Approximate Algorithms for Double Combinatorial Auctions for Resource Allocation in Clouds: An Empirical Comparison

Diana Gudu[1], Gabriel Zachmann[1,2], Marcus Hardt[1] and Achim Streit[1]
[1]*Karlsruhe Institute of Technology, Karlsruhe, Germany*
[2]*Baden-Wuerttemberg Cooperative State University, Karlsruhe, Germany*

Keywords: Combinatorial Auction, Resource Allocation, Cloud Computing, Approximate Algorithm.

Abstract: There is an increasing trend towards market-driven resource allocation in cloud computing, which can address customer requirements for flexibility, fine-grained allocation, as well as improve provider revenues. We formulate the cloud resource allocation as a double combinatorial auction. However, combinatorial auctions are NP-hard problems. Determining the allocation optimally is thus intractable in most cases. Various heuristics have been proposed, but their performance and quality of the obtained solutions are highly dependent on the input. In this paper, we perform an extensive empirical comparison of several approximate allocation algorithms for double combinatorial auctions. We discuss their performance, economic efficiency, and the reasons behind the observed variations in approximation quality. Finally, we show that there is no clear winner: no algorithm outperforms the others in all test scenarios. Furthermore, we introduce a novel artificial input generator for combinatorial auctions which uses parameterized random distributions for bundle sizes, resource type selection inside a bundle, and the bid values and reserve prices. We showcase its flexibility, required for thorough benchmark design, through a wide range of test cases.

1 INTRODUCTION

The pay-per-use, on-demand models promoted by cloud computing (Rappa, 2004) have enabled its ubiquity in today's technological landscape. From large businesses moving their services to the cloud, to developers running small tests for their applications, the requirements are becoming increasingly diverse. This diversity is challenging for both cloud providers seeking to improve their profits, as well as customers trying to find a cost-effective option that aligns with their requirements. The advent of open-source cloud technologies (CloudStack, 2017; Openstack, 2017), coupled with a wide availability of low-cost server hardware, has also led to an increase in cloud providers entering the market to meet the ever-growing user demands. This increases the burden on customers to make an informed decision when choosing and combining cloud resources from multiple providers.

Current trends (Buyya et al., 2008) point towards market-driven resource allocation and pricing for a fine-grained, customizable experience for cloud customers. Some commercial cloud providers have already adopted the concept of dynamic pricing in order to maximize their resource utilization and increase their revenue: Amazon is selling unused resources on the so-called spot market (Amazon, 2017), where price is regulated by the fluctuating demand and supply. There is still, however, more research required to move from existing market mechanisms which are fast, simple, but inflexible (such as single-good, one-sided auctions), towards more complex, but flexible and economically efficient mechanisms.

The use of two-sided combinatorial auctions for cloud resource provisioning is a research topic that has been gaining interest (Nejad et al., 2015; Samimi et al., 2014). The combinatorial aspect ensures flexibility in resource provisioning—clients can request exactly the amount of resources they need without being limited to a few predefined bundles offered by providers, such as the Amazon EC2 virtual machine instance types (Amazon, 2017). The two-sided aspect brings more fairness by considering both clients and providers when making allocation decisions, and offloading the decision to a central entity (the auctioneer) instead of each cloud provider.

The main reason why combinatorial auctions are not yet widely used in practice is their computational complexity: combinatorial auctions are $\mathcal{NP}$-hard problems (De Vries and Vohra, 2003). Finding

Gudu, D., Zachmann, G., Hardt, M. and Streit, A.
Approximate Algorithms for Double Combinatorial Auctions for Resource Allocation in Clouds: An Empirical Comparison.
DOI: 10.5220/0006593900580069
In *Proceedings of the 10th International Conference on Agents and Artificial Intelligence (ICAART 2018) - Volume 1*, pages 58-69
ISBN: 978-989-758-275-2

an optimal solution is intractable for large problems. Fast, approximate algorithms exist, but they incur a certain loss in efficiency that needs to be bounded. Due to the assumed scale of cloud allocation as presented above, fast allocation needs to be prioritized over optimal solutions. Nevertheless, any improvement in solution quality can translate into large savings for clients and increases in provider revenue.

Although there is a plethora of approximate algorithms for combinatorial auctions in the literature (Fujishima et al., 1999; Nejad et al., 2015; Lehmann et al., 2002; Holte, 2001; Zurel and Nisan, 2001; Hoos and Boutilier, 2000; Chu and Beasley, 1998; Khuri et al., 1994), to our knowledge there is no comprehensive comparison using a consistent problem formulation and benchmarks.

With this paper, we aim to lay the groundwork for a unified, consistent evaluation of heuristic algorithms for combinatorial auctions. We create a portfolio of algorithms based on existing work or well-known optimization methods, but adapted and improved for our proposed problem formulation. We then introduce a flexible tool for generating artificial datasets for combinatorial auctions, which can be the basis for thorough benchmarking of combinatorial auctions. We perform an extensive empirical comparison of the algorithms in the portfolio and discuss the differences in performance and approximation quality.

2 PROBLEM FORMULATION

A multi-good, multi-unit two-sided combinatorial auction is given by (Nisan et al., 2007b; Gudu et al., 2016): a set U of n buyers or users ($U = \{1,\dots,n\}$), a set P of m sellers or providers ($P = \{1,\dots,m\}$), each offering different quantities of resources G of l types ($G = \{1,\dots,l\}$), and an auctioneer that receives bids and asks, and decides upon the allocation and pricing of resources.

Each buyer $i \in U$ submits a bid for a bundle of resources to the auctioneer, expressed as $(\langle r_{i1},\dots,r_{il}\rangle, b_i)$, where b_i is the value the buyer is willing to pay for the bundle, and r_{ik} is the integer amount of requested resources of type k. Each seller $j \in P$ submits its ask to the auctioneer, expressed as $(\langle s_{j1},\dots,s_{jl}\rangle, \langle a_{j1},\dots,a_{jl}\rangle)$. That is, for each resource type $k \in G$, each seller offers a certain integer quantity s_{jk} at a reserve price a_{jk} (the minimum price for which it is willing to sell one item of the resource). We assume that a buyer can receive the resources in a single bundle from multiple sellers.

For example, a seller like Amazon EC2 might provide 3 types of virtual machines (*t2.small*, *t2.medium* and *t2.large*) at the current on-demand prices (Amazon, 2017) for the EU (London) region: 0.026$, 0.052$ and 0.106$ (per hour). If a seller provides 100 instances of each type, then its ask will be expressed as $(\langle 100,100,100\rangle, \langle 0.026,0.052,0.104\rangle)$. We can also use our problem formulation to model finer-grained allocation: the resources being sold are computing cores, memory, storage space, etc, and the customer can request exactly the amount of resources it needs instead of being constrained by prepackaged VMs. For example, a customer can request a VM with 40 cores, 64 GB of memory and 1 TB disk storage, and is willing to pay at most 10$. Then it submits the following bid: $(\langle 40,64,1000\rangle, 10)$. In this case, however, all the resources in a customer's bundle would have to be allocated by the same provider.

After collecting all bids and asks, the auctioneer must find the optimal allocation, which is defined as the allocation that maximizes the social welfare, i.e. the sum of all trade participants' utilities.

A buyer i's utility for a bundle S that it bids on is defined as: $u_i(S) = v_i(S) - p_i$, if i obtains bundle S, and 0 otherwise. p_i is the price the buyer pays at the end of the trade, and v_i is the buyer's valuation, or the real worth estimated by the buyer for the specific bundle. If the buyer is truthful, then the bid b_i it submits to the auctioneer is equal to its valuation. Seller utility can be defined in a similar way.

We assume that the buyers are *single-minded*: they need all the goods in the bundle (their valuation for any subset of the requested bundle is 0). Sellers, on the other hand, are not *single-minded*: any subset of the offered goods has a positive valuation. Moreover, the seller's valuation function is additive, meaning that the value of a bundle can be computed by summing the values of the goods in that bundle.

Then the winner determination problem (WDP) (Lehmann et al., 2006) can be written as an integer program that maximizes social welfare:

$$\max_{x,w}\left(\sum_{i=1}^{n} b_i x_i - \sum_{j=1}^{m}\sum_{k=1}^{l} a_{jk}\sum_{i=1}^{n} w_{ijk}\right) \tag{1}$$

subject to:

$$x_i \in \{0,1\}, \forall i \in U \tag{2}$$

$$\sum_{j=1}^{m} w_{ijk} = r_{ik}x_i, \forall i \in U, \forall k \in G \tag{3}$$

$$\sum_{i=1}^{n} w_{ijk} \le s_{jk}, \forall j \in P, \forall k \in G \tag{4}$$

where x_i and w_{ijk} define the final allocation: x_i indicates whether bidder i receives the bundle it requested, and w_{ijk} indicates the total number of resources of type k allocated by seller j to buyer i. Con-

straint (2) expresses the single-mindedness of buyers. Constraint (3) ensures that the total amount of resources of a certain type that all sellers allocate to a certain buyer is the same as the amount of resources of that type requested by the buyer. Finally, constraint (4) ensures that a seller cannot sell more than the amount of resources it offered.

2.1 Payment Scheme

Along with the WDP, the payment scheme is an essential part of mechanism design. Only through an appropriate pricing can the desired economic properties of the allocation be ensured. The following four properties are generally targeted, although they cannot be simultaneously satisfied (Myerson and Satterthwaite, 1983): incentive compatibility, individual rationality, economic efficiency and budget-balance.

Vickrey-Clarke-Groves (VCG) (Nisan et al., 2007a) is a class of mechanisms that are both truthful and achieve a socially optimal solution, but are not budget-balanced—the auctioneer has to subsidize the trade. Moreover, VCG pricing is computationally expensive: each winner pays its social cost, i.e. one must solve an instance of the WDP for each trade winner, to compute the social welfare for the case when it does not participate in the auction. This is why we did not use VCG in our work.

Instead, we relax the truthfulness constraint in favor of individual rationality and budget-balance. Depending on the algorithm used for WDP, economic efficiency might not be achieved. However, most heuristics are asymptotically economically efficient, guaranteeing a near-optimal solution (within a factor of $\sqrt{l}$ for the greedy algorithm (Lehmann et al., 2002)). The κ-pricing scheme (Schnizler et al., 2008) meets these requirements.

The general idea behind κ-pricing is to divide the trade surplus among the trade participants in order to ensure budget-balance. In the multi-good multi-unit case, for a buyer i that is allocated resources from multiple sellers, the surplus caused by the trade is:

$$\delta_i = b_i - \sum_{j=1}^{m} \sum_{k=1}^{l} a_{jk} w_{ijk} \tag{5}$$

Bidder i will thus receive a discount of a κ-th part of this surplus, resulting in payment $p_i = (b_i - \kappa\delta_i)x_i$.

The remaining $(1-\kappa)\delta_i$ is divided between all the sellers participating in the exchange generating the surplus, proportional to each seller's share. A seller's share in the trade with bidder i can be calculated by:

$$\beta_{ij} = \frac{\sum_{k=1}^{l} a_{jk} w_{ijk}}{\sum_{j=1}^{m} \sum_{k=1}^{l} a_{jk} w_{ijk}} \tag{6}$$

Adding up the surplus from all the exchanges a seller j is involved in, the final payment of seller j can be calculated by:

$$p_j = \sum_{i=1}^{n} \left(\sum_{k=1}^{l} a_{jk} w_{ijk} + (1-\kappa)\delta_i \beta_{ij} \right) \tag{7}$$

We used $\kappa = 0.5$ for an equal distribution of surplus.

Although the κ-pricing scheme is not incentive compatible, non-truthful bidding increases the risk of no allocation (Schnizler et al., 2008). Therefore, in most cases, the competition in the market motivates the participants to reveal their true valuations. Thus our mechanism is budget-balanced, individually rational, asymptotically economically efficient and, in practice, truthful.

3 ALGORITHM PORTFOLIO

In the following subsections we present different algorithms to solve the WDP. They are either based on existing work or use widely known optimization methods, but adapted to our problem formulation.

Except for branch-and-cut, the algorithms in the portfolio aim to approximate the optimal solution in a reasonable time rather than find the optimal solution. This is done by improving the ordering of bids and asks onto which greedy algorithms are finally applied.

3.1 Branch-and-cut

The WDP can be tackled as a mixed-integer linear program (MILP) (Gonen and Lehmann, 2000). The most used approach for solving MILPs is branch-and-cut (Padberg and Rinaldi, 1991). In the worst case, this approach has exponential complexity, but it always leads to an optimal solution.

We implemented WDP for our combinatorial auction problem using ILOG's CPLEX (IBM, 2017) software and named this algorithm MILP. CPLEX is the most used and most performant software for integer and linear programming problems. Even though it is proprietary software, IBM offers academic licenses.

3.2 Greedy Algorithms

As the name suggests, greedy algorithms are heuristics that make the locally optimal choice at every step, aiming for a globally optimal solution (Cormen et al., 2001). Greedy algorithms thus prioritize speed over solution quality. There is a rich literature of greedy algorithms for the WDP (Lehmann et al., 2002; Samimi et al., 2014; Nejad et al., 2015; Pfeiffer and Rothlauf,

2008), and they all share a simple idea: the bids are sorted according to a certain criterion and then bids are greedily allocated as long as there are no conflicts. In the rest of this section we present our greedy algorithms, adapted to our problem formulation.

We use bid and ask densities as sorting criteria, which are defined for buyer i and seller j in Eqn. 8.

$$d_i = \frac{b_i}{\sqrt{M_i}}, \qquad d_j = \frac{\sum_{k=1}^{l} a_{jk} s_{jk}}{\sqrt{M_j}} \tag{8}$$

Based on (Lehmann et al., 2002) and (Nejad et al., 2015), we redefine M_i and M_j in Eqn. 9.

$$M_i = \sum_{k=1}^{l} f_k^b r_{ik}, \qquad M_j = \sum_{k=1}^{l} f_k^a s_{jk} \tag{9}$$

f_k represents the relative weight of resource type k and it can be used to express differences in value for the resource types. We propose using different weights for bids and asks, f_k^b and f_k^a, respectively.

(Nejad et al., 2015) introduce three options for calculating f_k in one-sided auctions, which they name *relevance factors*. The first option is a generalization of the one-dimensional case of (Lehmann et al., 2002), namely uniform weights. The other options consider the scarcity of resources, either absolute value (the inverse of the provider's capacity of each resource) or relative scarcity (the difference between demand and supply, normalized by demand). We extend these options to the two-sided case as follows ($\forall k \in G$):

1. $f_k^b = f_k^a = 1$
2. $f_k^b = \frac{1}{\sum_{j=1}^{m} s_{jk}}$, $f_k^a = \frac{1}{\sum_{i=1}^{n} r_{ik}}$
3. $f_k^b = \frac{\left|\sum_{i=1}^{n} r_{ik} - \sum_{j=1}^{m} s_{jk}\right|}{\sum_{i=1}^{n} r_{ik}}$, $f_k^a = \frac{\left|\sum_{i=1}^{n} r_{ik} - \sum_{j=1}^{m} s_{jk}\right|}{\sum_{j=1}^{m} s_{jk}}$

Densities are used to sort bids descendingly and asks ascendingly, giving priority to clients with higher bids and providers offering cheaper goods. For each bid, the greedy algorithm then iterates through the asks and allocates resources to the client until its request is fully satisfied. When moving on to the next bid in the list, previous asks are not considered in order to preserve *monotonicity* (Lehmann et al., 2002)—a necessary (but not sufficient) condition for a truthful mechanism. The algorithm stops when it reaches the end of either the bid list (all requests can be satisfied) or the ask list (there are no more providers that can satisfy the requests), based on (Samimi et al., 2014). We call this algorithm Greedy-I. The main problem of Greedy-I is that it is not *individually rational*. That is, some participants might have negative utility by joining the auction, since the algorithm traverses the full lists of bids and asks; the stopping criterion does not take into account the bid and ask values, which might result in negative utility.

In Greedy-II, we propose stopping the algorithm when the surplus caused by satisfying the bid currently considered (as defined in Eqn. 5) becomes negative. We call this bid and the first unallocated ask *critical*. A critical value is defined as the value below which a bid would not be allocated (Lehmann et al., 2002). Although this might result in less goods being traded, the social welfare will be higher since we are mostly eliminating trades that will result in negative utility. We use the three options of calculating the relevance factors and name these variations Greedy-II fk1, Greedy-II fk2 and Greedy-II fk3.

Greedy-I performs much worse for sparse providers (offering only a subset of the resource types), yielding a social welfare at least two orders of magnitude smaller than the dense providers case (Gudu et al., 2016). The reason for this is that, while trying to preserve monotonicity, the algorithm skips providers that cannot satisfy the request of the currently considered bidder, and a large quantity of resources remain unallocated. In Greedy-III, we propose to fix this issue by considering partially allocated, previously considered (i.e. lower density) asks for each buyer request, therefore returning to the beginning of the ask list for each bid. This increases the complexity of the algorithm and violates the monotonicity property, but leads to a higher social welfare. Another consequence is that the goods allocated to one bidder will be less likely to come from the same provider (or a small number). Although this resource locality is not an explicit constraint in our problem definition, it might be a soft constraint in some cases.

Overlooking resource locality and monotonicity led to an improved algorithm, Greedy-IV, which still sorts the bids by density, but keeps an index of sorted asks for each resource type, sorting them by the asking price of the respective resource type (a total of l lists). The bids are then greedily allocated: the bidder with the highest density receives the cheapest resources over all providers. Greedy-IV has a higher time and memory complexity, but can also result in higher welfare than the other algorithms.

3.3 Relaxed Linear Program-based

Relaxing the integrality constraints of the decision variables x_i and quantities w_{ijk} transforms the integer program in Eqn. 1 into a linear program that can be solved faster (weakly polynomial time when using interior point methods, or exponential time for simplex methods (Luenberger and Ye, 2015)) and that

provides an upper bound for social welfare. We implemented two algorithms based on the relaxed linear problem, based on (Pfeiffer and Rothlauf, 2008).

The first one is called Relaxed LP Solution (RLPS) and it applies Greedy-II on a set of bids and asks sorted descendingly by their continuous decision variables, which in our case are x'_i and $\frac{\sum_{i=1}^{n}\sum_{k=1}^{l} w'_{ijk}}{\sum_{k=1}^{l} s_{jk}}$ for bids and asks, respectively. By x'_i and w'_{ijk} we denote the solutions of the relaxed linear program.

The second algorithm, Shadow Surplus (SS), uses the solution of the dual linear program (w''_{ijk}) to compute shadow surpluses for bids and asks as in Eqn. 10. Shadow surpluses are then used to sort bids descendingly and asks ascendingly, and then applying Greedy-II to compute the allocation.

$$SS_i = \frac{b_i}{\sum_{k=1}^{l}\sum_{j=1}^{m} w''_{ijk} r_{ik}}, SS_j = \frac{\sum_{k=1}^{l} a_{jk} s_{jk}}{\sum_{k=1}^{l}\sum_{i=1}^{n} w''_{ijk} s_{jk}} \quad (10)$$

Both algorithms use CPLEX to compute the solution of the relaxed linear problem.

3.4 Hill Climbing

Hill climbing algorithms (Russell and Norvig, 2010; Holte, 2001) typically perform a local search in the solution space by starting off at a random point and moving to a neighboring solution if the cost function of the neighbor is higher. The algorithm stops when it finds a (local) maximum. We define a solution in the search space as an ordering of bids (and asks) onto which a greedy algorithm can be applied, and the cost of a solution as its social welfare. We generate the random initial solution using a greedy algorithm. Based on (Zurel and Nisan, 2001), we came up with several ideas for generating a neighboring solution, which resulted in four algorithms.

In Hill-I, a neighboring solution is generated by moving an unallocated bid or ask to the beginning of the bid/ask list as proposed by (Zurel and Nisan, 2001), starting with the critical bid/ask and then going through the sorted lists. Greedy-II is used for allocation. In Hill-II, the unallocated bid/ask replaces the last allocated bid/ask instead. Greedy-II is used in this case as well. Hill-III uses Greedy-IV for allocation. This means that only the ordering of bids is part of the search space. Hill-III is similar to Hill-I by moving the bid to the top of the bid list. Hill-IV also uses Greedy-IV for allocation and moves the unallocated bid at the end of the list of allocated bids, before the last allocated bid.

3.5 Simulated Annealing

Simulated annealing (Kirkpatrick et al., 1983) is a well-known optimization algorithm that accepts (with some probability) worse solutions, in order to climb out of local optima and reach a global optimum. The acceptance probability depends on a temperature variable which decreases over time, allowing the algorithm to accept worse solutions with a higher probability in the beginning of the search, but to gradually converge as the temperature decreases.

We propose a simulated annealing algorithm with the following features: the initial solution is generated using Greedy-II; generating a neighboring solution is done by choosing a random x_i, toggling it, and then finding a feasible solution using Greedy-II-like allocation; the temperature decreases with a constant rate of $\alpha = 0.9$; the acceptance probability is computed using the formula $ap = e^{(w_{new}-w_{old})/(T|w_{old}|)}$, where w_{old} and w_{new} are the welfare values before and after moving to the neighboring solution. For each temperature, a constant number of iterations is executed; we named this algorithm SA-I. We derived a second algorithm SA-II by having a linear number of iterations for each temperature, proportional to the number of bids ($n/20$). Then in SA-III we use Greedy-IV for generating the initial solutions, as well as for computing the feasible allocations at every step.

3.6 Casanova

(Hoos and Boutilier, 2000) propose a stochastic local search algorithm named Casanova which, similar to SA, uses randomization to escape from local optima. The algorithm starts with an empty allocation and adds bids to reach a neighbor in the search space: with a walk probability wp, a random bid is chosen for allocation (we use a Greedy-IV-like algorithm to select the asks to satisfy the chosen bid); with a probability of $1-wp$, a bid is selected greedily by ranking the bids according to their score. The score is defined as the normalized bid price ($b_i/\sum_{k=1}^{l} r_{ik}$). From the sorted bids, the highest-ranked one is selected if its *novelty* is lower than that of the second highest ranked bid; otherwise, we select the highest ranked bid with a novelty probability np and the second highest one with a probability of $1-np$. We introduced the *novelty* of bids as a counterpart of the *age* defined by (Hoos and Boutilier, 2000), as a measure of how often the bid was selected instead of the number of steps since it was last selected. The search is restarted $maxTries = 10$ times and the best solution from all runs is chosen. As (Hoos and Boutilier, 2000), we set $wp = 0.15$ and $np = 0.5$.

3.7 Genetic Algorithms

The multi-unit multi-good combinatorial problem can be reduced to the multidimensional knapsack problem (Holte, 2001), which is an $\mathcal{NP}$-complete problem (Kellerer et al., 2004). There are several genetic algorithms for solving the multidimensional knapsack problem (Chu and Beasley, 1998; Khuri et al., 1994).

We propose a genetic algorithm where individuals are given by the ordering of bids and asks over which a greedy algorithm can be applied. This is different from other algorithms in the literature where individuals are given by the solution vectors, since our w representation would have increased the gene space considerably, as well as added complexity to ensure the feasibility of each solution. Next, we describe our steady-state (Whitley and Starkweather, 1990) approach, where offsprings replace the least fit individuals. The fitness function is defined as the welfare of an individual. We randomly generate an initial population of $N = 100$ individuals. Two parents are selected using the tournament selection method (Goldberg and Deb, 1991), each from a random pool of size $T = 5$. They produce a child through the crossover operation: a binary random number generator chooses which parent will give the child's next bid. Mutation consists of swapping two random bids and two random asks. We fix the mutation rate to $M = 2$. The process of parent selection, crossover, mutation and fitness evaluation is repeated $t_{max} = 1000$ times. The final solution is given by the fittest individual in the population. We implemented two versions: GA-I uses Greedy-II for allocation, while GA-II uses Greedy-IV.

3.8 Algorithm Properties

Table 1 summarizes the properties of each algorithm in the portfolio. The proofs for time complexities are omitted due to lack of space. However, we note that most of the approximate algorithms run in polynomial time, compensating with speed for the loss of welfare compared to the optimal exponential algorithm.

The algorithms in several classes (simulated annealing, stochastic local search and genetic algorithms) are stochastic: they use randomization to explore the search space, which means that they can compute different results when ran multiple times on the same input.

Another property included in Table 1 is resource locality. It expresses an algorithm's preference to allocate resources in a bidder's bundle on the same cloud provider. It should be noted that this is a soft constraint: it is enforced only when possible and it translates into bidders receiving resources from as few providers as possible, while preserving the monotonicity property. The algorithms that have this property use a Greedy-II-like allocation after optimizing the ordering of bids and asks.

4 INPUT GENERATION

Finding real-world data for auctions of cloud resources is a difficult task, since commercial cloud providers do not typically release user bidding information. Existing cloud auctions for dynamic pricing of resources, such as the Amazon spot instance market (Amazon, 2017), are simplistic and mostly one-sided single-good auctions. Although cloud prices are publicly available, the amount of resources offered and their relative scarcity are not, since providers claim infinite resources. Several cloud providers have released workload traces to be used for cloud scheduling (Wilkes, 2011; Facebook, 2012). These can provide insights into typical jobs running in the cloud, enabling us to extract the sizes of bundles requested by clients, as well as different resource quantities and their dependencies. However, resource types are generally limited to CPU load, memory, storage and possibly bandwidth, making the data too simple to thoroughly test combinatorial auctions.

Therefore, we use artificial data in all our evaluations, as most papers dealing with combinatorial auctions (Leyton-Brown et al., 2000; Sandholm, 2002; De Vries and Vohra, 2003) do. An added benefit of synthetic data generation is flexibility: a larger range of use cases can be covered, rather than having just a few real-world, but narrow-scoped datasets.

4.1 CAGE

Although there has been some work on artificial data generation for combinatorial auctions (Leyton-Brown et al., 2000; Sandholm, 2002; Fujishima et al., 1999; De Vries and Vohra, 2003), to our knowledge there is no work that deals with multi-good **and** multi-unit combinatorial auctions. The Combinatorial Auctions Test Suite (CATS) (Leyton-Brown et al., 2000) is actually the first work that tries to create a comprehensive tool for generating artificial data for combinatorial auctions. It supports legacy distributions (covered in older papers), but the authors also propose more realistic distributions by modeling complementarity and substitutability of goods using graphs. We do not deal with dependencies between goods, but the existing work is not sufficient to generate input data for our resource allocation problem with multi-unit multi-good bids. We also consider the differences between buyers

Table 1: Summary of algorithm properties.

Class	Algorithm	Stochastic	Resource locality	Time complexity
Branch-and-cut	MILP			exponential
Greedy	Greedy-I		✓	$O(n\log n + m\log m + l(n+m))$
	Greedy-II		✓	$O(n\log n + m\log m + l(n+m))$
	Greedy-III			$O(nml)$
	Greedy-IV			$O(nml)$
Linear relaxation	SS		✓	exponential
	RLPS		✓	exponential
Hill climbing	Hill-I		✓	$O(nml(n+m))$
	Hill-II		✓	$O(nml(n+m))$
	Hill-III			$O(n^2ml)$
	Hill-IV			$O(n^2ml)$
Simulated annealing	SA-I	✓	✓	$O(n\log n + m\log m + nl + it\frac{\log T_{min}}{\log \alpha}ml)$
	SA-II	✓	✓	$O(\frac{\log T_{min}}{\log \alpha}nml)$
	SA-III	✓		$O(\frac{\log T_{min}}{\log \alpha}nml)$
Genetic algorithms	GA-I	✓	✓	$O((N+l)(n+m)(N+t_{max}))$
Stochastic local search	Casanova	✓		$O(maxTries(nml+n^2))$

and sellers and hence require different strategies for generating input for each side. Thus, we propose an improved way of generating artificial data for multi-good multi-unit double combinatorial auctions, compatible with legacy distributions (Leyton-Brown et al., 2000). Our tool CAGE (Combinatorial Auctions input GEnerator) is introduced in the following.

A problem instance (defined by the set of bids and asks) can be generated using the following parameters: the number of bids n, the number of asks m, the number of resource types l and multiple random distributions with their parameters. We define the random distributions to generate the bids as follows.

A bundle size is randomly drawn from a given *bundle size distribution*, where the bundle size is the total number of items of any type of resource.

Resource type selection: for each bundle, the resource type of an item is selected with a probability given by a distribution over the number of resource types (e.g. uniform or normal). A special case is the *sparse uniform* distribution, where only a subset of resources are requested by each bidder, with the size of this subset given as input parameter; the resource types in the subset can be different for each bidder. The resource quantities of a bundle are then uniformly distributed among the resource types in the subset.

A bid value proportional to the number of items in each bundle is then generated; first, the *base price distribution* is used to generate a base price for each resource type, representing the known approximate market value of each resource type. These base prices are the same for all trade participants. The *price distribution* is then used to generate an average price per unit for each resource type, around its base price; the total bid value is then computed by combining these prices per unit and the number of items requested of each resource type in one of the following ways: 1) *additive*: the total bid value is computed as a linear combination of the unit prices per resource type, weighed by the number of items requested for each type; 2) *super-additive*: we compute the bid value as a quadratic function (De Vries and Vohra, 2003) proportional to the bid size and parameterized with an *additivity* factor (Eqn 11).

$$b_i = \sum_{k=1}^{l} v_{ik}^* r_{ik} + additivity \sum_{k \neq h} v_{ik}^* r_{ik} v_{ih}^* r_{ih} \qquad (11)$$

In Eqn. 11, by v_{ik}^* we denote a bidder i's reported (not necessarily *true*) valuation for one item of resource type k.

The same strategy is used for generating the asks, with the difference that we generate reserve prices for each resource type instead of a total ask value.

All the available random distributions and their default parameters are summarized in Table 2. We denote the base price of resource k as p_k^*.

Table 2: Distributions and default parameters in CAGE.

Category	Distribution	Default parameters
Bundle size	normal	$\mu = 1000, \sigma = 250$
	uniform	min=500, max=1500
	exponential	$\lambda = 0.005$, offset $20l$
	beta	$\alpha = 5, \beta = 1$, $\times 200l$
	constant	1000
Resource type	normal	$\mu = l/2, \sigma = l/4$
	uniform	$min = 0, max = l-1$
	sparse	2 resource types
Base prices	uniform	$p_k^* : min = 0.1$, $max = 0.9$
	normal	$p_k^* : \mu = 0.5, \sigma = 0.2$
Unit prices	uniform	$v_{ik}^* : min = p_k^* - 0.1$, $max = p_k^* + 0.1$
	normal	$v_{ik}^* : \mu = p_k^*, \sigma = 0.05$

5 EVALUATION

We compared all the algorithms in the portfolio under various input conditions, in order to evaluate their performance and solution quality. To that end, we selected two essential metrics: execution time and social welfare. In most cases, we normalized the welfare by the optimal welfare (computed with MILP). Below we present our evaluation scenarios.

5.1 Average Case

We wish to find the best algorithm in the average case. Therefore, we generated 3226 problem instances using CAGE's default parameters and different combinations of the available distributions. We used both additive bids, as well as three different positive additivity parameters. We fixed the number of bids, asks and resources as follows: $n = m = 200, l = 50$. We did not increase the problem size further in order to be able to include in our comparison the optimal algorithm, which has exponential time complexity. The results are depicted in Fig. 1.

We notice large differences in both welfare and execution time between the algorithms in the portfolio. The algorithms based on the linear relaxation of the integer program (SS and RLPS) are the least performant, since the execution time is longer than most algorithms in the portfolio, while the social welfare is on average less than 25% of the optimal welfare (with high standard deviation).

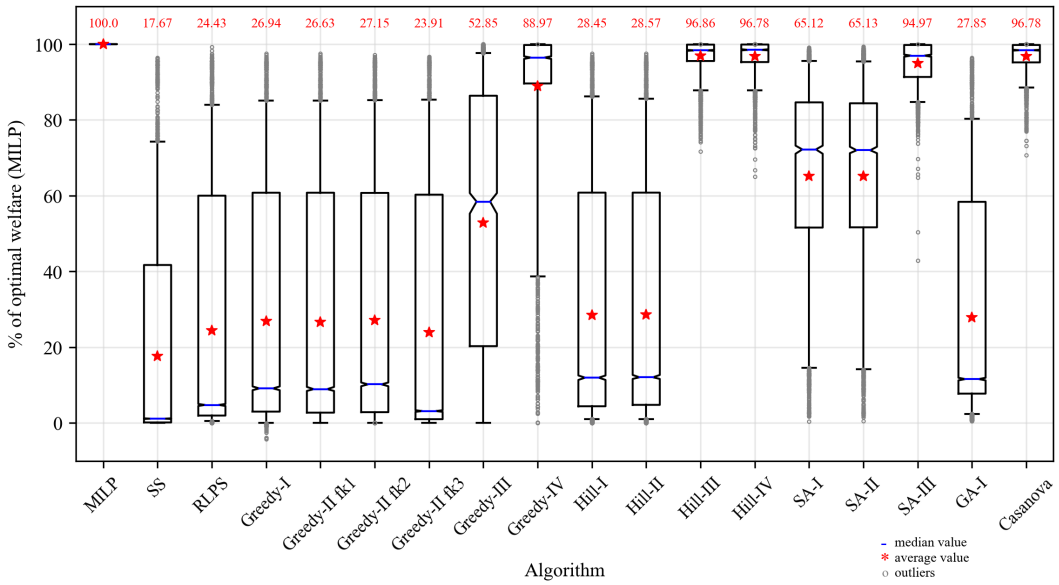
The algorithms with the highest welfare are the ones based on Greedy-IV: Hill-III, Hill-IV, SA-III and Casanova. Although Casanova and SA-III achieve, on average, the same or slightly lower welfare than the hill climbing algorithms III and IV, they are 12 to 15 times faster, on average, and thus equally attractive options. Even the Greedy-IV algorithm computes, on average, a welfare that is almost 90% of the optimal one, which is surprisingly high, as well as extremely fast. We notice, however, a high standard deviation for Greedy-IV due to the fact that ordering the bids by density is a coarse approximation that does not work well in all possible input scenarios. The Greedy-IV-based algorithms achieve near-optimal welfare, which can be explained by the fact that provider goods are treated separately and all constraints of resource locality and bid monotonicity are relaxed. We believe that adding such constraints to our problem formulation, namely having each provider offering a bundle to be sold to a single bidder, would make the problem harder and most likely would make the approximations worse. This interesting hypothesis is, however, the subject of future work, and will require adapting all the algorithms to the new problem formulation.

The Greedy-II-based algorithms roughly enforce the resource locality constraints, in the sense that for each bid, they try as much as possible to allocate all resources from the same provider. If this is a desired feature, it must be noted that the resulting social welfare is low, around 27% of the optimal welfare, for Greedy-II (variations 1,2,3), Hill-I, Hill-II and GA-I. The fact that the hill climbing algorithms do not significantly improve the greedy allocation suggests that they get stuck in a local maximum close to the initial greedy solution. This hypothesis is confirmed by the more than doubling of welfare brought on by the simulated annealing algorithms (SA-I and SA-II), which mitigate this exact problem through randomization.

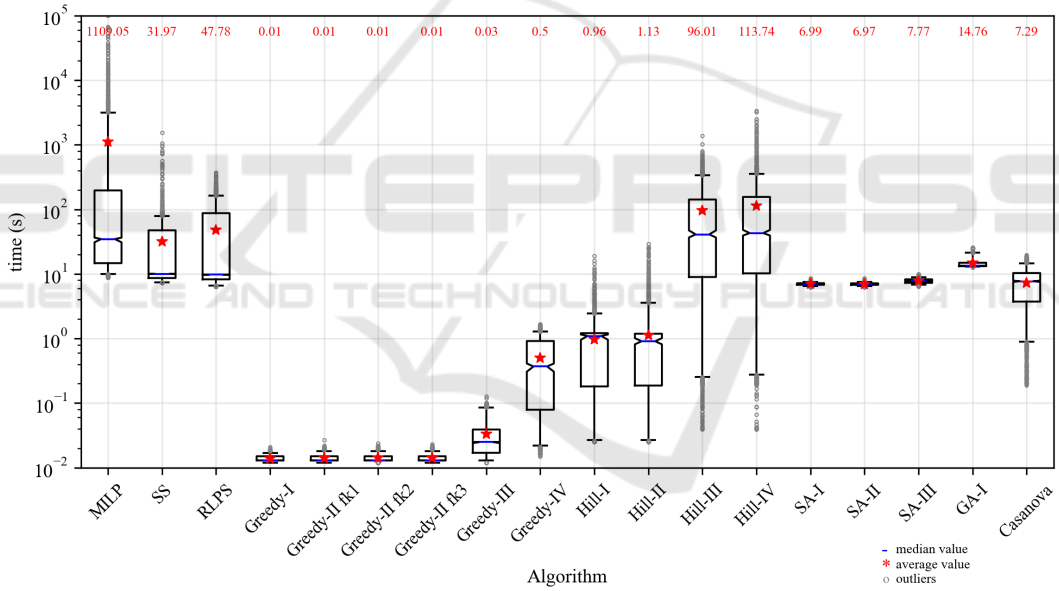
5.2 Best Algorithm

As shown in Fig. 1(a), several algorithms yield near-optimal welfare on average. At the same time, even the algorithms that perform poorly on average (computing a welfare $< 30\%$ of the optimal one) can in a few cases result in high welfare. These cases are depicted by outliers found even above the 90% line.

To obtain a clear picture of which algorithm in the portfolio actually performs best, we analyzed the dataset constructed for our first test case. For each problem instance, we compared the social welfare computed by all the approximate algorithms and selected the algorithm with the highest welfare, and, when multiple algorithms yielded the same welfare, the fastest algorithm. We counted the number of instances for each algorithm and summarized the results in Table 3. We notice that there is no clear winner of the portfolio, with the results being almost equally



(a) Social welfare, normalized by the optimal welfare computed with MILP



(b) Execution time

Figure 1: Results from a comprehensive parameter sweep through the input space: different combinations of available distributions in CAGE with default parameters and 4 different additivity parameters. The average value for each algorithm is represented by a red star, with the actual value attached at the top of each box. The boxes extend from the lower to the upper quartile values of the data, with a blue line at the median. The notches around the median represent the confidence interval around the median. The whiskers reach from 5% to 95% of the data. The remaining data are represented as outliers with gray circles. A logarithmic scale was used for the *y*-axis of Fig. 1(b) for better readability.

Table 3: Breakdown of results by best algorithm: number of instances where each algorithm outperforms all the other algorithms in the portfolio (absolute numbers and percentage of total instances on which the portfolio was ran).

Algorithm	RLPS	Greedy-IV	Hill-III	Hill-IV	SA-I	SA-III	Casanova
Number of instances	6	753	1021	566	2	54	824
Percentage of total instances	0.18%	23.34%	31.64%	17.54%	0.06%	1.67%	25.54%

distributed among a few Greedy-IV-based algorithms: Greedy-IV, Hill-III, Hill-IV and Casanova. Furthermore, in a few cases (1.89%), other algorithms (SA-I, SA-III, RLPS) output the highest social welfare.

This result reinforces the need for consistent benchmarking efforts of approximate algorithms for the WDP. A fair comparison can only be ensured through applying the algorithm portfolio on a unique problem formulation, coupled with a wide range of test cases that can uncover each algorithm's strengths. Our work on standardizing all the existing algorithms on the proposed problem formulation (tailored to cloud resource allocation), as well as the novel input generator for combinatorial auctions CAGE, aims at tackling this issue.

5.3 Effect of Randomization

Since some algorithms in the portfolio are stochastic (Casanova, genetic and simulated annealing), they will yield different results for different runs on the same input. It is desirable to have small variations in welfare between runs. Thus, for each stochastic algorithm we performed 100 runs on the same problem instance, and recorded the social welfare.

Fig. 2 shows the percentual variations for each algorithm with respect to the mean of the 100 runs. We performed this normalization in order to have a comparative overview over all the algorithms. The plot contains events from 100 different instances with 100 runs each. The highest variation is seen in the Greedy-II-based algorithms, most pronounced for the GA-I algorithm, where the lower and upper quartile values of the data are at about -9% and $+8.6\%$ of the mean value. This result reinforces the idea that the solution quality of Greedy-II-based algorithms, where soft locality constraints are enforced, is highly dependent on ordering of bids and asks, and even small changes can lead to very different results. In order to use these algorithms reliably, multiple runs are necessary, and the best solution can be used in the end. This would, however, increase the execution time.

Fig. 3 shows the absolute welfare values from 100 runs on a single instance for Casanova and SA-III, the algorithms based on Greedy-IV. For this particular instance, SA-III computes a higher welfare than Casanova. Although the standard deviation is low in both cases (0.012% and 0.36% from mean), SA-III also produces a lower number of unique solutions, converging to a few solutions in the search space: 9 local maxima for 100 runs. The SA-III algorithm is more likely to converge than Casanova due to the decreasing temperature variable which forces a decrease in the acceptance probability of worse solutions in the search space. The results in Fig. 3 confirm this assertion. However, there is no guarantee that SA-III will always lead to better solutions than Casanova.

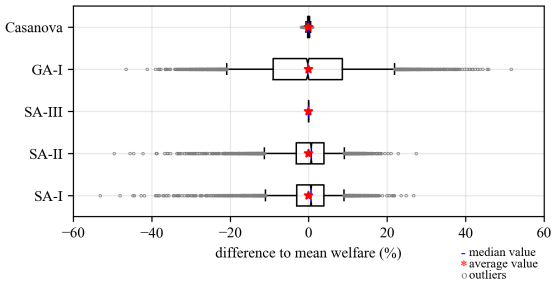


Figure 2: Variation in social welfare for the stochastic algorithms: results from 100 different input instances, with 100 runs per instance. The difference of each run to the average value of the respective instance, normalized by that average value, is plotted. The average value (red star) is always at 0. The boxes extend from the lower to the upper quartile, with a blue line at the median. The whiskers reach from 5% to 95% of the data, and the rest are outliers (gray circles).

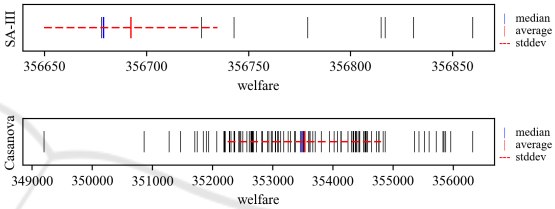


Figure 3: Absolute welfare values computed by two algorithms (SA-III and Casanova) on a single input instance, with 100 repetitions. Average value (vertical red line), median value (vertical blue line) and $\pm$ one standard deviation from the mean (horizontal dashed red line) are also plotted.

6 RELATED WORK

In the previous sections, we referred to existing work on various approximate algorithms for solving the WDP (Nejad et al., 2015; Lehmann et al., 2002; Holte, 2001; Zurel and Nisan, 2001; Bertocchi et al., 1995; Hoos and Boutilier, 2000; Chu and Beasley, 1998; Khuri et al., 1994; Fujishima et al., 1999). While each aforementioned paper compares its newly introduced algorithm to other algorithms in the literature, there is no attempt, to our knowledge, to experimentally compare all the existing algorithms in a more systematic and consistent manner.

(Leyton-Brown et al., 2000) have worked in this direction by proposing CATS, a "universal test suite": a wide range of economically motivated test scenarios consisting of artificially generated data. However, they focus on optimal algorithms and compare their runtime to predict the hardness of a problem instance.

In contrast, we focus on approximate algorithms to accommodate large-scale auctions and use social welfare to compare them. Furthermore, our input generator can deal with multi-unit multi-good double auctions, as opposed to single-unit multi-good one-sided auctions for CATS, but we simplified our approach by not considering dependencies between goods.

7 CONCLUSION

In this paper, we performed a systematic and comprehensive comparison of approximate algorithms for winner determination in double combinatorial auctions. We created an algorithm portfolio and found that only a subset of the algorithms compute near-optimal welfare in the average case. However, our analysis revealed that there is no clear portfolio winner, and the algorithms' performance highly depends on the input. In the future, we will perform a deeper analysis to identify the input characteristics which influence the solution quality and we will employ machine learning methods to predict the algorithms' performance. Moreover, we argue for the need to harmonize benchmarking efforts and propose a flexible approach to generate artificial data. In the future, we will integrate real cloud data in the input generator by using up-to-date prices, as well as analyzing public cloud workloads and extracting relevant parameters to fit them to certain random distributions.

REFERENCES

Amazon (2017). Amazon ec2 pricing. https://aws.amazon.com/ec2/pricing. Accessed: 10. 11. 2017.

Bertocchi, M., Butti, A., Słomiñ ski, L., and Sobczynska, J. (1995). Probabilistic and deterministic local search for solving the binary multiknapsack problem. *Optimization*, 33(2):155–166. doi: 10.1080/02331939508844072.

Buyya, R., Yeo, C. S., and Venugopal, S. (2008). Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. IEEE. doi: 10.1109/HPCC.2008.172.

Chu, P. C. and Beasley, J. E. (1998). A genetic algorithm for the multidimensional knapsack problem. *Journal of heuristics*, 4(1):63–86. doi: 10.1023/A:1009642405419.

CloudStack (2017). Apache cloudstack. https://cloudstack.apache.org/. Accessed: 10. 11. 2017.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). Greedy algorithms. *Introduction to algorithms*, 1:329–355.

De Vries, S. and Vohra, R. V. (2003). Combinatorial auctions: A survey. *INFORMS Journal on computing*, 15(3):284–309. doi: 10.1287/ijoc.15.3.284.16077.

Facebook (2012). Facebook workload repository. https://github.com/SWIMProjectUCB/SWIM/wiki. Accessed: 10. 11. 2017.

Fujishima, Y., Leyton-Brown, K., and Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI*, volume 99, pages 548–553.

Goldberg, D. E. and Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, 1:69–93. doi: 10.1016/b978-0-08-050684-5.50008-2.

Gonen, R. and Lehmann, D. (2000). Optimal solutions for multi-unit combinatorial auctions: Branch and bound heuristics. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 13–20. ACM. doi: 10.1145/352871.352873.

Gudu, D., Hardt, M., and Streit, A. (2016). On mas-based, scalable resource allocation in large-scale, dynamic environments. In *Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016 Intl IEEE Conferences*, pages 567–574. IEEE. doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0097.

Holte, R. C. (2001). Combinatorial auctions, knapsack problems, and hill-climbing search. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 57–66. Springer. doi: 10.1007/3-540-45153-6_6.

Hoos, H. H. and Boutilier, C. (2000). Solving combinatorial auctions using stochastic local search. In *AAAI/IAAI*, pages 22–29.

IBM (2017). Ilog cplex 12.6.3. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud. Accessed: 10. 11. 2017.

Kellerer, H., Pferschy, U., and Pisinger, D. (2004). Introduction to np-completeness of knapsack problems. In *Knapsack problems*, pages 483–493. Springer.

Khuri, S., Bäck, T., and Heitkötter, J. (1994). The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the 1994 ACM symposium on Applied computing*, pages 188–193. ACM. doi: 10.1145/326619.326694.

Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P., et al. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680. doi: 10.1126/science.220.4598.671.

Lehmann, D., Müller, R., and Sandholm, T. (2006). The winner determination problem. *Combinatorial auctions*, pages 297–318. doi: 10.7551/mitpress/9780262033428.003.0013.

Lehmann, D., Oćallaghan, L. I., and Shoham, Y. (2002). Truth revelation in approximately efficient combinato-

rial auctions. *Journal of the ACM (JACM)*, 49(5):577–602. doi: 10.1145/585265.585266.

Leyton-Brown, K., Pearson, M., and Shoham, Y. (2000). Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pages 66–76. ACM. doi: 10.1145/352871.352879.

Luenberger, D. G. and Ye, Y. (2015). *Linear and nonlinear programming*, volume 228. Springer. doi: 10.1007/978-3-319-18842-3.

Myerson, R. B. and Satterthwaite, M. A. (1983). Efficient mechanisms for bilateral trading. *Journal of economic theory*, 29(2):265–281. doi: 10.1016/0022-0531(83)90048-0.

Nejad, M. M., Mashayekhy, L., and Grosu, D. (2015). Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds. *Parallel and Distributed Systems, IEEE Transactions on*, 26(2):594–603. doi: 10.1109/TPDS.2014.2308224.

Nisan, N. et al. (2007a). Introduction to mechanism design (for computer scientists). *Algorithmic game theory*, 9:209–242.

Nisan, N., Roughgarden, T., Tardos, E., and Vazirani, V. V. (2007b). *Algorithmic game theory*, volume 1. Cambridge University Press Cambridge.

Openstack (2017). Openstack. https://www.openstack.org/. Accessed: 10. 11. 2017.

Padberg, M. and Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100. doi: doi.org/10.1137/1033004.

Pfeiffer, J. and Rothlauf, F. (2008). Greedy heuristics and weight-coded eas for multidimensional knapsack problems and multi-unit combinatorial auctions. In *Operations Research Proceedings 2007*, pages 153–158. Springer. doi: 10.1007/978-3-540-77903-2_24.

Rappa, M. A. (2004). The utility business model and the future of computing services. *IBM systems journal*, 43(1):32–42. doi: 10.1147/sj.431.0032.

Russell, S. and Norvig, P. (2010). Beyond classical search. *Artificial Intelligence, A Modern Approach*, pages 125–128.

Samimi, P., Teimouri, Y., and Mukhtar, M. (2014). A combinatorial double auction resource allocation model in cloud computing. *Information Sciences*. doi: 10.1016/j.ins.2014.02.008.

Sandholm, T. (2002). An algorithm for optimal winner determination in combinatorial auctions. *Artificial intelligence*, 135(1):1–54. doi: 10.1016/S0004-3702(01)00159-X.

Schnizler, B., Neumann, D., Veit, D., and Weinhardt, C. (2008). Trading grid services–a multi-attribute combinatorial approach. *European Journal of Operational Research*, 187(3):943–961. doi: 10.1016/j.ejor.2006.05.049.

Whitley, D. and Starkweather, T. (1990). Genitor ii: A distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 2(3):189–214. doi: 10.1080/09528139008953723.

Wilkes, J. (2011). More Google cluster data. Google research blog. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

Zurel, E. and Nisan, N. (2001). An efficient approximate allocation algorithm for combinatorial auctions. In *Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 125–136. ACM. doi: 10.1145/501158.501172.