Combinatorial Auction Algorithm Selection for Cloud Resource Allocation Using Machine Learning

Diana Gudu(✉), Marcus Hardt, and Achim Streit

Karlsruhe Institute of Technology, Karlsruhe, Germany
{diana.gudu,marcus.hardt,achim.streit}@kit.edu

Abstract. Demands for flexibility, efficiency and fine-grained control for the allocation of cloud resources have steered the research in this field towards market-inspired approaches. Combinatorial auctions can fulfill these demands, but their inherent $\mathcal{NP}$-hardness makes them impractical if an optimal solution is desired in a reasonable time. Various heuristic algorithms that yield good allocations fast have been proposed, but their performance and solution quality are highly dependent on the input. In this paper, we investigate which features of a problem instance are predictive of algorithm performance and quality, and propose an algorithm selection method that uses machine learning to find the best heuristic for each given input. We introduce a new cost model for the trade-off between execution time and solution quality, which enables quantitative algorithm comparison. Using feature-based classification to train the algorithm selection model, we can show that our approach outperforms the single best algorithm, as well as a random algorithm selection.

Keywords: Cloud resource allocation · Combinatorial auction
Algorithm selection · Feature-based classification

1 Introduction

Cloud computing leverages economies of scale to provide resources as a utility. Therefore, market-oriented approaches are necessary to regulate the demand and supply of cloud resources, as well as provide economic incentives for both providers and customers [1].

The concept of dynamic pricing is gaining interest, as cloud providers such as Amazon use single-good auctions to sell their unused resources on the spot market [2]. Moreover, dynamic pricing is an essential part of smart contracts [3], an emerging alternative to broker-based matchmaking for cloud service selection, which support changes in agreements while offering quality and security guarantees through their self-executing nature.

Combinatorial auctions [4] can offer more flexibility and fine-grained control, by affording customers to request and pay only for the combination of resources that fits their requirements. However, their applicability to resource allocation has been limited to the realm of academic research [5], due to their $\mathcal{NP}$-hardness. The forecast growth of public cloud services market [6] will further impede their practical use due to scalability concerns – as long as optimal solutions are desired. Therefore, for real-world adoption, it is necessary to sacrifice optimality requirements by using heuristics.

Existing heuristic algorithms for combinatorial auctions [7–9] perform differently depending on the input characteristics, in terms of both runtime and solution quality [10]. A more robust usage is essential, since any performance gain can translate into high increases in revenue for cloud providers. Furthermore, the quality-speed trade-off of each algorithm needs to be reliably controlled, in order to fit any particular needs.

In this paper, we address these challenges by using machine learning to select the most suited heuristic for each individual auction instance, while introducing a quantitative definition for this suitability – based on a runtime and welfare-dependent cost model. Furthermore, we propose a feature set tailored to combinatorial auctions to aid the learning process. We perform an extensive evaluation and show that the proposed approach outperforms single algorithms, as well as a random algorithm selection.

2 Related Work

Various approaches for algorithm selection have been applied to combinatorial search problems, as summarized in [11]. The techniques are categorized according to the type of algorithm portfolio (static or dynamic), features (low or high-knowledge, static or dynamic), performance models, and prediction types. Their applicability is exemplified across a range of application domains: SAT, Mixed Integer Programming, machine learning, etc. However, these methods focus on a single optimization objective, usually runtime.

The only work where algorithm selection was applied to combinatorial auctions [12] is concerned with optimal algorithms and minimizing the execution time, whereas we look at heuristic algorithms and optimize both social welfare and execution time. Leyton-Brown et al. [12] studied the empirical hardness of combinatorial auctions and devised a methodology to understand this hardness using feature-based supervised learning. They identified certain structural features of the WDP that are predictive of running time, and used this runtime prediction to select the fastest algorithm for each problem instance, outperforming the best algorithm in the average case. We note that this work used regression-based learning techniques to predict the runtime of algorithms instead of classification, in order to penalize mispredictions differently.

Beck and Freuder [13] use algorithm selection for scheduling problems. They optimize the performance of a portfolio of optimal algorithms only based on low-knowledge information, obtained by running all the algorithms for a short time, recording their performance, and using this information to inform the prediction.

3 Formal Problem Definition

To model the cloud resource allocation using market-inspired concepts, we make the following assumptions:

1. there are multiple cloud providers offering computing resources and multiple customers requesting resources from any provider at a centralized marketplace,
2. there is a fixed number of resource types on the market, known apriori by all market participants,
3. all resources requested by a customer need to come from the same provider,
4. all requests and offers are independent, and
5. no partial allocations or floating point quantities are allowed.

The problem can then be formalized as a multi-unit, double combinatorial auction, consisting of: a set of n bidders $U = \{1, \ldots, n\}$, a set of m providers $P = \{1, \ldots, m\}$, a set of l goods $G = \{1, \ldots, l\}$, and an auctioneer that decides the allocation and pricing of resources based on the bids and asks.

Each customer i submits a single bid for a bundle of resources, expressed as $(\langle r_{i1}, \ldots, r_{il} \rangle, b_i)$, where r_{ik} is the number of items of resource type k that the bidder i requests, and b_i is the maximum amount bidder i is willing to pay for the entire bundle. Similarly, a seller j submits its ask expressed as $(\langle s_{j1}, \ldots, s_{jl} \rangle, a_j)$, where s_{jk} are the quantities offered by seller j of each resource type k. Seller j offers its bundle of resources at a price a_j, which is the minimum acceptable.

In the context of cloud computing, a bundle represents a virtual machine (VM), consisting of resources such as CPU cores, memory, disk storage, GPU cores, etc. This model makes assumption 3 indispensable, since a VM cannot contain resources from different providers. Furthermore, we showed that the resource locality constraint makes the allocation problem harder in terms of time complexity [10], and thus requires the use of heuristic algorithms. Therefore, the algorithm selection approach is aimed at this specific use case.

The auctioneer collects the bids and asks, and finds the best allocation of resources that maximizes the social welfare of the system. To that end, it first determines which bidders will receive the requested bundles and which providers can sell their resources – also called the Winner Determination Problem (WDP) [14] – and then decides the trading prices – also called the payment scheme.

The social welfare is defined [15] as the sum of all the participants' utilities, where the utility is a measure of a trader's satisfaction. For example, a bidder i's utility for a requested bundle S is defined as $u_i(S) = v_i(S) - p_i$, if i wins the auction, and 0 otherwise, where $v_i(S)$ (valuation) is the true value bidder i is willing to pay for bundle S, and p_i is the actual price paid at the end of the auction. When a bidder is truthful, $v_i(S) = b_i$.

We assume that customers and providers are single-minded, which means that they are only interested in buying or selling the full bundle, and have 0 valuation for all the other bundles.

Then the WDP can be written as the following integer program:

$$\max_{x,y} \left(\sum_{i=1}^{n} b_i x_i - \sum_{j=1}^{m} \sum_{i=1}^{n} a_j y_{ij} \right) \tag{1}$$

subject to:

$$x_i, y_{ij} \in \{0, 1\}, \forall i \in U, \forall j \in P \tag{2}$$

$$\sum_{i=1}^{n} y_{ij} \leq 1, \forall j \in P \tag{3}$$

$$\sum_{j=1}^{m} y_{ij} = x_i, \forall i \in U \tag{4}$$

$$r_{ik} x_i \leq \sum_{j=1}^{m} s_{jk} y_{ij}, \forall i \in U, \forall k \in G \tag{5}$$

where constraint (2) expresses the single-mindedness of bidders and sellers, constraint (3) ensures that a seller can allocate its bundle to at most one bidder, and constraint (4) ensures that each customer receives the resources in its bundle from a single provider. Finally, constraint (5) ensures that a provider cannot sell more than the amount of resources it offered.

The auctioneer then uses a κ-pricing scheme [10] to set the bundle prices by distributing the trade surplus among the auction winners, thus ensuring budget-balance. The truthfulness requirement is relaxed, but it was shown that it can be achieved in practice, since non-truthful bidding increases the risk of no allocation [16].

4 Algorithm Selection

Combinatorial auctions are $\mathcal{NP}$-hard [4], hindering their wide adoption in real-world applications. Existing heuristic algorithms mitigate the scalability and efficiency issues posed by optimal algorithms, but their solution quality varies with the input [10]. For a more robust usage of heuristic algorithms for combinatorial auctions, we propose using an algorithm selection approach [11]: selecting the most suitable algorithm on a case-by-case basis. To predict which heuristic will perform best on each problem instance, we propose the use of supervised machine learning in conjunction with an algorithm portfolio.

The workflow for algorithm selection, based on similar approaches for runtime prediction [12], is depicted in Fig. 1. We first build an algorithm portfolio by assembling a collection of complementary heuristic algorithms for combinatorial auctions. The data are collected in two sub-steps: (1.a) generating a large number of auction instances (defined by a set of bids and asks) that covers a representative part of the input space, by using our artificial input generator for combinatorial auctions CAGE [10]; (1.b) running all the algorithms in the

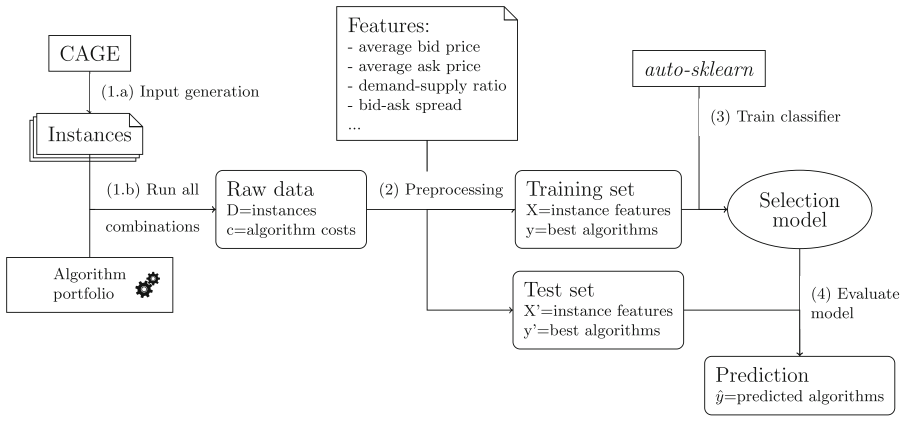

Fig. 1. Algorithm selection workflow

portfolio on all the generated instances to record their runtime and resulting social welfare. Since using the raw input for learning can be computationally expensive or even intractable, we propose to use domain knowledge to extract a set of features that contain sufficient information; the features are mainly statistics related to bid and ask values, quantities or demand-supply balance (see Sect. 4.2). The preprocessing step (2) includes feature extraction, labeling the data by selecting the best algorithm for each instance – the algorithm with the lowest cost, as defined in Sect. 4.3 – and splitting the dataset into training and test data for supervised learning. We formulate the algorithm selection problem as a multi-class classification problem: given observations (a set of instances defined by their features) whose class labels (best algorithm) are known, we (3) train a model that can predict the class label of any new observation. The model is then (4) tested on unseen data. At this step, several appropriate metrics to evaluate the quality of the prediction should be considered (see Sect. 4.4).

4.1 Algorithm Portfolio

In [10], we investigated and compared different algorithms for approximating the solutions of WDPs. We built an algorithm portfolio by either adapting various combinatorial auction algorithms [7–9], or applying well-known optimization methods [17,18] to combinatorial auctions. The experiments revealed that algorithm runtime and solution quality are highly dependent on the input, and no single algorithm outperformed the others in all test cases. This result was most pronounced when resource locality was a desired property – all resources requested by a customer being allocated on the same cloud provider. The problem formulation presented in this paper already includes the resource locality constraint, and the algorithms were adapted accordingly.

In the rest of this section, we briefly describe the 12 algorithms included in our portfolio. Based on the employed optimization approach, we can group the algorithms into four families: greedy, hill climbing, simulated annealing, and stochastic local search.

The greedy algorithms sort the bid and ask lists according to a certain criteria (e.g. bid density), and then traverse the list to greedily match bids with asks. Based on [7], we used three different sorting criteria to implement algorithms GREEDY1, GREEDY2 and GREEDY3. A greedy algorithm that gives priority to sellers was also implemented (we denote seller priority by a '-S' suffix: GREEDY1S).

Hill climbing algorithms perform a local search in the solution space, similar to gradient descent. We included two methods of exploring the neighborhood of a solution: first, by changing the ordering of bids or ask onto which a greedy allocation is performed [8] (algorithms HILL1 and HILL1S), and second, by toggling the allocation of a bid through the x_i variables [19] (HILL2 and HILL2S).

Simulated annealing algorithms (SA and SAS) use the same method of generating a neighboring solution as HILL2, but randomly accept worse solutions to escape from local optima.

Finally, to mitigate the same problem, stochastic local search algorithms (CASANOVA and CASANOVAS) use random walks with restarts, while exploring the solution neighborhood by adding bids based on their ranking and novelty [9].

The algorithms based on simulated annealing and stochastic local search techniques are stochastic, yielding different results for multiple runs on the same input. For reliable usage, the average welfare and execution time over 10 runs were used in our experiments.

The portfolio is easily extensible, and more algorithms can be added as they are developed. However, this affects the rest of the algorithm selection pipeline: the prediction models need to be retrained for every portfolio change.

4.2 Features

Using domain knowledge – insights into the inner workings of combinatorial auctions, as well as each individual algorithm – we defined a number of 75 features that can be extracted from any problem instance. The features are mainly statistics, and can be computed in $\mathcal{O}\left(l\left(m+n\right)\right)$, which is faster than any of the algorithms in the portfolio. The defined features can be grouped in four categories: price related, quantity related, quantity per resource related (as measures of heterogeneity of requests) and demand-supply balance related.

We give some examples of features in the following. First, statistics of the distribution of the asking price per unit over all asks were included (mean, standard deviation, skewness and kurtosis). Similarly, we looked at the distribution of the bid price per unit over all bids, as well as the corresponding quantity related features: the total bundle sizes of bids and asks. Moreover, we included economics concepts such as the bid-ask spread, defined as the difference between the minimum ask and maximum bid, and used as a measure of the market liquidity. Similarly, we defined a quantity spread per resource, as the difference between the maximum requested quantity and the minimum offered quantity per resource, and computed the first four central moments of the distribution of quantity spread over all the l resource types. Other features in the group of demand-supply balance related features deal with quantity surpluses, either total surplus (the difference between the total number of resources offered and requested), or a quantity surplus per resource type.

4.3 Cost Model

Since the algorithms in the portfolio are heuristic, they generally trade solution quality for speed. Thus, labeling the dataset requires, for each problem instance, a comparison of all algorithms with respect to both social welfare and execution time, which can then yield the best algorithm for both criteria. We propose modeling this as a multi-objective optimization problem [20], whose objectives are a maximum social welfare and a minimum execution time. In order to find a Pareto optimal solution, we use the idea of a compromise solution [21], which minimizes the distance between the potential optimal point and a utopia (or ideal) point.

As welfare and time are measured on different scales, they should first be normalized to obtain non-dimensional objective functions. We normalize the welfare objective function, and call it welfare cost $c_{\mathrm{w}}(o, a)$, as defined in Eq. 6, where $w(o, a)$ is the welfare computed by algorithm a on instance o, while $w_{\min}(o)$ and $w_{\max}(o)$ are, respectively, the minimum and maximum welfare obtained for instance o by any algorithm in the portfolio. Thus, the best algorithm when only welfare objective is considered will have zero welfare cost.

$$c_{\mathrm{w}}(o, a) = \frac{w_{\max}(o) - w(o, a)}{w_{\max}(o) - w_{\min}(o)} \tag{6}$$

Similarly, in Eq. 7 we define the time cost $c_{\mathrm{t}}(o, a)$ as the normalized time objective, where $t(o, a)$ is the execution time of algorithm a on instance o, and $t_{\min}(o)$ and $t_{\max}(o)$ are the execution times of the fastest and slowest algorithms in the portfolio on instance o. The best algorithm with respect to time will also have zero time cost.

$$c_{\mathrm{t}}(o, a) = \frac{t(o, a) - t_{\min}(o)}{t_{\max}(o) - t_{\min}(o)} \tag{7}$$

Then the multi-objective function is defined as a vector in the two-dimensional objective space, $C = \left[c_w \; c_t\right]^\top$. Furthermore, we introduce a user-defined preference parameter $\lambda \in [0, 1]$ that reflects the relative importance of the two objectives, in order to provide more control over the decision of selecting the best algorithm. This changes the multi-objective vector to $C_\lambda = \left[\lambda \; (1 - \lambda)\right] C$. A value of $\lambda = 1$ implies that solely the welfare objective should be considered, while $\lambda = 0.5$ places equal importance on welfare and time.

Finally, we find the optimal solution (best algorithm) by minimizing the distance to the utopian vector C°, whose components are the lower bounds of each objective function – in this case $\left[0 \; 0\right]^\top$. We use the Euclidean distance to compute the scalar cost metric that will ultimately be used to select the best algorithm, as defined in Eq. 8.

$$c_\lambda(o,a) = \|C_\lambda - C^\circ\| = \sqrt{(\lambda c_\mathrm{w}(o,a))^2 + ((1-\lambda)\, c_\mathrm{t}(o,a))^2} \tag{8}$$

In Fig. 2, we exemplify the use of λ on a random problem instance. For different λ values, different algorithms have minimum cost and are thus selected as the best: when speed is more important ($\lambda = 0.1$), the CASANOVAS algorithm is selected, while an algorithm based on hill climbing (HILL2S) is best when welfare has a higher priority ($\lambda = 0.9$). A simulated annealing algorithm (SAS) is the best when time and welfare are weighted equally ($\lambda = 0.5$).

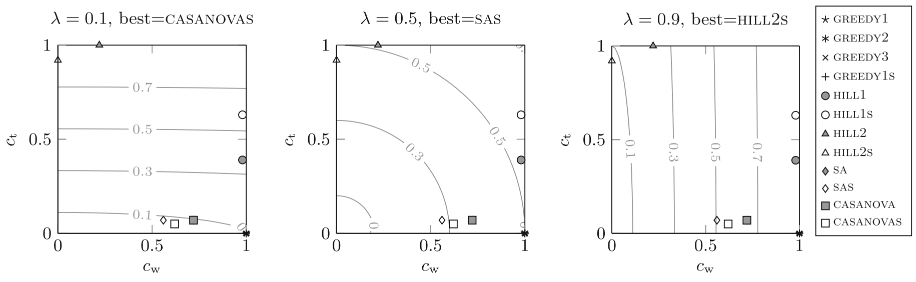

Fig. 2. Visualization of a problem instance in the two-dimensional objective space. Isolines represent scalar cost c_λ. Different algorithms emerge as best depending on λ.

4.4 Evaluation Metrics

There are several success measures when evaluating a classification model. The most intuitive measure is the accuracy, namely how often the model correctly predicts the algorithm with the lowest cost. More specifically, we define the accuracy in Eq. 9, for a given dataset O, as the fraction of the instances for which the predicted algorithm $\hat{y}_o$ is the same as the algorithm with the lowest cost y_o.

$$\mathrm{accuracy}_\lambda(y,\hat{y}) = \frac{1}{|O|}\sum_{o\in O} 1(\hat{y}_o = y_o) \tag{9}$$

However, the accuracy does not give a quantitative evaluation of a model's mispredictions: it penalizes all misclassifications equally, irrespective of their associated costs. To that end, we introduce a metric that considers the cost of the predicted algorithm: the mean relative error (MRE), as defined in Eq. 10.

$$MRE_\lambda(y,\hat{y}) = \frac{1}{|O|}\sum_{o\in O} (c_\lambda(o,\hat{y}_o) - c_\lambda(o,y_o))^2 \tag{10}$$

For a meaningful evaluation, we also compare our portfolio-based algorithm selection against a single algorithm a^*. Therefore, we introduce the relative mean relative error (RMRE) metric, defined in Eq. 11 as the ratio between the MRE of the classification model and the MRE of using algorithm a^* on the entire dataset. The classification model can be similarly compared to a random selection model.

$$RMRE_\lambda(y, \hat{y}, a^*) = \frac{MRE_\lambda(y, \hat{y})}{MRE_\lambda(y, a^*)} \tag{11}$$

5 Evaluation

We evaluated our machine learning-based algorithm selection on an artificially generated dataset, as real data for combinatorial auctions of cloud resources (e.g. user bidding data) is not available. We used CAGE [10], a flexible input generator designed specifically for multi-unit, multi-good double combinatorial auctions.

We created a dataset of 5970 auction instances by varying input parameters such as the number of bids, asks and resource types, sparsity of resources inside a bundle, additivity, and distributions used for generating base prices. Given that we model cloud resources, we assume that bundle sizes for both bids and asks are drawn from exponential random distributions, which means that most of the bundles are small. This is in accordance with Google cloud traces [22], where most task are short, while only a few tasks are long running with high resource demands. Regarding bidding strategies, we assume a normal distribution around base prices, meaning that bidders are willing to pay, per unit, a price close to a resource's known market price.

5.1 Dataset Analysis

We analyze the dataset by evaluating the relevance of the defined features to the prediction, as well as the distribution of class labels. The dataset was labeled by selecting, for each problem instance, the algorithm that yielded the lowest cost. Since the cost is λ-dependent, so are the labels.

Figure 3 shows the support for each class, over 11 values of λ equidistantly distributed over $[0, 1]$. Note that the dataset is imbalanced for all λ. Furthermore, for small λ values, when time is more important than welfare, greedy algorithms were selected more frequently, as they are fast, but have poor quality, while at the other end hill climbing algorithms, although slower, were selected for their higher welfare. For $\lambda \in [0.1, 0.6]$, simulated annealing algorithms were often selected as best – not surprising, since they are similar to hill climbing, but randomly accept worse solutions to climb out of local optima and reach to a solution faster. An interesting result is the λ-independent number of instances for which the greedy algorithms are selected as best (e.g. 47 instances for GREEDY1). These are the infeasible instances, or the auctions where no match exists and the social welfare is 0—thus the fastest algorithm is always selected as best.

Figure 3 hence demonstrates the input-dependent performance of heuristic algorithms, and the potential for improvement by using algorithm selection.

Next, we investigated which features are more relevant to the prediction. The aim is to identify irrelevant or redundant features, and remove them to

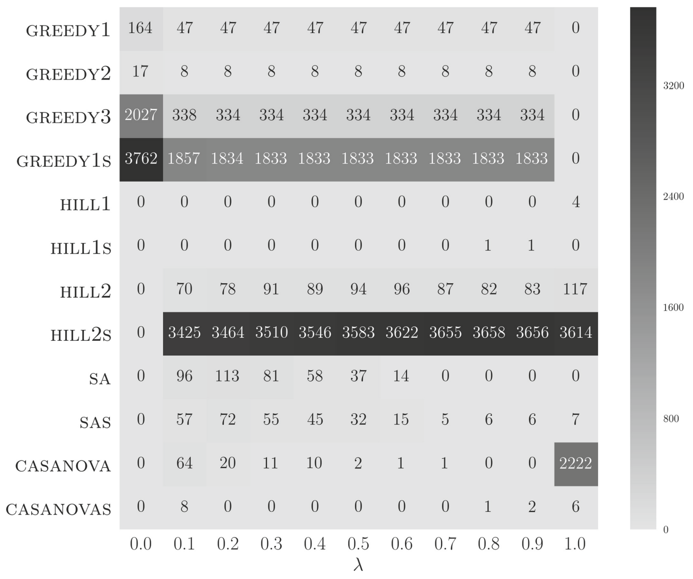

Fig. 3. Algorithm selection dataset: breakdown by class labels for several λ values.

reduce the dimensionality of the input space and prevent over-fitting. We used tree-based estimators to compute relative feature importances to the model's performance.

In Fig. 4, all 75 features are sorted based on their importance and the first 20 are shown. Note that only a few are relevant, e.g. 16 features have an importance over 0.02. The most relevant features are related to the quantities per resource, demanded or supplied on the market – minimum, maximum, and average values – as well as the mean and standard deviation of bundle sizes. This can be explained by the fact that quantities per resource are instrumental in assessing the feasibility of a solution, as enforced by constraint (5), and influence the way algorithms move in the search space. From the price-related features, only the minimum and standard deviation of the asking price per unit have a certain effect on the prediction.

5.2 Classification Evaluation

The dataset was split into a training set (70%) and a test set (30%), used to test how the model generalizes on unseen data. Because of the imbalanced dataset, the splitting was performed using stratified sampling, to ensure that the train and test sets have the same percentage of samples of each class as the full set.

Using *auto-sklearn* [23], we trained a classification model for each λ value. The *auto-sklearn* library implements an automated machine learning approach, which relies on Bayesian optimization methods to construct an ensemble of classifiers and find their best hyperparameters and preprocessing steps. The preprocessing,

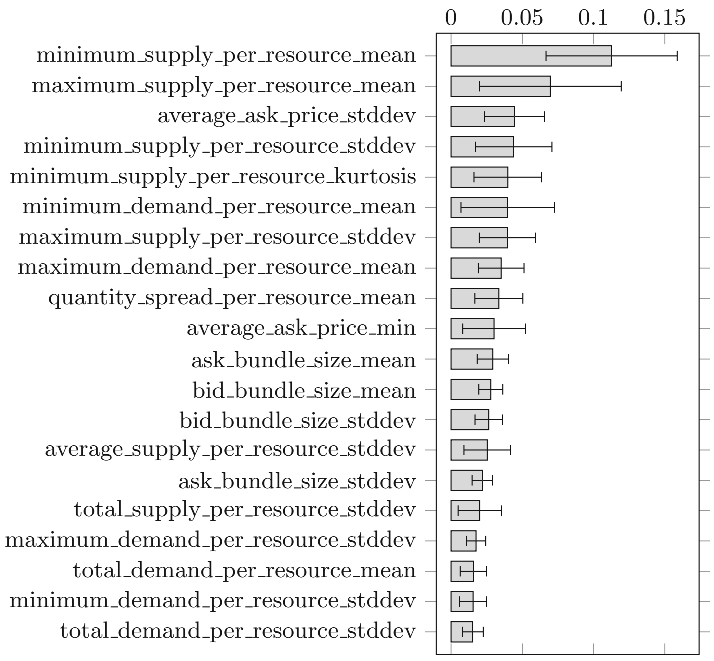

Fig. 4. Relative feature importances averaged over all λ values, computed using ExtraTreesClassifier in *scikit-learn* with 500 estimators. Only the first 20 most relevant features are shown.

in this case, includes feature scaling and feature selection for dimensionality reduction, based on their relevance as described in Sect. 5.1.

Figure 5 shows the accuracy of the models for each λ, on both training and test set. Good accuracies over 93% are obtained for most λ preferences, with higher accuracy for higher λ, suggesting that the selected features are more relevant to the welfare objective rather than the time objective.

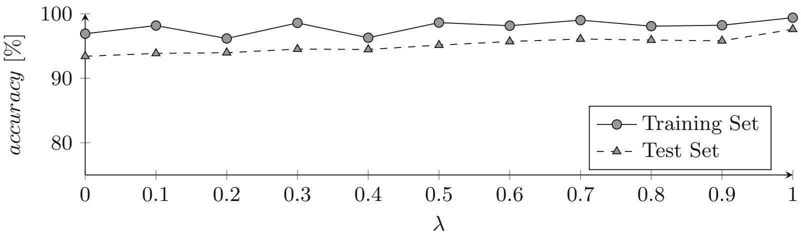

Fig. 5. Accuracy of ML-based algorithm selection for different λ values.

More importantly, a comparison between the trained models for each λ and random selection (see Fig. 6) shows that our algorithm selection approach is always 2 to 4 orders of magnitude better than a random selection approach. Note that an RMRE value below 1 implies that the algorithm selection using machine learning is better than its counterpart in the comparison.

Similarly, a comparison between our models and the best pure algorithm a^* for each λ, where a^* is defined as the algorithm selected most often as the best in the labeling phase (cf. Fig. 3), showed that our approach outperforms the best pure algorithm for all values of λ except 0.7 and 0.8 (see Fig. 6), with overall lower RMRE for smaller λ. The best pure algorithm method can also be seen as a rule-based system that uses domain knowledge to select an algorithm per λ value, e.g. when speed is the most important, greedy algorithms are always used.

Therefore, our machine learning approach yields higher welfare, but also higher cost error MRE with increasing λ, leading to worse performance than a single algorithm when only the welfare obective is considered. This can be explained by the fact that the algorithms' performances vary more in the welfare dimension than the runtime, but classification penalizes all mispredictions eaqually, ultimately leading to the paradox of 96% accuracy with $RMRE \geq 1$.

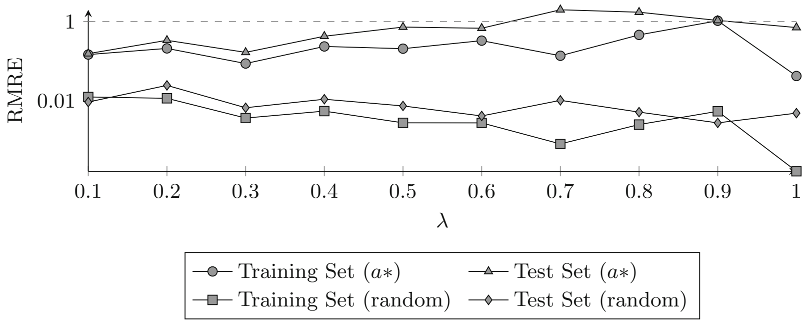

Fig. 6. RMRE comparison of ML-based algorithm selection to random selection and best pure algorithm for different λ values.

6 Conclusions

In this paper, we proposed an algorithm selection approach to improve the performance and solution quality of combinatorial auctions, applied to the problem of cloud resource allocation. We introduced a machine learning approach that selects the best heuristic algorithm for each problem instance, where the *best algorithm* is defined by our proposed multi-objective cost model. Another contribution of this paper is a feature set to aid in the learning process, engineered using domain knowledge. We showed that our proposed approach predicts the best algorithm per instance with an accuracy of up to 99%. This approach also outperforms a random algorithm selection approach, as well as the best pure algorithm, in most cases.

To further improve the prediction, in the future we will integrate low-knowledge, dynamic features, that can be obtained by running all the algorithms on a small sample of the problem instance.

References

1. Buyya, R., Yeo, C.S., Venugopal, S.: Market-oriented cloud computing: vision, hype, and reality for delivering IT services as computing utilities. In: 10th IEEE International Conference on High Performance Computing and Communications, 2008. HPCC 2008, pp. 5–13. IEEE (2008). https://doi.org/10.1109/HPCC.2008.172
2. Amazon: Amazon EC2 spot instaces (2017). https://aws.amazon.com/ec2/spot/
3. Scoca, V., Uriarte, R.B., De Nicola, R.: Smart contract negotiation in cloud computing. In: 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), pp. 592–599. IEEE (2017). https://doi.org/10.1109/CLOUD.2017.81
4. De Vries, S., Vohra, R.V.: Combinatorial auctions: a survey. INFORMS J. Comput. **15**(3), 284–309 (2003). https://doi.org/10.1287/ijoc.15.3.284.16077
5. Zaman, S., Grosu, D.: Combinatorial auction-based allocation of virtual machine instances in clouds. J. Parallel Distrib. Comput. **73**(4), 495–508 (2013). https://doi.org/10.1016/j.jpdc.2012.12.006
6. Smith, D.M.: Predicts 2017: cloud computing enters its second decade. Gartner Special report (2017)
7. Nejad, M.M., Mashayekhy, L., Grosu, D.: Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds. IEEE Trans. Parallel Distrib. Syst. **26**(2), 594–603 (2015). https://doi.org/10.1109/TPDS.2014.2308224
8. Zurel, E., Nisan, N.: An efficient approximate allocation algorithm for combinatorial auctions. In: Proceedings of the 3rd ACM conference on Electronic Commerce, pp. 125–136. ACM (2001). https://doi.org/10.1145/501158.501172
9. Hoos, H.H., Boutilier, C.: Solving combinatorial auctions using stochastic local search. In: AAAI/IAAI, pp. 22–29 (2000)
10. Gudu, D., Zachmann, G., Hardt, M., Streit, A.: Approximate algorithms for double combinatorial auctions for resource allocation in clouds: an empirical comparison. In: Proceedings of the 10th International Conference on Agents and Artificial Intelligence, ICAART, pp. 58–69 (2018). https://doi.org/10.5220/0006593900580069
11. Kotthoff, L.: Algorithm selection for combinatorial search problems: a survey. In: Bessiere, C., De Raedt, L., Kotthoff, L., Nijssen, S., O'Sullivan, B., Pedreschi, D. (eds.) Data Mining and Constraint Programming. LNCS (LNAI), vol. 10101, pp. 149–190. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50137-6_7
12. Leyton-Brown, K., Nudelman, E., Shoham, Y.: Empirical hardness models: methodology and a case study on combinatorial auctions. J. ACM (JACM) **56**(4), 22 (2009). https://doi.org/10.1145/1538902.1538906
13. Beck, J.C., Freuder, E.C.: Simple rules for low-knowledge algorithm selection. In: Régin, J.-C., Rueher, M. (eds.) CPAIOR 2004. LNCS, vol. 3011, pp. 50–64. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24664-0_4
14. Lehmann, D., Müller, R., Sandholm, T.: The winner determination problem. In: Combinatorial Auctions, pp. 297–318 (2006). https://doi.org/10.7551/mitpress/9780262033428.003.0013
15. Shoham, Y., Leyton-Brown, K.: Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press, Cambridge (2008). https://doi.org/10.1145/1753171.1753181

16. Schnizler, B., Neumann, D., Veit, D., Weinhardt, C.: Trading grid services-a multi-attribute combinatorial approach. Eur. J. Oper. Res. **187**(3), 943–961 (2008). https://doi.org/10.1016/j.ejor.2006.05.049
17. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., et al.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983). https://doi.org/10.1126/science.220.4598.671
18. Russell, S., Norvig, P.: Beyond classical search. In: Artificial Intelligence: A Modern Approach, pp. 125–128 (2010)
19. Bertocchi, M., Butti, A., Słomiñ ski, L., Sobczynska, J.: Probabilistic and deterministic local search for solving the binary multiknapsack problem. Optimization **33**(2), 155–166 (1995). https://doi.org/10.1080/02331939508844072
20. Deb, K.: Multi-objective optimization. In: Burke, E., Kendall, G. (eds.) Search Methodologies, pp. 403–449. Springer, Boston (2014). https://doi.org/10.1007/978-1-4614-6940-7_15
21. Marler, R.T., Arora, J.S.: Survey of multi-objective optimization methods for engineering. Struct. Multidiscip. Optim. **26**(6), 369–395 (2004). https://doi.org/10.1007/s00158-003-0368-6
22. Mishra, A.K., Hellerstein, J.L., Cirne, W., Das, C.R.: Towards characterizing cloud backend workloads: insights from Google compute clusters. ACM SIGMETRICS Perform. Eval. Rev. **37**(4), 34–41 (2010)
23. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Advances in Neural Information Processing Systems, pp. 2962–2970. Curran Associates, Inc. (2015)