individuals every given number of generations. This improves the variability of individuals, usually speeding up convergence, and reduces the risk that the algorithm will get stuck in local optima. Note that it is not necessary that this process be done in parallel in the sense of using several parallel machines or CPUs. It is straightforward to adapt the BRKGA framework: π separate populations are created such that they are evolved simultaneously applying the evolutionary process in lines 3–8 of Algorithm 1 to each population. In this case, we will have $P_1, \ldots, P_\pi$ populations, $E_1, \ldots, E_\pi$ elite sets, and $Q_1, \ldots, Q_\pi$ next-generation pools. The individual exchanges occur when a given threshold is reached, for instance, at every δ generations. For each population P_i, the η best individuals are copied from other populations $P_{j \neq i}$ and replace the $\eta(\pi - 1)$ worst individuals in P_i.

In conclusion, the parameters that must be specified beforehand are the size of the chromosomes t, the size of the population p, the size of elite set p_e, the number of mutants p_μ introduced at each generation, and the inheritance probability ρ_e. If using parallel populations, one must set the number of populations π and the generation threshold δ to exchange the η best individuals. Advice for the parameter setup can be found in Gonçalves and Resende (2011a).

4 Decoding the Winner Determination Problem

We now focus on BRKGA decoders for the winner determination problem. Recall that an instance of the WDP consists of finite sets of bidders $N = \{1, \ldots, n\}$, goods $M = \{1, \ldots, m\}$, and a collection of bids $\mathcal{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_n)$, where $\mathcal{B}_i$ is the bid set of bidder $i \in N$. As mentioned, we have not addressed the bidder identities; instead, we consider the bids $\hat{\mathcal{B}}$, as defined in Section 2, but in some fixed order such that $\hat{\mathcal{B}} = (B_1, B_2, \ldots, B_t)$, where $t = |\hat{\mathcal{B}}|$. Each bid B_j has value b_j. The decoders select a subset of bids that is maximal with respect to the sum of their values while respecting the pairwise-disjoint constraints among selected bids.

We develop three approaches for decoding a solution. These approaches are related in how they select and analyze the bids based on chromosome values and structural information of the problem. Define the size of each chromosome to be t. Our decoders associate each bid with an allele, that is, the value of the jth random key is associated with the jth bid. The first step is to sort the bids in some particular order, generating a permutation of bids.

- **Chromosomal approach.** The keys are sorted in nonincreasing order of their values. Ties are broken by element indices.
- **Greedy approach.** We first choose the keys whose values are greater than or equal to a threshold τ; then these keys are sorted in nonincreasing order of the cost/benefit of their respective bids, that is, $b_j/|B_j|$. Notice that in the greedy approach the relative order of the bids is fixed for all possible chromosomes, and in fact a permutation of the bids is not generated. Instead, we generate an ordered list containing a subset of the original bids. Ties are broken by element indices.
- **Surrogate duality approach.** This is similar to the greedy approach, but the cost/benefit is calculated differently. Let α be the dual solution vector of the relaxation of Formulation (2) when $x \in [0, 1]^t$. Note that each α_i is tied to good i and represents the shadow price of i. The cost/benefit of B_j is $b_j / \sum_{i \in B_j} \alpha_i$. This surrogate duality approach was first proposed by Pirkul (1987). As in the greedy approach, the dual vector α may be computed only once, and ordered lists can then be generated from it. Again, ties are broken by element indices.

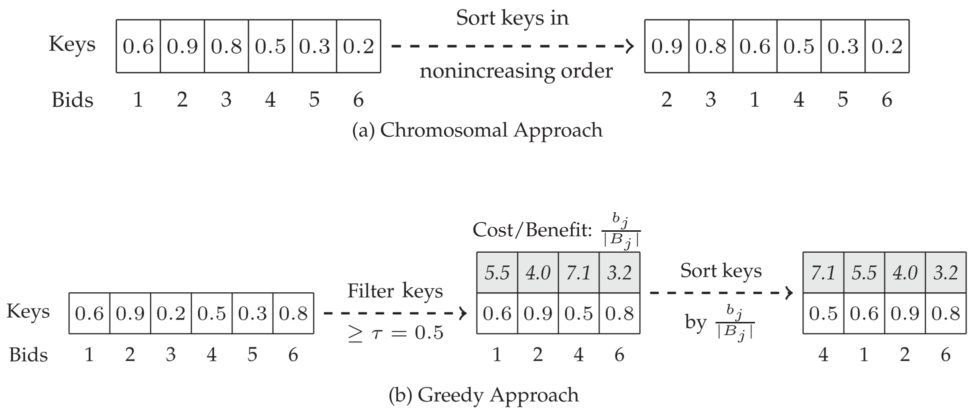


Figure 1: Example of sorting keys using the chromosomal and greedy approaches.

Note that in the greedy and surrogate approaches, the parameter τ induces an implicit binary chromosome encoding and does not take advantage of the magnitude of the keys as does the chromosomal approach. The rationale behind the greedy approach is that the algorithm will take the most efficient bundles at first. This means that it will prefer the bids that most value the goods individually. The surrogate duality approach tries to capture the aggregate consumption levels of goods, meaning that bid efficiency is a measure of how much the bid impacts the entire system when it is chosen as the winner. In other words, if the marginal cost of the goods for a given bid is high, then this bid considers goods with high demand, and it may not be worthwhile to choose it as winner if it were to offer a low value for these goods.

As an example, consider the chromosome in Figure 1. In chromosomal approach, we simply sort the keys generating a permutation of bids, as shown by the indices of the vector in Figure 1a. In the greedy and surrogate duality approaches, we first filter the bids by their keys (using $\tau = .5$ in this example) and then sort the remaining bids in nonincreasing order of their cost/benefit (as shown in the gray vector in Figure 1b).

One can note that in the chromosomal approach, the chromosome is used to generate *a permutation* of bids to be used in the subsequent processing. In the greedy and surrogate duality cases, the chromosome is used to generate a *subset* of bids whose size is controlled by parameter τ. Note that if $\tau = 0$, then all bids are considered at once, and since the relative order of bids is fixed a priori because of the cost/benefit relation, the decoder always returns the same solution. This way, $\tau > 0$ can be viewed as a *separation threshold*. Lines 3–7 of Algorithm 2 summarize these procedures.

The next phase (lines 8–14) uses the sorted list of bids to construct a solution for the WDP. Initially, no bid is selected and all goods are unmarked. The decoder iterates over the bids in the supplied list, selecting a bid whenever all its goods are not yet marked, thus maintaining the property that the winning bids are mutually exclusive with respect to their goods. If the current bid B_j is selected, its corresponding goods are then marked. Otherwise, if bid B_j has a conflicting good with another bid already selected, it is ignored and the value of the corresponding key, say κ_j, is reset to $1 - \kappa_j$ if $\kappa_j > .5$, discouraging this bid from being considered in the following generations. Note that if $\kappa_j \leq .5$, this bid is already discouraged and its key value need not change.

After this primary construction phase, the algorithm has checked all bids in the chromosomal approach. Therefore, in this case, it returns the solution value. In the

Algorithm 2 Decoder for the winner determination problem

```
Let $S$ be an empty list to hold the solution;
Let $\kappa_j$ be the key associated with bid $B_j$;
if the chromosomal approach is used then
    Let $L$ be a list of bid indexes ordered in nonincreasing order of keys $\kappa$;
else
    Let $L$ be a list of bid indexes such that $\kappa_j \geq \tau$ for all bid $B_j$;
    Sort $L$ in nonincreasing order of cost/benefit according to greedy or surrogate
    dual approach;
foreach $j \in L$ in the given order do
    if $B_j$ has no marked goods then
        $S \leftarrow S \cup \{j\}$;
        Mark all goods of $B_j$;
    else if $\kappa_j >$ .5 then
        $\kappa_j \leftarrow 1 - \kappa_j$ ; // discourage bid $B_j$
    $L \leftarrow L \setminus \{j\}$;
if the chromosomal approach is used then
    Go to Line 25;
// Process the remaining bids
Let $L'$ be the list of indexes of remaining bids with unmarked goods;
Sort $L'$ in nonincreasing order of cost/benefit according to greedy or surrogate dual
approach;
foreach $j \in L'$ do
    if $B_j$ has no marked goods then
        $S \leftarrow S \cup \{j\}$;
        Mark all goods of $B_j$;
        if $\kappa_j <$ .5 then
            $\kappa_j \leftarrow 1 - \kappa_j$ ; // encourage bid $B_j$
return the fitness $\sum_{j \in S} b_j$.
```

greedy and surrogate duality cases, some bids are not visited because of filtering by τ. Therefore, the algorithm builds a secondary list containing those bids that include only unmarked goods (disregarding those that were not selected in the previous phase because of conflicts). The bids are then sorted according to one of the preceding criteria, and the algorithm iterates over this list adding the bids that do not create conflict with the bids already selected. Each added bid B_j has its corresponding key κ_j reset to $1 - \kappa_j$ if $\kappa_j < .5$, encouraging this bid to be considered in further generations. This secondary phase is described in lines 17–24.

The running time for this procedure to obtain the sorted list of bids is bounded by $O(t \log t)$, where t is the number of bids.[1] In the worst case, the sum of the number of iterations in the two **foreach** loops is at most t, given that all bids may be analyzed. Checking and marking of goods can be implemented in O(1) using a simple binary vector indexed by the goods. This implies that for each iteration we have $O(m)$ checks

[1]Note that this term depends on the sort algorithm used and in fact can be reduced to $\Theta(t \frac{\log t}{\log \log t})$ using fusion trees (Fredman and Willard, 1993).

and markings in the worst case, where m is the number of goods. Therefore, in the chromosomal case, we can bound the running time of the decoder by $O(t \log t + tm)$.

The running times for the greedy and surrogate duality cases are different from the previous cases, since we have an extra sorting procedure and a second traversal over the bids. As mentioned, the relative order of bids is fixed and need only be calculated once a priori. In this case, the sort procedures of lines 7 and 18 can be done in linear time using an indicator vector, where each position corresponds to the position of a bid in the precalculated order. Note that each sort procedure is done over a partition of the bids, and therefore both together have running times that can be bounded by $O(t)$. The first filtering traversal in line 6 takes t steps. The second traversal in line 17 is a function of τ and takes fewer than t steps. Both loops together take t iterations over m goods. Thus, we can bound the total running time of these decoders by $O(t) + 2t + tm = O(tm)$.

5 Initializing the Population of a BRKGA

The most common approach to initializing the population of a BRKGA is to generate its chromosomes with uniformly drawn random keys over the interval $[0, 1]$. In an attempt to speed up the search, we introduce a novel approach where we use solutions to the linear programming relaxations of Equation (2) as chromosomes, given that the decision variables of these relaxations are such that $0 \leq x_k \leq 1$ for all $k \in \hat{\mathcal{B}}$, where $\hat{\mathcal{B}}$ represents the set of all bids and $t = |\hat{\mathcal{B}}|$. Therefore, a solution to the relaxed LP is a vector $x \in [0, 1]^t$ that is compatible with the requirement of the keys of a BRKGA, and therefore we simply use the values of the optimal relaxed variables x_k as the corresponding alleles of an initial chromosome. An advantage of using such an individual in the initial population is that it is perhaps closer to a good solution than are most random individuals. In addition to the pure relaxation we use relaxations generated by the insertion of cutting planes in the original formulation. A *cutting plane* is an inequality that eliminates an infeasible solution for the original integer program. The insertion of cutting planes leads to tighter formulations with respect to the integer solutions (see, e.g., Wolsey (1998) for more details). We expect that chromosomes generated from these tighter relaxations will be decoded into solutions that are even closer to good integer solutions.

This process consists of two nested phases as shown in Algorithm 3. In the first phase (lines 5–7), cutting planes are generated and added to the formulation and its linear relaxation solved. This results in vector $\tilde{x}$ such that $0 \leq \tilde{x_k} \leq 1$, for all $k = 1, \ldots, t$ except the fixed variables, which have their values defined in next phase. Cut generation procedures have been widely studied in the mathematical programming literature and can be implemented in different ways. In this paper we do not make use of any particular cut generation procedure but rather delegate their generation to the mixed integer programming (MIP) solver. To date, most modern MIP solvers, such as IBM ILOG Cplex, Gurobi Optimizer, and Fico Xpress are able to generate general strong cuts, such as clique cuts (Nemhauser and Wolsey, 1988) and Gomory fractional cuts (Gomory, 1958) known for the tight relaxations they produce. The task of finding cutting planes and reoptimization can be time-consuming, and therefore we limit this procedure to at most a predetermined number of steps or stop after a maximum time limit is reached. The relaxed solution $\tilde{x}$ is added to the initial population.

To generate several different chromosomes, we fix variables iteratively, generating other relaxations (lines 10–16). In alternating iterations, we fix some variable x_s to 0, meaning that the corresponding bid will not belong to any solution, or to 1, implying that the corresponding bid will belong to all solutions. This way, two consecutively generated chromosomes enforce the decision to select or not select the bid in question.

Algorithm 3 Initialization by LP relaxations

```
1  Let $x_1, \dots, x_t$ be a vector such that $x_k$ is the variable associated to bid $B_k \in \hat{\mathcal{B}}$;
2  Let $P$ be the empty initial population;
3  $k \leftarrow 1$;  $bound \leftarrow 0$;
4  while $k < t$ and a stopping criterion is not reached do
5      while maximum cutting iterations or the time limit are not reached do
6          Insert cutting planes in the formulation if possible;
7          Solve the LP relaxation;
8      Let $\tilde{x} \in [0,1]^t$ be the relaxed optimal solution;
9      $P \leftarrow P \cup \{\tilde{x}\}$;
       // Do variable fixing
10     Fix $x_k$ to $bound$;
11     if $bound = 0$ then
12         $bound \leftarrow 1$;
13         if $k \geq 2$ then
14             Unfix $x_{k-1}$;
15     else
16         $bound \leftarrow 0$;  $k$++;
17 if $P$ is not complete then
18     Generate random chromosomes to complete $P$;
```

Note that we fix the variables in the order that they appear in the set of bids. Another possible strategy is to choose a variable to fix uniformly at random, saving the last fixed variable to restore its bounds. Both types of variable-fixing procedures do not guarantee any solution quality but diversify the search. Note that in the first iteration of Algorithm 3, no variable is fixed and a full relaxation of the model is solved. It is also possible, although unlikely, that two or more distinct variable fixings result in the same relaxation. In this case, we discard the duplicates.

Although the initialization with LP relaxations can speed up the convergence of the BRKGA, the time to generate these initial chromosomes is not negligible. It is worthwhile mentioning that solving an LP relaxation is a polynomial time process, but finding cutting planes can be slow in certain situations, and several practical issues can contribute to this slowdown (Wolsey, 1998). In this regard, we set the stopping criterion for this type of initialization to a specific running time or number of chromosomes, whichever comes first (line 4). The remaining chromosomes are generated at random, as is usual in the standard BRKGA.

6 Experimental Setup

We conducted several experiments with three objectives. The first objective was to investigate the effectiveness of biased random-key genetic algorithms to find optimal solutions for instances where exact algorithms succeeded in finding one. The second was to evaluate the solution quality for those instances where an optimal solution could not be found. The third was to investigate the effectiveness of the initialization of BRKGA with LP relaxations. Throughout the experiments we compared our results with state-of-the-art algorithms for the WDP and the MDKP.

Table 1: Instance classes and their sizes.

	CATS								LG		
Bids	40	80	200	400	1,000	1,024[a]	2,000	4,000	1,000	1,000	1,500
Goods	10	10	50	50	256	256	512	1,024	500	1,000	1,500
No. of instances	30	30	30	30	30	27	30	30	100	100	100

[a]Generated using default_hard flag.

6.1 Instances

For the following experiments, we used two sets of instances. We first generated several instances using the Combinatorial Auction Test Suite, or CATS (Leyton-Brown et al., 2011), a standard generator of instances for combinatorial auction, largely adopted in the literature. The advantage of this suite lies in its ability to generate instances for several scenarios, such as time-scheduling auctions, matching auctions, region-border auctions, and even legacy distributions used in earlier papers. We generated two blocks of instances: one of smaller instances containing from 40 to 400 bids whose number of goods vary between 10 and 100; and another comprising larger instances with 1,000 to 4,000 bids and 256 to 1,500 goods. Preliminary experiments showed that the number of goods does not appreciably affect the running time of the algorithms. This fact was also observed by Buer and Pankratz (2010). Then, we set the number of goods to be smaller than the number of bids in the test problems, seeking auctions with relevant conflicts among the bids, i.e., with several bids competing for the same sets of goods.

From CATS we used legacy distributions L2, L3, L4, L6, L7 and the "arbitrary," "matching," "paths," "regions," and "scheduling" distributions (a total of 10 classes). We did not use the L1 and L5 distributions because of problems generating nondominated bids. These instances broadly cover general combinatorial auctions. For further details, see Cramton et al. (2006, chapters 18, 19). For each distribution, we generated three instances of each type using the default parameters supplied by CATS (see Table 1). We also used the CATS hard mode (default_hard flag), which generates three instances with approximately 1,024 bids and 256 goods for each distribution with the objective of being hard to solve. The suite does not generate hard instances for "path" distributions. In summary, we used CATS to generate 120 small and 117 large instances.

A drawback of CATS is that instances appear to be easy in the sense that they can generally be solved by exact algorithms in reasonable time (see Boughaci et al. [2009]; Guo et al. [2006]). In fact, such studies adopted a set of instances provided by Lau and Goh (2002), which are indeed harder than instances generated by CATS. These instances were generated using several factors observed in real brokering systems, such as pricing of a bundle, preference of each bidder, and fairness of goods distributions. We selected three classes of such instances and called them LG (see Table 1). Each class contains 100 instances, all having more than 1,000 bids.

In short, we experimentally analyzed the proposed algorithms on 537 instances, of which 417 are large with respect to number of bids, that is, they have more than 1,000 bids. Table 1 summarizes the instances adopted in the subsequent analysis. The last row of this table shows the number of instances in each class.[2]

[2]These instances can be found at http://www.loco.ic.unicamp.br/instances/wdp.html

6.2 Algorithms

In our evaluation we considered specialized algorithms for both the winner determination problem and the multidimensional knapsack problem. We tested two exact algorithms and two heuristics, both considered to be state-of-the-art for both problems.

To tune the parameters of the heuristics, we used the iterated racing procedure (Birattari et al., 2010). This method consists of sampling configurations from a particular distribution, evaluating them using either the Friedman test or the *t*-test, and refining the sampling distribution with repeated applications of F-race. We used the irace package (López-Ibáñez et al., 2011), implemented in R, for parameter tuning. For each heuristic, we used a budget of 2,000 experiments in the tuning procedure, where each experiment was limited to one hour. For this purpose, we chose one instance of each size from each CATS class, and ten instances from each LG class, totaling 109 instances.

6.2.1 Boughaci et al. Memetic Algorithm, $\mathbf{BO_{MA}}$

Boughaci et al. (2009) present a specialized memetic algorithm for WDP. It is a genetic algorithm that uses random-key encoding tied to a local search procedure for exploitation. Their representation and decoding phase is similar to our chromosomal approach. But in the reproduction phase the individuals are chosen to crossover only if they differ sufficiently based on a similarity metric. In this case, the similarity is the size of the intersection of the winning bid sets induced by these individuals.

For crossover, the algorithm chooses an individual X from the set C_1 of best individuals and another individual Y from a set C_2 that contains individuals with small similarity with respect to individuals in C_1. The crossover is done traversing the concatenation XY and choosing no-conflict bids in this order. The local search that characterizes the memetic flavor is the following. With probability wp choose the best bid (one that maximizes the auctioneer's revenue) or with probability $1 - wp$, a random bid. This bid is added to the solution, and all other conflicting bids are removed. This process is repeated for a given number of iterations and returns the best individual found.

This algorithm outperformed other algorithms for the WDP that were previously proposed in the literature (Casanova of Hoos and Boutilier [2000] and SAGII of Guo et al. [2006]) and indeed presents competitive results.

We used the original C implementation provided to us by the author of Boughaci (2013). Their implementation was slightly modified to support timing limits. In parameter tuning, we used the following ranges: population size $pop\_size \in [300, 2, 000]$; $|C_1| \in [5, 20]$; $|C_2| \in [7, 30]$; $wp \in [.1, .5]$; and maximum local search iterations $max\_lsi \in [100, 500]$. The best setup indicated by irace was $pop\_size = 1, 400$; $|C_1| = 12$; $|C_2| = 24$; $wp = .3$; and $max\_lsi = 150$.

6.2.2 Raidl and Gottlieb Weight-Biased Genetic Algorithm, $\mathbf{RG_{RK}}$

Raidl and Gottlieb (2005) proposed a genetic algorithm for the MDKP where a solution is represented by a weight-biased real vector using surrogate duality information in the decoding phase. The authors used several probability distributions to generate the biased vectors. Their experiments showed that following a log-normal distribution often works best. The weighted vector w is generated such that $w_j = (1 + \gamma)^{\mathcal{N}(0,1)}$, where $\mathcal{N}$ denotes a normally distributed random number with mean 0 and unit standard deviation, and $\gamma > 0$ is a parameter that controls the intensity of biasing. Thus, the item j is biased by a new price $p'_j = p_j w_j$.

The decoding phase used the approach of Pirkul (1987) and is similar to our surrogate duality ordering. In this case, the pseudo utility of a item j is $u_j = p'_j / \sum_{i=i}^{m} \alpha_i r_{ij}$, where m is the number of dimensions, α_i is the dual value associated with dimension i, and r_{ij} is the demand of item j in dimension i.

The offspring generation is obtained by selecting two parents via binary tournaments, performing uniform crossover in their characteristic vectors, flipping each bit with probability $1/n$ (mutation probability), performing repair if a capacity constraint is violated, and always applying local improvement. If such a new candidate solution is different from all solutions in the current population, it replaces the worst of them only if the new candidate has a better fitness than the worst solution (Puchinger et al., 2010). This algorithm is to date one of the best heuristics for the MDKP.

We used the Java code provided to us by the authors of Pfeiffer and Rothlauf (2007). In parameter tuning with irace, we used the following ranges: population size $pop\_size \in [300, 2,000]$; tournament size $tourn\_size \in [10, 30]$; and $\gamma \in [.01, .20]$. The best results were obtained with $pop\_size = 500$; $tourn\_size = 20$; and $\gamma = .15$.

6.2.3 Mansini and Speranza Exact Algorithm, CORAL

Mansini and Speranza (2012) presented an exact algorithm for MDKP using the idea of core items. This algorithm divides the problem into subproblems with a limited number of variables. For each subproblem, a recursive variable-fixing procedure is applied trying to fix as many variables as possible. The remaining unfixed variables represent the core items for which it is difficult to decide whether they belong to an optimal solution. For these items, a restricted core problem is built and solved with a branch-and-bound procedure. To speed up the branch-and-bound, several pruning conditions are introduced. This algorithm has a nontrivial implementation, and we omit several details here that can be found in the original publication.

CORAL is considered to be a state-of-the-art exact algorithm for the MDKP. It works particularly well on instances with a large number of items. Its key feature is the ability to continually improve lower bounds using the optimal solutions from their restricted subproblems. However, in instances with a large number of constraints, CORAL has difficulty in finding good solutions, as observed in Mansini and Speranza (2012).

We used the original Java implementation provided to us by the authors. Slight modifications were done to their implementation to support timing limits. All parameters were set as in the original paper.

6.2.4 Standard Mixed Integer Programming Solver, CPLEX

We used the IBM ILOG CPLEX Optimizer as a standard mixed integer programming solver to deal with Formulation (2) directly. CPLEX uses a branch-and-cut algorithm that is a deterministic enumerative procedure that explores a solution space using a bounding process in the solution values of the tree built during the search. Further detail can be found in Wolsey (1998). This type of algorithm has exponential running time in the worst case. The implementation of the IBM ILOG CPLEX Optimizer uses linear programming relaxations to bound the solution values in addition to using primal heuristics to produce integer solutions. According to its documentation, the IBM ILOG CPLEX Optimizer is fully deterministic with default parameters that are used in our experiments.

Note that we use the cut generation procedure in our LP initialization approaches. In that situation, however, we only use solutions from the root node and the first level of the branching tree. In fact, we do not use the CPLEX branching mechanism there but only solve the LP and apply cut generation procedures. All variable fixing is controlled by our procedure, as shown in Algorithm 3.

We used the IBM ILOG CPLEX Optimizer version 12.5.0.0. All default control parameters were used, except time limit, which was set to 3,600 wall-clock seconds, and number of threads, set to four. Using the default settings, CPLEX performs a preprocessing step to try to eliminate variables and constraints and to calculate initial bounds. Unfortunately, we cannot know what methods are used to perform this preprocessing, since CPLEX is a closed-source commercial package.

It is important to note that these settings are used only in the case where CPLEX is run stand-alone. To create the initial chromosomes for our algorithms based on LP relaxations, we set up CPLEX differently, as described next.

6.2.5 Our Approaches

Our algorithms are as follows:

CA_{RA}. The proposed algorithm using the chromosomal approach and random initialization (see Section 4)

CA_{LP}. The proposed algorithm using the chromosomal approach but initialized with the optimal variables from the LP relaxations

GA_{RA}. The proposed algorithm using the greedy approach and random initialization

GA_{LP}. The proposed algorithm using the greedy approach and initialized with the optimal variables from the LP relaxations

SD_{RA}. The proposed algorithm using the surrogate duality approach and random initialization

SD_{LP}. The proposed algorithm using the surrogate duality approach and initialized with the optimal variables from the LP relaxations

The proposed algorithms were written in C++ on top of the BRKGA API of Toso and Resende (2015), which implements all the problem-independent components described in Section 3. Random numbers were generated by an implementation of the Mersenne twister (Matsumoto and Nishimura, 1998), and we used the standard sort algorithm of the C++ Standard Template Library. In particular, this version implements the introsort algorithm whose worst case running time is $O(n \log n)$ (Musser, 1997). Our algorithms used four cores for simultaneous decoding (see Section 3).

To tune the BRKGA parameters with irace, we used the following ranges: elite percentage $\in [0.10, 0.30]$; percentage of mutants introduced at each generation $\in [0.05, 0.20]$; probability of inheriting each allele from elite parent $\rho_e \in [.5, .8]$; number of independent populations $\pi \in [1, 3]$; exchange interval $\delta \in [50, 200]$; and number of elite individuals in an exchange $\eta \in [1, 2]$. The population size was set to $p = \min(10t, 2{,}000)$, where t is the number of bids. The main reason for this upper bound is to bound the running time of each generation and allow the BRKGA to evolve for several generations. We noted that populations with over 2,000 individuals had slow convergence because of the time needed to evaluate each generation. This is true mainly on large instances. The tuning results obtained with irace were very close to the values suggested by Gonçalves and Resende (2011a). The elite size was set to $p_e = \lceil 0.20p \rceil$, the number of mutants to $p_\mu = \lfloor 0.15p \rfloor$, and inheritance probability to $\rho_e = .70$. We evolved $\pi = 3$ populations simultaneously, and once every $\delta = 100$ generations each population exchanged its $\eta = 2$ best solutions with the other populations.

For the greedy and surrogate duality approaches, we set the filter threshold $\tau = .5$. In these cases, we expect that half of the bids are assigned to the first phase allocation and the other half to the second phase. As the algorithm evolves, good bids will have random-key values greater than or equal to τ, and the impact of the second phase will diminish, since bad bids will have their goods marked in the first phase. Note that τ has more impact on initial and mutant chromosomes than on others, since on the latter the random keys evolved to a better solution.

For the approaches with LP-based initialization, we relaxed the integrality constraints of Formulation (2) to $0 \leq x_k \leq 1$. Since this procedure is time-consuming, the restrictions were as follows. The number of initial chromosomes was set to $lp_{init} = \lfloor 0.1p \rfloor$. Note that in Algorithm 3 this number may be small, since it is limited by the number of bids and duplications. If we obtained no duplicates, $lp_{init} \leq 2t + 1$, where t is the number of bids (the additive factor 1 arises from the initial unrestricted relaxation). We allowed two iterations of cut generation or 5 s to generate each chromosome. Both cut generation and the solution of the LP relaxation were done with the IBM ILOG CPLEX Optimizer version 12.5.0.0. We did not use other CPLEX features, such as variable fixing and branch-and-bound.

6.3 Computational Environment and Algorithm Settings

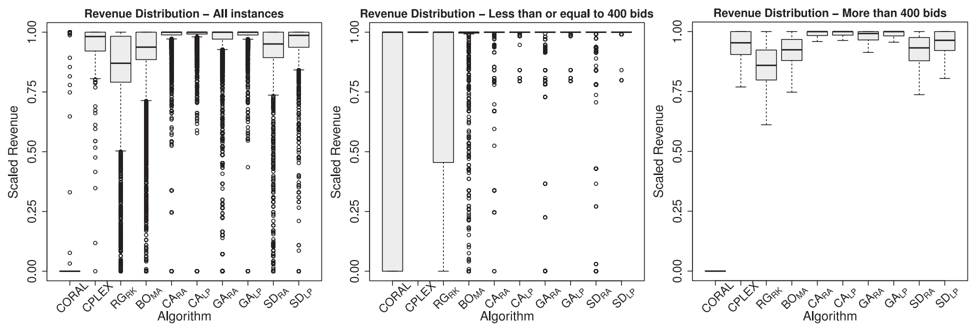
The experiments were conducted on identical machines with quad-core Intel Xeon E5530 2.4 GHz CPUs and 32 GBytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance. Each run was limited to 3,600 s for all algorithms. There are two reasons for our choice of time limit. On the application side, procurement auctions are often managed in a short period of time to limit the response time of the bidders in order to achieve economic efficiency (for further details, see Cramton et al. [2006, chapters 2, 23]). On the algorithmic side, the exact approaches produced a pattern of slow convergence characterized by minimal decrease in the optimality gap throughout the execution on all instances where an optimal solution was not found. This pattern can be clearly identified in the first hour of computation according to preliminary experiments. Notice in the results that follow that some running times are slightly over 3,600 s. This is because we wait for each algorithm to complete its current iteration before actually stopping it.

For the heuristics, we use an additional stopping criterion: 1,000 generations without improvement of the best solution found so far. In preliminary experiments no instance presented an offset greater than 1,000 between two subsequent improvements. In fact, the average offset was about 126.20 ± 138.00, and the largest offset was 791 iterations. This way, we reduced the total computation time, though we may have also reduced the long-term effect of mutation in BO_{MA} and RG_{RK} and, perhaps more seriously, the effect of the introduction of mutants in the BRKGAs.

To compile the C/C++ code, we used the GNU g++ compiler version 4.8.1 and libstdc++ version 6.0.18. To compile the Java code, we used the Oracle Java 64-Bit JDK runtime environment version 1.7.0_45. To allow for full memory utilization, we ran the Java bytecodes with JVM parameter -Xmx32g.

7 Experimental Results and Discussion

For CPLEX and CORAL, we performed one run per instance since both are exact and deterministic algorithms. For the remaining algorithms, we performed 30 independent runs for each instance. Note that this experimental setup is huge, and in fact it took more than 100 CPU days running over 32 identical machines. Each experiment was



(a) (b) (c)

Figure 2: Dispersion of revenue for each algorithm.

conducted individually on one machine to ensure the algorithm had exclusive use of all the machine's resources. This way, we minimized external effects. All reported times are wall-clock times, and we reported only the optimization time, excluding the effort to load the instance and log the run.

7.1 Comparing Revenue

To compare the algorithms with respect to revenue, it is necessary to scale the results, since each instance can have very different revenue values and even different orders of magnitude. For each instance $\mathcal{I}$, let $\chi_{\mathcal{I}}$ be the set of values of the solutions found for $\mathcal{I}$, and $D_{\mathcal{I}} = \max(\chi_{\mathcal{I}}) - \min(\chi_{\mathcal{I}})$. The scaling is done by the simple transformation

$$\chi'_{\mathcal{I}} = \begin{cases} (x - \min(\chi_{\mathcal{I}}))/D_{\mathcal{I}} & \forall x \in \chi_{\mathcal{I}} \text{ and } D_{\mathcal{I}} > 0, \\ 1 & \text{otherwise}, \end{cases}$$

where $\chi'_{\mathcal{I}}$ is the set of scaled values. Note that all values are scaled to the range $[0, 1]$.

With this scaling process Figure 2 shows the distribution of revenues for each algorithm. The box plots show the location of the first quartile, the revenue median, and the third quartile. The whiskers extend to the most extreme revenue no more than 1.5 times the length of the box. The dots are the outliers.

Figure 2a shows the revenue distribution over all instances. Note that CORAL presented poor results when compared to the other algorithms. Indeed, this behavior was expected, since Mansini and Speranza (2012) reported large solution times for instances with 500 bids (items, in their case). This can be clearly observed if we compare the distribution for small instances ($\leq$ 400 bids) in Figure 2b with the distribution for large instances ($\geq$ 1,000 bids) in Figure 2c. CORAL is able to achieve a large range of values on small instances with a median of 0.99, close to the other results. For the large instances, CORAL rarely found a good solution, as shown by its outliers: it obtained only 20 optimal solutions on the large instances. One could argue that these instances are hard to solve by exact algorithms; however, we observe that CPLEX does very well on them. CPLEX's distributions are consistently good, with 113 optimal solutions found on large instances. The heuristics presented overall good results. Among them, RG_{RK} experienced the worst results, followed by BO_{MA}. Our approaches did better than the other algorithms, although among our approaches, it is tricky to determine their relative performance simply by examining the box plots. Note that the utilization of the pseudoutility derived from surrogate duality vectors did not, as expected, have a favorable impact on the results, since SD_{RA} and SD_{LP} presented greater variances and lower medians than our approaches that did not use surrogate duality.

The box plots help shape our intuition that our approaches obtained better results than those of previous methods. To confirm our conclusions, we tested the normality of these distributions using the Shapiro-Wilk test and applied the Mann-Whitney-Wilcoxon U test, considered more effective than the *t*-test for distributions sufficiently far from normal and for sufficiently large sample sizes (Conover, 1980; Fay and Proschan, 2010). For all tests, we assumed a confidence interval of 99%. For small, large, and full distributions, the Shapiro-Wilk tests revealed that no revenue distribution fits a normal distribution, since the p-values for all tests are less than 2.2×10^{-16}. Therefore, we applied the U test, which assumes as null hypothesis that the location statistics are equal in both distributions. As several statistical tests were performed, we used a p-value correction procedure based on false discovery rate (FDR) to minimize the number of false positives (Type I error) as indicated by Benjamini and Hochberg (1995).

We tested the results of each pair of algorithms for small instances, large instances, and the full instance dataset.[3] For a confidence level of 99%, almost all comparisons were statistically significant, indicating differences among the results of the different algorithms (almost all p-values are less than .01). CORAL presented significantly poor performance, and we believe there are two major reasons for this: first, CORAL cannot handle instances with a large number of bids and goods, as previously shown by Mansini and Speranza (2012) and confirmed in Figure 2c. Second, CORAL takes advantage of the cardinality of each dimension constraint. In MDKP, each constraint j is limited to be at most equal to c_j (Formulation (2)), where in general $c_j \geq 1$. In single-good WDPs all constraints have cardinality $c_j = 1$, which does not allow CORAL to take advantage of them, limiting its major feature. Only on small instances does CORAL present good results, although worse than the other approaches.

CPLEX produced solid results that were in general better than those of RG_{RK} and BO_{MA}. With respect to the algorithms proposed in this paper, CPLEX was worse except when compared to SD_{RA}. Comparison between CPLEX and SD_{LP} was inconclusive (p-value $>$ .07). For all small instances, CPLEX obtained an optimal solution. Although the U test returned a value of 0.00 for the differences among the algorithms and CPLEX, we believe this difference is very small in favor of CPLEX, since the other algorithms display more variance than does CPLEX, as shown in Figure 2b. Note that for the approaches using LP-based initialization applied to small instances, the tests against CPLEX were inconclusive (p-values $>$ .08). We discuss this effect in Section 7.5. For large instances, its behavior was similar to that of the general case, where all instances are considered.

The performance of RG_{RK} was worse than that of all heuristics and even CPLEX. This is surprising, since this algorithm has been considered to be one of the best heuristics for the MDKP (Pfeiffer and Rothlauf, 2007). We argue that, like CORAL, RG_{RK} uses in its favor the cardinality of each constraint, since cardinality is correlated with the dual variables used to build the pseudoutilities. In the MDKP this information is very rich, since the multiplicity of the demand of each dimension is weighted by its corresponding dual cost. The WDP has unit cardinalities, and therefore the pseudoutilities are less effective. Note that even in our BRKGAs, both SD_{RA} and SD_{LP}, which make use of surrogate duality, performed worse than variants that did not use this. BO_{MA} performed

[3]The full results are available at http://www.loco.ic.unicamp.br/results/wdp/supplementary_material.pdf

Table 2: Algorithm performance on instances with known and unknown optimum solutions.

	Known Optima (202 instances)					Unknown Optima (335 instances)				
	Optima			Prop. Diff.		Best			Prop. Diff.	
Algorithm	No. of Opt.	% Opt.	% Run	%	σ	No. of Best	% Best	% Run	%	σ
CPLEX	202	100.00	100.00	0.00	0.00	31	9.25	9.25	4.26	2.70
CORAL	52	25.74	25.74	24.49	35.41	3	0.90	0.90	58.36	16.44
RG_{RK}	123	60.89	19.93	3.02	3.22	3	0.90	0.56	8.64	3.57
BO_{MA}	128	63.37	18.90	2.96	4.00	93	27.76	4.05	4.92	2.76
CA_{RA}	171	84.65	24.48	0.95	0.95	200	59.70	26.36	1.00	1.29
CA_{LP}	187	92.57	30.33	0.79	0.80	201	60.00	24.51	1.02	1.28
GA_{RA}	184	91.09	23.66	2.33	3.68	142	42.39	18.89	1.54	1.55
GA_{LP}	186	92.08	28.87	0.89	0.84	200	59.70	23.67	1.14	1.36
SD_{RA}	144	71.29	24.24	7.99	14.74	42	12.54	4.02	4.24	2.45
SD_{LP}	186	92.08	37.17	1.29	1.09	64	19.10	5.90	3.52	2.23

quite well on the entire set of instances, producing results that are slightly worse than those of our approaches but better than previous algorithms.

Most variants of BRKGA outperformed all other algorithms in the general case. Among these variants, one can only point to small differences. The exception to this observation is SD_{RA}, whose location statistics are slightly lower those of CPLEX. It is surprising that the chromosomal approach (of CA_{RA} and CA_{LP}), our most basic approach, showed better results than our other variants. We believe this is so because the greedy and surrogate duality strategies can lead the algorithms to premature convergence to poor local maxima, from which they are unable to escape. Note that for small instances, the tests comparing the LP-based initialized algorithms with CPLEX displayed p-values $> .08$, and therefore we cannot reject the hypothesis that these algorithms have similar performance. If we consider only the large instances, the behavior was similar to the general case. In general, the results for CA_{RA} and GA_{LP} are inconclusive, since in their test the p-value $> .29$.

Table 2 reports the performance of the algorithms considering the instances partitioned into two sets. The first group of columns (2–6) shows the performance considering 202 instances for which an optimal solution was found by CPLEX. There, column 2 represents the number of instances for which the algorithm found an optimal solution; column 3, the percentage of the number of optimal solutions found; and column 4, the percentage of the number of runs on which the algorithm found an optimal solution. The two columns under the label Prop. Diff. show, respectively, the average of the proportional difference between the optimal solution value and the achieved value (%), and its corresponding standard deviation (σ). As we used the optimum solutions obtained by CPLEX, its entries are presented at the maximum levels (minimum levels, in columns 5–6). CORAL found few optimal solutions (25.74% of the 202 instances for which optimal solutions are known), while the other solutions it produced varied widely with respect to quality. Note that both % Opt. and % Run have the same value, since a single run was performed per instance. The heuristics performed well, finding more than 60% of the optimal solutions. With the exception of SD_{RA}, they were never off by more than 4% of the optimal. As expected, the approaches using LP-based initialization often found optimal solutions. However, in some cases they did not reach an optimal solution, suggesting that the relaxation and variable-fixing process did not always produce chromosomes that when evolved are decoded into an optimal solution (see Section 7.5).

The second group of columns (7–11) in Table 2 reports the performance of the algorithms on the 335 instances where no optimal solution is known. It follows the same structure of columns 2–6 but instead of comparing the algorithms with the optimum solution values, we compared them using the best known solutions. CPLEX was able to find a best known solution on 9.25% of the 335 instances, while on the remaining instances it was about 4.26% off the best values. CORAL and RG_{RK} only found three best known solutions, while the average gaps of the solutions it found with respect to the best known solution values were about 58.36% and 8.64%, respectively. Again, we emphasize that the pseudoutility approach did not work well, since RG_{RK}, SD_{RA}, and SD_{LP} presented the worst results among all heuristics. The LP-based initialization approaches again found the best results (with the exception of SD_{LP}), but not as good as the quality of the solutions it found on instances with known optima.

7.2 Iterations and Runtime Analyses

Table 3 shows the average number of iterations taken by the heuristics to find a best solution. The last iterations without improvement performed in the value of the best solution found are disregarded. The first two columns of this table list, respectively, the instance classes and their corresponding sizes. Each following pair of columns shows the average number of iterations to find a best solution and standard deviation for each algorithm, respectively. For instances with 40 and 80 bids, all algorithms converged very early to an optimal solution. An analysis of CA_{RA}, the "most random" of all the algorithms, shows that all instances having 40 and 80 bids are easy. This is so because all but six runs show only a single iteration to reach an optimal solution on 40 bids instances (one took three iterations and another five took two). Note that for the LP-based approaches on instances with 40 and 80 bids, the BRKGA framework did not play any role in the optimization, since all runs took a single iteration (see Section 7.5).

Figure 3 shows performance profiles (Dolan and Moré, 2002) for all algorithms. In performance profiles the abscissa shows the time needed to reach a target solution value (in log scale), while the ordinate shows the cumulative probability to reach a target solution value for the given time in the abscissa. Each algorithm is characterized by a different performance profile curve made up of (time, cumulative probability) pairs, one for each execution of the algorithm on a particular instance. Runs that took over 3,600 s are not shown in the figure. Therefore, the percentage of runs that concluded within the time limit can be seen as the intersection of the profile with the right-hand side of the figure.

Figure 3a shows performance profiles considering only target values of instances for which an optimum solution was found. Figure 3b has as target the values of the best solution found on instances with unknown optimal solution. Finally, Figure 3c takes as target the values of best solutions found for all instances. The solid black line with white squares shows the performance profile for CPLEX. As previously reported, CPLEX is quite fast to find these optimal solutions: in 82% of runs it required less than 1 s, and in 99% less than 1,000 s. Only one run took 1,230 s. Since CPLEX spent about one hour on instances with unknown optima, it does not appear in Figure 3b. In general, CPLEX has the empirical probability of approximately 37% to find a best solution in less than 1,000 s. CORAL is represented by the solid black line with filled

Table 3: Average of iterations in finding the best solution. The last iterations without improvement in the best solution found are disregarded.

		RG_{RK}		BO_{MA}		CA_{RA}		CA_{LP}	
Class	Size	Iterations	σ	Iterations	σ	Iterations	σ	Iterations	σ
CATS	40	2	0.07	2	14.25	1	0.14	1	0.00
	80	2	0.47	21	95.38	1	0.45	1	0.00
	200	17	64.24	287	516.44	29	114.36	1	0.00
	400	64	167.19	348	675.30	109	227.47	30	137.06
	1,000	243	360.07	523	629.76	322	456.50	118	271.27
	1,024	257	345.40	480	561.63	284	448.99	121	282.51
	2,000	232	204.81	435	572.54	651	767.01	220	422.79
	4,000	121	86.07	155	147.78	579	521.75	319	459.41
LG	1,000	95	195.78	509	515.63	197	311.99	197	317.58
	1,500	46	91.45	469	531.82	178	258.24	166	240.78

		GA_{RA}		GA_{LP}		SD_{RA}		SD_{LP}	
Class	Size	Iterations	σ	Iterations	σ	Iterations	σ	Iterations	σ
CATS	40	1	0.55	1	0.00	1	2.65	1	0.00
	80	4	38.27	1	0.00	2	8.25	1	0.00
	200	44	142.41	1	0.00	76	182.31	1	0.00
	400	71	189.38	20	115.19	93	215.54	22	104.42
	1,000	322	511.15	143	361.45	352	447.98	106	256.41
	1,024	308	450.09	127	277.23	287	449.97	100	270.16
	2,000	585	689.49	209	399.79	700	765.56	153	330.10
	4,000	748	745.53	393	567.50	777	643.72	413	637.15
LG	1,000	173	286.25	194	303.76	262	371.30	186	303.13
	1,500	140	231.15	150	224.75	212	280.29	171	248.40

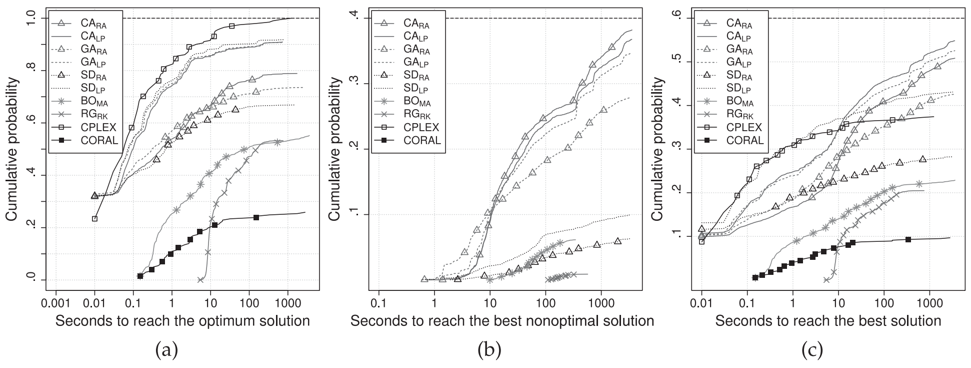


Figure 3: Running time distributions to optimal assignment of winner bids. The identification marks correspond to 0.2% of the points plotted for each algorithm.

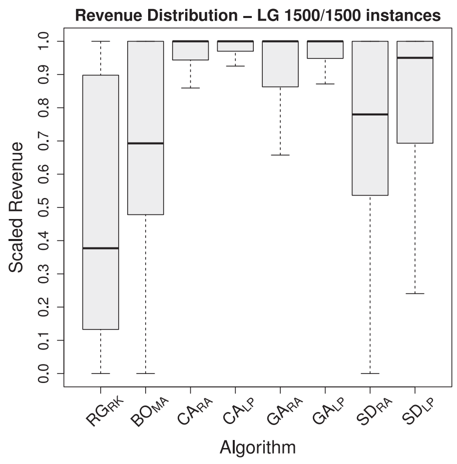


Figure 4: Dispersion of revenue for each algorithm on LG 1,500/1,500 instances.

squares. It found 20% of optimal solutions in less than 10 s but only 26% in less than 3,600 s. For the same reason as CPLEX, CORAL does not appear in Figure 3b and has about a 10% probability of finding a best solution in less than 1,000 s. BO_{MA} (solid green line with asterisks) and RG_{RK} (solid purple line with crosses) are slower than the other algorithms (except CORAL) in most cases. Note in Figure 3b that BO_{MA} has a small advantage over SD_{RA} (dense dashed black line with triangles). RG_{RK} and BO_{MA} found about 55% of the optimal solutions in less than 3,600 s. But, in general, BO_{MA} presents a slightly better probability than RG_{RK}, as shown in Figure 3c. In general, BRKGA variants using LP-based initialization (lines with solid dots) are slower in the first 10 s because of the initialization process, but they outperformed their corresponding counterparts after this. This fact is due to the time needed to create the first LP-based individuals. In fact, the average time of this procedure is 50.71 ± 78.13 s, and the maximum time was 1,377 s. The 377 additional seconds are due to instance setup as a CPLEX model. Considering optimal solutions, CPLEX presents the best time/probability trade-off. Among the heuristics, SD_{LP} (dense dashed line with solid dots) presents the highest probability (approximately 92%). Considering instances with unknown optima, CA_{RA} (solid blue line with triangles) presents the highest probability (approximately 38%). Overall, the best empirical probability was approximately 55% for CA_{LP} (solid blue line with solid dots).

7.3 Comparing the Heuristics on Hard Instances

Since the exact methods can only solve to optimality the small or easy instances, the heuristics play a major role in solving the large instances. The following analysis uses the set of LG 1,500/1,500 instances which proved to be the hardest instances considered in this paper. All algorithms except BO_{MA} reached the time limit on most runs and presented a relatively small number of iterations.

Figure 4 shows the distributions of revenues considering only the heuristics on the LG 1,500/1,500 instances. The values were scaled in the same fashion as was done in Section 7.1. We also performed the U test for each pair of algorithms at confidence level of 99%.[4] For the pairs of algorithms ($\mathrm{CA}_{\mathrm{RA}}, \mathrm{CA}_{\mathrm{LP}}$), ($\mathrm{CA}_{\mathrm{LP}}, \mathrm{GA}_{\mathrm{LP}}$), and ($\mathrm{SD}_{\mathrm{RA}}, \mathrm{SD}_{\mathrm{LP}}$), we cannot reject the hypothesis that the results of each pair are similar, since the *p*-values obtained from the U tests are greater than .01. For the other pairs, the differences presented in Figure 4 are statistically significant. Note that the revenues of $\mathrm{BO}_{\mathrm{MA}}$ are inferior to those of the BRKGAs, which confirms our previous suspicion that $\mathrm{BO}_{\mathrm{MA}}$ converges prematurely, as shown by its number of iterations in Table 3. As in the previous analyses, the algorithms using surrogate duality presented results below those of the other approaches, while $\mathrm{CA}_{\mathrm{RA}}$, $\mathrm{CA}_{\mathrm{LP}}$, and $\mathrm{GA}_{\mathrm{LP}}$ found the best values.

7.4 Comparing the Heuristics on Small Number of Generations

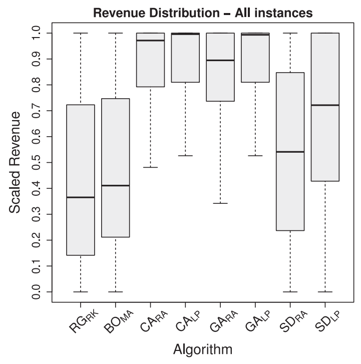
Standard genetic algorithms, such as $\mathrm{RG}_{\mathrm{RK}}$ and $\mathrm{BO}_{\mathrm{MA}}$, often converge quickly, namely, in a small number of generations, to locally optimal, globally suboptimal solutions. Most of the diversification in their population is due to the initial population, since during evolution new individuals are created only by crossover or mutation. Besides creating new individuals by crossover, BRKGAs insert new genetic material into the population at each generation in the form of mutants. This diversification can lead to long runs, that is, having a large number of generations. Tables 2 and 3 show that the BRKGAs and $\mathrm{BO}_{\mathrm{MA}}$ are able to find better solutions using systematically more iterations than $\mathrm{RG}_{\mathrm{RK}}$, suggesting that the adopted stopping criterion of 1,000 generations without improvement of the best solution may favor BRKGAs and $\mathrm{BO}_{\mathrm{MA}}$ over $\mathrm{RG}_{\mathrm{RK}}$.

To address this possible bias, we limit ourselves in this section to considering only the best solutions found by the algorithms in their first 100 generations. We chose 100 generations because this value is close to 95.49, the average number of generations taken by $\mathrm{RG}_{\mathrm{RK}}$ to first find the best solution in a given run. This way, we expect to reduce the impact of inserting new genetic material into the population of a BRKGA. We used the experimental results of previous sections but extracted the values after 100 generations. Note that for large instances the algorithms were not able to reach the one hundredth generation because of the time limit. This is particularly true on large instances with 4,000 bids (and also on some instances with 2,000 bids). Therefore, these large instances are omitted from the following discussion.

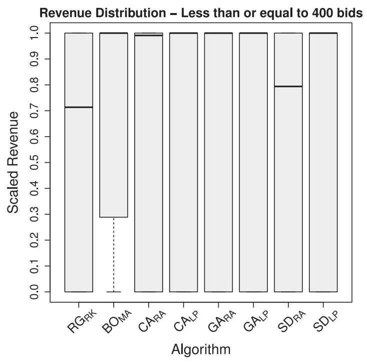
Figure 5 shows the box plots for these results. Their description is similar to Figure 2. We also performed U tests for each pair of algorithms using a confidence level of 99%. In general, $\mathrm{BO}_{\mathrm{MA}}$ outperformed $\mathrm{RG}_{\mathrm{RK}}$ but not the BRKGAs, as observed in previous sections. The exception here is that $\mathrm{BO}_{\mathrm{MA}}$ is significantly better than all other algorithms on small instances (Figure 5b). In general, $\mathrm{CA}_{\mathrm{LP}}$ and $\mathrm{GA}_{\mathrm{LP}}$ presented the best results, although the test between them was inconclusive (*p*-value $> .84$). For large instances, $\mathrm{CA}_{\mathrm{LP}}$ and $\mathrm{GA}_{\mathrm{LP}}$ reached the best results. As before, we cannot affirm which of the two is best (*p*-value > 0.77). For hard LG 1,500/1,500 instances, the tests for $\mathrm{CA}_{\mathrm{RA}}$, $\mathrm{CA}_{\mathrm{LP}}$, and $\mathrm{GA}_{\mathrm{LP}}$ were inconclusive: *p*-values $> .66$.

We observe that even with a small number of generations, the BRKGAs outperformed the other algorithms (except for $\mathrm{BO}_{\mathrm{MA}}$ on small instances). Another interesting observation was the performance of the algorithms with LP initialization, which produced better results than random initialization.

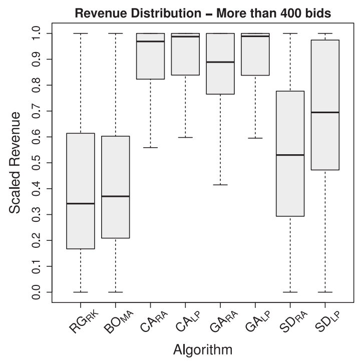
[4]The full results are available at http://www.loco.ic.unicamp.br/results/wdp/supplementary_material.pdf



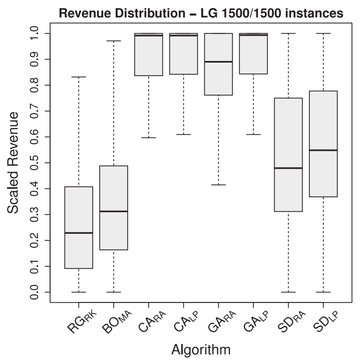
(a) All instances



(b) Small instances



(c) Large instances



(d) LG 1,500/1,500instances

Figure 5: Dispersion of revenue for each heuristic using 100 generations at most.

7.5 Effect of LP-Based Initialization

Notice in Tables 2 and Figure 2 that the approaches using LP-based initialization performed better than the approaches using only random vectors as the initial population. This could suggest that some chromosomes generated by the LP relaxation are often decoded into an optimum solution. In fact, this is not the case, as shown in Table 4. This table shows the average ratio R_{LP} of the revenues of the best chromosome generated by LP relaxations and the best chromosome in the final population, namely, $R_{LP} = best\_lp/best\_final$, for each algorithm and all the instances (hard and easy). The best LP-based chromosome values were obtained with Algorithm 3 and do not include any random individuals. Among all LP-based chromosomes, we select the one with the highest revenue. Note that we also consider random initialized algorithms. In this case, we consider the best random chromosome from the first generation.

Table 4 contains three blocks with respect to the size of the instances, and for each block the Ratio column shows the average ratio R_{LP}, and the σ column shows the corresponding standard deviation. On small instances the LP-based chromosomes generate revenues very close to those in the final population, indicating a possible dominance of LP initial solutions. But the ratio of random initialization and LP-based initialization revenues is very small. This ratio is about 4% larger when we consider large instances exclusively. It is interesting to note that GA_{RA} displays a better ratio than that of GA_{LP}, implying that the results between the first and last generations of GA_{RA}

Table 4: Ratio between the revenues of LP-based chromosomes and the best chromosome.

	Size ≤ 400		Size $\geq 1{,}000$		All	
Algorithm	Ratio	σ	Ratio	σ	Ratio	σ
CA_{RA}	.9848	0.02	.9035	0.06	.9183	0.06
CA_{LP}	.9999	0.00	.9471	0.03	.9574	0.03
GA_{RA}	.9963	0.00	.9669	0.02	.9724	0.02
GA_{LP}	.9999	0.00	.9529	0.03	.9617	0.03
SD_{RA}	.9931	0.01	.9295	0.05	.9427	0.05
SD_{LP}	.9998	0.00	.9640	0.03	.9721	0.03

Table 5: Ratio between the revenues of LP-based chromosomes and the best chromosome by instance class.

							General	
Algorithm	CA_{RA}	CA_{LP}	GA_{RA}	GA_{LP}	SD_{RA}	SD_{LP}	r	σ
L2	.99	**1.00**	.99	**1.00**	.92	**1.00**	.99	0.05
L3	.83	.99	.95	.99	.94	.99	.95	0.08
L4	.92	.99	.98	.99	.97	.99	.97	0.04
L6	.88	.99	.96	.99	.95	.99	.96	0.06
L7	.98	.99	.99	.99	.97	.99	.99	0.02
Arbitrary	.89	.96	.95	.96	.93	.97	.94	0.05
Matching	.89	**1.00**	.97	**1.00**	.96	**1.00**	.97	0.06
Paths	.90	.99	.96	.99	.95	.99	.96	0.05
Regions	.89	.97	.95	.97	.93	.97	.94	0.06
Scheduling	.99	**1.00**	.99	**1.00**	.99	**1.00**	.99	0.01
LG	.91	.93	.97	.94	.93	.94	.94	0.03

are closer together than the corresponding ones for GA_{LP}. However, GA_{LP} presented better results than GA_{RA} (see Section 7.1).

Table 5 presents the R_{LP} ratios by instance class. Each column lists the average ratio of each instance class for each algorithm, with the exception of the last two, which are the overall average r and standard deviation σ. The results are very close to those of Table 4. There are, however, some important points to consider. Note that on the instance classes L2, Matching, and Scheduling, the LP-based initialization algorithms achieved a ratio of $R_{LP} = 1.00$ with a standard deviation of 0.00. This implies that for all instances of these classes, the best solution was found in the first generation. This, in turn, implies that the LP relaxations are able to produce the best solution. This is expected, since such instance classes are known to be easy to solve. In fact, CPLEX was able to find the optimal solutions for these instances in the root node of the branch-and-bound tree, that is, its heuristics were able to find these solutions without enumeration. In these cases, the evolutionary mechanism of BRKGA plays no role in the optimization. However, for other instance classes, BRKGA improves the solution value.

Note that for the CATS instances, the algorithms with LP-based initialization have very tight R_{LP} ratios, namely, around .99, especially when compared with the random initialization approach (for which they are around .95). For LG instances, the R_{LP} ratios are larger than those for CATS; for both initialization approaches, the ratios are around .94. These results were expected, since the CATS instances are easier than the LG instances. Again, it is interesting to note that GA_{RA} presented tighter ratios than GA_{LP} but only on the LG instances. This is not in contradiction with the results of Table 4, since the number of LG instances total more than half of all instances.

8 Concluding Remarks

In this paper we introduced six variants of biased random-key genetic algorithms applied to the winner determination problem in combinatorial auctions. We also proposed a novel initialization scheme for BRKGAs based on intermediate solutions to the LP relaxation of the integer programming model for that problem. Such a scheme can be easily applied in BRKGAs for other 0-1 integer linear programs, since solutions for the LP relaxation of such problems are natural vectors of random keys and thus BRKGA chromosomes. The proposed algorithms have outperformed the standard LP model using a commercial mixed integer programming solver and other heuristic approaches on large CATS instances (Leyton-Brown et al., 2011) as well as on the LG instances (Lau and Goh, 2002). On small CATS instances, where optimal solutions were found with the commercial mixed integer programming solver but not the BRKGAs, the maximum gap between the two was less than 3%, showing that the BRKGAs can still obtain high-quality solutions. Another advantage of BRKGAs is their ability to find good solutions in a very short time, enabling their application in iterative auctions with thousands of goods and bids.

Acknowledgments

The authors thank Dalila Boughaci for kindly providing the instances used in Lau and Goh (2002) and the source code of BO_{MA}. We also are grateful to Renata Mansini and Grazia Speranza for providing the source code of CORAL, and to Jella Pfeiffer for providing her implementation of RG_{RK}. We thank the anonymous reviewers for providing comments that improved this paper significantly. Carlos E. Andrade is supported by São Paulo Research Foundation (FAPESP) grants 2010/05233-5 and 2012/08222-0. Flávio K. Miyazawa is supported by National Council for Scientific and Technological Development (CNPq) grants 306860/2010-4 and 477692/2012-5.

References

Alaya, I., Solnon, C., and Ghédira, K. (2004). Ant algorithm for the multi-dimensional knapsack problem. In *Proceedings of the International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, pages 63–72.

Andrade, C. E., Miyazawa, F. K., and Resende, M.G.C. (2013). Evolutionary algorithm for the *k*-interconnected multi-depot multi-traveling salesmen problem. In *Proceeding of the 15th Annual Conference on Genetic and Evolutionary Computation*, pp. 463–470.

Andrade, C. E., Resende, M.G.C., Karloff, H. J., and Miyazawa, F. K. (2014). Evolutionary algorithms for overlapping correlation clustering. In *Proceedings of the 16th Annual Conference on Genetic and Evolutionary Computation*, pp. 405–412.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 2(6): 154–160.

Benjamini, Y., and Hochberg, Y. (1995). Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society, Series B (Methodological)*, 57(1): 289–300.

Bichler, M., Pikovsky, A., and Setzer, T. (2009). An analysis of design problems in combinatorial procurement auctions. *Business & Information Systems Engineering*, 1:111–117.

Bikhchandani, S., and Ostroy, J. M. (2006). From the assignment model to combinatorial auctions. In P. Cramton, Y. Shoham, and R. Steinberg (Eds.), *Combinatorial auctions*, pp. 189–214. Cambridge, MA: MIT Press.

Birattari, M., Yuan, Z., Balaprakash, P., and Stützle, T. (2010). F-race and iterated F-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and M. Preuss (Eds.), *Experimental methods for the analysis of optimization algorithms*, pp. 311–336. Berlin: Springer.

Blumrosen, L., and Nisan, N. (2007). Combinatorial auctions. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani (Eds.), *Algorithmic game theory*, pp. 267–299. New York: Cambridge University Press.

Boughaci, D. (2013). Metaheuristic approaches for the winner determination problem in combinatorial auction. In X.-S. Yang (Ed.), *Artificial intelligence, evolutionary computing and metaheuristics*, pp. 775–791. Berlin: Springer.

Boughaci, D., Benhamou, B., and Drias, H. (2009). A memetic algorithm for the optimal winner determination problem. *Soft Computing*, 13:905–917.

Boyer, V., Baz, D. E., and Elkihel, M. (2010). Solution of multidimensional knapsack problems via cooperation of dynamic programming and branch and bound. *European Journal of Industrial Engineering*, 4:434–449.

Buer, T., and Pankratz, G. (2010). Solving a bi-objective winner determination problem in a transportation procurement auction. *Logistics Research*, 2:65–78.

Carr, R. D., and Lancia, G. (2014). Ramsey theory and integrality gap for the independent set problem. *Operations Research Letters*, 42(2): 137–139.

Chardaire, P., McKeown, G. P., and Maki, J. A. (2001). Application of GRASP to the multiconstraint knapsack problem. In *Applications of Evolutionary Computing, Evo Workshops 2001*, pp. 30–39. Lecture Notes in Computer Science, Vol. 2037.

Chu, P. C., and Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86.

Conover, W. J. (1980). *Practical nonparametric statistics*, 2nd ed. New York: Wiley.

Cramton, P., Shoham, Y., and Steinberg, R. (Eds.). (2006). *Combinatorial auctions*. Cambridge, MA: MIT Press.

Dobzinski, S., Nisan, N., and Schapira, M. (2005). Approximation algorithms for combinatorial auctions with complement-free bidders. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing*, pp. 610–618.

Dolan, E. D., and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2): 201–213.

Escudero, L. F., Landete, M., and Marín, A. (2009). A branch-and-cut algorithm for the winner determination problem. *Decision Support Systems*, 46(3): 649–659.

Fay, M. P., and Proschan, M. A. (2010). Wilcoxon-Mann-Whitney or *t*-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys*, 4:1–39.

Feige, U., and Vondrák, J. (2010). The submodular welfare problem with demand queries. *Theory of Computing*, 6(1): 247–290.

Fredman, M. L., and Willard, D. E. (1993). Surpassing the information theoretic bound with fusion trees. *Journal of Computer and System Sciences*, 47:424–436.

Garey, M. R., and Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of np-completeness*. San Francisco: Freeman.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Boston: Addison-Wesley.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278.

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

Gonçalves, J. F., and Resende, M.G.C. (2011b). A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22:180–201.

Guo, Y., Lim, A., Rodrigues, B., and Zhu, Y. (2006). Heuristics for a bidding problem. *Computers & Operations Research*, 33(8): 2179–2188.

Halldórsson, M. M. (2000). Approximations of weighted independent set and hereditary subset problems. *Journal of Graph Algorithms and Applications*, 4(1): 1–16.

Holte, R. (2001). Combinatorial auctions, knapsack problems, and hill-climbing search. In *Advances in Artificial Intelligence, AI 2001*, pp. 57–66. Lecture Notes in Computer Science, Vol. 2056.

Hoos, H. H., and Boutilier, C. (2000). Solving combinatorial auctions using stochastic local search. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pp. 22–29.

Krishna, V. (2010). *Auction theory*, 2nd ed. Waltham, MA: Academic Press.

Lau, H. C., and Goh, Y. G. (2002). An intelligent brokering system to support multi-agent web-based 4th-party logistics. In *Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence*, pp. 154–161.

Lehmann, D., O'Callaghan, L. I., and Shoham, Y. (2002). Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49:577–602.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., and Shoham, Y. (2011). CATS: The Combinatorial Auction Test Suite. http://www.cs.ubc.ca/∼kevinlb/CATS/

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, Iterated Race for Automatic Algorithm Configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

Mansini, R., and Speranza, M. G. (2012). CORAL: An exact algorithm for the multidimensional knapsack problem. *INFORMS Journal on Computing*, 24(3): 399–415.

Matsumoto, M., and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30.

Musser, D. R. (1997). Introspective sorting and selection algorithms. *Software Practice and Experience*, 27(8): 983–993.

Nemhauser, G. L., and Wolsey, L. A. (1988). *Integer and combinatorial optimization*. New York: Wiley.

Nisan, N. (2000). Bidding and allocation in combinatorial auctions. In *Proceedings of the 2nd ACM Conference on Electronic Commerce*, pp. 1–12.

Norman, B. A., and Bean, J. C. (1999). A genetic algorithm methodology for complex scheduling problems. *Naval Research Logistics*, 46(2): 199–211.

Padberg, M. W. (1973). On the facial structure of set packing polyhedra. *Mathematical Programming*, 5:199–215.

Parsons, S., Rodriguez-Aguilar, J. A., and Klein, M. (2011). Auctions and bidding: A guide for computer scientists. *ACM Computing Surveys*, 43(2):article 10.

Pfeiffer, J., and Rothlauf, F. (2007). Analysis of greedy heuristics and weight-coded EAs for multidimensional knapsack problems and multi-unit combinatorial auctions. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, p. 1529.

Pirkul, H. (1987). A heuristic solution procedure for the multiconstraint zero-one knapsack problem. *Naval Research Logistics*, 34(2): 161–172.

Puchinger, J., Raidl, G. R., and Pferschy, U. (2010). The multidimensional knapsack problem: Structure and algorithms. *INFORMS Journal on Computing*, 22(2): 250–265.

Raidl, G. R. (1999). The multiple container packing problem: A genetic algorithm approach with weighted codings. *SIGAPP Applied Computing Review*, 7(2): 22–31.

Raidl, G. R., and Gottlieb, J. (2005). Empirical analysis of locality, heritability and heuristic bias in evolutionary algorithms: A case study for the multidimensional knapsack problem. *Evolutionary Computation*, 13(4): 441–475.

Rothkopf, M. H., Pekec, A., and Harstad, R. M. (1998). Computationally manageable combinational auctions. *Management Science*, 44(8): 1131–1147.

Rothlauf, F., Goldberg, D. E., and Heinzl, A. (2002). Network random keys: A tree representation scheme for genetic and evolutionary algorithms. *Evolutionary Computation*, 10:75–97.

Sandholm, T. (2002). Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2): 1–54.

Sandholm, T. (2006). Optimal winner determination algorithms. In P. Cramton, Y. Shoham, and R. Steinberg (Eds.), *Combinatorial auctions*, pp. 337–368. Cambridge, MA: MIT Press.

Schwind, M., Stockheim, T., and Rothlauf, F. (2003). Optimization heuristics for the combinatorial auction problem. In *Congress on Evolutionary Computation*, Vol. 3, pp. 1588–1595.

Spears, W. M., and DeJong, K. A. (1991). On the virtues of parameterized uniform crossover. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pp. 230–236.

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

Vasquez, M., and Vimont, Y. (2005). Improved results on the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, 165(1): 70–81.

Whitley, D., Rana, S., and Heckendorn, R. B. (1998). The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47.

Wolsey, L. A. (1998). *Integer programming*. New York: Wiley.