Algorithm Selection in Auction-based Allocation of Cloud Computing Resources

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Diana Gudu
aus Slobozia, Rumänien

Tag der mündlichen Prüfung:	17. Juli 2019
Erster Gutachter:	Prof. Dr. Achim Streit
Zweiter Gutachter:	Prof. Dr. Dorothea Wagner

Erklärung zur selbständigen Anfertigung der Dissertationsschrift

Hiermit erkläre ich, dass ich die Dissertationsschrift mit dem Titel

Algorithm Selection in Auction-based Allocation of Cloud Computing Resources

selbständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittelbe-nutzt sowie die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemach-tund die Regeln zur Sicherung guter wissenschaftlicher Praxis am Karlsruher Institut für Technologie (KIT) beachtet habe.

Ort, Datum Diana Gudu

To my family.

Abstract

The hallmarks of Cloud Computing—pay-as-you-go, on-demand provisioning, and well-defined service level agreements—have fostered a strong relation between its usage and the financial aspects, making it the ideal candidate for a market-based resource allocation. Market-based approaches can increase provider revenue and resource utilization by providing appropriate economic incentives to cloud providers and customers.

This thesis addresses limitations of current cloud resource allocation approaches: on the one hand, in dealing with fluctuating demand and supply, and on the other hand, in providing greater control to cloud customers. The goal is to ultimately enable a more flexible and efficient allocation through market-based mechanisms. As a result, I propose and motivate a model for the cloud resource allocation problem as a double combinatorial auction. The principal reason why a combinatorial auction-based approach has not yet been used in practice is its intractability. This thesis is focused on making combinatorial auctions practicable, in the context of cloud resource allocation. This is based on the assumption that sacrificing optimality in favor of speed is acceptable, but it needs to be bounded, which opens up the possibility of using approximate, heuristic algorithms.

The key contributions of this thesis are two meta-heuristic approaches, that enable the selection of the most appropriate heuristic algorithm for any given input.

These algorithm selection approaches are founded on a portfolio of heuristic algorithms. The portfolio is created by adapting existing approaches, as well as applying generic optimization methods, to the proposed problem model. However, a systematic and comprehensive comparison of existing work was lacking. I perform such a comparative study in this thesis, harmonized through a common problem formulation and test data. A pivotal finding is the fact that no algorithm outperforms the others in all the test cases.

To assist in the evaluation, a new approach for input data generation is proposed, which enables the creation of realistic data for cloud auctions, while considering the double combinatorial aspect. This was necessary, as there is no cloud-specific auction data available, and existing work on auction data generation is not applicable to the modeled auction flavor.

The two algorithm selection approaches distinguish themselves from related work through the fact that they target heuristic algorithms rather than optimal ones, and implicitly consider two typically conflicting objectives when selecting the best algorithm—fast execution and high solution quality. A cost model is proposed, which quantifies this described trade-off, and facilitates algorithm comparison in the multi-objective space. The first algorithm selection approach uses domain knowledge and supervised learning to create a prediction model of the best algorithm, while the second approach uses algorithm properties and probes the search space to create per-algorithm performance models—which are ultimately used to decide which algorithm is most suitable. The two approaches complement each other, and each has its own merits. While the second approach is domain-independent and thus more general, the first approach has a faster selection (since the training is done beforehand) and thus more suitable when algorithm speed is essential.

Finally, an extensive evaluation shows that both approaches can predict the best algorithm fairly accurately. More importantly, the evaluation reveals that they perform better than any individual algorithm, thus demonstrating the value of the proposed approaches, and of algorithm selection in general.

Zusammenfassung

Die Merkmale von Cloud Computing—Pay-as-you-go, On-Demand-Provisioning und klar definierte Service Level Agreements—haben einen starken Zusammenhang zwischen der Nutzung und den finanziellen Aspekten geschaffen und machen es zum idealen Kandidaten für eine marktbasierte Ressourcenallokation. Marktbasierte Ansätze können den Umsatz und die Ressourcenausnutzung von Anbietern steigern, indem sie geeignete wirtschaftliche Anreize für Cloud-Anbieter und Kunden schaffen.

Diese Arbeit befasst sich mit den Grenzen der aktuellen Ansätze der Cloud-Ressourcenallokation: mit dem Umgang schwankender Nachfrage und Angebot auf der einen Seite, und mit der Bereitstellung einer besseren Kontrolle für Cloud-Kunden auf der anderen Seite. Ziel ist es, letztlich eine flexiblere und effizientere Zuordnung durch marktbasierte Mechanismen zu ermöglichen. Als Ergebnis schlage ich ein Modell für das Problem der Cloud-Ressourcenallokation als doppelte kombinatorische Auktion vor. Der Hauptgrund, warum ein kombinatorischer auktionsbasierter Ansatz in der Praxis noch nicht angewendet wurde, ist seine Komplexität. Diese Arbeit konzentriert sich darauf, kombinatorische Auktionen im Rahmen der Cloud-Ressourcenallokation praktikabel zu machen. Dies basiert auf der Annahme, dass es akzeptabel ist, die Optimalität zugunsten der Geschwindigkeit zu opfern, wodurch es möglich wird, approximative, heuristische Algorithmen einzusetzen.

Die wichtigsten Beiträge dieser Arbeit sind zwei meta-heuristische Ansätze, die es ermöglichen, den am besten geeigneten heuristischen Algorithmus für jeden Input zu wählen.

Diese Algorithmus-Auswahlverfahren basieren auf einem Portfolio von heuristischen Algorithmen. Das Portfolio besteht aus angepassen bestehenden Ansätze sowie generischer Optimierungsmethoden die auf das vorgeschlagene Problemmodell angewendet werden. Da ein systematischer und umfassender Vergleich der vorhandenen Arbeiten jedoch fehlte, führe ich eine solche vergleichende Studie in dieser Arbeit, erstellte durch eine gemeinsame Problemformulierung und gleichen Testdaten. Eine entscheidende Erkenntnis ist die Tatsache, dass kein Algorithmus die anderen in allen Testfällen übertrifft.

Zur Unterstützung der Bewertung wird ein neuer Ansatz für die Generierung von Input-Daten vorgeschlagen, der die Erstellung realistischer Daten für Cloud-Auktionen unter Berücksichtigung des doppelten kombinatorischen Aspekts ermöglicht. Dies war notwendig, da keine cloudspezifischen Auktionsdaten verfügbar sind und bestehende Arbeiten zur Generierung von Auktionsdaten für die modellierten Auktionsvarianten nicht anwendbar sind.

Die beiden Algorithmus-Auswahlverfahren unterscheiden sich von verwandten Arbeiten dadurch, dass sie auf heuristische statt auf optimale Algorithmen abzielen und bei der Auswahl des besten Algorithmus zwei typischerweise gegensätzliche Ziele berücksichtigen—

schnelle Ausführung und hohe Lösungsqualität. Es wird ein Kostenmodell vorgeschlagen, das diesen beschriebenen Kompromiss quantifiziert und den Algorithmenvergleich im mehrkriteriellen Raum erleichtert. Der erste Algorithmus-Auswahlansatz verwendet Domänenwissen und überwachtes Lernen, um ein Vorhersagemodell des besten Algorithmus zu erstellen, während der zweite Ansatz Algorithmeneigenschaften verwendet und den Suchraum untersucht, um Leistungsmodelle pro Algorithmus abzuleiten, die letztlich verwendet werden, um zu entscheiden, welcher Algorithmus am besten geeignet ist. Die beiden Ansätze ergänzen sich gegenseitig und haben jeweils ihre eigenen Vorzüge. Während der zweite Ansatz domänenunabhängig und damit allgemeiner ist, hat der erste Ansatz eine schnellere Auswahl (da das Training vorher durchgeführt wird) und ist somit besser geeignet, wenn die Geschwindigkeit des Algorithmus entscheidend ist.

Schließlich zeigt eine umfangreiche Auswertung, dass beide Ansätze den besten Algorithmus in den meißten Fällen vorhersagen können. Noch wichtiger ist, dass sie besser abschneiden als jeder Algorithmus einzeln abschneiden würde und so den Wert der vorgeschlagenen Ansätze, als auch der Algorithmusauswahl im Allgemeinen, demonstrieren.

Acknowledgments

This thesis would not have been possible without the help of a number of people, to whom I would like to give my sincere thanks.

First of all, I would like to express my deep gratitude to my supervisor Prof. Achim Streit, for giving me the opportunity to pursue my PhD under his guidance, for his support and valuable feedback throughout this entire process, and above all for providing an environment where I learned about all the facets of research. I would like to sincerely thank Prof. Dorothea Wagner for accepting to be my co-supervisor, and for providing me with constructive feedback that contributed to significant improvements in my work.

I am particularly thankful to Marcus Hardt for all the time, patience, and energy he dedicated to supervising my work, from inception to the final write-up. His enthusiasm for discussing any new ideas, his always on-point questions, his patience for reading and correcting all my writings, or for going through the abundance of plots I tend to create, his continuous encouragements, all made my work better, and gave me confidence in my abilities as a researcher. I am also thankful for the team feeling he managed to create, which contributed to a great working atmosphere.

A very special thank you goes to Peter Krauß, whose contribution to successfully finishing this work cannot be overstated. From the ideas we came up with over countless cups of coffee, embedded all over this thesis, to the intense programming sessions on the joint project, he made the PhD pursuit less lonely. I am especially grateful for his invaluable infrastructure support and moral support during the thesis writing, which helped me stay sane and successfully reach the finish line.

I owe many thanks to Yoshiyuki Sakai, whose love for science and incredible ability to look at the big picture helped discover and shape the topic of this thesis early on, through countless stimulating discussions. I also am grateful for his support in carefully correcting many of my writings, and patiently listening to my conference presentations.

I am also thankful to all my current and former colleagues, for countless enjoyments, lively lunch break discussions, which created a warm and inspiring atmosphere. I would also like to extend my thanks to Gabriel, for his hard work in implementing the initial algorithm portfolio.

Last, but not least, I would like to thank my family, Stelian, Marioara, Tania, Andrei, for their unconditional love and support throughout this long journey, ever since I decided to move to Germany and follow my passion for science. I would not be here without them. Haristo!

Diana Gudu
Karlsruhe, May 2019

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Cloud Computing's defining features, such as the pay-as-you-go model, on-demand provisioning, and well-defined services, continue to attract more and more businesses seeking to take advantage of the low-costs, elasticity, availability, and flexibility that the cloud offers. This has led to a rapid growth of the cloud market, as more cloud infrastructures are being built and deployed to meet these rising demands.

By now, Cloud Computing has become a mature technology, ubiquitous in today's technological landscape. Nevertheless, there are still a number of issues that have not been sufficiently addressed. I focus on the allocation of cloud resources, and discuss the identified issues in the following.

Problem 1

Current approaches for cloud resource allocation are not specifically designed to seamlessly regulate demand and supply, and deal with variable load.

Cloud Computing promises to offer elasticity and infinite scalability to customers. Behind the scenes, cloud providers need to ensure that they have enough spare resources to accommodate peak load times (for example, during Black Friday sales at Amazon or any other online store). This also means that, during normal or low load times, providers have idle resources that nonetheless consume energy and cooling. Some providers try to monetize these resources by selling them at lower prices, with some limitations in terms of lifetime and service quality guarantees. The most prominent example is the Amazon EC2 Spot market (Amazon, 2017).

Although this is a step into the right direction, there is no resource allocation in production today that is entirely based on market concepts. This would regulate demand and supply *by design*.

Problem 2

Currently used pricing models are not flexible enough.

The fixed-price models currently used in Cloud Computing are inefficient when resource values change dynamically, as in the case of fluctuating customer demands for resources. Admittedly, some commercial cloud providers complement their fixed-price models with dynamic pricing schemes (e.g., the Amazon Spot instances mentioned above), but these prices are not market-driven (Agmon Ben-Yehuda et al., 2013): even though single-good auctions are being run to designate who will receive the resources, the prices are randomly drawn from a tight interval via a hidden dynamic reserve price.

A truly market-driven pricing would provide appropriate economic incentives to both cloud providers and customers, while increasing revenue and resource utilization.

Problem 3

Cloud customers lack fine-grained control, aggravated by the burden of choice and lack of interoperability.

Customers lack control in the sense of being able to fully customize the resources they buy, rather than choosing from a list of predefined options offered by the providers. Furthermore, customers have the burden of finding the provider that matches their needs—this problem is addressed by brokering solutions (Grozev & Buyya, 2014)—, augmented by the fact that there is no standardization for cloud interfaces and interoperability. The cloud market is clearly imbalanced in terms of control, biased towards cloud providers.

A marketplace where providers and customers can meet and trade resources would effectively address this issue, ensuring that market forces shape the interactions and provide proper incentives for both sides to participate.

Essentially, all these problems can be addressed by employing certain market-inspired approaches. I will show in Chapter 3 that double combinatorial auctions fulfill all these demands. The auction aspect ensures market-driven allocation and pricing, the combinatorial aspect ensures fine-grained control, while the double aspect restores the power balance. However, double combinatorial auctions are complex problems, accompanied by their own set of challenges.

Problem 4

Combinatorial auctions are impracticable due to their intractability.

Combinatorial auctions are a thoroughly studied topic. However, their applicability to cloud resource allocation has, so far, been limited to the realm of academic research, due to their intractability. They are $\mathcal{NP}$-hard problems (Sandholm, 2002), and the time for finding an optimal solution scales exponentially with the problem size.

The ever-growing cloud market requires a scalable approach for resource allocation. For the adoption of combinatorial auctions in practice, it is necessary to sacrifice the optimality requirements in favor of speed, by using heuristic algorithms.

Problem 5

Heuristic algorithms for combinatorial auctions yield highly input-dependent results.

A wide spectrum of algorithms can be employed for approximating the solution of a combinatorial auction. However, due to their heuristic nature, these algorithms perform differently depending on their input data. Depending on the techniques used, certain algorithms might perform better on a specific class of problems, and worse on others.

1.1 RESEARCH QUESTIONS

Given these challenges, a number of research questions were identified and studied. The main question underlying this thesis is:

How can market-based mechanisms be used to make cloud resource allocation more flexible, scalable, and efficient?

This question can be broken down into 4 different research questions, in an attempt to address all the issues above.

Research Question 1

How can the resource allocation problem be modeled as a combinatorial auction?

To answer this question, a thorough analysis is needed, in order to derive the essential requirements of the resource allocation. Cloud resources and customer preferences need to be abstracted and modeled. Next, it should be investigated which economic properties can be satisfied and what assumptions are necessary for designing an appropriate auction mechanism.

Finally, based on these analyses, rules for allocation and pricing can be designed. This research question addresses the first three problems discussed above.

Research Question 2

Which auction mechanisms exist and how can they be applied to the modeled problem?

An extensive literature review of heuristic algorithms is required to identify existing approaches that are applicable to the modeled problem. Existing approaches need to be adapted, or new methods need to be developed.

Research Question 3

How can heuristic algorithms for combinatorial auctions be evaluated and compared, and what input data is suitable for a fair comparison?

Having a set of heuristic algorithms that perform differently depending on the input data produces the need to consistently compare and evaluate these algorithms. Given that they typically trade solution quality for execution speed, it is necessary to first model this trade-off. Furthermore, evaluating the heuristic algorithms is challenging due to the lack of empirical data for cloud auctions.

Research Question 4

To what extent can automated algorithm selection yield better results than a single auction mechanism?

Algorithm selection is a meta-algorithmic technique where the best algorithm is chosen from a portfolio on a case-by-case basis. Applying algorithm selection to heuristic algorithms for combinatorial auctions can improve the auction outcome. Since these improvements can be directly translated to significant revenue gains, any improvement in solution quality is paramount. It is necessary to study the extent of such improvements, as well as the features in the input space that could be predictive of algorithm performance. Further investigations are necessary into how machine learning techniques or dynamic information can aid in the selection of the best algorithm.

1.2 SCIENTIFIC CONTRIBUTIONS

Overall, the contributions of this thesis focus on making cloud resource allocation more flexible, scalable and efficient by employing market-based mechanisms. To tackle the computational challenges of combinatorial auctions, I propose two approaches for algorithm selection for combinatorial auctions: a high-knowledge, machine learning-based approach, and a low-knowledge, probing-based approach. These methods are supported by an underlying model for the cloud resource allocation as an auction problem, as well as a multi-objective cost model to evaluate auction mechanisms. Each contribution is discussed in more detail in the following.

Contribution 1

An extensible model for cloud resource allocation as a combinatorial auction.

Following an analysis of the cloud computing landscape, a number of requirements were derived for the resource allocation and pricing, such as scalability, market-driven pricing, or tractability. To meet these requirements, the cloud resource allocation problem is modeled as a double combinatorial auction, consisting of an allocation mechanism and a pricing scheme. The proposed model enables customers to define their resource requests in a fine-grained manner, represent resource complementarity relations, and ultimately only pay for the amount of resources they actually need, while cloud providers can sell otherwise idle resources to other customers. The dynamic pricing scheme minimizes the economic loss that occurs with fluctuating demand and supply. The model is generic and can be easily extended to support additional requirements. This contribution was published in (Gudu et al., 2016) and (Gudu et al., 2018a), and it addresses Research Question 1.

Contribution 2

A unified comparative study of heuristic algorithms for combinatorial auctions.

The use of combinatorial auctions in practice is hindered by their computational complexity. To fulfill the scalability requirements, fast, approximate algorithms must be employed, but they incur a certain monetary loss that needs to be bounded. Although there is a plethora of heuristic algorithms for combinatorial auctions in the literature, their problem formulations are not consistent, and neither are the test scenarios and application domains, which makes systematic comparison practically not possible. A systematic and comprehensive comparison was highly desirable. One contribution of this thesis consists of a unified benchmarking approach for heuristic algorithms for combinatorial auctions, under the umbrella of a consistent problem formulation and a variety of common test cases. By adapting existing algorithms or applying well-known optimization methods to my proposed model, an algorithm portfolio was created. Furthermore, an extensive empirical evaluation over a wide range of test scenarios was performed. This contribution addresses Research Question 2 and the first part of Research Question 3, and was published in (Gudu et al., 2018b) and (Gudu et al., 2019).

Contribution 3

A flexible model for generating input data for multi-good, multi-unit combinatorial auctions.

Since commercial cloud providers do not typically release information on customer requests, there is a lack of real-world data that can be used in scientific research of auctions of cloud resources. Synthetic input was thus required, but existing work on artificial data generation for combinatorial auctions was not suitable for the model studied in this thesis. Therefore, a novel auction data generator was proposed. The flexible, easily extensible tool considers the two-sided aspect of the auction to generate multi-good, multi-unit bids and asks. An approach which is based on public datasets of cloud traces was also proposed, to generate realistic resource bundles. Artificial data generation has a number of benefits: flexibility, wider coverage of the input space, as well as more control over input parameters, making it an integral part of the benchmarking efforts. This contribution also assists in answering Research Question 3.

Contribution 4

A multi-objective cost model to quantitatively evaluate and compare approximate algorithms for combinatorial auctions.

As heuristic algorithms sacrifice solution quality in favor of execution speed, it was necessary to quantify this trade-off for a quantitative comparison of the algorithms over the two-dimensional space. Therefore, a cost model that expresses the trade-off between execution time and solution quality was introduced, which relies on scalarization of a multi-objective optimization problem. Increased control over the relative importance of solution quality to time is afforded through user-defined weights. This contribution was published in (Gudu et al., 2018a), and is also associated with Research Question 3.

Contribution 5

A high-knowledge, machine learning-based algorithm selection method for combinatorial auctions.

The comprehensive evaluation of various heuristics for combinatorial auctions revealed that the results are highly dependent on the input. Thus, there is no single algorithm that outperforms all other algorithms on all input data. In order to minimize costs, the most suitable heuristic can be automatically selected on an instance-by-instance basis. A central contribution of this thesis is an approach for algorithm selection for combinatorial auctions, which employs supervised learning techniques to train a classification model that can predict, based on individual instance features, which algorithm in the portfolio is the most suitable for the given problem. The proposed multi-objective cost model is used to measure the suitability of an algorithm. A range of features of the problem domain were investigated in order to find out which features are predictive of algorithm performance and solution quality. The training was performed using automatic machine learning, which optimizes the hyper-parameters of the classification model. The machine learning approach resulted in high accuracy over a comprehensive dataset, which was artificially generated using the proposed input generator. Moreover, even when the cost of the mispredictions was taken into account to compare against a perfect prediction, this approach was better than the best single auction mechanism, as well as better than a random selection. This contribution was published in (Gudu et al., 2018a), and addresses Research Question 4.

Contribution 6

A low-knowledge, probing-based algorithm selection method for combinatorial auctions.

An alternative method for algorithm selection was proposed, that requires less domain knowledge and is dynamic. Rather than relying on domain-specific features, this approach uses a probing technique to run each algorithm on a sample of the problem. It then extrapolates the result to predict each algorithm's behavior on the full instance. Insights into the algorithms' time complexity and scaling of social welfare over problem size were considered in the extrapolation, to account for differences in algorithm behavior over problem size. In contrast to the machine learning-based approach, no training is necessary, making this method more robust against changes in the algorithm portfolio. On the other hand, the probing phase introduces a runtime overhead that has to be taken into account. An evaluation of the sample size was performed, in order to balance the prediction quality against the runtime overhead of the probing phase. Extensive evaluation showed that this method improves upon the single best mechanism approach in most cases. This contribution also addresses Research Question 4.

1.3 LIST OF PUBLICATIONS

Most of the contributions in this thesis (Contributions 1, 2, 4, 5) have already been published in peer-reviewed conference proceedings and journals. The publications relevant to the content presented in this thesis are listed in the following.

- **Diana Gudu**. *MAS-based, Scalable Allocation of Resources in Large-scale, Dynamic Environments.* In Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS), Doctoral Consortium, pages 1530–1531. IFAAMAS, 2016. http://dl.acm.org/citation.cfm?id=2936924.2937240.

- **Diana Gudu**, Marcus Hardt, and Achim Streit. *On MAS-based, Scalable Resource Allocation in Large-scale, Dynamic Environments.* 2016 International IEEE Conference on Scalable Computing and Communications (ScalCom), pages 567-574. Toulouse, 2016. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0097

- **Diana Gudu**, Gabriel Zachmann, Marcus Hardt, and Achim Streit. *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), pages 58–69. INSTICC, SciTePress, 2018. doi:10.5220/0006593900580069.

- **Diana Gudu**, Marcus Hardt, and Achim Streit. *Combinatorial Auction Algorithm Selection for Cloud Resource Allocation Using Machine Learning.* European Conference on Parallel Processing (Euro-Par), pages 378–391. Lecture Notes in Computer Science, vol 11014. Springer, Cham, 2018. doi:10.1007/978-3-319-96983-1_27

- **Diana Gudu**, Marcus Hardt, and Achim Streit. *A Unified Comparative Study of Heuristic Algorithms for Double Combinatorial Auctions: Locality-constrained Resource Allocation Problems.* In Agents and Artificial Intelligence, pages 3–22. Lecture Notes in Computer Science, vol 11352. Springer, Cham, 2018. doi:10.1007/978-3-030-05453-3_1

1.4 THESIS OUTLINE

The remainder of this thesis is organized as follows.

In Chapter 2, the background relevant to the research in this thesis is presented. I start by introducing the fundamentals of Cloud Computing, outline current trends, and lay out the challenges related to cloud resource allocation. Given the trend towards market-based allocation, the underlying market concepts are introduced. The market structure and interactions between market components are described. Next, I highlight the steps and challenges of designing mechanisms that guide market behavior. A family of competitive mechanisms (auctions) is then discussed. Finally, the algorithm selection problem is formalized, and existing approaches are presented. The chapter ends with related work on algorithm selection for combinatorial auctions, and their shortcomings.

Chapter 3 presents the requirements analysis that motivates the use of double combinatorial auctions for cloud resource allocation. The proposed formalization of the allocation problem as an auction is introduced.

In Chapter 4, I investigate how existing approaches can be applied to solve the allocation problem of the previously introduced combinatorial auction. The algorithms are grouped in an algorithm portfolio and described in detail. Next, an approach for generating artificial data for double combinatorial auctions is introduced, based on real cloud workloads. Finally, the algorithm portfolio is evaluated under a comprehensive range of test cases, using the proposed input data generator.

In Chapter 5, I propose a machine learning-based approach for algorithm selection, applied to heuristic algorithms for combinatorial auctions. A novel multi-objective cost model is presented, used to assess the best algorithm with respect to time and solution quality. I examine which features of the input data are predictive of algorithm performance, and evaluate the accuracy of the approach, as well as the improvement it brings compared to using a single heuristic algorithm.

Chapter 6 presents the second proposed approach for algorithm selection, based on probing information. I describe how the probing is performed, and detail the algorithm properties fundamental to the prediction quality. Finally, after a sample size study, the approach is evaluated on the same dataset as the machine learning-based approach, and the results are discussed.

Chapter 7 summarizes the work in this thesis, emphasizing my contributions. I conclude with an overview of open questions and sketch out directions for future research.

Theory is the essence of facts. Without theory scientific knowledge would be only worthy of the madhouse.

Oliver Heaviside

2 Background

In this chapter, the necessary background to understand the research described in this thesis is provided. Section 2.1 gives a brief overview of cloud computing. It introduces some basic concepts and terminology, and proceeds with an incursion in the evolution of cloud computing, focusing on the emerging related challenges for cloud resource allocation, in the larger context of an unfolding paradigm shift towards market-oriented cloud computing. Section 2.2 then defines fundamental market concepts, delves into mechanism design, and presents a taxonomy of auctions. Finally, Section 2.3 explains the algorithm selection problem, by detailing the basic model and classifying existing approaches. The chapter concludes with related work in algorithm selection for combinatorial auctions.

2.1 EVOLUTION OF CLOUD COMPUTING

Clouds are large-scale distributed systems that enable on-demand access over the network to virtualized resources. This section does not aim to describe Cloud Computing in all its complexity, but rather provide the necessary concepts that will allow us to focus on the challenges that the cloud is currently facing, particularly regarding resource allocation.

2.1.1 CLOUD FUNDAMENTALS

Since Cloud Computing is constantly evolving, there is no consensus on an all-encompassing definition of Cloud Computing. Therefore, the early attempt by the National Institute of Standards and Technology (NIST) to define Cloud Computing (Mell & Grance, 2011) is provided below, touching on points still relevant today:

> *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*

In the midst of this continuous transformation, a number of invariant characteristics remain. According to NIST (Mell & Grance, 2011), Cloud Computing provides *on-demand self-service* (consumers can automatically and unilaterally provision resources) over the network to *pools of resources*, which are shared using a multi-tenant model. Moreover, resources appear to be infinite and can be *elastically* provisioned and released to cope with different loads. Finally, the *metering capability*—a concept leveraged from utility computing (Parkhill, 1966)—allows for transparent monitoring, control, and optimization of resource usage. The benefits of the cloud model for customers are indisputable (Armbrust et al., 2009): reduced IT costs, no up-front payments, and the ability to pay per use of resources as needed—the so-called *pay-as-you-go* model.

The main enabling technologies for Cloud Computing are virtualization (Marshall, 2007) and web services (Endrei et al., 2004). Virtualization abstracts out the resources of a machine by partitioning them into multiple execution environments. Irrespective of the techniques used (binary translation, paravirtualization, hardware-assist) or the resources virtualized (CPU, memory, device), virtualization abstracts the underlying hardware, provides performance and security isolation, and facilitates resource customization. Web services are self-contained, self-describing, modular applications, accessible over a network through standard Internet protocols (Endrei et al., 2004). Advantages of using web services include: reduced complexity by encapsulation, reusability, interoperability, accessibility, and abstraction.

As such, clouds deliver everything as a service, ensuring service quality through formal contracts named Service Level Agreements (SLA) (Wu & Buyya, 2012). The original NIST classification (Mell & Grance, 2011) only identified three service delivery models, depending on the capabilities provided: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and

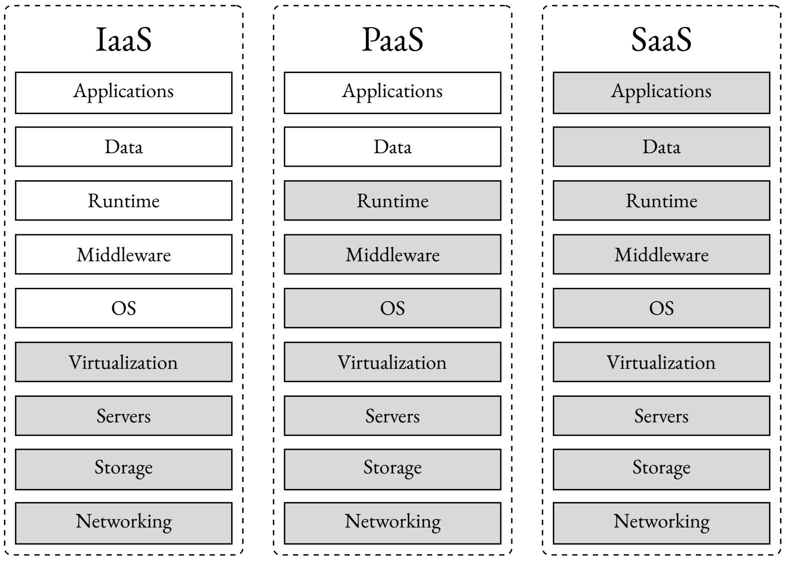


Figure 2.1: Key differences between the three main cloud service delivery models[1]: IaaS, PaaS, SaaS. The components managed by the cloud provider in each model are highlighted through the gray boxes, while the customer capabilities are depicted by white boxes.

Software as a Service (SaaS). Figure 2.1 depicts the key difference between these models, focusing on which components of the hardware and software stack are managed by the cloud provider. Although the examples in this thesis will mostly refer to IaaS—where fundamental resources are offered in the form of virtual machine (VM) instances, storage, etc.—, my work is generic enough to be applied to other types of cloud services.

In fact, this categorization is now largely extended, as the ever-changing cloud landscape accommodates concepts like Blockchain as a Service[2,3] (BaaS), Function as a Service[4] (FaaS or serverless computing (Baldini et al., 2017)), or Machine Learning as a Service[5,6] (MLaaS).

2.1.2 CLOUD COMPUTING TRENDS

As the cloud enters its second decade, it has established itself as a key technology that is here to stay, as depicted by the mainstream adoption of certain cloud delivery models (IaaS,

[1]https://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/
[2]https://aws.amazon.com/blockchain/
[3]https://azure.microsoft.com/en-us/solutions/blockchain/
[4]https://aws.amazon.com/de/lambda/
[5]https://aws.amazon.com/machine-learning/
[6]https://cloud.google.com/ml-engine/

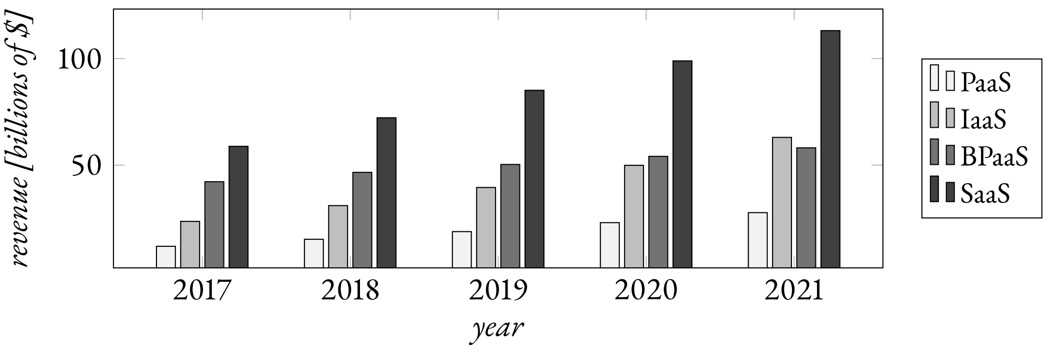


Figure 2.2: Revenue forecast for public cloud services worldwide, according to Gartner (Graham et al., 2018). *BPaaS* is the term used to denote Cloud Business Process Services.

SaaS) (Smith & Anderson, 2018). This is also apparent from the steady yearly growth in revenue, which is forecast to continue at a rapid pace for several cloud market segments. Gartner (Graham et al., 2018) forecast a 17.3% growth for the worldwide public cloud services market in 2019, with the growth of individual market segments portrayed in Figure 2.2. The fastest growing segment is IaaS, fueled by demands for integrated IaaS and PaaS offerings. On the other hand, SaaS is the largest segment of the cloud market, which will continue to bring in more revenue by providing enterprise content management services.

Among the trends (Khnaser et al., 2018) and the hype cycle analysis (Smith & Anderson, 2018) envisioned by Gartner for Cloud Computing in 2019, the terms *multicloud* and *hybrid IT* were identified as increasingly important.

More and more IT organizations are adopting a multicloud strategy, that is, they are using different services from several public cloud providers. The initial drivers for multicloud were redundancy and avoiding vendor lock-in. Although still relevant, multicloud is nowadays driven by business and technical concerns, i.e., price and performance. Besides the obvious advantages, a multicloud strategy also poses some challenges: a wide reaching expertise is required from IT staff, across multiple providers and types of services; the lack of standardization among the various providers makes workload and application management difficult.

Gartner (Khnaser et al., 2018) advises IT organizations to prepare for multicloud adoption by establishing their own governance and policies over the cloud services consumed. Therefore, there is a trend towards developing tooling that can achieve this in an automated manner, with focus on federated identity, access, and cost management. This new area is termed Cloud Management, and brokering is an important component. This paves the way towards hybrid IT, which Gartner (Khnaser et al., 2018) defines as the process of IT departments becoming *"brokers of IT-based services, blending traditional services, public cloud services and private cloud services"*.

In a general sense, a cloud service broker is a *"cloud service partner that negotiates relationships between cloud service customers and cloud service providers"* (Commission et al., 2014). A broker covers a wide range of activities, with open issues identified (Guzek et al., 2015) in the areas of: price, security, and trust optimization, as well as multi-objective matching of customer requirements. In the following I will address challenges of cloud brokerage from the perspective of resource allocation.

2.1.3 CLOUD RESOURCE ALLOCATION

Resource allocation is concerned with *"the distribution of resources economically among competing groups of people or programs"* (Manvi & Shyam, 2014). Brokering of cloud resource allocation is a matter of matching customer requests for cloud services with offers of cloud providers in a mutually optimal way.

The majority of cloud brokering research (Grozev & Buyya, 2014) have taken an SLA-based approach: customers specify their requirements in an SLA, which are then matched automatically and transparently by the broker. However, this leaves customers with no direct control over the provisioning of their application. Moreover, the requirements that can be specified are limited: pricing, data location, legislation / policy constraints. More complex brokering approaches (Jrad, 2014, Zhang et al., 2012) also consider non-functional Quality of Service (QoS) requirements. Outside the academic realm, cloud service brokers[7,8] are simpler in terms of requirements optimized, but manage to take into account particularities of real-world scenarios in a more effective and reliable way (Guzek et al., 2015).

Nevertheless, these brokering solutions are one-sided, since they assist single cloud users in their service selection, independent of other users and thus oblivious to the cloud market dynamics.

2.1.4 TOWARDS MARKET-INSPIRED ALLOCATION

Early on, Buyya et al. (2008) envisioned a future of market-oriented Cloud Computing, where economic incentives and demand and supply regulate the terms of cloud resource

[7]https://www.ibm.com/us-en/marketplace/cloud-brokerage-solutions
[8]https://www.appdirect.com/

management. A few years later, Ben-Yehuda et al. (2014) reinforced this economic model of Cloud Computing and termed it Resource as a Service (RaaS). They identified three trends in the operation and use of IaaS (shorter rental periods, finer grained resources, and differentiated SLAs), as well as economic forces acting on cloud providers (competition-driven need for efficiency and dynamic pricing) and customers (individual economic incentives drive client needs for better cost control and for computerized agents to negotiate on their behalf). The outlined trends and economic forces were predicted to ultimately lead to the rise of RaaS, where fine-grained resources would be continuously traded at market-driven prices. Although the cloud landscape has since changed in ways that could not have been predicted five years ago, the opportunities and challenges engendered by RaaS are eminently relevant today. This is supported by myriad research works, as well as by a shift of the industry sector in this direction. I present a few relevant examples below.

Amazon sells unused VMs from their IaaS compute service—Elastic Compute Cloud (EC2)—as spot instances[9]. The dynamic price resulted from running provider-side single-good auctions enables customers to save up to 90% of the on-demand price if they are willing to forego any lifetime guarantees. Although Amazon has not released any details of the market mechanism employed, it is the only IaaS provider so far that publicly provides an auction-like mechanism, in this way using demand and supply to regulate prices. However, this is somewhat controversial, since Agmon Ben-Yehuda et al. (2013) reverse engineered Amazon's pricing scheme and showed evidence that it is not market-driven, but rather uses a dynamic hidden reserve price.

Google (Stokely et al., 2009) proposed and tested internally a market-based solution to provision resources across clusters, based on an ascending clock combinatorial exchange mechanism, but they have not implemented it in any of their cloud products yet.

Nejad et al. (2015a) addressed the problem of bidding on bundles of different VM instance types through combinatorial auctions. The authors proposed several mechanisms that consider heterogeneity and scarcity of resources. They approached the problem from the perspective of mechanism design, by investigating which incentives can be provided to enforce truthful bidding, but in exchange compute sub-optimal solutions.

Mashayekhy et al. (2014) consider the two-sided scenario for trading of computing resources for big data, i.e., when multiple cloud customers and providers wish to trade with each other, and design a novel truthful auction mechanism. This work does not deal with bundling of resources, but it does take into account customer preferences for the time slot

[9] `https://aws.amazon.com/ec2/spot/`

when a job should be run.

Toosi et al. (2016a) implemented an open-source alternative to the Amazon spot instance market. Their framework, termed *Spot instance pricing as a Service* (SipaaS), implements dynamic pricing for IaaS markets, and can be easily integrated with existing cloud platforms. The authors also provide an extension for integration with OpenStack[10]. The employed market mechanism (Toosi et al., 2016b) is a multi-unit auction, truthful with a high probability, that leads to near-optimal provider revenue. Similar to Amazon's spot instance pricing, they use a dynamic reserve price.

Watzl (2014) developed a framework for trading cloud resources on an exchange. This research spans multiple topics, including comparability of cloud services through quantification, classification, and quality and performance rating. The vendor-neutral marketplace came into operation in 2015 as the Deutsche Börse Cloud Exchange AG[11], and was unfortunately discontinued in 2016. Hermann (2016) speculated that despite the ambitious goals and pertinent ideas, the cloud market was not prepared for it, while end users did not yet understand the added value of such a platform.

In summary, there is sustained interest and need for market-based cloud resource allocation, but a number of open issues impede its practical implementation. For example, there are issues of standardization across providers, partly addressed by Watzl (2014). In this thesis, however, I deal with issues related to the market mechanisms, since practicable yet fine-grained and flexible mechanisms are still missing. Although fine-grained mechanisms (combinatorial auctions) are well-studied in academia, in the real world simplicity and speed have won against flexibility—as exemplified by the Amazon spot market. Nevertheless, if providers could reliably increase their revenue, they would be willing to adopt more complex mechanisms.

2.2 FUNDAMENTALS OF MARKETS

In order to incorporate market-based concepts in the design of cloud resource allocation mechanisms, a good understanding of markets and mechanism design is required.

This section aims to provide a comprehensive overview of market concepts. Section 2.2.1 introduces the market terminology used throughout the thesis, as well as the microeconomic system framework that formalizes, at a high level, the market structure. Section 2.2.2 assumes

[10]https://www.openstack.org/

[11]http://cloud.exchange/

the mechanism designer's perspective, and discusses how the implemented rules affect market behavior. Finally, Section 2.2.3 describes a type of competitive market mechanisms—auctions—with a focus on double combinatorial auctions.

2.2.1 MICROECONOMIC SYSTEM FRAMEWORK

An economic system, to the extent that it determines the allocation of resources, can be seen as a machine (Reiter, 1977), which takes as input an economy's basic data and outputs an allocation of commodities among the participants in the economy. The first attempt to create a consistent terminology for describing any economic system (Smith, 1982) identified two main components: the economic environment and the institution. The microeconomic system framework is depicted in Figure 2.3, and described in more detail in the following, based on (Smith, 1982, Neumann, 2007).

ECONOMIC ENVIRONMENT

Definition 1 (Economic environment). The *economic environment* is defined as the set of initial circumstances that have an impact on the system performance, but cannot be changed by the agents or the institutions in the system.

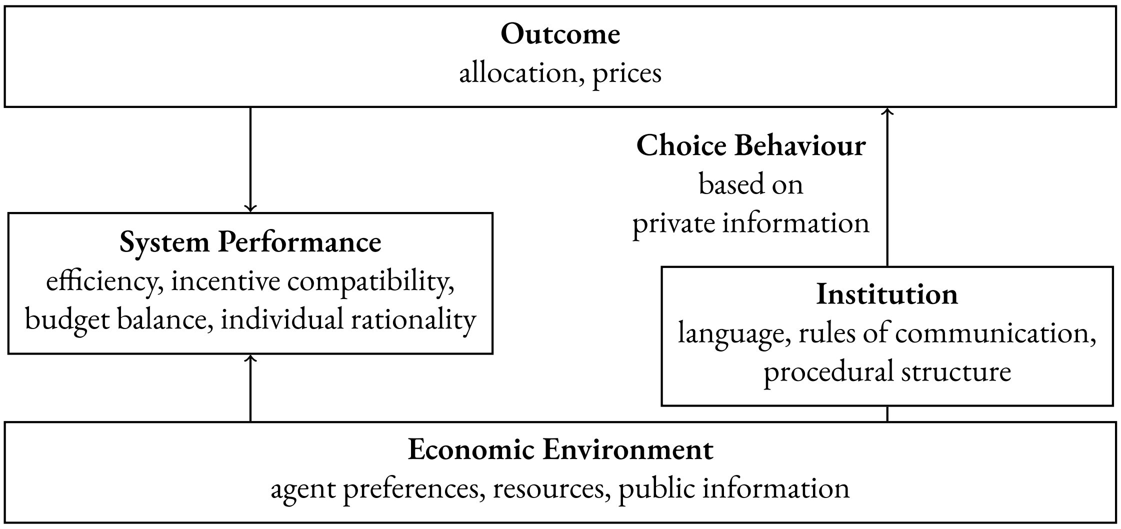

Outcome
allocation, prices
Choice Behaviour
based on
private information
System Performance
efficiency, incentive compatibility,
budget balance, individual rationality
Institution
language, rules of communication,
procedural structure
Economic Environment
agent preferences, resources, public information

Figure 2.3: Microeconomic System Framework (Smith, 1982, Neumann, 2007).

More specifically, an economic environment consists of a set of agents, a set of resources, and certain private characteristics of each agent, such as preferences, utility, knowledge, and endowment.

In this work, I only consider non-cooperative environments, since they are more appropriate for modeling a cloud resource economy where cloud providers compete to sell resources to individually-motivated customers. As such, the modeling unit of an environment is the *agent*, which is (assumed to be) a self-interested and rational entity.

An agent is self-interested not in the sense that it only cares about itself, but rather that it has its own description of which states of the world it likes and acts accordingly to induce these states (Shoham & Leyton-Brown, 2008). The most common way to model an agent's interest stems from utility theory: a *utility function* that quantifies the agent's preference over a set of alternative outcomes. More intuitively, a utility function measures the level of satisfaction obtained by an agent from an outcome. Agents then act in ways that maximize their expected utility (von Neumann et al., 1944).

While utility is a useful metric as a uni-dimensional function, it is actually grounded in a more complex concept called *preference*.

Definition 2 (Preference). Let $\mathcal{O}$ denote a finite set of possible outcomes. For a given pair $o_1, o_2 \in \mathcal{O}$, let $o_1 \succcurlyeq o_2$ denote the fact that an agent weakly prefers outcome o_1 to o_2. Then $\succcurlyeq$ is called the *preference* relation.

Definition 3 (Utility function). A *utility function* $u\colon \mathcal{O} \to \mathbb{R}$ ranks each outcome based on preference: if an agent weakly prefers outcome o_1 over o_2, then $u(o_1) \geq u(o_2)$.

An important finding in microeconomics is that, if an agent's preference relation is complete and transitive, then the agent is rational (von Neumann et al., 1944). Formally, it can also be said that the utility function *u represents* the preference relation $\succcurlyeq$, or:

$$u(o_1) \geq u(o_2) \iff o_1 \succcurlyeq o_2. \tag{2.1}$$

In addition to preference, an agent's utility is also affected by an agent's response to uncertainty, or its *risk attitude* (Shoham & Leyton-Brown, 2008). The risk attitude encodes an agent's value for money: a risk averse agent has a sublinear value for money and thus prefers a definite situation to a risky one with the same expected value (losing and gaining the same amount of money is not valued the same in its utility function), while a risk seeking agent has a superlinear value for money. Nevertheless, it is common and quite reasonable to assume risk-neutral agents who have a linear value for money, as I do in this thesis.

While preference and risk attitudes are in essence private agent characteristics, there can be *public information* influencing an agent's utility, either observable by all agents, or vaguely known and therefore estimated by each agent differently. For example, in cloud computing, an agent does not know how much one virtual machine is worth, but it might roughly estimate its price based on the model and amount of CPUs, the time the machine is needed, current energy prices, and other assumptions about the size and characteristics of the data center where it is running. As such, each agent will have its own estimation for the value of the same VM. On the other hand, historical VM prices can also constitute public information, equally available to all agents.

Finally, an important constituent of the economic environment is the *set of resources* to be allocated, with different characteristics (Neumann, 2007), such as: number of resources, discreetness, number of units, substitutability, complementarity. The characteristics are publicly known to all agents and can have an impact on the agents' utility. For example, two complementary resources are valued more when sold together rather than individually, while substitutable resources might serve the same purpose or have similar technical attributes, therefore being less valuable when sold together than separately.

INSTITUTION

Definition 4 (Institution). The *institution* is the entirety of rules for establishing an outcome, as well as for defining agent messaging behavior. A *market* is the most prominent example of an institution.

The institution provides a *language*, which defines the nature and content of feasible messages for agents to communicate their preferences. Examples of messages include bids and offers. For practical use, a language should be expressive, concise, natural, and tractable.

Furthermore, the institution defines rules that govern the transition from messages to outcomes, grouped in three categories:

- *allocation (choice) rules*, which map the submitted messages to a final resource allocation among the agents,
- *payment (transfer) rules*, which state the payments to be made by each agent as a function of the submitted messages; usually institutions offer incentives to nudge agents to behave in desired ways,

- *adjustment process rules*, which govern the message exchange procedure: the starting rule specifies the time and conditions for the message exchange to begin, transition rules describe the sequencing and exchange of messages, and the stopping rule specifies the exchange termination conditions.

Compared to the economic environment, the institution is the component that can be modified—through the defined rules—in order to enforce certain agent behaviors or system performance requirements. This enables closing the loop between environment, institution and outcome, as depicted in Figure 2.3, and is the subject of mechanism design. Therefore, I will cover the remaining components in Figure 2.3—system performance, choice behavior, and outcome—in the following section, through the lens of mechanism design.

2.2.2 MECHANISM DESIGN

In this section I take on the market designer's perspective: how can market rules be defined without any knowledge of the agents' private preferences, but which guide the market to behave in accordance to certain performance metrics? What kind of incentives can be put in place to make agents reveal enough private information that will lead to the desired outcome?

Mechanism design hence deals with aggregating preferences of strategic agents in order to find an optimal outcome (Shoham & Leyton-Brown, 2008). The assumption that agents are strategic entails that they will behave in a way that maximizes their individual utility, for example, even by misrepresenting their preferences (lying).

It can also be said that a mechanism defines the "rules of the game" (Parkes, 2001), where a game is defined as the totality of circumstances that have a result dependent on the actions of multiple strategic players (or agents). In its normal form (Shoham & Leyton-Brown, 2008), a game is defined by the set of players, each with a finite set of actions available to them, as well as an associated utility function.

Definition 5 (Strategy). A strategy is a complete contingent plan defining the actions an agent will take in every state of the game (Parkes, 2001).

Game theory studies such systems of strategic agents, attempting to find equilibrium states—states where agents cannot increase their utility by unilaterally changing their strategy. This concept is called a *Nash equilibrium* (Nash, 1950). Although fundamental in game theory, it assumes that agents have perfect information about the preferences of the other agents, and that all agents will select the same Nash equilibrium. In games with incomplete information, where agents only have prior knowledge about the *distribution* of the other agents' preferences, a *Bayesian-Nash* equilibrium can be reached when each agent selects a strategy that maximizes their *expected* utility.

A stronger type of equilibrium is the *dominant strategy* equilibrium, where each agent has a single utility-maximizing strategy, independent of the other agents' strategies. This is a desirable solution, as it is very robust since it makes no assumptions about the information available to each agent, but it might not be available in all games.

An ideal mechanism provides agents with a dominant strategy that will ultimately lead to an optimal outcome. The optimal outcome for the entire system is defined with a social choice function.

Definition 6 (Social choice function). A *social choice function* $f \colon \Theta_1 \times \ldots \times \Theta_N \to \mathcal{O}$ chooses an outcome $f(\theta)$ given the agents' preferences $\theta = (\theta_1, \ldots, \theta_N)$.

Definition 7 (Market mechanism). A *market mechanism* $\mathcal{M} = (\Omega, g(\cdot))$ defines the strategies $\Omega = \Omega_1 \times \ldots \times \Omega_N$ available to each agent, and an outcome rule $g \colon \Omega \to \mathcal{O}$ that maps strategy profiles to possible outcomes.

Thus the mechanism can restrict agent behavior through the defined strategies (e.g., by allowing only bids above the asking price of a resource), and then select the outcome based on these strategies (e.g., the highest bidder wins and pays the second highest bid).

Then, it can be said that a mechanism implements a social choice function if the outcome computed with equilibrium strategies is a solution to the social choice function for all possible agent preferences.

Definition 8 (Mechanism implementation). Given the mechanism $\mathcal{M} = (\Omega, g(\cdot))$ and the social choice function $f(\cdot)$, $\mathcal{M}$ *implements* f if $g(s^*(\theta)) = f(\theta), \forall \theta \in \Theta_1 \times \ldots \times \Theta_N$, where $s^* \in \Omega$ is an equilibrium strategy profile.

Mechanisms are designed to reach certain objectives, which should be defined upfront by the mechanism designer. In the following, I outline the most relevant objectives (or system performance metrics, cf. Figure 2.3) for a cloud computing economy, while a comprehensive overview can be found in the literature (Neumann, 2007).

Definition 9 (Incentive compatibility). A mechanism is *incentive compatible*, or *truthful*, if it is direct (agents are asked to report their individual preferences), and each agent's equilibrium strategy is to report its true preference.

Definition 10 (Economic efficiency). A mechanism is *economically efficient* if it produces a strictly Pareto efficient outcome: there is no other allocation that improves any individual preference without making at least one preference worse off. This can also be expressed as the allocation that maximizes the social welfare, or the total utility of all agents, if quasilinear utilities are assumed.

Definition 11 (Budget balance). A mechanism is *budget balanced* if it breaks even: it makes neither a profit, nor a loss. The relaxation of this condition allows the mechanism to make a profit, and is called *weak budget balance*.

Definition 12 (Individual rationality). A mechanism is *individually rational* if no agent loses by participating in the mechanism.

2.2.3 AUCTIONS

One family of market mechanisms—auctions—provides a general solution to the resource allocation problem in systems of self-interested agents.

Auctions have been thoroughly studied from a theoretical point of view, due to their long tradition of practical use. Owing to their omnipresence in day-to-day life, the term *auction* instantly invokes in any reader's mind some well-known examples: the antique business, fine art collectibles, the Amsterdam flower market, electromagnetic spectrum licenses, or governmental contracts. With the advent of the internet, the reach of auctions has expanded to trading almost any type of goods through online platforms such as eBay [12].

The reasons to use auctions are abundant; for example, for goods that have no standard value (such as rare paintings), or for fair decision-making in situations where there is intense competition; more recently, for efficiency in allocating computational resources. As a result, there is an equally diverse ecosystem of auctions, each designed to attain a different goal. In this section, I give a brief overview of the taxonomy of auctions.

Definition 13 (Auction). McAfee & McMillan (1987) define an auction as *"a market institution with an explicit set of rules determining resource allocation and prices on the basis of bids from the market participants"*.

Formally, auctions provide a bidding language for potential buyers or sellers to express their interest, and define an allocation rule and a payment rule to decide, based on the collected bids, the auction winners and the corresponding payments.

[12] `https://www.ebay.com`

Auctions can be classified according to several criteria (Shoham & Leyton-Brown, 2008):

- *one-sided vs. two-sided*, depending on whether there are both buyers and sellers whose bids the auctioneer has to match, or only one side, typically buyers.
- *sealed-bid vs. open-outcry*, if the bids are only known to the auctioneer or openly available to the other participants as well.
- *first-price vs. k-th price*, if the winner pays the price of the winning bid or the price of the k-th ranked bid.
- *single-unit vs. multi-unit*, if it is possible to bid for a single good or for multiple units of the same good (homogeneous resources).
- *single-item vs. multi-item*, if the bidding can be done for a single good or bundles of multiple goods (heterogeneous resources).

One-sided single-unit auctions are the most known, most used, and also the simplest type of auctions. Although there are many variations, such as the English auction (where the bids are announced in ascending order, usually sell-side with reserve price, and the winner pays first-price; a classic type of auction, used in most auction houses, such as Sotheby's), the Dutch auction (similar to the English auction, but the bids are announced in descending order; used in the Dutch flower markets), the Japanese auction (where the auctioneer calls out ascending prices and bidders announce when they drop out of the auction until a single bidder remains, hence open-outcry first-price; it originated in the Japanese fish markets), or the Vickrey auction (second-price sealed-bid; first used in stamp auctions (Lucking-Reiley, 2000), now very popular with Internet auctions, e.g., on eBay), to just name a few, they are not that different in terms of expected revenue, according to the Revenue Equivalence Theorem (Vickrey, 1961). In practice, however, factors such as risk attitudes and incomplete information might lead to different revenues for different auction formats.

COMBINATORIAL AUCTIONS

Combinatorial auctions (De Vries & Vohra, 2003) are one-sided multi-item mechanisms that allow agents to express more complex preferences over arbitrary combinations (bundles) of multiple goods, and thus encode complementarity and substitutability of resources.

There are several aspects that make combinatorial auctions difficult: defining an appropriate bidding language, solving the allocation rule, and putting proper incentives in place for reaching a number of desired objectives.

Bidders use the bidding language to communicate their valuation, which should theoretically be defined for all possible bundles. Unfortunately, the number of possible combinations of goods grows exponentially with the number of goods, rendering this task impossible. It is necessary to have a concise but expressive language, that is also tractable for the auctioneer's allocation algorithm. Common ways to express bids are (Shoham & Leyton-Brown, 2008): atomic bids (a particular bundle of goods accompanied by the price the bidder is willing to pay), OR bids (disjunction of atomic bids), and XOR bids (exclusive OR bids).

The allocation rule has been named the Winner Determination Problem and determines which bids are to be accepted.

Definition 14 (Winner Determination Problem). The Winner Determination Problem (WDP) (Lehmann et al., 2006) for a combinatorial auction, given the agents' valuations, is to find a feasible allocation of goods to agents that maximizes the total utility of agents (the social welfare).

Consequently, considering that resources are discrete and bundled together, the WDP is formulated as an integer program, which is an $\mathcal{NP}-$hard problem. The WDP is actually an instance of the Set Packing Problem, as stated by Sandholm (2002).

Since polynomial time algorithms do not exist for the general-case WDP, there are two ways of dealing with the computational issue (De Vries & Vohra, 2003): either restricting the problem to a special class of problems that can be solved in polynomial time (such as total unimodularity (Nemhauser & Wolsey, 1988)), or using heuristics that sacrifice the optimality of the solution. As a result, economic efficiency cannot be guaranteed in most cases.

Another important incentive issue relates to truthfulness. Allocative efficiency and incentive compatibility can only be achieved simultaneously through a mechanism named Vickrey-Clarke-Groves (VCG) (Nisan et al., 2007a), where each agent pays its *social cost*. This means that, for each agent, it is necessary to also compute the social welfare of the system if the agent would not participate in the auction, essentially solving another (exponential) WDP-like problem. The computational complexity makes this payment scheme unusable in practice. Furthermore, VCG violates the budget balance property by forcing the auctioneer to subsidize the trade.

EXCHANGES

When competitive bidding occurs on both seller and buyer side, we are dealing with two-sided, or double auctions, also called *exchanges*. The most notable example of an exchange is the stock market.

Unfortunately, exchanges are not as well-studied as their one-sided counterparts, as they are difficult to model game-theoretically. One important, albeit negative, theoretical result is the impossibility theorem by Myerson & Satterthwaite (1983), which states that no mechanism can simultaneously satisfy the following four properties: individual rationality, truthfulness, economic efficiency, and budget balance. Consequently, it is left to the mechanism designer to decide which properties must be satisfied, in accordance to other problem-specific requirements.

A significant aspect for exchanges is the timing of clearing the market. Two types of markets can be distinguished here: the continuous double auction (CDA) and the periodic double auction (call market). In a CDA, bids are submitted asynchronously and the auctioneer tries to satisfy them immediately, while the call market is cleared after fixed time intervals, during which bids are collected. It is clear that CDAs are more suitable when response time is essential, while call markets typically yield more efficient allocations.

In summary, this section demonstrated the theoretical breadth of auction mechanisms and their wide applicability for real-world problems, being especially suited for resource allocation. The multitude of auction flavors facilitates the design of mechanisms tailored to any given problem's specifications.

2.3 ALGORITHM SELECTION PROBLEM

The algorithm selection problem has been widely studied in different areas of artificial intelligence, often under different names, such as hyper-heuristics (Burke et al., 2013) or meta-learning (Lemke et al., 2015). This sustained interest is prompted by the ubiquity of $\mathcal{NP}-$hard combinatorial optimization problems, for which optimal algorithms are intractable, but a multitude of heuristics are being developed. However, in most cases, a new heuristic improves upon existing algorithms for certain classes of problems, only to perform worse for other classes. This idea was formalized through the No Free Lunch (NFL) theorems (Wolpert & Macready, 1997), showing that, usually, one single algorithm cannot yield superior performance across all possible problems. Algorithm selection promises to close the performance gap between different heuristic algorithms by selecting the best heuristic for each given instance.

Definition 15 (Algorithm Selection Problem). The algorithm selection problem deals with choosing the most suitable algorithm for solving a problem instance on a case-by-case basis (Kotthoff, 2014).

2.3.1 BASIC MODEL

Rice (1976) was the first to formulate an abstract model for algorithm selection, depicted in Figure 2.4. Essentially, the model states that, given a space of algorithms $\mathcal{A}$ and a space of problems $\mathcal{X}$, one can construct a mapping $P : \mathcal{A} \times \mathcal{X} \rightarrow \mathcal{R}^n$ from each pair of algorithm–problem (A, X) to its performance $P(A, X)$, where performance can be a multi-dimensional measure. Then the algorithm selection problem can be reformulated as the problem of finding the selection mapping $S : \mathcal{X} \rightarrow \mathcal{A}$ that maximizes performance (normalized as $||P||$). In the following, I discuss the model components in more detail.

The set of problem instances is large, possibly infinite, and diverse. The *problem space* $\mathcal{X}$ can be defined as a highly-dimensional space that describes the input data, where by dimensions I understand the various independent problem characteristics that have an impact on algorithm performance. Since the problem space cannot be fully known, algorithm selection typically works with samples drawn from this space for empirical evaluation. The difficulty is to ensure that the samples are representative. In a refined model of algorithm selection, Rice (1976) added a *feature space*, in order to reduce the dimensionality of the problem space while extracting features that are relevant to the selection mapping. He also argues that,

> *"The determination of the best (or even good) features is one of the most important, yet nebulous, aspects of the algorithm selection problem".*

Nevertheless, almost all modern approaches base their algorithm selection on feature extraction (usually fed to a Machine Learning system).

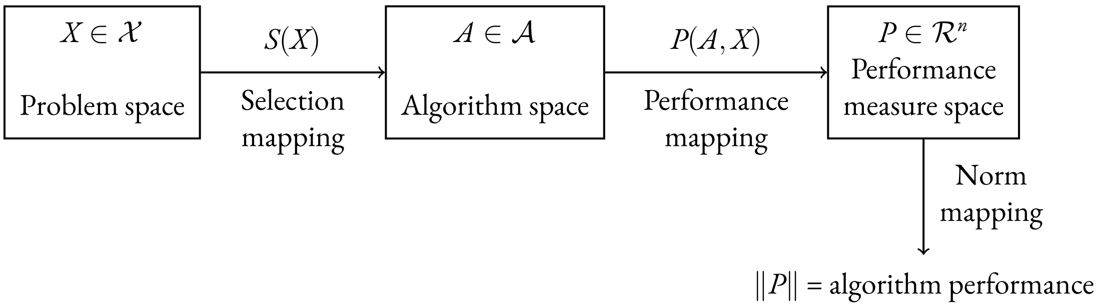


Figure 2.4: Basic algorithm selection model introduced by (Rice, 1976).

The *algorithm space* $\mathcal{A}$ can be equally diverse and large, even though in practice only a small number of algorithms are considered. Different parameters for the same algorithm are counted as different algorithms, since their effect on performance can be significant. The algorithms are typically organized in portfolios.

The *performance measure space* consists of the multitude of criteria used to evaluate algorithm performance, contingent on the desired objectives. The criteria might not be comparable, and might have some degree of uncertainty. Execution time and solution accuracy are the most commonly used, but the space can also include metrics such as: memory usage, ease of use, interpretability, etc.

Rice (1976) also identified four different (non-exhaustive) criteria for the *selection*:

- *best selection*, equivalent to choosing the selection mapping that results in maximum performance for ech problem.
- *best selection for a subclass of problems*: a single algorithm is selected for every subclass of problems, such that the performance degradation per subclass is minimized.
- *best selection from a subclass of mappings*: the selection mapping is restricted to be of a certain form (i.e. from a subclass of mappings); then the best selection is chosen from this subclass such that performance degradation for all problems is minimized.
- *best selection from a subclass of mappings and problems* combines the last two criteria.

The *best selection* is the ideal case, but difficult to attain in practice, due to incomplete information about the problem space or algorithm behavior.

The basic model in Figure 2.4 can be extended (Kotthoff, 2014) to include a feedback loop from performance measure back to the selection model, to incorporate past algorithm performance into the model.

2.3.2 CLASSIFICATION OF APPROACHES

Kotthoff (2016) surveyed the literature of existing approaches for algorithm selection, applied to combinatorial search problems. Despite the large number of publications on the topic and the growing interest over the last years, according to Kotthoff (2014), the research community is fragmented and there is little exchange of ideas, leading to some techniques being re-invented. This is also due to the fact that researchers tend to publish in domain-specific venues, as there is no algorithm selection community *per se*.

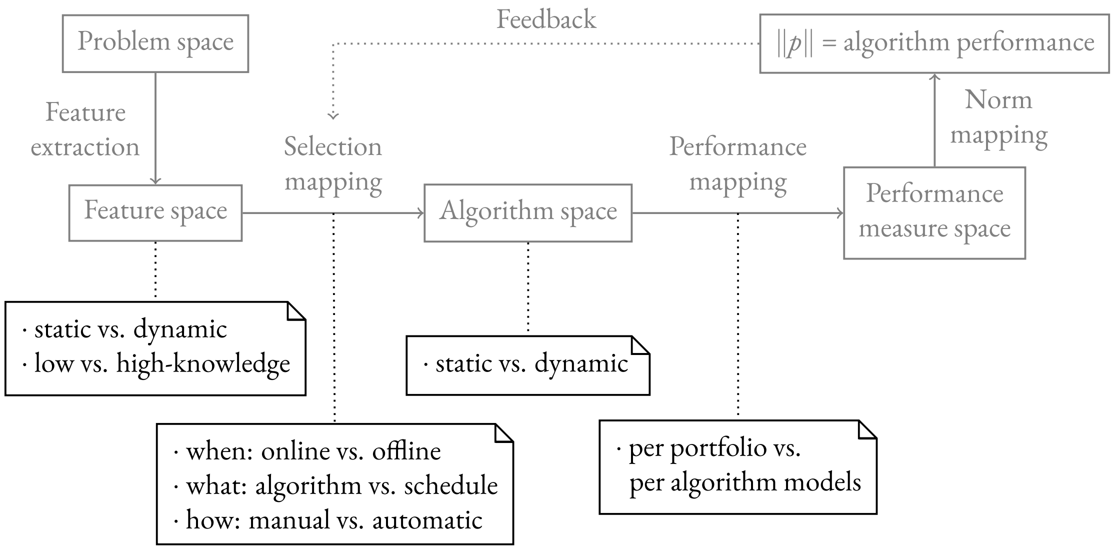


Figure 2.5: Categorization of algorithm selection techniques based on different aspects along the selection pipeline.

As a result, there is a proliferation of approaches, often similar, but with variations imposed by problem-specific challenges. I categorize the existing approaches below, according to various criteria, based on the survey performed by Kotthoff (2016). Figure 2.5 places these criteria along the algorithm selection pipeline.

FEATURES

The feature space characterizes the problems or the algorithms in a way that is meaningful for the selection. Since defining these features is the most difficult stage of algorithm selection, a myriad of approaches exist, each with its individual strengths. The features can be classified according to two criteria: how much domain knowledge is needed to define them (low or high-knowledge), and when and how are the features computed (static or dynamic).

High-knowledge features are the most common (Leyton-Brown et al., 2002, Xu et al., 2007, Kadioglu et al., 2010, Xu et al., 2011), yet complex type. Domain-specific expertise informs the definition of a diverse sets of features, gathered under the all-encompassing term of *instance features*. For example, Kadioglu et al. (2010) apply instance-specific algorithm configuration to several problem domains and define features for each domain, such as: number of variables and of constraints, percentage of binary variables, mean and standard deviation of coefficients of objective function (Mixed Integer Programs), number of clauses, node degree statistics for the variable graph (satisfiability problems or SAT), statistics for vectors of normalized costs, item costs, bag densities (Set Covering Problem).

In comparison, low-knowledge features are more generic, hence applicable to other domains. They can be extracted from: past performance of the algorithms on previous instances (Fitzgerald & O'Sullivan, 2017), probing the search space—e.g. by running the portfolio for a short time on the current instance (Beck & Freuder, 2004)—, or search statistics computed at runtime to guide the search to solution—e.g. for constraint satisfaction problems, conflict related information (Stergiou, 2008). It is also possible to combine low- and high-knowledge features (Xu et al., 2011).

Static features are computed offline, before beginning to solve the problem instance. Most instance features fit into this category, since they are numerous and costly to compute. However, they fail to take into account the performance of the algorithms on the current instance.

Alternatively, dynamic features are computed during the solution process, which is then adapted based on these features by e.g. changing the algorithm configuration (Xu et al., 2011) or selecting a different heuristic to continue solving the instance (Stergiou, 2008).

Some probing features (Beck & Freuder, 2004) fall in-between these categories and are termed semi-static, since they are computed before the actual solving of the instance (i.e. statically), but they do explore a part of the search space for the current instance.

ALGORITHM PORTFOLIO

Two types of algorithm portfolios can be distinguished, depending on how the portfolio is constructed: static and dynamic. This classification is similar to the distinction between heuristic selection and generation made in the survey on hyper-heuristics (Burke et al., 2013).

Static portfolios are the most common (Leyton-Brown et al., 2002, Beck & Freuder, 2004, Xu et al., 2007, Fitzgerald & O'Sullivan, 2017). A static portfolio is constructed before solving any problem instance and it does not change at runtime. As a result, the number of algorithms is fixed and the algorithms cannot be adapted based on information gained at runtime, which makes this approach inflexible. On the upside, this approach is also straightforward and has no runtime overhead. The said inflexibility can be mitigated by thoughtfully deciding on the composition of the portfolio. Perhaps counter-intuitively, rather than including the algorithms with good overall performance, it is preferable that algorithms complement each other. For that reason, Xu et al. (2007) select candidate solvers whose runtimes are relatively uncorrelated. Xu et al. (2012) quantify an algorithm's value with regard to its ability to solve instances that cannot be solved by other algorithms, i.e. the algorithm's marginal contribution to the state-of-the-art, and propose constructing portfolios based on this measure.

Dynamic portfolios are modified at runtime, either by assembling heuristic building blocks into algorithms (Elsayed & Michel, 2011), or through automatic parameter tuning (Xu et al., 2011, Kadioglu et al., 2010).

SELECTION

Depending on when and what is selected, and how the selection is performed, algorithm selection approaches can be categorized as follows.

When. The distinction between offline and online selection is linked to the distinction between static and dynamic features, and it indicates whether the selection is performed before (offline) or while solving the problem (online). The latter has the benefit of a finer-grained control, since it can mitigate bad choices by monitoring the performance of the selected algorithm on the current instance and selecting a more suitable algorithm. However, the increased overhead and complexity of implementation make this approach (Kadioglu et al., 2010, Fitzgerald & O'Sullivan, 2017, Stergiou, 2008, Elsayed & Michel, 2011) less explored than the offline one (Leyton-Brown et al., 2002, Beck & Freuder, 2004, Xu et al., 2007, 2011). It is also possible to combine both approaches: Stern et al. (2010) complement an offline selection (based on instance features) with runtime information, which is fed back into the system for online training, as well as for adapting or switching the algorithms at runtime.

What. Most selection models choose a single algorithm to solve the instance. Alternatively, a schedule of algorithms can be devised, to be run in parallel (Yun & Epstein, 2012), or in the order and for a duration proportional to the algorithms' expected performance (Roberts & Howe, 2006). Both methods can work equally well, depending on the use case: selecting a single algorithm has the advantage of simplicity of implementation and lower computational cost, while a schedule brings robustness.

How. Early research on algorithm selection used explicit rules, handcrafted based on expert knowledge, to decide when to switch an algorithm while solving an instance (Borrett & Tsang, 2009) or to estimate the best performing algorithm based on probing information (Beck & Freuder, 2004). In contrast, recent work automates the creation of performance models (discussed in the next section) through various machine learning methods, be it classification (Roberts & Howe, 2006, Xu et al., 2011), regression (Leyton-Brown et al., 2002, Xu et al., 2007), clustering (Stergiou, 2008, Kadioglu et al., 2010), or reinforcement learning (Fitzgerald & O'Sullivan, 2017).

PERFORMANCE MODELS

The performance models are closely linked to what is predicted and how the selection is made. The prediction can be a categorical value (the best algorithm), a performance value (the runtime/cost/utility of each algorithm in the portfolio), or the solution quality (used as a criteria for when to switch the algorithm in online settings). Accordingly, one can differentiate between per-portfolio and per-algorithm performance models.

A performance model for an entire portfolio can be constructed from training data, usually to predict the best algorithm for a given instance, without actually modelling each algorithm's performance. For example, Kadioglu et al. (2010) cluster a training set of representative instances and assign suitable algorithm configurations to each cluster; then, for each new instance, the configuration of the closest cluster (in the defined feature space) is used.

Another approach is to build a performance model for each algorithm; then, for any new problem instance, one can use these models to predict the performance of each algorithm on the instance, and select the best-performing one. For example, Leyton-Brown et al. (2002) use regression to build per-algorithm performance models in order to predict running time. Per-algorithm models have the benefit that it is easy to add and remove algorithms from the portfolio, since it is not necessary to retrain the model for the entire portfolio, but rather just for a single algorithm. However, they fail to consider interactions between algorithms.

2.3.3 COMBINATORIAL AUCTION ALGORITHM SELECTION

Even though there is a substantial amount of research on algorithm selection, there is not much particularly aimed at combinatorial auctions (the winner determination problem), or simply generic but tested on auctions. Moreover, existing work is designed towards optimizing runtime as a single measure of performance and does not consider other objectives.

Leyton-Brown et al. (2002) build performance models for optimal algorithms for the WDP (so-called empirical hardness models) to predict algorithm runtime—more specifically, the log of runtime, i.e. order of magnitude. They do so by training a regression model on domain-specific instance features extracted from artificially generated data, coupled with the measured algorithm performance on the training instances. Individual models are trained for the three algorithms in the portfolio, which are then used to predict each algorithm's runtime for any new problem instance (after computing its feature values). The algorithm predicted to be fastest is finally selected. The authors show that the portfolio approach outperforms the algorithm that is fastest on average by roughly a factor of 3 (Leyton-Brown et al., 2003). They further tune their approach by smart feature computation and capping runs.

Fitzgerald & O'Sullivan (2017) introduce an online automatic algorithm configuration system, which is tested, among other datasets, on combinatorial auction instances. The system selects multiple solver configurations from a set of possible candidates, which are run in parallel on each instance. The algorithm that finishes first is considered the winner, and by means of reinforcement learning, this information is fed back into the ranking system used in the selection procedure for future instances. The influence of ordering and grouping of instances over the solving time is also investigated.

Stern et al. (2010) propose a novel approach to portfolio-based algorithm selection that combines two sources of information: static instance features with dynamically updated algorithm features. The algorithm features are learned by a Bayesian model, trained online with past algorithm performance. The approach is tested on the same combinatorial auctions dataset as Leyton-Brown et al. (2003), and yields similar improvements.

Essentially, all models are wrong, but some are useful.

George Box

3 Problem Modeling

Section 2.1 reviewed the state-of-the-art in Cloud Computing and emphasized the discernible trend towards market-driven resource allocation. In this chapter, I formulate the requirements expected from a such a resource allocation approach, in accordance with the presented trends and unanswered needs. I then formally describe the proposed market-based model for the cloud resource allocation problem in Section 3.2, first introduced in (Gudu et al., 2018a). Finally, in Section 3.3, the model is compared against existing approaches, with respect to the elicited requirements.

3.1 REQUIREMENT ANALYSIS

Having established the need for market-inspired approaches in Section 2.1, there are two significant issues that need to be addressed in the allocation of cloud resources.

1. **Inflexibility:** any novel approach should be flexible enough for both customers and providers. For customers, this means the ability to customize the resources they pay for, with a fine-grained control that supports a wide range of preferences. For providers, flexibility is reflected in the pricing scheme: dynamic prices that adapt with changes in demand and supply lead to higher revenues. In contrast, the industry standard is to provide predefined combinations of resources (e.g. so-called VM instance types for EC2) at fixed prices.

2. Impracticability: crucial for practical adoption, the proposed approach should reconcile the two conflicting goals of allocative efficiency and speed in a satisfying manner. This has been the primary hindrance for implementing such flexible mechanisms in practice, otherwise well studied in theory.

Consequently, several requirements for a flexible and practicable market-based allocation of cloud resources can be extracted, as depicted in Figure 3.1. The reasoning behind each requirement is detailed in the following.

3.1.1 RESOURCE ALLOCATION-DRIVEN REQUIREMENTS

Requirement 1 (Market-driven pricing). In competitive environments with variable demand like cloud computing, a fixed pricing scheme leads to inefficiencies (Lai, 2005): a demand below the fixed price causes unrealized utility for the customers who are unwilling to pay such a high price. On the other side, a demand above the fixed price produces unrealized profit on the provider side, since some customers would be willing to pay more for the resources. What is more, the fixed pricing prevents providers from differentiating between the preferences of potential buyers, and thus fail to sell the resources to the customers that want them the most; this is more unrealized utility on the client side. A variable price that follows the variable demand and supply of resources (i.e. market-driven) would eliminate these inefficiencies.

Requirement 2 (Fine-grained control). Customer needs are quite diverse, since cloud customers themselves are diverse. As a result, customers should be able to specify the combination and characteristics of the resources they need, and only pay for those, rather than selecting from a small number of predefined options. This would also increase resource utilization on the provider side.

Requirement 3 (Resource bundling). Customers typically need several types of resources to be able to perform a single task. For example, the NASA image and video library[1] uses several Amazon cloud services[2], including resources such as VMs, object storage, message queue, and relational database. If a customer is not allocated one of the resources, then it will not be able to run its task, and thus have no use for the other resources. Hence customers should have the possibility to request and buy resources in bundles, which are then allocated simultaneously.

[1] https://images.nasa.gov/

[2] https://aws.amazon.com/partners/success/nasa-image-library/

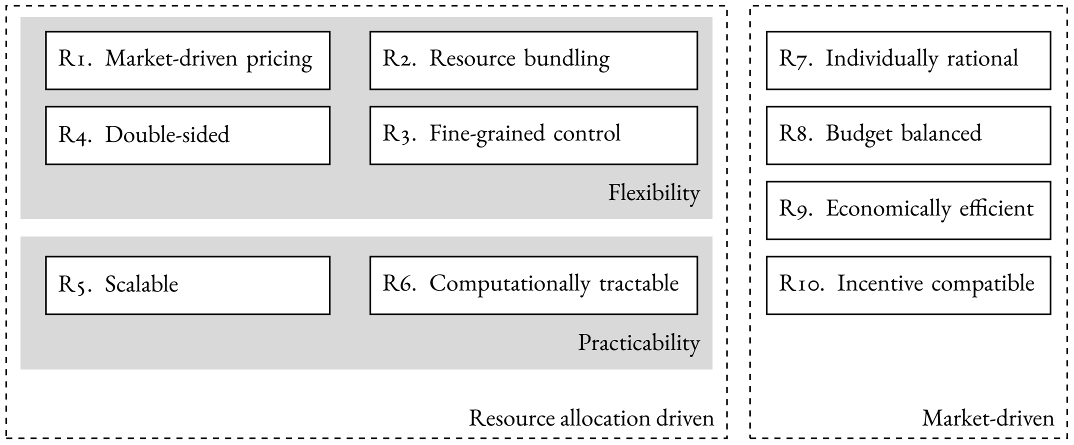

R1. Market-driven pricing
R2. Resource bundling
R4. Double-sided
R3. Fine-grained control
Flexibility
R5. Scalable
R6. Computationally tractable
Practicability
Resource allocation driven
R7. Individually rational
R8. Budget balanced
R9. Economically efficient
R10. Incentive compatible
Market-driven

Figure 3.1: Requirements for a flexible and practicable market-based allocation of cloud resources.

Requirement 4 (Double-sided). Current allocation mechanisms and many proposed approaches are one-sided. On the one hand, each provider runs its own market-based mechanism, and customers have to submit bids to all the providers they are interested in. On the other hand, brokering is done for each customer, in order to match one customer's needs to the available resources from several providers. A double sided approach would include multiple customers and providers in the decision process. This would take the decision burden off either side to offload it onto a third objective party, and ensure a healthy competition among all the players in the marketplace.

Requirement 5 (Scalable). As the cloud market is constantly growing (cf. Section 2.1), any novel allocation mechanism should be able to deal with requests from many customers and providers. The mechanism should thus be scalable in all stages of allocation: receipt and processing of incoming requests, matchmaking of requests to offers, and communicating the decision and payments.

Requirement 6 (Computationally tractable). Related to the scalability requirement, the mechanism should also be computationally tractable. This refers specifically to the problem of finding the best allocation, for which polynomial time algorithms should be used.

3.1.2 PROPOSED APPROACH

Double combinatorial auctions satisfy the requirements, with the caveat that fast heuristic algorithms are needed to solve the otherwise intractable WDP. My proposed approach assumes a call market model—auctions are run at predefined intervals rather than continuously matching requests as they are submitted.

I now discuss how double combinatorial auctions comply with the requirements above.

The market-driven pricing requirement is satisfied by auctioning off resources and setting their price based on submitted bids and asks. The combinatorial aspect deals with fine-grained control and resource bundling, since customers can express complementarity of resources by submitting bids on their preferred combination and quantities of resources. The two-sided aspect of the auction enables the satisfaction of the double-sided requirement.

Scalability can be ensured by employing a bidding language that is simple, easy to use, and tractable. Furthermore, since the WDP is $\mathcal{NP}-$hard, using heuristic algorithms to approximate the solution rather than solving the problem optimally guarantees tractability and scalability of the matchmaking part of the allocation.

Be that as it may, the problem with using heuristics is not just that they are not optimal, but for combinatorial auctions they provide no guarantees of how far from optimal the solution is. Even more, heuristics can perform quite differently across the input space. This is the issue at the core of this thesis; I address it by proposing two approaches for algorithm selection, in Chapter 5 and Chapter 6.

3.1.3 MARKET-DRIVEN REQUIREMENTS

Due to the decision using market mechanisms for resource allocation, there are some additional requirements driven by mechanism design. When designing an auction mechanism, four essential economic properties need to be considered, as defined in Section 2.2.2: incentive compatibility, individual rationality, economic efficiency, and budget balance. Since no mechanism can simultaneously satisfy all four properties, their significance for cloud resource allocation is discussed below.

Requirement 7 (Individually rational). In a real world environment, it is unacceptable for either buyers or seller to lose any money by participating in the auction, since there are no other incentives for participants. As such, individual rationality must be met.

Requirement 8 (Budget balanced). The auctioneer should not incur any loss by running the auction. Assuming that this is a piece of software ran by a third-party, the auctioneer cannot subsidize the trade, since it needs an incentive to set-up and maintain the necessary infrastructure. Thus, the market mechanism should either be weakly budget balanced (the auctioneer makes a profit from every auction it runs) or budget balanced (assuming a different business model for the auctioneer, e.g. where providers pay a fixed fee to join the market).

Requirement 9 (Economically efficient). A mechanism is economically efficient if it maximizes the social welfare of the system. This property typically conflicts with revenue maximization (Myerson, 1981), but it is usually preferred (Neumann, 2007) since it encourages participation (Parkes, 2001). As a result, this thesis also aims for an economically efficient mechanism. For combinatorial exchanges, this requirement unfortunately conflicts with the tractability requirement, which is essential in a real world scenario. Under the circumstances, the efficiency requirement can be relaxed; I nonetheless take steps to improve efficiency as much as possible.

Requirement 10 (Incentive compatible). A mechanism is incentive compatible if each agent's equilibrium strategy is to reveal its true valuation. Although possible in theory, such mechanisms are difficult to implement. This is another requirement that can be relaxed. Section 3.2.2 discusses the reasons why existing truthful schemes can not be used, as well as the payment scheme chosen to mitigate, to some extent, any attempts to game the system.

3.2 PROBLEM FORMULATION

The participants of a double combinatorial auction for cloud resource allocation are:

- a set of n cloud customers, or buyers, $\mathcal{U} = \{1, \ldots, n\}$,
- a set of m cloud providers, or sellers, $\mathcal{P} = \{1, \ldots, m\}$,
- and an auctioneer that decides upon the allocation and pricing of resources based on the requests received from the buyers and sellers.

The object of the exchange is a set of l goods, or types of cloud resources, $\mathcal{G} = \{1, \ldots, l\}$, which can be traded in various integer amounts, packaged in bundles.

Each trade participant, be it a buyer or a seller, submits a single request to the auctioneer, expressing its interest in acquiring or selling a particular bundle of cloud resources. A buyer's request is called a bid, while a seller's request is called an ask, and they are defined below.

Definition 16 (Bid). A buyer i's bid for a bundle of cloud resources is defined as a vector of quantities and an associated bundle value:

$$(\langle r_{i1}, \ldots, r_{il} \rangle, b_i), \forall i \in \mathcal{U} \tag{3.1}$$

where r_{ig} is the (integer) amount of resource type g in the bundle requested by buyer i, and b_i is the maximum amount buyer i is willing to pay for the bundle.

Definition 17 (Ask). An ask or offer of a seller j for a bundle of resources is defined as:

$$(\langle s_{j1}, \dots, s_{jl} \rangle, a_j), \forall j \in \mathcal{P} \tag{3.2}$$

where s_{jg} is the (integer) amount of resource type g in the bundle offered by seller j, and a_j is the minimum payment seller j is willing to accept for the bundle.

After collecting the bids and asks, the auctioneer computes the resource allocation and pricing according to two rules:

1. Allocation rule: the auctioneer solves the Winner Determination Problem (WDP) by deciding which bidders and sellers will trade goods such that the social welfare is maximized.
2. Payment rule: the auctioneer computes the prices at which the bundles are traded by using a payment scheme that satisfies certain economic properties.

The social welfare is defined (Shoham & Leyton-Brown, 2008) as the total utility of all trade participants, where utility is, informally, a quantitative measure of an agent's happiness (formal definition in Section 2.2.1).

A significant impact on utility is carried by an agent's *valuation function*. Briefly, the valuation $v\colon \mathcal{O} \to \mathbb{R}^+$ is the true monetary value that the agent places on each possible bundle, typically derived by combining public information with private preferences.

For simplicity of bid representation, I assume that both bidders and sellers are single-minded, which means that they are only interested in buying or selling one particular bundle in full, and have zero valuation for any other bundle.

Definition 18 (Single-mindedness). A valuation v is *single-minded* if there exists a bundle S^* and a value $v^* \in \mathbb{R}^+$ such that $v(S) = v^*, \forall S \supseteq S^*$, and $v(S) = 0$ otherwise.

Given these assumptions, the utility of bidders and sellers, as well as the social welfare can be formally defined as follows (Nisan et al., 2007b, Shoham & Leyton-Brown, 2008).

Definition 19 (Bidder utility). Let $v_i(S)$ be a buyer i's valuation for a bundle S. If the buyer pays a price p_i for the bundle, then its utility $u_i(S)$ can be defined as the difference between the bundle valuation and the actual price paid by buyer i at the end of the auction.

$$u_i(S) = \begin{cases} v_i(S) - p_i, & \text{if } i \text{ wins bundle } S \text{ in the auction} \\ 0, & \text{otherwise.} \end{cases} \tag{3.3}$$

Definition 20 (Seller utility). Let $v'_j(S)$ be a seller j's valuation for a bundle S, and p'_j the payment seller j receives for the bundle at the end of the auction. Then its utility $u'_j(S)$ can be defined as the difference between the bundle's price and its valuation, also viewed as the profit of seller j.

$$u'_j(S) = \begin{cases} p'_j - v'_j(S), & \text{if } j \text{ wins bundle } S \text{ in the auction} \\ 0, & \text{otherwise.} \end{cases} \tag{3.4}$$

Definition 21 (Social welfare). A market's social welfare is defined as the sum of the utilities of all trade participants. For a double combinatorial auction with n buyers and m sellers, the social welfare w can be expressed as:

$$w = \sum_{i=1}^{n} u_i + \sum_{j=1}^{m} u'_j. \tag{3.5}$$

3.2.1 ALLOCATION RULE

In a combinatorial exchange, maximizing the social welfare is equivalent to maximizing the surplus, or profit (Kothari et al., 2004), which is essentially the difference between the revenue brought in by the buyers and the amount paid to the sellers. Therefore, the allocation rule determines the auction winners such that the surplus is maximized.

Let $x_i \in \{0, 1\}, \forall i \in \mathcal{U}$ denote the decision variable for bidder i, whose value represents whether bidder i will win the requested bundle in the auction ($x_i = 1$) or not ($x_i = 0$).

The winning sellers are designated in relation with the buyers that they trade with. As such, let $y_{ij} \in \{0, 1\}, \forall i \in \mathcal{U}, \forall j \in \mathcal{P}$ denote the decision variable stating whether bidder i will buy a bundle of resources from seller j ($y_{ij} = 1$) or not ($y_{ij} = 0$).

Definition 22 (Surplus). Given a combinatorial double auction, let $v_i\colon \mathcal{O} \to \mathbb{R}^+, \forall i \in \mathcal{U}$ be the valuation functions of the bidders, and $v'_j\colon \mathcal{O} \to \mathbb{R}^+, \forall j \in \mathcal{P}$ the valuation functions of the sellers. Then the net monetary gain realized by trading the goods is called a *surplus*, and is defined as the difference between the sum of the valuations of the winning bidders and the sum of the valuations of the winning sellers:

$$\Delta = \sum_{i=1}^{n} v_i x_i - \sum_{j=1}^{m} \sum_{i=1}^{n} v'_j y_{ij}. \tag{3.6}$$

Note that the auctioneer does not know the true valuations, but only has access to the *revealed* valuations submitted by bidders and sellers: b_i and a_j, respectively. Accordingly, the allocation rule will maximize the *revealed surplus* (Kothari et al., 2004).

Then the WDP is formalized as the surplus-maximizing integer program:

$$\max_{x,y} \left(\sum_{i=1}^{n} b_i x_i - \sum_{j=1}^{m} \sum_{i=1}^{n} a_j y_{ij} \right) \tag{3.7}$$

subject to the following constraints:

$$x_i, y_{ij} \in \{0,1\}, \forall i \in \mathcal{U}, \forall j \in \mathcal{P} \tag{3.8}$$

$$\sum_{i=1}^{n} y_{ij} \leq 1, \forall j \in \mathcal{P} \tag{3.9}$$

$$\sum_{j=1}^{m} y_{ij} = x_i, \forall i \in \mathcal{U} \tag{3.10}$$

$$r_{ig} x_i \leq \sum_{j=1}^{m} s_{jg} y_{ij}, \forall i \in \mathcal{U}, \forall g \in \mathcal{G}. \tag{3.11}$$

Constraint (3.8) expresses the single-mindedness of bidders and sellers and forbids partial bundle allocations. Constraint (3.9) ensures that a seller allocates its bundle to at most one bidder, while Constraint (3.10) ensures that each bidder receives the resources in its bundle from a single provider. Finally, Constraint (3.11) ensures that, in the eventuality of a trade, a seller's bundle contains at least the quantity of resources requested by the bidder for each resource type.

3.2.2 PAYMENT RULE

The Vickrey-Clarke-Groves (VCG) payment scheme (Nisan et al., 2007a) discussed in Section 2.2.3 might guarantee truthfulness, but is computationally expensive, and runs a deficit which requires auctioneer subsidies (thus violating the budget balance property). Both issues conflict with the requirements discussed in Section 3.1. As a result, VCG cannot be used for modeling this problem.

Satterthwaite (1993) proposed the k-pricing scheme, which engendered a new class of mechanisms called k-double auctions (k-DA), where the market clearing price is chosen from the interval of possible market clearing prices using a weight $k \in [0, 1]$. This interval is derived from the intersection of the demand and supply curves. The scheme is proven to be worst-case asymptotic optimal (Satterthwaite & Williams, 2002). Although valid for single-unit exchanges, adapting this payment scheme to combinatorial exchanges, while preserving its properties, is not straightforward.

Schnizler et al. (2008) introduced a k-pricing scheme for a combinatorial exchange which, for each trade, essentially distributes the resulting surplus among the trade participants, in a proportion given by the factor $k \in [0, 1]$. In (Schnizler et al., 2008), buyers can trade with multiple sellers at the same time, resulting in complex calculations for the seller payments: for each buyer, the sellers with which it trades need to distribute the surplus among themselves, proportionally to their contribution to the trade. Since my proposed problem formulation constrains each buyer to trade with a single seller, the payment rule is vastly simplified.

Let δ_{ij} denote the surplus resulting from a trade between a buyer i and seller j:

$$\delta_{ij} = b_i - a_j, \forall y_{ij} = 1, i \in \mathcal{U}, j \in \mathcal{P}. \tag{3.12}$$

Then bidder i will receive a discount of a k-th part of this surplus, resulting in payment p_i:

$$p_i = b_i - k\delta_{ij} = (1-k)b_i + ka_j, \forall y_{ij} = 1. \tag{3.13}$$

Similarly, seller j will receive the following payment:

$$p'_j = a_j + (1-k)\delta_{ij} = (1-k)b_i + ka_j, \forall y_{ij} = 1. \tag{3.14}$$

In this work, the trade surplus is equally distributed among the winning buyers and sellers ($k = 0.5$).

3.3 MECHANISM PROPERTIES

This section discusses which economic properties are satisfied, and concludes with a comparison of the proposed model to related work, with respect to all the presented requirements.

Theorem 1 (BB). *The mechanism is budget balanced.*

Proof. In the k-pricing scheme, the trade surplus is distributed among the trade participants. As a result, the payments made by the buyers are equal to the ones received by the sellers. Hence the auctioneer neither makes a profit, nor subsidizes the trade. The detailed proof is presented below.

Formally, budget balance means that the payments made by the (winning) buyers must be equal to the payments received by the (winning) sellers:

$$\sum_{i=1}^{n} p_i x_i = \sum_{j=1}^{m} p'_j \sum_{i=1}^{n} y_{ij}. \tag{3.15}$$

On the basis of Constraint 3.10 and the price definitions in Equation 3.13 and Equation 3.14, Equation 3.15 can be reformulated as follows:

$$\sum_{i=1}^{n} p_i \sum_{j=1}^{m} y_{ij} = \sum_{j=1}^{m} p'_j \sum_{i=1}^{n} y_{ij} \tag{3.16}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} (p_i - p'_j) y_{ij} = 0 \tag{3.17}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{m} \left((1-k)b_i + ka_j - (1-k)b_i - ka_j\right) y_{ij} = 0 \tag{3.18}$$

$$0 = 0 \tag{3.19}$$

□

Theorem 2 (IR). *The mechanism is individually rational.*

Proof. The individual rationality property holds when each agent's utility after taking part in the auction is non-negative, irrespective of it winning the auction or not.

According to the utility definitions in Equation 3.3 and Equation 3.4, an agent's utility is 0 if it does not buy or sell a bundle in the auction, thus non-negative.

Let us analyze the case when a buyer i is allocated a bundle S in the auction. Then, cf. Equation 3.3, its utility is:

$$u_i(S) = v_i(S) - p_i. \tag{3.20}$$

Using the price definition in Equation 3.13 and assuming truthfulness ($v_i(S) = b_i$), the utility can be reformulated as:

$$u_i(S) = b_i - (b_i - k\delta_{ij}) = k(b_i - a_j). \tag{3.21}$$

Moreover, the trade between buyer i and seller j can only occur when the bid value exceeds the ask value ($b_i \geq a_j$), otherwise the objective function would not be maximized—a greater welfare can be obtained when the trade does not occur. Then, it follows that:

$$u_i(S) = k(b_i - a_j) \geq 0. \tag{3.22}$$

That is, the utility of a buyer that won a bundle in the auction is also non-negative.

Similarly, it can be shown that a winning seller's utility is non-negative (using Equation 3.4 and Equation 3.14):

$$u_j'(S) = p_j' - v_j' = a_j + (1-k)\delta_{ij} - a_j = (1-k)(b_i - a_j) \geq 0. \tag{3.23}$$

□

Theorem 3 (EE). *The mechanism is at most asymptotically economically efficient.*

Proof. The economic efficiency property is satisfied only when an optimal algorithm is used to solve the WDP. However, tractability is typically desired for cloud resource allocation, i.e. the algorithm should execute in polynomial time. Heuristic algorithms do not always yield the optimal solution, but it was shown for algorithms such as Greedy (Lehmann et al., 2002) that a solution within a factor of $\sqrt{l}$ of the optimal solution can be guaranteed. □

Theorem 4 (TF). *The mechanism is not truthful or incentive compatible.*

Proof. According to Schnizler et al. (2008), the chosen k-pricing scheme cannot achieve incentive compatibility. This means that agents are not motivated to reveal their true valuations and might try to game the system to their advantage. However, they also showed that non-truthful bidding increases the risk of no allocation because of competition in the market. Thus, in practice, most agents are truthful. □

Table 3.1 gives an overview of the requirements satisfied by the model described in this chapter, and compares it to related work in market-based cloud resource allocation, as well as to industry-standard approaches for resource allocation (all discussed in Section 2.1). To that end, three approaches from academia were selected, which are based on auction mechanisms (Mashayekhy et al., 2014, Nejad et al., 2015a, Toosi et al., 2016b), and the one commercial implementation of a market-based allocation—the Amazon EC2 spot market. For reference, the dominant standard fixed-pricing resource allocation was also included, exemplified by Amazon EC2's on-demand pricing.[3]

The requirements driven by mechanism design are not applicable to EC2 on-demand, since the allocation is not market-based. The EC2 spot pricing scheme is not public, as such one can only speculate on the economic properties: individual rationality is clearly satisfied, since customers do not lose money by participating; otherwise, the other properties cannot be proven.

For the budget balance requirement, weak budget balance was also considered (e.g. for Mashayekhy et al. (2014)). Mashayekhy et al. (2014) allow bidding on a whole cluster offered by a provider, implementing in this way a single-good auction where customers do not have fine-grained control over the resources they need, and cannot request bundles of different resources.

The approach proposed by Toosi et al. (2016b) satisfies resource bundling, but only for resources of the same type, since they implement a single-good multi-unit auction. Moreover, it is not truthful and economically efficient in the absolute sense, but the authors prove that it has high probability of being truthful. They nevertheless choose to optimize provider revenue over social welfare, and achieve near optimal revenue.

[3]`https://aws.amazon.com/de/ec2/pricing/on-demand/`

Table 3.1: Comparison of the proposed model to related work w.r.t. requirements satisfied.

	Requirement	Proposed model	Mashayekhy et al. (2014)	Nejad et al. (2015a)	Toosi et al. (2016b)	EC2 on-demand	EC2 spot
R1	Market-driven pricing	✓	✓	✓	✓		✓
R2	Fine-grained control	✓		✓			
R3	Resource bundling	✓		✓	✓		
R4	Double-sided	✓	✓				
R5	Scalable	✓	✓	✓	✓	✓	✓
R6	Computationally tractable	✓	✓	✓	✓	✓	✓
R7	Individually rational	✓	✓	✓	✓	N/A	✓
R8	Budget balanced	✓	✓		✓	N/A	
R9	Economically efficient					N/A	
R10	Incentive compatible		✓	✓		N/A	

Although this may seem a paradox, all exact science is dominated by the idea of approximation.

Bertrand Russell

4 Approximate Winner Determination

This chapter partially addresses Research Questions 2 and 3, defined in Section 1.1. Namely, I investigate how existing heuristic approaches can be applied to solving the Winner Determination Problem for double combinatorial auctions (as defined in Section 3.2.1), as well as how these approaches can be evaluated and compared in a comprehensive and consistent way.

To that end, an algorithm portfolio was constructed, presented in detail in Section 4.1. Section 4.2 describes a novel, flexible input generator for combinatorial auctions, based on realistic cloud workloads. Finally, in Section 4.3, the portfolio is evaluated using a wide range of test cases and artificially-generated instances. The results and key insights derived from the evaluation are discussed. Work on the algorithm portfolio and its evaluation was published in (Gudu et al., 2018b) and (Gudu et al., 2019). The contributions presented in this section consist of a unified comparative study of heuristic algorithms for double combinatorial auctions, as well as a new approach for generating realistic input data for double combinatorial auctions of cloud resources.

4.1 ALGORITHM PORTFOLIO

I built a portfolio of heuristic algorithms for solving the Winner Determination Problem. This section gives a detailed account of the algorithms, the optimization methods used, as well as my specific contributions. Generally, the algorithms are based on the vast literature on

Table 4.1: Algorithm portfolio: algorithms and families. A reference to related work is provided if the algorithm is based on existing research on algorithms for WDP. If a reference is missing, it means that I applied a general optimization method to combinatorial auctions.

Family	Related work	Algorithms
Optimal		CPLEX
Greedy	Nejad et al. (2015a)	GREEDY1, GREEDY2, GREEDY3, GREEDY1S
Relaxed LP based	Luenberger & Ye (2015)	RLPS
Hill climbing	Zurel & Nisan (2001)	HILL1, HILL1S
	Bertocchi et al. (1995)	HILL2, HILL2S
Simulated annealing		SA, SAS
Stochastic local search	Hoos & Boutilier (2000)	CASANOVA, CASANOVAS

approximate algorithms for combinatorial auctions or generic optimization algorithms, but they are normalized to a single problem formulation, described in Chapter 3. This enables a consistent comparison.

The portfolio consists of 14 algorithms belonging to 6 families of optimization approaches. Table 4.1 provides an overview of the portfolio, and each algorithm family is described in the following sections. The code was open-sourced and is available at (Gudu, 2019b).

4.1.1 OPTIMAL ALGORITHM

The optimal algorithm was included in the portfolio, to be used as a reference throughout the evaluation. The optimal algorithm treats the WDP as a mixed-integer linear program (MILP) (Gonen & Lehmann, 2000), a class of problems which is typically solved using branch-and-cut techniques (Padberg & Rinaldi, 1991). The optimal algorithm is implemented using IBM's commercial software CPLEX (IBM, 2015), hailed as the most performant solver for MILPs.

4.1.2 GREEDY ALGORITHMS

Greedy algorithms are heuristics that make the locally optimal choice at every step, aiming for a globally optimal solution (Cormen et al., 2001). The rich literature of greedy algorithms for the WDP (Nejad et al., 2015b, Lehmann et al., 2002, Pfeiffer & Rothlauf, 2008) hinges on a common idea: the bids (and asks for two-sided auctions) are sorted according to a certain criterion, and then they are greedily allocated as long as there are no conflicts.

The presented implementation use bid and ask densities as sorting criteria, which are defined for buyer i and seller j in Equation 4.1 and Equation 4.2, respectively.

$$d_i = \frac{b_i}{\sqrt{\sum_{g=1}^{l} f_g r_{ig}}}, \forall i \in \mathcal{U} \tag{4.1}$$

$$d'_j = \frac{a_j}{\sqrt{\sum_{g=1}^{l} f'_g s_{jg}}}, \forall j \in \mathcal{P} \tag{4.2}$$

Compared to the average price per unit, the density gives priority to smaller customer requests and was shown to yield higher welfare (Lehmann et al., 2002).

Based on the work of Nejad et al. (2015b), I also employed the concept of relevance factors, or relative weights of resource types, used to express differences in value for the different resource types. In (Gudu et al., 2018b), I proposed using different weights for bids and asks: f_g and f'_g, respectively. I hence extended the three calculation methods proposed by Nejad et al. (2015b) to the case of two-sided auctions as follows.

1. Uniform weights, as a generalization of the one-sided case (Lehmann et al., 2002):

$$f_g = f'_g = 1, \forall g \in \mathcal{G} \tag{4.3}$$

2. Weights based on the absolute scarcity of each resource, defined as the inverse of the supply of the resource on the market (for bids) or the inverse of the demand for the respective resource (for asks):

$$f_g = \frac{1}{\sum_{j=1}^{m} s_{jg}}, \forall g \in \mathcal{G} \tag{4.4}$$

$$f'_g = \frac{1}{\sum_{i=1}^{n} r_{ig}}, \forall g \in \mathcal{G} \tag{4.5}$$

3. Weights based on the relative scarcity of each resource, defined as the difference between demand and supply, normalized by the demand (for bids), and normalized by the supply (for asks):

$$f_g = \frac{\left|\sum_{i=1}^{n} r_{ig} - \sum_{j=1}^{m} s_{jg}\right|}{\sum_{i=1}^{n} r_{ig}}, \forall g \in \mathcal{G} \tag{4.6}$$

$$f_g' = \frac{\left|\sum_{i=1}^{n} r_{ig} - \sum_{j=1}^{m} s_{jg}\right|}{\sum_{j=1}^{m} s_{jg}}, \forall g \in \mathcal{G} \tag{4.7}$$

In this work, the greedy algorithms for each method of calculating the relevance factors will be referred to as: GREEDY1, GREEDY2, and GREEDY3. The pseudocode is shown in Algorithm 1.

Additionally, due to the two-sided aspect, a greedy algorithm that gives priority to sellers was implemented, which moves through the list of bids until one that satisfies the considered ask is found. In Algorithm 1, this is achieved by simply swapping lines 10 and 11. Throughout this work, the naming convention is to add an "'-s" prefix for algorithms that prioritize asks, in this case GREEDY1S. The pseudocode is included in Appendix B (Algorithm 8).

Algorithm 1 Greedy algorithms for different calculation methods of relevance factors.

```
1:  function GREEDYX($n, m, l, b, r, a, s$)                 ▷ $X \in \{1, 2, 3\}$
2:      compute $f_g$, $f_g'$, $\forall g \in \mathcal{G}$ with method X          ▷ relevance factors
3:      compute $d_i$, $\forall i \in \mathcal{U}$ and $d_j'$, $\forall j \in \mathcal{P}$      ▷ bid and ask densities
4:      sort bids descendingly by $d$
5:      sort asks ascendingly by $d'$
6:      $i \leftarrow 1; j \leftarrow 1$
7:      while $i \leq n$ and $j \leq m$ do
8:          if $r_{ig} \leq s_{jg}, \forall g \in \mathcal{G}$ and $b_i \geq a_j$ then   ▷ if ask j can satisfy bid i
9:              $x_i \leftarrow 1; y_{ij} \leftarrow 1$                 ▷ allocate resources offered by
                                                          seller j to bidder i
10:             $i \leftarrow i + 1$                            ▷ move to next bidder
11:         $j \leftarrow j + 1$                                ▷ move to next seller
12:     return $(x, y)$
```

4.1.3 RELAXED LINEAR PROGRAM-BASED

Relaxing the integrality constraints of the decision variables x_i and y_{ij} transforms the integer program in Equation 3.7 into a linear program that can be solved faster—weakly polynomial time when using interior point methods, or exponential time for simplex methods (Luenberger & Ye, 2015)—and that provides an upper bound for social welfare. An algorithm based on the relaxed linear problem was added to the portfolio, similar to the work of Pfeiffer & Rothlauf (2008), but adapted to the two-sided case, called Relaxed LP Solution (RLPS). The algorithm sorts the bids and asks descendingly by their continuous decision variables, which in this case are $x_i^*, \forall i \in \mathcal{U}$ and $\sum_{i=1}^{n} y_{ij}^*, \forall j \in \mathcal{P}$, respectively, and then applies a greedy algorithm similar to GREEDY1. x_i^* and y_{ij}^* denote the solutions of the relaxed linear program, computed using the CPLEX library (IBM, 2015).

4.1.4 HILL CLIMBING ALGORITHMS

Hill climbing algorithms (Russell & Norvig, 2009, 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 new solution is better. The algorithm stops when it finds a (local) optimum. This is depicted in Algorithm 2, where the initial solution is determined using a greedy algorithm.

Algorithm 2 Hill climbing.

```
1: function HILL($n, m, l, b, r, a, s$)
2:     $(x, y) \leftarrow$ GREEDY1($n, m, l, b, r, a, s$)          ▷ generate initial solution
3:     while solution improves do
4:         while solution has unexplored neighbors do
5:             $(x', y') \leftarrow$ NEIGHBOR($x, y$)                ▷ get neighboring solution
6:             if welfare($x', y'$) $>$ welfare($x, y$) then        ▷ if new solution is better
7:                 $(x, y) \leftarrow (x', y')$                       ▷ move to new solution
8:                 break
9:     return $(x, y)$
```

I devised two different methods of exploring the neighborhood of a solution in the solution space.

Firstly, as shown in Algorithm 3, a neighboring solution is found by changing the ordering of bids or asks, and applying a greedy algorithm onto this ordering. Based on an idea by Zurel & Nisan (2001), an unallocated bid is moved to the beginning of the bid list to generate a neighboring solution, starting with the first bid after the critical bid (or first unallocated bid in the ordered list) and then going through the sorted list. This algorithm is called HILL1. The version that prioritizes sellers, HILL1S, changes the ordering of asks to explore the neighborhood of a solution and uses GREEDY1S.

A more generic hill climbing algorithm, where neighboring solutions are generated by toggling the x_i variables, as proposed by Bertocchi et al. (1995) for the multiknapsack problem, is depicted in Algorithm 4. This means that a randomly selected bid is allocated. The corresponding y_{ij} variable is adjusted by greedily searching through the list of unallocated asks, already sorted by density. The neighborhood exploration stops after a number of n unsuccessful attempts of increasing the welfare. This algorithm is called HILL2.

Algorithm 3 Function that returns the neighbor in the solution space of a given solution, by changing the bid ordering. Used in HILL1.

```
1: function NEIGHBOR1($x, y$)
2:     move bid $i$ to beginning of list                 ▷ $i \leftarrow$ critical $i + 1, n$
3:     return GREEDY1($n, m, l, b, r, a, s$)[6 : 12]     ▷ apply on new ordering
                                                          lines 6–12 in Algorithm 1
```

Algorithm 4 Function that returns the neighbor in the solution space of a given solution, by toggling a random x_i. Used in HILL2.

```
1: function NEIGHBOR2($x, y$)
2:     $i \leftarrow$ random$(1, n)$                                   ▷ randomly select bid
3:     if $x_i = 0$ then                                               ▷ if bid i not already allocated
4:         for $j \leftarrow 1, m$ do                                  ▷ greedy-like search for ask
5:             if $y_{qj} = 0, \forall q \in U$ then                   ▷ if ask j not allocated
6:                 if $r_{ig} \le s_{jg}, \forall g \in G$ and $b_i \ge a_j$ then   ▷ if ask j can satisfy bid i
7:                     $x_i \leftarrow 1; y_{ij} \leftarrow 1$         ▷ match bid i and ask j
8:                     break                                           ▷ stop—match found
9:     return $(x, y)$                                                 ▷ return neighboring solution
```

A hill climbing algorithm that prioritizes sellers, HILL2S, was also implemented. Algorithm HILL2S explores the neighborhood of a solution by allocating a random ask j, and uses GREEDY1S for allocation.

4.1.5 SIMULATED ANNEALING ALGORITHMS

Simulated annealing (Kirkpatrick et al., 1983) is an optimization method that tries to mitigate the issues of gradient-based methods (such as hill climbing) of being stuck in local optima. To that end, it accepts worse solutions during the search process, with a probability that decreases over time.

Inspired by annealing of solids, physical terms are used to describe the optimization process (Aarts et al., 2014): the system consists of discrete states (feasible solutions), each with its own energy level (cost). A system can transition from one state to the other (move to a neighboring solution) based on an acceptance criteria. The annealing process starts at a certain temperature (where temperature denotes a control parameter) and is gradually cooled until the system is frozen (no change in cost for a certain number of temperature values). Guided by Aarts et al. (2014), I discuss below the four key parameters of a simulated annealing algorithm: cost function, acceptance criteria, cooling schedule, and neighborhood structure.

COST FUNCTION

This is a measure of solution quality. In this case, the cost is simply the social welfare. Since it is calculated in every iteration of the algorithm, it is more efficient to use delta evaluation: only computing the cost difference between the current and neighboring solution.

ACCEPTANCE CRITERIA

State transitions are probabilistic processes: the system moves towards lower-energy states, but at high enough temperatures it might also transition to higher-energy states. This is expressed by the following acceptance criteria: the system always accepts a lower-energy state, and, with a probability that decreases with the temperature, also accepts a higher-energy state. Consequently, the acceptance probability is computed using the formula in Equation 4.8, where (x, y) is the current solution in the search space, while (x', y') is the explored neighboring solution. As a result, better solutions are always accepted (the probability is always higher or equal to 1), while for worse solutions, the probability that they are accepted is in $[0, 1]$ but decreases with the temperature variable T, ultimately allowing the search to converge.

$$ap = \mathrm{e}^{(\mathrm{welfare}(x',y') - \mathrm{welfare}(x,y))/T} \tag{4.8}$$

COOLING SCHEDULE

The control parameter (temperature) changes according to the cooling schedule, which thus specifies the sequence of possible values for the temperature. More specifically, it is defined by a starting and a final temperature, a decrement function, and a finite number of iterations to be performed at each temperature. In this thesis, a static schedule is employed, where the parameters are fixed before the algorithm execution.

Choosing a suitable starting temperature is not an easy task: a low temperature would lead to rarely accepting worse solutions, effectively behaving like a hill climbing algorithm, while too high temperatures will transform the algorithm into a random search, thus inherently inefficient until the temperature is low enough. Typically, a good starting temperature is the maximum possible cost difference between two neighboring solutions. Since its computation for the WDP is time consuming, I estimate this value using the formula in Equation 4.9: the highest welfare increase can be obtained by matching the highest bid to the lowest ask. This does not consider if the allocation is feasible, or if the requests are already allocated, but it provides a reasonable upper bound quickly.

$$T_{\max} = \max_{i \in \mathcal{U}} b_i - \min_{j \in \mathcal{P}} a_j \tag{4.9}$$

The temperature decreases at a constant rate $\alpha \in (0, 1)$, according to the geometric function in Equation 4.10. Values for α that have shown to yield good results lie between 0.8 and 0.99. Higher values lead to higher solution quality, but slow convergence.

$$T_{k+1} = \alpha \cdot T_k, k = 0, 1, \ldots \tag{4.10}$$

In the implementation included in the proposed portfolio, a constant number of iterations N_{it} is executed at each temperature value. This has to be balanced against the decrement α: an appropriate number of iterations allows the system to stabilize at each temperature. The following values were used with good results: $\alpha = 0.9, N_{it} = 100$.

The search stops when the system is frozen—the temperature reached zero (or a suitably low value), or no state transitions happened for a while. I set the final temperature to $T_{\min} = 10^{-5}$, and consider that a system is frozen if the solution has not changed for three consecutive temperature values.

NEIGHBORHOOD STRUCTURE

Similar to HILL2, a neighboring solution is generated by toggling a randomly selected x_i variable (cf. function NEIGHBOR2 in Algorithm 4). The only difference is that an already allocated bid can be removed from the solution, since a simulated annealing algorithm can accept worse solutions. The modified function, NEIGHBOR_SA, is shown in Algorithm 11 in Appendix B.

Two algorithms were implemented, SA and SAS, by giving priority to bidders and sellers, respectively, and using the appropriate greedy algorithm. The SA algorithm is shown in Algorithm 5.

Algorithm 5 Simulated annealing.

```
1: function SA($n, m, l, b, r, a, s$)
2:     $(x, y) \leftarrow$GREEDY1$(n, m, l, b, r, a, s)$          ▷ generate initial solution
3:     $T_{\max} = \max_{i \in \mathcal{U}} b_i - \min_{j \in \mathcal{P}} a_j$          ▷ initialize temperature
4:     while $T > T_{\min}$ and system not frozen do          ▷ decreasing temperature
5:         for $it \leftarrow 1, N_{it}$ do          ▷ fixed number of iterations
6:             $(x', y') \leftarrow$NEIGHBOR_SA$(x, y)$          ▷ get neighboring solution
7:             $ap = \mathrm{e}^{(\mathrm{welfare}(x', y') - \mathrm{welfare}(x, y))/T}$          ▷ acceptance probability
8:             if $ap > \mathrm{rand}(0, 1)$ then          ▷ with probability ap...
9:                 $(x, y) \leftarrow (x', y')$          ▷ ...move to new solution
10:        $T \leftarrow \alpha T$          ▷ fixed rate $\alpha = 0.9$
11:    return $(x, y)$
```

4.1.6 CASANOVA ALGORITHMS

Two stochastic local search algorithms were also included in the portfolio, based on the Casanova algorithm introduced by Hoos & Boutilier (2000). Similar to SA, CASANOVA uses randomization to escape from local optima. However, it differs substantially in the strategies used for neighborhood exploration and randomization. What is more, the algorithm uses random restarts, which were shown to mitigate the large performance variability of stochastic combinatorial search methods (Gomes et al., 1997).

The approach proposed by Hoos & Boutilier (2000), adapted to my problem formulation, is described in the following. The pseudocode is shown in Algorithm 6.

The algorithm is based on scoring each state by the revenue per good that it brings. Since a neighbor in the search space can be reached by adding a bid to the solution, the scoring function can be defined (cf. Equation 4.11) as the bid value normalized by the total quantity of resources in the requested bundle. Note that for the proposed double-sided auction, this is a rough approximation, since the corresponding ask is not included in the score. It nevertheless allows us to rank and process bids based on their score.

$$score(i) = \frac{b_i}{\sum_{g=1}^{l} r_{ig}}, \forall i \in \mathcal{U} \tag{4.11}$$

The search starts with an empty allocation and moves to a neighboring solution in the search space by allocating a single bid: with a walk probability wp, a random bid is chosen; with a probability of $1 - wp$, a bid is selected from the list of bids sorted by score—either the

Algorithm 6 Casanova algorithm, based on stochastic local search.

```
1:  function CASANOVA($n, m, l, b, r, a, s$)
2:      for $try \leftarrow 1, maxTries$ do                              ▷ restart search try times
3:          $(x, y) \leftarrow 0$                                        ▷ empty allocation
4:          sort bids descendingly by score
5:          sort asks ascendingly by density
6:          for $step \leftarrow 1, maxSteps$ do
7:              if $wp > \text{rand}(0, 1)$ then                         ▷ with walk probability wp
8:                  $(x, y) \leftarrow$INSERT(random unallocated bid $i, x, y$)
9:              else if age(first unallocated bid) $>$ age(second unallocated bid) then
10:                 $(x, y) \leftarrow$INSERT(first unallocated bid, $x, y$)
11:             else if $np > \text{rand}(0, 1)$ then                    ▷ with novelty probability np
12:                 $(x, y) \leftarrow$INSERT(second unallocated bid, $x, y$)
13:             else
14:                 $(x, y) \leftarrow$INSERT(first unallocated bid, $x, y$)
15:             if $step > \theta_r$ and no improvement in last $\theta_r/2$ steps then
16:                 break                                                ▷ soft restart
17:     return best $(x, y)$ found
```

highest or second highest ranked bid. The highest-ranked bid is selected if its age is larger than that of the second highest-ranked bid; otherwise, I select the second highest-ranked bid with a novelty probability *np* and the highest one with a probability of $1 - np$. The age of a bid is defined as the number of steps since it was last selected in the current try.

The search is performed for at most *maxSteps* iterations and is restarted *maxTries* times. The best solution from all runs is chosen. The soft restart strategy proposed by Hoos & Boutilier (2000) is used, which restarts the search if at least θ_r steps have been performed since the last restart, but no improvement occurred within the last $\theta_r/2$ steps.

The *maxSteps* and θ_r variables are problem-size dependent, and need to ensure that the search space is sufficiently explored. Thus, $maxSteps = n$ and $\theta_r = n/4$ are used. The other parameters are fixed to values shown by Hoos & Boutilier (2000) to yield good results: $wp = 0.15, np = 0.5, maxTries = 10$.

Algorithm 7 shows how a bid is added to a partial solution: a greedy-like algorithm is used to find an unallocated ask that can satisfy it. If no match is found, the search continues among the already allocated asks: the bid can replace an already allocated bid only if it improves the social welfare.

An algorithm that prioritizes sellers, CASANOVAS, was also implemented, with the follow-

Algorithm 7 Neighbor function for Casanova algorithm. The function adds an unallocated bid and best matching ask to the current solution.

```
1:  function INSERT($i, x, y$)
2:      for $j \leftarrow 1, m$ do                                        ▷ greedy-like search for ask
3:          if $y_{qj} = 0, \forall q \in \mathcal{U}$ then                            ▷ j not allocated
4:              if $r_{ig} \leq s_{jg}, \forall g \in \mathcal{G}$ and $b_i \geq a_j$ then            ▷ if ask j can satisfy bid i
5:                  $x_i \leftarrow 1; y_{ij} \leftarrow 1$                                ▷ match bid i and ask j
6:                  reset age($i$)
7:                  return $(x, y)$                                    ▷ match found
8:      for $j \leftarrow 1, m$ do                                        ▷ no match, restart search
9:          if $\exists q \in \mathcal{U}, y_{qj} = 1$ then                            ▷ j already allocated
10:             if $r_{ig} \leq s_{jg}, \forall g \in \mathcal{G}$ and $b_i \geq a_j$ then            ▷ if ask j can satisfy bid i
11:                 if $b_i > b_q$ then                                ▷ if bid i improves allocation
12:                     $x_i \leftarrow 1; y_{ij} \leftarrow 1$                            ▷ match bid i and ask j
13:                     $x_q \leftarrow 0; y_{qj} \leftarrow 0$                            ▷ undo allocation of bid q
14:                     reset age($i$)
15:                     return $(x, y)$                                ▷ match found
16:     return $(x, y)$                                                ▷ no match found
```

ing differences: asks are sorted by their score (the ask value normalized by the total quantity of resources in the offered bundle), while bids are sorted by density; the insert function takes an ask *j* and tries to find a matching bid *i*.

4.2 INPUT DATA GENERATION

Although most research on combinatorial auctions uses artificial data for evaluation (Sandholm, 2002, Fujishima et al., 1999, De Vries & Vohra, 2003), the generation approaches are typically simplistic, and not representative of real data. The Combinatorial Auctions Test Suite (CATS) (Leyton-Brown et al., 2000) is the only work that aims to create a comprehensive tool for generating artificial data for combinatorial auctions, and to provide support for more realistic distributions—by modeling complementarity and substitutability of goods using graphs. Nevertheless, this work is not sufficient to generate input data adequate for the problem model proposed in this thesis.

This is because, to my knowledge, there is no work that addresses all of the following aspects:

- *multi-unit and multi-good aspect*: generating bids for bundles of resources (combinatorial bids), but allowing multiple units for each resource type.
- *two-sided aspect*: generating both bids and asks, while considering the interactions or dependencies between buyers and sellers.
- *cloud aspect*: generating data for cloud resource auctions, by creating bundles that are similar or representative to some extent of real-world cloud workloads.

A new approach for input generation that could address these issues was necessary. This approach should enable flexible input generation, be easily parameterizable, generate realistic cloud bundles, and provide a way to model the dependency between the two sides when it comes to generating bid and ask values.

Thus, I propose an improved way of generating artificial data for multi-good multi-unit double combinatorial auctions for cloud resources. The developed input generation tool CAGE (Combinatorial Auctions input GEnerator) (Gudu, 2019a) is introduced in the following.

Given the desired amount of bids n, the desired amount of asks m, as well as the number of possible resource types l, generating an auction instance means generating all the bids and asks, expressed as:

$$\text{bids} : (\langle r_{i1}, \ldots, r_{il} \rangle, b_i), \forall i \leq n \tag{4.12}$$

$$\text{asks} : (\langle s_{j1}, \ldots, s_{jl} \rangle, a_j), \forall j \leq m \tag{4.13}$$

The input generation is done in two steps:

1. generating the vectors of resource quantities, through the realistic cloud bundle generation approach proposed in Section 4.2.1, and
2. generating the bid and ask values, by using the previously generated bundles, with the pricing model and valuation generation approach proposed in Section 4.2.2.

4.2.1 BUNDLE GENERATION

The goal is to create artificial bundles resembling the characteristics of real bundles of computing resources, while maintaining flexibility by allowing for parameterizable ranges and granularities of resources. The bundle generation presented in this section is the result of joint work with Peter Krauß (publication in preparation).

TRACES

It must first be defined what *realistic* means. Since there is no public dataset of resource requests submitted to cloud providers, public cluster traces are used to extract information on the kind of jobs that *might* typically run inside a datacenter. It cannot be proven that these traces are representative of all workloads running in the cloud nowadays, but they are real workloads of specific use cases, thus ensuring that realistic bundles are generated. Another plus is the fact that they are publicly available and often used in cloud research, paving the way towards standardization and comparability of research in this field.

Two sources of traces were selected: the Google cluster data (Wilkes, 2011, Reiss et al., 2011), containing several traces of Google cluster management software and systems, and the fastStorage trace (Shen et al., 2015, Bitbrains IT Services Inc., 2014), provided by the Bitbrains IT Services provider, specializing in business computation for enterprises (such as financial reporting). The latter dataset is hosted at the Grid Workloads Archive[1].

These traces measure various performance metrics over time (CPU and memory usage, disk and network I/O performance), for all the jobs running in a distributed datacenter in a predefined period of time (e.g. 29 days for the Google traces). The values of these performance metrics are normalized for anonymization. Next, the raw traces are processed to extract the set of jobs that were run in the datacenter, each with its total resource usage.

At this point, I make the following assumptions:

1. Each job and its resource usage can be mapped to a submitted request for the same amount of cloud resources. Then the full cluster workload can be mapped to a set of bids.

2. The relevant characteristics for cloud workloads are the relations between the resources and their distribution throughout the whole domain space.

Regarding the second assumption, let us consider an example: if the resources are CPU and memory, then it is relevant how they are combined in bundles—e.g. bundles with 8 CPUs and 8 GB of memory exist, but bundles with 100 CPUs and 1 GB of memory do not—, and how often they appear in a workload consisting of multiple bundles—e.g. the (8 CPUs, 8 GB) bundles are 10 times more common than (64 CPUs, 64 GB) bundles.

[1] `http://gwa.ewi.tudelft.nl/`

APPROACH

An approach is proposed, that can create a model of these relevant characteristics for any real workload, which can then be used to generate artificial data that resembles the real workload, but allowing user-defined parameters for the generated bundles (e.g. number of bundles, resource types and amount ranges, quantization).

The advantages of modelling instead of directly using the real data concern size constraints, and flexibility and variety demands. Real datasets can be hard to handle due to their large sizes, as opposed to models, which are small, while capturing most of the characteristics of the source data. Models are therefore increasingly more portable, since once they are created, they are independent from the data source and directly usable to generate artificial data of any size. Moreover, models enable parameterized generation, by allowing e.g., different quantizations of resources, different amounts of bundles, giving the possibility to generate as many different datasets as necessary. Real data, however, is limited with respect to the number of available datasets and variety within datasets, and can be noisy.

Figure 4.1 describes the bundle generation approach.

First, a model is created from a given data source. The data model is a multi-dimensional probability density function, where each dimension represents a resource, while each point in this space represents the probability of generating a bundle with the resource composition given by its coordinates. The model is created in two steps: discretization and interpolation.

The discretization consists of creating a multi-dimensional histogram of the real data, where the resources are binned in a configurable number of bins. The binning directly impacts the model size and quality: the more bins are used, the more closely the model resembles the real data; however, this also increases the model size. An example 2-dimensional histogram created from one of the Google traces is shown in Figure 4.2: only the CPU usage and memory resources are considered, with 16 regular bins in each dimension. Irregular bins can also be used, for example to increase the granularity only in a certain region on the resource domain. Even more, by applying clustering on the real data, one can determine a binning

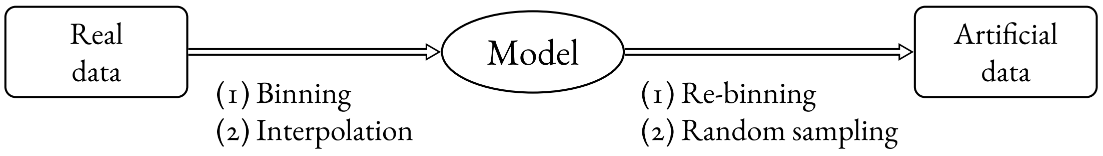


Figure 4.1: Approach for artificially generating realistic cloud bundles.

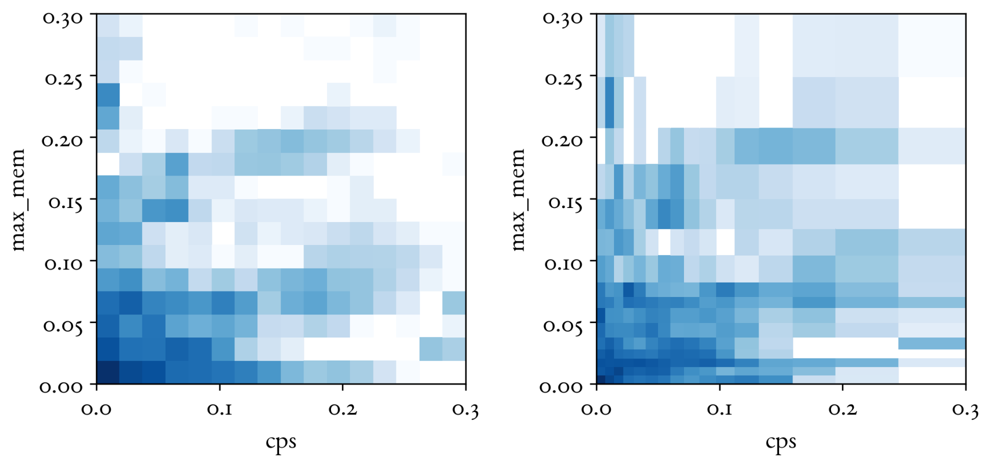


Figure 4.2: Two example histograms used to create a model, with different binnings: regular (left) and clustered (right). The two dimensional space denotes the two resource types: maximum memory used and average cycles per second. Both binnings use 16 bins in each dimension.

that tries to capture the data distribution and is thus finer where the clusters are located (cf. Figure 4.2).

In the interpolation step, the histogram values are used, together with the bin centers as support points, to create the probability density function: a function defined on the full domain, that returns the probability of any bundle in this domain.

Finally, once a model is created, artificial data can be generated. The following parameters can be changed: the number and type of bins in each dimension, the number of bundles; the domain can also be re-scaled to any desired range. The data generation is done in two steps: first, re-binning from the model bins to the desired bins, and computing the probability of each possible bundle; second, randomly sampling the desired amount of bundles from the possible bundles, according to the computed probabilities.

Figure 4.3 shows an example of artificial data, generated with 64 bins in each dimension, from a model based on clustered binning with 16 bins in each dimension. The real data binned in the same way is also showed for comparison. It can be seen that the generated data maintains the characteristics of the real data, while smoothing out irregular data.

Finally, in addition to the real workloads, two flavors of artificial models are included, to aid in the evaluation of bundle generation. The first model is called UNIFORM, and assumes a uniform probability of bundles throughout the resource space: any combination of resources is equally possible. The second model, HOTSPOTS, assumes only a specific set of bundles are possible, with equal probabilities, while other resource combinations are not allowed. Due to

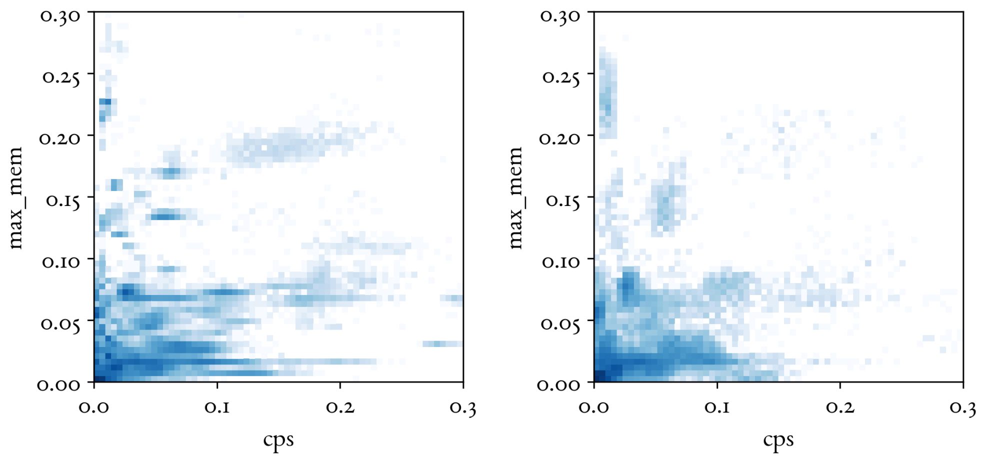


Figure 4.3: Real (left) vs artificial (right) data, both binned with 64x64 regular bins, generated from a model of only 16x16 clustered bins.

their simplicity, these models can be used to quickly generate bundles in any resource domain space. These models are appropriate to generate cloud offers (for example, the hotspots in the resource space can be mapped to the predefined VM instance types offered by Amazon).

4.2.2 VALUATION GENERATION

Applying the bundle generation method, with the desired models and parameters, outputs the vectors of resource quantities for all the bids and asks. To complete the generation of an auction instance—a set of bids and asks—, the bid values $(b_i)_{i \leq n}$ and ask values $(a_j)_{j \leq m}$ are left to generate.

First, let us consider each resource type individually. To describe how the price of a resource changes with the number of units of said resource, I define a resource pricing model. This can be any function $\Upsilon : \mathbb{R} \to \mathbb{R}$, that takes a quantity of resources of type $g \in \mathcal{G}$, and assigns a value to this quantity.

A simple and straightforward model is a linear pricing model: the value of a resource increases linearly with quantity. A fixed cost can also be included, for example to model the cost of provisioning a certain type of resource, which does not change with quantity. In this thesis, linear pricing models are used for all the resources, with different parameters. Nevertheless, the input generator can be easily extended with more complex, non-linear pricing models.

Moving from one type of resources to l types, let us define the linear pricing model Υ on the l-dimensional resource domain, returning an l-dimensional value of resources:

$$\Upsilon : \mathbb{R}^l \rightarrow \mathbb{R}^l, \quad \Upsilon(q) = slope \cdot q + fixed, \tag{4.14}$$

where q is a vector of resource quantities, the slope vector contains the unit prices of each resource type, and fixed is a vector of fixed costs for provisioning each type of resources (it can also be zero).

Given base values for resources (through their pricing models), I now wish to encode the difference in valuations between the two sides, bidders and sellers: in order to have a chance at receiving a desired bundle, a bidder will likely bid a higher value than what providers offer. A fixed value *dist* is introduced that expresses the distance between the pricing models of bidders and sellers (relative to the slope). Then the bidder and seller pricing models have different slopes, each at half of a given distance *dist* from the base slope. This results in pricing models Υ_b and Υ_a for bidders and sellers, where the quantity vectors are r and s:

$$slope_b = slope \cdot (1 + \frac{dist}{2}) \implies \Upsilon_b(r) = slope_b \cdot r + fixed \tag{4.15}$$

$$slope_a = slope \cdot (1 - \frac{dist}{2}) \implies \Upsilon_a(s) = slope_a \cdot s + fixed \tag{4.16}$$

Additive valuations for bids and asks are assumed when combining resource types. This means that, to compute bid and ask values, the values of all the resources in the bundle—according to the pricing model—are added up. This can be expressed formally as the L_1 norm of the Υ vectors.

Finally, to simulate the fact that auction participants have different preferences and risk attitudes, and for example they might be willing to pay more for some resources than other bidders, random perturbations are added around the computed values, according to a normal distribution with a given standard deviation (relative to the mean value, e.g. 5% around the mean).

Let σ_b and σ_a be the user-defined standard deviations for biders and sellers, respectively. Then the bid and ask values are calculated as follows:

$$b = |\Upsilon_b(r)|_1(1 \pm \sigma_b) \tag{4.17}$$

$$a = |\Upsilon_a(s)|_1(1 \pm \sigma_a) \tag{4.18}$$

The user-defined parameters $dist, \sigma_b, \sigma_a$ determine the amount of overlap between the bid and ask values, and indirectly the likelihood of finding matches between bids and asks.

4.3 EVALUATION

I performed an extensive and systematic evaluation of the algorithm portfolio, and present the results in the rest of this section.

Since real-world auction data for cloud resources is not available, artificial data was used in all the tests, a common practice in the area of combinatorial auctions (Leyton-Brown et al., 2000, Sandholm, 2002, De Vries & Vohra, 2003). However, in contrast to other work, I ensure that the data are realistic, through the proposed approach of bundle generation based on extracting relevant characteristics from real cloud traces. The synthetic aspect affords a coverage of a wider scope of scenarios, while providing full control over the input.

Using CAGE—the novel tool for generating artificial combinatorial auction data introduced in Section 4.2—, two different datasets for evaluating the algorithm portfolio were created:

D1: the bundles in this dataset have 3 resource types (e.g., CPU cycles, memory, disk) and they were generated using four different models based on the Google cluster traces, as well as the artificial models UNIFORM and HOTSPOTS. The cost model parameters were varied to support different spreads of agent valuations around a common value, as well as different degrees of overlap between bidder and seller valuations.

D2: the bundles in this dataset have 4 resource types (e.g. CPU usage, memory, disk I/O, network I/O) and are based on the Bitbrains workload traces, as well as the UNIFORM and HOTSPOTS models. The other parameters are the same as for dataset D1.

Table 4.2 summarizes the pricing model parameters used to generate the auction instances in D1 and D2. The number of bids and asks were fixed ($n = m = 1000$), as well as the binning type (regular) and binning domain ($[0, 128]$ in each dimension) for both bid and ask bundles.

4.3.1 AVERAGE CASE

First, the average behavior of the algorithm portfolio was studied on each dataset. The execution time and the computed welfare were recorded, and the results are depicted in Figure 4.4 and Figure 4.5.

Table 4.2: Resource pricing model parameters used in generating datasets D1 and D2.

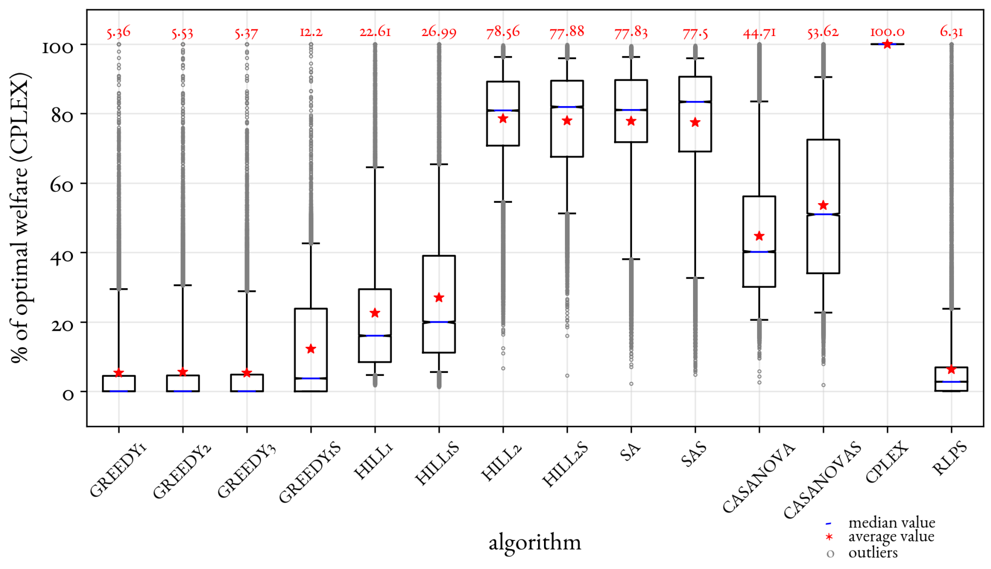
Parameter	**D1**	**D2**
slope	$\{[1,1,1],$ $[1,0.9,0.5],$ $[1.1,1,0.9]\}$	$\{[1,1,1,1],$ $[1,0.9,0.5,1],$ $[1.1,1,0.9,1]\}$
fixed	$\{[0,0,0],$ $[0,0,0.1]\}$	$\{[0,0,0,0],$ $[0,0,0.1,0]\}$
dist	$\{0,0.1,0.5\}$	$\{0,0.1,0.5\}$
σ_a	$\{0.05,0.1,0.25\}$	$\{0.05,0.1,0.25\}$
σ_b	$\{0.05,0.1,0.25\}$	$\{0.05,0.1,0.25\}$
# instances	46,656	8440

Note that during the analysis of results, to refer to both variations of an algorithm (i.e. with bidder and seller priority), the term <ALGO>(S) will be used, where <ALGO> can be for example HILL2. Then, HILL2(S) denotes HILL2 and HILL2S in a shorthand manner.

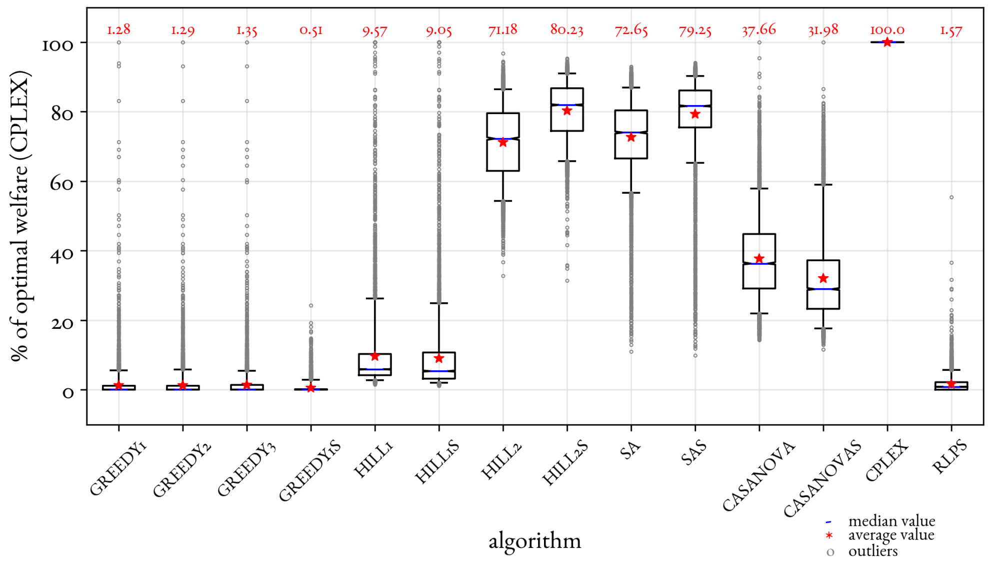
Figure 4.4 shows that there are large differences between the algorithms in terms of computed welfare (normalized to the optimal welfare). The greedy algorithms generally yield a low welfare (on average, 5−12% on D1, and only ≈ 1.3% on D2), but have a large variation over the dataset, sometimes reaching near-optimal solutions. Algorithms HILL2(S) and SA(S) compute the highest welfare in the portfolio, with averages of 71-80% of the optimal welfare. The remaining algorithms, CASANOVA(S) and HILL1(S) give inconclusive results, having the largest spread of welfare value of all algorithms. On D1, the CASANOVA(S) algorithms yield, on average, half of the optimal welfare, and the HILL1(S) algorithms only about a quarter.

This behavior is consistent over the two datasets, with one notable exception: the algorithms based on optimizing the ordering of bids and asks, onto which a greedy allocation can be applied (all from the greedy family, RLPS, CASANOVA(S), HILL1(S)) yield up to one order of magnitude lower welfare on D2 than on D1. This is due to the fact that instances in D2 have more resource types ($l = 4$), increasing the complexity of finding a match, since for an ask to be able to satisfy a bid, all the resource types in the ask should be in a quantity higher than the ones requested in the bid. However, ordering the bid and ask lists typically relies on average values over a bundle, such as density or price per unit.

The immediate conclusion of the results in Figure 4.4 is that the HILL2(S) or SA(S) algorithms are the best in the portfolio, but the execution time analysis in Figure 4.5 paints a different picture.

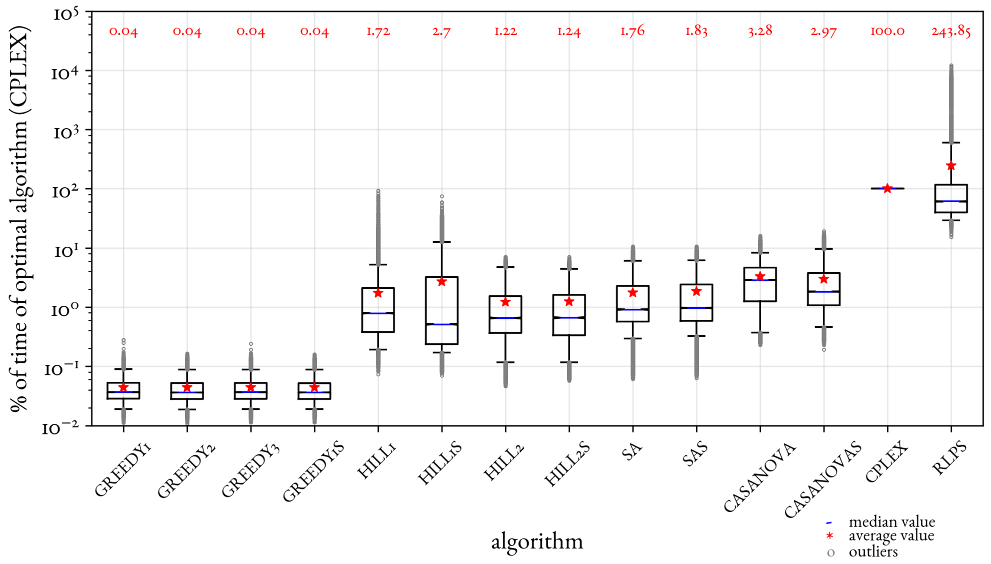


(a) Dataset D1: Social welfare, normalized by the optimal welfare computed with CPLEX

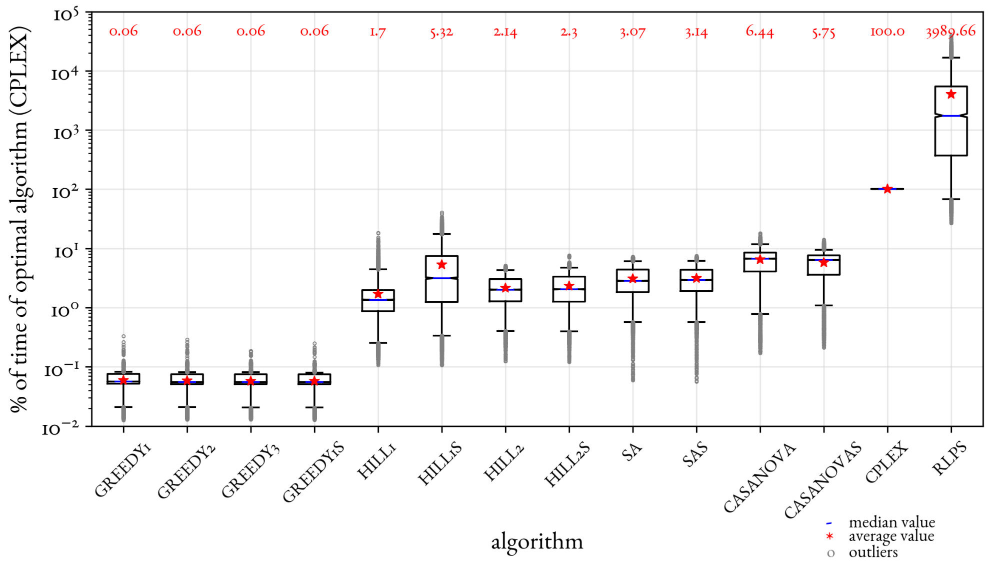


(b) Dataset D2: Social welfare, normalized by the optimal welfare computed with CPLEX

Figure 4.4: Social welfare results for the two datasets, normalized by the optimal welfare computed with the optimal algorithm CPLEX. 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) Dataset D1: Execution time, normalized by the execution time of the optimal algorithm CPLEX



(b) Dataset D2: Execution time, normalized by the execution time of the optimal algorithm CPLEX

Figure 4.5: Execution time results for the two datasets, normalized by the execution time of the optimal algorithm CPLEX. 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 for better readability.

Fig. 4.5 shows that the greedy algorithms are at least 2 orders of magnitude faster than any other algorithm in the portfolio. Furthermore, while HILL2(S) and SA(S) all have an average time of $\approx$ 1-3% of CPLEX's time, they scale differently with the problem size—HILL2(S) have a time complexity of $\mathcal{O}(n^2)$, vs. $\mathcal{O}(n \log n)$ for SA(S). Also note that on D2, HILL1 is, on average, faster than all the other algorithms (with the exception of the greedy family), and can therefore not be ruled out as a viable option for solving combinatorial auctions when the speed is an important factor.

Nevertheless, the average case behavior confirms my hypothesis that algorithm performance and solution quality are highly dependent on the input, and thus there is no clear portfolio winner, especially when considering both execution time and welfare. Each algorithm has its strengths and weaknesses. I explore this further in Section 4.3.3.

The only exception is the RLPS algorithm, which is often slower than CPLEX (due to the use of simplex methods for solving the linear relaxation problem), and yields low welfare (6.3% of the optimal welfare on D1, and 1.5% on D2) with a large spread. As a result, is was excluded from the portfolio.

4.3.2 EFFECT OF RANDOMIZATION

The CASANOVA(S), SA(S), and HILL2(S) algorithms are stochastic, causing them to yield different results for different runs on the same input. For a reliable usage, it is desirable to minimize welfare variations between runs. Thus, the robustness of the four algorithms with respect to randomness was evaluated. The D1 dataset was used, and 100 runs were performed on each problem instance.

Figure 4.6 shows the percentual variations for each algorithm with respect to the mean of the 100 runs. This normalization was performed in order to have a comparative overview over all the instances and algorithms. Over all the evaluated algorithms, it can be observed that the lower and upper quartile are inside the interval [-4.28%, 3.46%] with respect to the mean, with 5% to 95% of the data in [-38%,31%]. However, outliers can be seen even beyond $\pm$ 100%.

The highest variations were observed for the CASANOVA(S) algorithms, since they work with an improved score-based ordering of bids and asks, and the computed welfare is very sensitive to ordering changes (as Figure 4.4 shows).

The more interesting result concerns the difference between HILL2(S) and SA(S) algorithms. Even though SA(S) have a slightly more narrow range of values that contains the 50% of the data (the boxes in Figure 4.6 extend from the lower to the upper quartile), and even

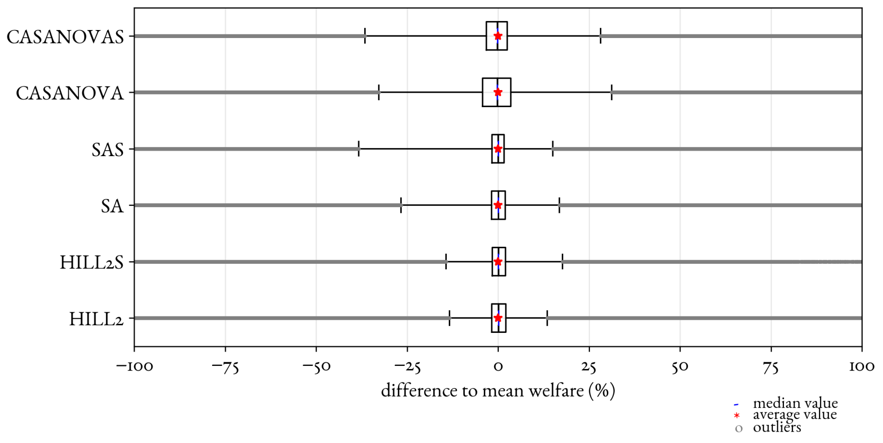


Figure 4.6: Variation in social welfare for the stochastic algorithms: results for dataset D1, with 100 runs per instance. The difference of each run to the average value of the respective instance, normalized by that average value, is plotted. The average value (red star) is always at 0. The boxes extend 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).

though their welfare distributions are more skewed towards the higher values (depicted by the whiskers in Figure 4.6, which extend from 5% to 95% of the data, and are shorter to the right side of the mean value compared to the left side), they also exhibit long tail behavior towards the lower end. This means that the HILL2(S) algorithms are more robust, yielding more consistent results between runs.

The difference can also be explained by the fact that the HILL2(S) algorithms use randomization to select a neighbor for each solution in space, but otherwise follow a gradient descent approach, whereas SA(S) randomly accept worse solutions and thus explore the search space much more before settling towards gradient descent—which can sometimes end in worse solutions overall.

In conclusion, even though the variation is small in most cases, in order to use these algorithms more reliably, multiple runs are necessary, and the best solution can be used in the end. This would, however, increase the execution time.

4.3.3 BEST ALGORITHM

Even though in Section 4.3.1 it was observed that some algorithms, such as HILL2, perform better than others on average, a deeper investigation is necessary to find out whether they perform best on any given instance.

Consequently, I analyzed the two datasets by checking, for each instance, which algorithm

Table 4.3: Breakdown of datasets D1 and D2 by best algorithm: number of instances where each heuristic algorithm computes the highest welfare in the portfolio (excluding CPLEX)—absolute numbers and percentage of total instances on which the portfolio was run.

	D1		D2	
	# instances	% instances	# instances	% instances
GREEDY1	3681	7.89 %	42	0.49 %
GREEDY2	15	0.03 %	0	0 %
GREEDY3	4	0.01 %	0	0 %
GREEDY1S	60	0.13 %	0	0 %
HILL1	1472	3.16 %	65	0.77 %
HILL1S	80	0.17 %	2	0.02 %
HILL2	6048	12.96 %	312	3.67 %
HILL2S	5592	11.99 %	3078	36.23 %
SA	12103	25.94 %	1846	21.73 %
SAS	14310	30.67 %	3134	36.89 %
CASANOVA	708	1.52 %	9	0.11 %
CASANOVAS	2583	5.54 %	8	0.09 %

computes the highest welfare. The results of the breakdown are summarized in Table 4.3.

Note that, even though SAS computes the highest welfare more often than other algorithms (30.67% and 36.23% for D1 and D2, respectively), there is no single best algorithm in the portfolio. No algorithm has majority, and the title of best algorithm is relatively balanced among SA(S) and HILL2(S) algorithms. Furthermore, considered underdogs such as GREEDY1, HILL1, and CASANOVAS succeed in winning in a significant number of cases (a total of 16.6% on D1). Overall, all the algorithms are winners in certain cases.

This result reinforces the idea that there is no clear winner of the algorithm portfolio, especially if preferences for the time-quality trade-off are considered. It is also a strong motivation for an algorithm selection approach which selects the best algorithm on a case-by-case basis.

Effective algorithms make assumptions, show a bias toward a simple solutions, trade off the costs of error against the cost of delay, and take chances.

Brian Christian, Tom Griffiths

5 High-knowledge Algorithm Selection

Double combinatorial auctions are $\mathcal{NP}$-hard (De Vries & Vohra, 2003), hindering their wide adoption in real-world applications. In Chapter 4, I showed that heuristic algorithms can mitigate the tractability and scalability issues posed by optimal algorithms. However, the evaluation in Section 4.3 also revealed that the solution quality of heuristics is highly input-dependent. Algorithm selection enables a more robust usage of heuristic algorithms for combinatorial auctions, by selecting the most suitable algorithm on a case-by-case basis.

In this thesis I propose two approaches for algorithm selection: a high-knowledge approach (MALAISE), which uses supervised machine learning to train a per-portfolio performance model, and a low-knowledge approach (PRAISE), which builds per-algorithm models using probing information.

This chapter is devoted to the high-knowledge approach, published in (Gudu et al., 2018a). It starts with a high-level description and an outline of my contributions in Section 5.1. It then dives deeper into the building blocks of the algorithm selection: the novel cost model is introduced in Section 5.1.1, which enables a quantitative multi-objective algorithm comparison; the feature space is described in Section 5.1.2. Section 5.2 formulates the selection as a classification problem, and delineates my methodology for constructing and evaluating a MALAISE selection model. Finally, an evaluation of the approach in Section 5.3 shows that it yields better results than a single algorithm—in response to Research Question 4.

5.1 APPROACH

MALAISE (MAchine Learning-based AlgorIthm SElection) is an algorithm selection approach that uses supervised learning techniques to automatically extract relevant information from large amounts of data, in order to predict which algorithm in a portfolio will perform best for a given instance.

For a more detailed description of MALAISE, the algorithm selection is broken down into four building blocks (problem, feature, algorithm, and performance measure space)—similar to Figure 2.5 in the Background chapter. Then, MALAISE can be categorized according to the characteristics of each building block, as visualized in Figure 5.1.

A defining aspect of this approach concerns the *feature space* design: as the chapter title suggests, the features are domain-specific, statistical information extracted from the problem instances. As such, high-knowledge of the problem domain—double combinatorial auctions—is required to define the features. This also means that the approach is not directly applicable to other problem domains.

The feature space is derived from the *problem space*, which consists of auction instances that, in the presented experiments, are artificially generated using CAGE, the input generator introduced in Section 4.2. The artificial nature of the problem instances allows a comprehensive coverage of the problem space, while ensuring realistic cloud resource bundles.

Another important aspect is the offline *selection*—the selection model is constructed in advance, in the initial training phase. Furthermore, the feature extraction and prediction are also performed before running any algorithm on the given instance. Only after the selection is done, the predicted best algorithm is run to completion. Even though the training phase can be costly, once the model is constructed, the selection is fast, since the feature extraction and applying the model are designed for fast execution (linear time complexity).

For each instance, a single algorithm is selected from a static *algorithm portfolio*. The proposed MALAISE approach distinguishes itself from related work (see Section 2.3.3) by dealing with heuristic algorithms for combinatorial auctions instead of optimal algorithms. This comes with its own set of challenges, among which the most significant being algorithm comparability: since heuristics generally trade solution quality for speed, it is often the case that the most accurate algorithm is not the fastest. To be able to decide which algorithm is the best, I introduce a cost model that considers both objectives—speed and accuracy—and an adjustable importance of each objective, in order to evaluate algorithm performance. In contrast, related work is only concerned with speed, as it aims to predict execution time or problem hardness.

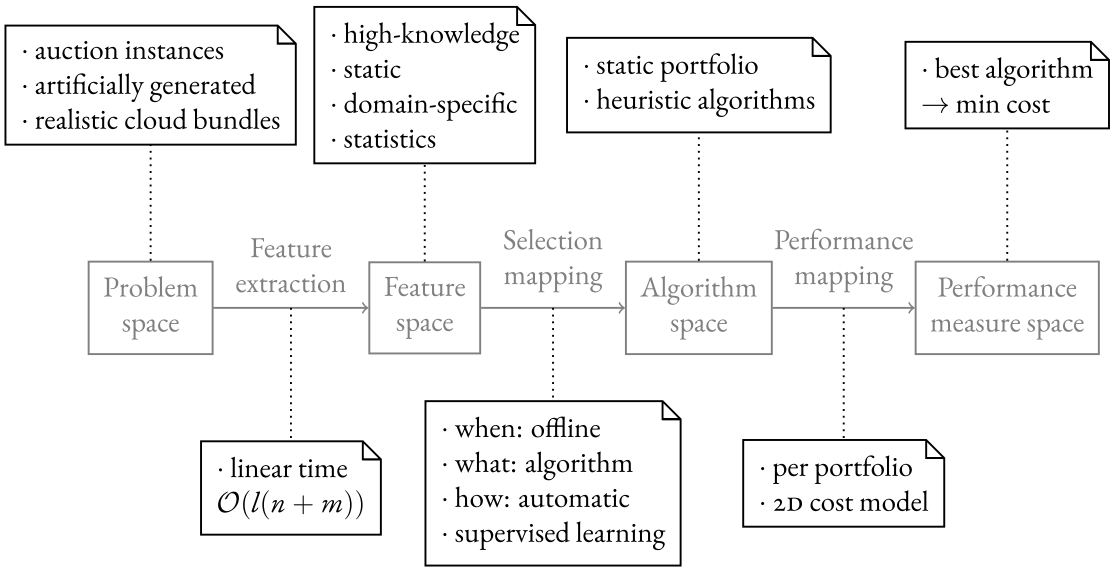


Figure 5.1: Overview of MALAISE approach for algorithm selection. Categorization and description of each building block (cf. Figure 2.5).

Finally, the cost model is used to build a *performance model* for the algorithm portfolio. Since the prediction is a categorical value (the best algorithm according to the cost model), modelling each algorithm's performance is not necessary. As a result, this approach has the benefit of considering the algorithms relative to each other rather than independently. However, this also means that adding or removing algorithms to/from the portfolio requires rebuilding the performance model.

CONTRIBUTIONS

I summarize the contributions of MALAISE to the field of algorithm selection below, and detail them in the following sections:

- MALAISE is the only approach, to my knowledge, that selects from a portfolio of *heuristic* algorithms for double combinatorial auctions.
- To that end, the proposed approach builds a performance model that considers *both* time and welfare objectives.
- The relative importance of the two objectives in selecting the best algorithm can be configured by the auctioneer.

- Moreover, features specific to the combinatorial auction domain were engineered; even more, objective-specific feature relevance was investigated in order to build selection models tailored to the desired relative importance of the time and welfare objectives.

5.1.1 COST MODEL

Since the algorithms in the portfolio are heuristic, they generally trade solution quality for speed. For a quantitative comparison of the algorithms with respect to both objectives, it is necessary to quantify the trade-off between accuracy and execution time. Finding the best algorithm is then equivalent to minimizing the proposed measure of trade-off. The cost model is based on an idea developed in collaboration with Peter Krauß.

I propose modeling this as a multi-objective optimization problem (Deb, 2014), whose objectives are a maximum social welfare (or solution quality) and a minimum execution time. In order to find a Pareto optimal solution, the idea of a compromise solution (Marler & Arora, 2004) is used, which minimizes the distance between the potential optimal point and a utopia (or ideal) point.

Firstly, as welfare and time are measured on different scales, they should be normalized to obtain non-dimensional objective functions.

The welfare objective function is normalized, and called welfare cost $c_{\mathrm{w}}(X, A)$, as defined in Equation 5.1, where $w(X, A)$ is the welfare computed by algorithm A on instance X, while $w_{\mathrm{min}}(X)$ and $w_{\mathrm{max}}(X)$ are, respectively, the minimum and maximum welfare obtained for instance X by any algorithm in the portfolio. Thus, the best algorithm when only welfare objective is considered (with maximum welfare) will have zero welfare cost.

$$c_{\mathrm{w}}(X, A) = \frac{w_{\mathrm{max}}(X) - w(X, A)}{w_{\mathrm{max}}(X) - w_{\mathrm{min}}(X)} \tag{5.1}$$

Similarly, Equation 5.2 defines the time cost $c_{\mathrm{t}}(X, A)$ as the normalized time objective, where $t(X, A)$ is the execution time of algorithm A on instance X, and $t_{\mathrm{min}}(X)$ and $t_{\mathrm{max}}(X)$ are the execution times of the fastest and slowest algorithms in the portfolio on instance X. The best algorithm with respect to time will also have zero time cost.

$$c_{\mathrm{t}}(X, A) = \frac{t(X, A) - t_{\mathrm{min}}(X)}{t_{\mathrm{max}}(X) - t_{\mathrm{min}}(X)} \tag{5.2}$$

Then the multi-objective function is defined as a vector C in the two-dimensional objective space:

$$C = \begin{bmatrix} c_{\mathrm{w}} \\ c_{\mathrm{t}} \end{bmatrix} \tag{5.3}$$

Furthermore, a user-defined preference parameter $\lambda \in [0,1]$ is introduced, which reflects the relative importance of the two objectives, in order to provide more control over the decision of selecting the best algorithm. This decision over the value of λ is entrusted to the auctioneer, who nevertheless has to make this information public, as it influences the bidding strategies of bidders and sellers.

As such, the welfare cost is weighted by a factor λ, while the time cost is weighted by $1 - \lambda$. Then the multi-objective vector can be written as a λ-dependent vector C_λ:

$$C_\lambda = \begin{bmatrix} \lambda c_{\mathrm{w}} \\ (1-\lambda) c_{\mathrm{t}} \end{bmatrix} \tag{5.4}$$

A value of $\lambda = 1$ implies that solely the welfare objective should be considered, while $\lambda = 0$ ignores the welfare objective and focuses on the algorithm's execution time. An equal importance is placed on welfare and time with $\lambda = 0.5$.

Finally, the optimal solution (best algorithm) is found by minimizing the distance to the utopian vector C°, whose components are the lower bounds of each objective function—in this case both zero:

$$C^\circ = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{5.5}$$

This distance, denoted by $c_\lambda(X, A)$, is computed using the L^2 norm (Euclidean distance):

$$\begin{aligned} c_\lambda(X, A) &= \|C_\lambda - C^\circ\| && (5.6) \\ &= \sqrt{(\lambda c_{\mathrm{w}}(X, A))^2 + ((1-\lambda)\, c_{\mathrm{t}}(X, A))^2} && (5.7) \end{aligned}$$

Then the best algorithm for a problem instance $X \in \mathcal{X}$ and a given preference $\lambda \in [0,1]$ is the algorithm with a minimum cost c_λ:

$$\mathrm{best}_\lambda(X) = \operatorname*{arg\,min}_{A \in \mathcal{A}} c_\lambda(X, A) \tag{5.8}$$

Figure 5.2 exemplifies the use of λ on a problem instance. For specific λ values, different al-

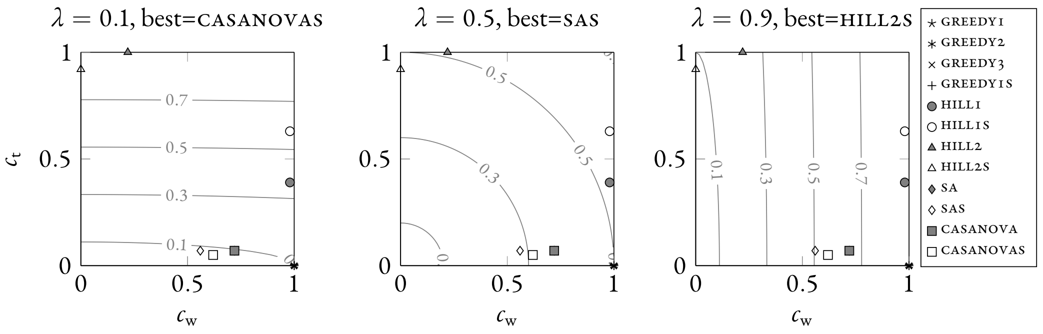


Figure 5.2: Visualization of a problem instance in the two-dimensional cost space. Isolines represent scalar cost c_λ. Different algorithms emerge as best depending on the chosen value for λ.

gorithms 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 ($\lambda = 0.5$).

5.1.2 FEATURES

Using domain knowledge—insights into the inner workings of combinatorial auctions, as well as of each individual algorithm—I defined a number of 75 features that can be extracted from any problem instance. The features are mainly statistics, and can be computed in linear time ($\mathcal{O}\left(l\left(m+n\right)\right)$), 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. The features in each group are listed in Table 5.1. I discuss a few examples 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, I 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, economics concepts were included, such as the bid-ask spread, defined as the difference between the maximum bid value and the minimum ask value, and used as a measure of market liquidity. Similarly, a quantity spread per resource was defined, as the difference between the maximum requested quantity and the minimum offered quantity per resource, and the first four central moments of the distribution of quantity spread over all the l resource types were computed. Other features in the group of demand-supply balance related features

Table 5.1: Instance features used for algorithm selection. The first four moments of various probability distributions (mean, variance, skewness, kurtosis) are denoted by $\mu_1, \mu_2, \mu_3, \mu_4$.

price related	1-4.	average bid price: $\mu_1, \mu_2, \mu_3, \mu_4$
	5-8.	average ask price: $\mu_1, \mu_2, \mu_3, \mu_4$
	9.	average bid price: max
	10.	average ask price: min
	11.	mid price
	12.	bid-ask spread
	13.	bid-ask spread over mid price
quantity related	14-17.	bid bundle size: $\mu_1, \mu_2, \mu_3, \mu_4$
	18-21.	ask bundle size: $\mu_1, \mu_2, \mu_3, \mu_4$
quantity per resource related	22-25.	total demand per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	26-29.	average demand per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	30-33.	minimum demand per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	34-37.	maximum demand per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	38-41.	total supply per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	42-45.	average supply per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	46-49.	minimum supply per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	50-53.	maximum supply per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
demand-supply balance related	54.	surplus value per surplus quantity
	55.	demand-supply value ratio
	56.	demand-supply quantity ratio
	57-60.	demand-supply ratio of total quantity per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	61-64.	demand-supply ratio of average quantity per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	65.	surplus quantity
	66-69.	surplus of total quantity per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	70-73.	quantity spread per resource: $\mu_1, \mu_2, \mu_3, \mu_4$
	74.	mean average price bid to ask ratio
	75.	mean bundle size bid to ask ratio

deal with quantity surpluses, either total surplus (the difference between the total quantity of resources offered and requested), or a quantity surplus per resource type.

5.2 METHODOLOGY

The MALAISE algorithm selection can be formulated as a multi-class classification problem. Classification (Russell & Norvig, 2009) is a supervised learning approach that takes labeled training data and constructs a model that can predict the labels of any new, unseen data.

Given a training set (X, Y) consisting of N training examples, in the form:

$$(X, Y) = \{(X_1, Y_1), \ldots, (X_N, Y_N)\}, \tag{5.9}$$

where X_i is the feature set of the i-th training example and Y_i is its label (or class), a learning algorithm tries to approximate the function $F : \mathcal{X} \to \mathcal{A}$ that maps any feature set from the feature space $\mathcal{X}$ to a label from the set of possible labels $\mathcal{A}$. For MALAISE, the feature space is the 75-dimensional space described in Section 5.1.2; the label set is the algorithm portfolio $\mathcal{A}$ presented in Section 4.1.

The methodology for training and evaluating a MALAISE selection model is described in the following, and depicted in Figure 5.3.

Having constructed an algorithm portfolio $\mathcal{A}$ of heuristics, first a raw dataset is build, that will later be processed and used for training and evaluating the learned model. 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 my artificial input generator for combinatorial auctions CAGE (cf. Section 4.2);

(1.b) running all the algorithms in the portfolio on all the generated instances to record their runtime and resulting social welfare.

The next step—(2) Preprocessing—includes feature extraction, labeling the data, and splitting the dataset into training and test data. Since using the raw input for learning can be computationally expensive or even intractable, it is necessary to use domain knowledge to extract a set of features that contain sufficient information to aid the learning process (see Sect. 5.1.2). Labeling the dataset means selecting the best algorithm for each problem instance—the algorithm with the lowest cost, as defined in Sect. 5.1.1. The processed dataset is then split into a training set (X, Y) and a test set (X', Y').

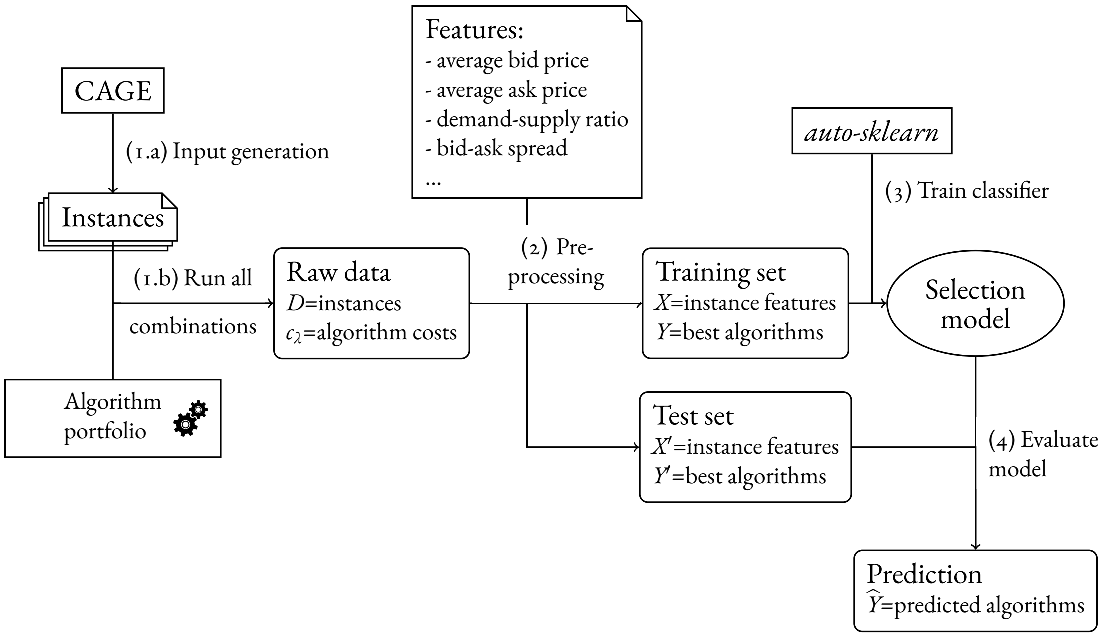


Figure 5.3: MALAISE methodology.

In step (3), a model is trained that can predict the class label of any new problem instance, using the *auto-sklearn* (Feurer et al., 2015) library, which implements an automated machine learning approach. The library relies on Bayesian optimization methods to construct an ensemble of classifiers and find their best hyperparameters and preprocessing steps. The preprocessing, in this case, includes feature scaling and feature selection for dimensionality reduction, based on their relevance as described in Sect. 5.3.1. The 15 classification algorithms in *auto-sklearn* fall into seven categories: general linear models, support vector machines, discriminant analysis, nearest neighbors, naïve Bayes, decision trees, and ensembles.

For MALAISE, ensemble models were built, which combine several learning algorithms into one predictive model, in order to decrease variance or bias (Russell & Norvig, 2009). The ensemble methods considered by *auto-sklearn* are: AdaBoost, gradient boosting, random forests, and extremely randomized trees. These methods use decision trees as base classifiers, which have the benefit of interpretability and fast execution on test data. This motivates the use of ensemble models over, for example, neural networks. Although neural networks do not require feature engineering, but rather infer the features during training, they are slower for both training and prediction. Furthermore, their black-box nature makes them impracticable for auction design: interpretable models are important for market participants, in order to devise appropriate bidding strategies.

Finally, the model is (4) tested on unseen data. At this step, several appropriate metrics to evaluate the quality of the prediction are considered, and discussed in the following.

5.2.1 PREDICTION EVALUATION METRICS

There are several success measures when evaluating a classification model. The most intuitive metric is the accuracy, namely how often the model correctly predicts the algorithm with the lowest cost. More specifically, I define the accuracy in Equation 5.10, for a given dataset X, as the fraction of the instances for which the predicted algorithm $\widehat{Y}_i$ is the same as the algorithm with the lowest cost Y_i.

$$accuracy_\lambda(Y, \widehat{Y}) = \frac{1}{|Y|} \sum_{i=1}^{|Y|} 1(\widehat{Y}_i = Y_i) \tag{5.10}$$

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 address this, I introduce a metric that considers the cost of the predicted algorithm: the mean squared error (*MSE*), defined in Equation 5.11 as the mean squared differences between the predicted and true best algorithms. The MSE can therefore detect when a wrongly predicted algorithm has the same cost as the best algorithm—and not count it towards the total error. On the other hand, large cost differences have a significant impact on the error due to squaring.

$$MSE_\lambda(Y, \widehat{Y}) = \frac{1}{|Y|} \sum_{i=1}^{|Y|} \Big(c_\lambda(X_i, \widehat{Y}_i) - c_\lambda(X_i, Y_i) \Big)^2 \tag{5.11}$$

However, the absolute *MSE* values needs to be put into context in order to be a meaningful indicator of prediction quality: only by comparing it to the error of other approaches (a single auction mechanism or random selection), it can be said if and to what extent the algorithm selection improves upon these approaches.

First, to compare the MALAISE algorithm selection against a single algorithm A^*, I introduce the relative mean squared error (*RMSE*) metric, defined in Equation 5.12 as the ratio between the *MSE* of the classification model and the *MSE* of using algorithm A^* on the entire dataset.

$$RMSE_\lambda(Y, \widehat{Y}, A^*) = \frac{MSE_\lambda(Y, \widehat{Y})}{MSE_\lambda(Y, A^*)} \tag{5.12}$$

The classification model can be similarly compared to a random selection model, where a random algorithm $\widetilde{Y}_i$ is selected for each instance i.

$$RMSE_\lambda(Y, \widehat{Y}, \widetilde{Y}) = \frac{MSE_\lambda(Y, \widehat{Y})}{MSE_\lambda(Y, \widetilde{Y})} \tag{5.13}$$

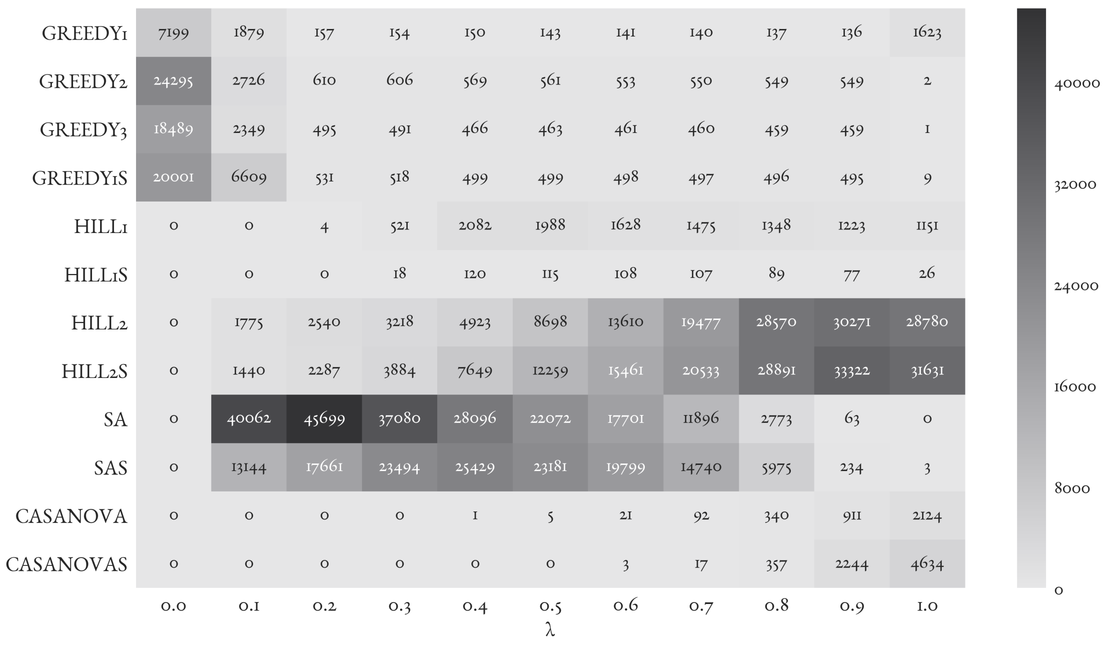
5.3 EVALUATION

Using the proposed input generator CAGE, a dataset D3 was created, consisting of 69,984 problem instances. Similar to the dataset D1 in Section 4.3, used to compare the algorithms in the portfolio, the bundles in this dataset have 3 resource types (e.g., CPU cycles, memory, disk) and they were generated using four different models based on the Google cluster traces, as well as the artificial models UNIFORM and HOTSPOTS. The valuation parameters were similarly varied to support different spreads of agent valuations around a common value, as well as different degrees of overlap between bidder and seller valuations. The main difference stems from the larger problem sizes in D3: 10,000 bids and 10,000 asks. This is now possible since only the heuristic algorithms need to be run on this dataset. The script and parameters used to create D3 are listed in Appendix C (Listing C.3).

5.3.1 DATASET ANALYSIS

The dataset was analyzed 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 5.4 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 less accurate, 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 most 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 a solution faster. Even though they were selected less often, the HILL1(S) and CASANOVA(S) algorithms have a non-negligible coverage for several λ values. Thus, they cannot be excluded from the portfolio, as



	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
GREEDY1	7199	1879	157	154	150	143	141	140	137	136	1623
GREEDY2	24295	2726	610	606	569	561	553	550	549	549	2
GREEDY3	18489	2349	495	491	466	463	461	460	459	459	1
GREEDY1S	20001	6609	531	518	499	499	498	497	496	495	9
HILL1	0	0	4	521	2082	1988	1628	1475	1348	1223	1151
HILL1S	0	0	0	18	120	115	108	107	89	77	26
HILL2	0	1775	2540	3218	4923	8698	13610	19477	28570	30271	28780
HILL2S	0	1440	2287	3884	7649	12259	15461	20533	28891	33322	31631
SA	0	40062	45699	37080	28096	22072	17701	11896	2773	63	0
SAS	0	13144	17661	23494	25429	23181	19799	14740	5975	234	3
CASANOVA	0	0	0	0	1	5	21	92	340	911	2124
CASANOVAS	0	0	0	0	0	0	3	17	357	2244	4634

Figure 5.4: Algorithm selection dataset: breakdown by class labels for several λ values.

they seem to serve a certain (albeit niche) type of problems. An interesting result is that there is a number of instances for which the greedy algorithms were selected as best even when execution time was not important ($\lambda = 1$). This means that the other algorithms could not improve upon the greedy solution (since they typically use it as an initial solution). In some cases, this is because no match exists and the social welfare is zero (the so-called infeasible instances)—thus the fastest algorithm is always selected as best.

Figure 5.4 hence demonstrates the input-dependent performance of heuristic algorithms, and the potential for improvement by using algorithm selection.

Next, I investigated which features are more relevant to the prediction. The aim is to identify irrelevant or redundant features, and remove them to reduce the dimensionality of the input space and prevent over-fitting. Tree-based estimators were used to compute relative feature importances for the model's performance.

In Figure 5.5, all 75 features are sorted based on their importance and the first 20 are shown. Note that only a few are relevant, e.g. 17 features have an importance over 0.02.

The most relevant features are related to the asking price per unit—the spread (standard deviation) over the set of asks, the minimum value. Other important features are concerned with the relationship between the asking and bidding price per unit (their ratio, the bid-ask spread,

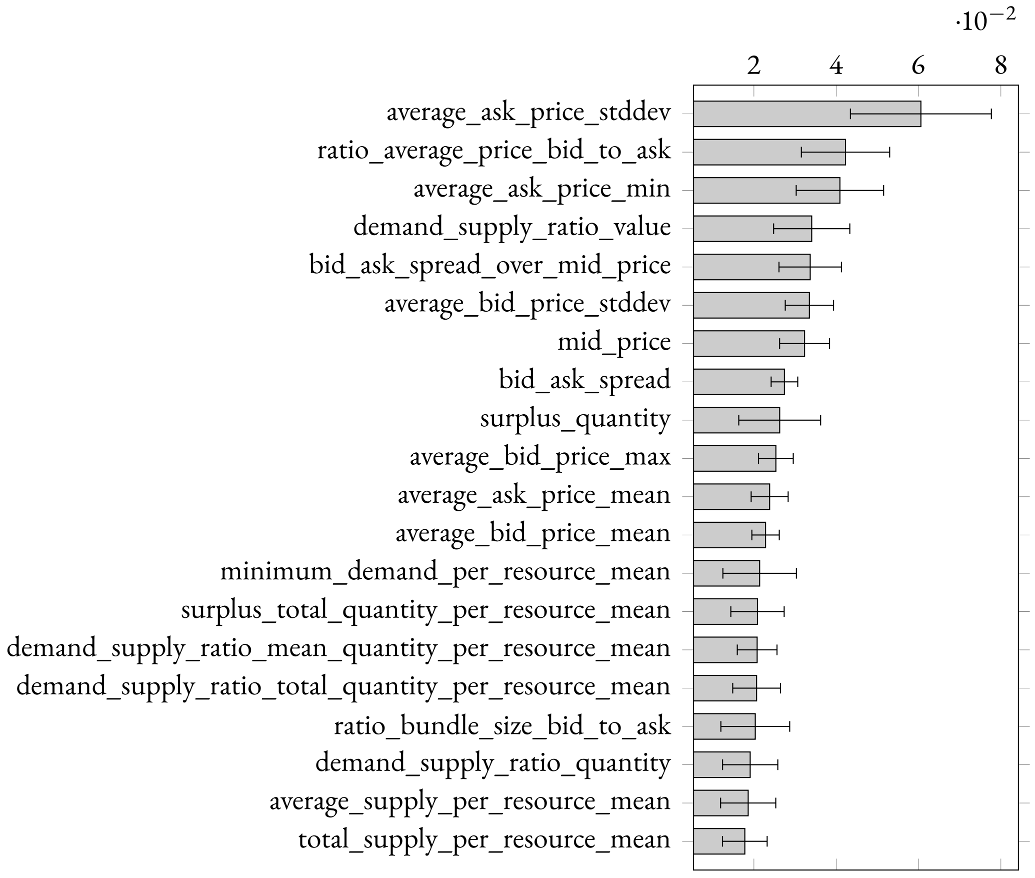


Figure 5.5: Relative feature importances averaged over all λ values, computed using ExtraTreesClassifier in *scikit-learn* with 500 estimators. Only the first 20 most relevant features are shown.

the mid price, the value demand-supply ratio). This is not surprising, since this relationship is inextricably embedded in the problem: a bid and ask can be matched only if the bid value is larger than the ask value; expensive offers (with high asking values) are hard to match, and thus often not part of the solution. Nevertheless, stochastic algorithms might consider them during the search as a way to escape local optima, while hill-climbing algorithms only move to better solutions. It is clear that these features affect each algorithm's behavior differently.

From the quantity-related features, only the ones that encode the relation between the quantities of bids and asks, either overall (surplus quantity) or per resource (surplus/demand-supply ratio quantity per resource), have a certain effect on the prediction. This can be explained by the fact that quantities per resource are instrumental in assessing the feasibility of a solution, as enforced by constraint (3.11), and influence the way algorithms move in the search space.

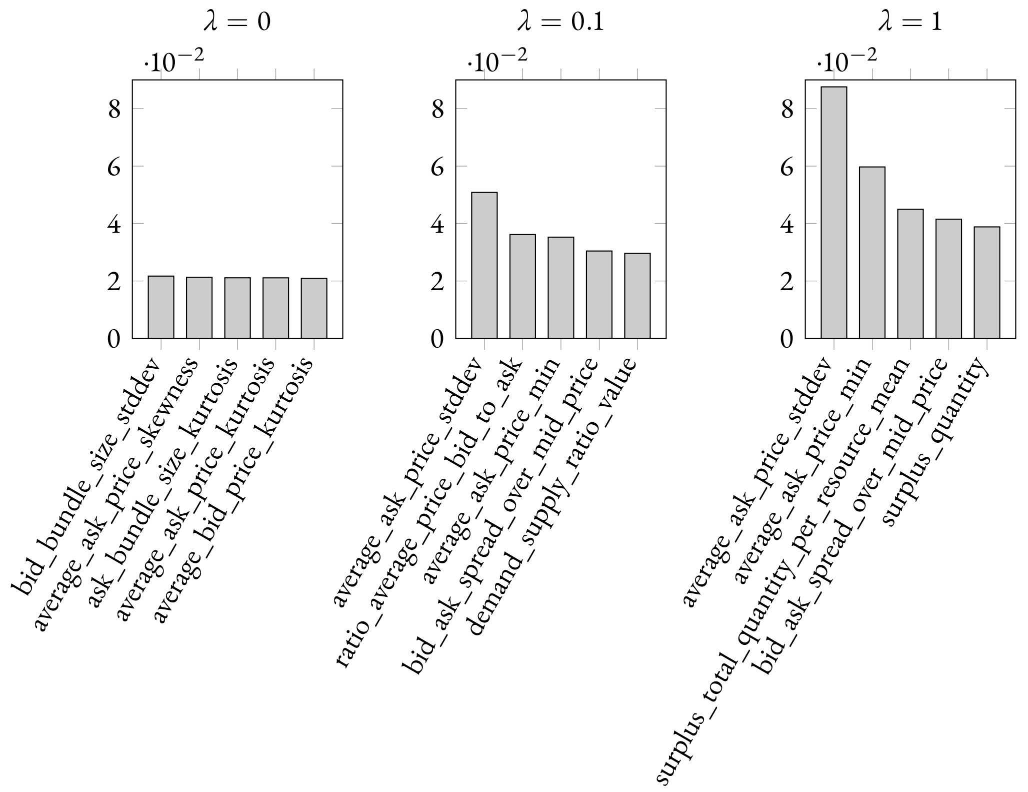


Figure 5.6: Relative feature importances for $\lambda \in \{0, 0.1, 1\}$, computed using ExtraTreesClassifier in *scikit-learn* with 500 estimators. Only the first 5 most relevant features are shown.

The error bars in Figure 5.5 suggest that different features might be relevant for different λ values. As a result, I investigate feature relevance for extreme λ values ($\lambda = 0$ and $\lambda = 1$), and plot the 5 most relevant features for each λ in Figure 5.6. When only welfare is considered in the selection ($\lambda = 1$), the features discussed above (spread and minimum value of ask price per unit, and surplus quantity—total or per resource) are still the most relevant. What is more, their impact on the prediction compared to the rest of the features is more pronounced than for other λ values. Therefore, it can be said that my proposed feature set contains sufficient information to reliably predict which algorithm will result in the highest welfare. On the other hand, when $\lambda = 0$ and only execution time is considered, all the features have a similar, low relevance. In this case, the best algorithms are the greedy ones, even if they do not find any matches; the feature set is not able to discriminate between the different greedy algorithms—mostly because their runtimes are strikingly similar. Finally, I also looked at $\lambda = 0.1$, to investigate whether the features are predictive of execution time: for such a low λ, runtime is still the most important objective for algorithm selection, but welfare is also considered, resulting in a more balanced label distribution. In this case, the value-related features are more relevant than the quantity-related ones.

It can be concluded that the defined feature set is useful in the prediction of the best algorithm, with the caveat that irrelevant and redundant features should be eliminated before training, for faster training and enhanced generalization (reduced over-fitting). This is done automatically by *auto-sklearn*, which selects the relevant features from the 75 given features.

5.3.2 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 training and test sets have the same percentage of samples of each class as the full set.

Using *auto-sklearn*, a classification model was trained for each λ value. This is necessary since the labels are different for each λ value.

Figure 5.7 shows the accuracy of the models for each λ, on both training and test set. Good accuracies (between 90–97% on the training set, 80–92% on the test set) 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. The only exception is $\lambda = 0$, where MALAISE only achieves 42% and 38% accuracy on the training and test set, respectively. The reasons for this were explained in Section 5.3.1: the fastest algorithms all belong to the greedy family, and the feature set cannot discriminate between the execution times of the greedy algorithms, since the differences in runtime are small and essentially random.

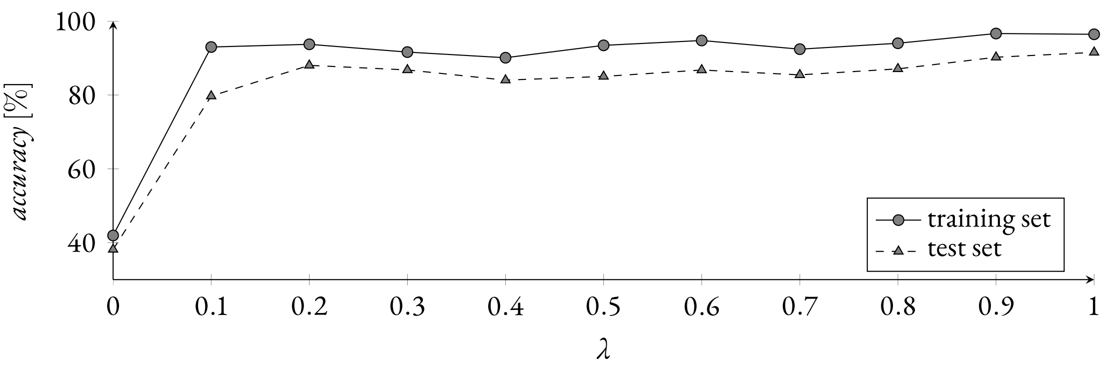


Figure 5.7: Accuracy of MALAISE for different λ values.

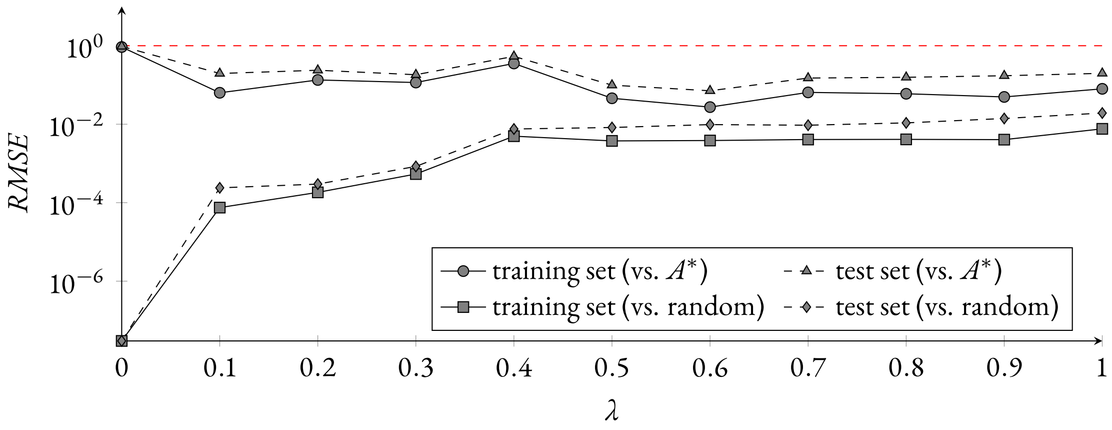


Figure 5.8: *RMSE* comparison of MALAISE to random selection and best pure algorithm for different λ values.

Next, the mispredictions are quantified by calculating the prediction error *MSE* (cf. Equation 5.11), and compared to a random selection, as well as a single algorithm, using the *RMSE* metric (cf. Equation 5.12). Note that an *RMSE* value below 1 implies that the MALAISE approach is better than its counterpart in the comparison.

The comparison between the trained models for each λ, and random selection (see Figure 5.8) shows that MALAISE is always 2 to 7 orders of magnitude better than a random selection approach. This is true especially for $\lambda = 0$: since the greedy algorithms are orders of magnitude faster than the others, randomly selecting a non-greedy algorithm will have a large impact on the error. Similarly, I compared the MALAISE models to the best pure algorithm A^* for each λ, where A^* is defined as the algorithm selected most often as best in the labeling phase (cf. Figure 5.4). 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. The comparison revealed that MALAISE outperforms the best pure algorithm for all values of λ (see Figure 5.8), with overall lower *RMSE* for larger λ. On the test set, my approach results in an error that is 2 to 14 times lower. The only exception is, again, $\lambda = 0$: with an *RMSE* of 0.96, MALAISE is only slightly better than the best pure algorithm approach. This confirms that the runtime differences between the greedy algorithms are small and unpredictable.

All in all, the trend of better prediction for higher λ values is confirmed by both accuracy and *RMSE* evaluations. The evaluation showed that the proposed MALAISE approach yields accurate predictions for all user-preferences λ, and it improves upon the single algorithm approach.

In your thirst for knowledge, be sure not to drown in all the information.

Anthony J. D'Angelo

6 Low-knowledge Algorithm Selection

While the MALAISE algorithm selection approach presented in Chapter 5 delivers good results by accurately selecting the best algorithm, and improves upon the best (on average) single algorithm in the portfolio, it has a few limitations. First, domain knowledge is needed to define combinatorial auction-specific features that facilitate the machine learning process. This makes it hard to generalize the approach to other problem domains. Second, a training phase is necessary in order to build the selection model; what is more, the model needs to be re-trained after every change to the algorithm portfolio.

In this chapter, I propose a second approach for algorithm selection, PRAISE, which does not require any knowledge of the problem domain, or training. PRAISE uses probing information to build per-algorithm performance models. As a result, adding an algorithm to the portfolio only involves modelling the performance of the new algorithm. However, this approach requires some knowledge of the algorithms in the portfolio, namely time complexity and welfare scaling over problem size.

Section 6.1 gives an overview of the method and contributions. In Section 6.2, the methodology behind PRAISE is detailed: I describe how the probing is done and how the acquired information is used to predict algorithm performance. Updated metrics for evaluating the prediction are presented in Section 6.2.4. The proposed method is evaluated in Section 6.3. I show the cases where it outperforms a single algorithm, and discuss the reasons why it does not in other cases. Finally, the two algorithm selection approaches are compared in Section 6.4, and the benefits and drawbacks of each method are discussed.

6.1 APPROACH

PRAISE (PRobing-based AlgorIthm SElection) is an algorithm selection approach that predicts individual algorithm performance, based on information obtained by running the algorithm portfolio on a sample of the problem instance; the best algorithm is then selected after computing the predicted cost of each algorithm.

As with MALAISE (in Figure 5.1), the algorithm selection is broken down into four building blocks: problem, feature, algorithm, and performance measure space. Then, PRAISE can be described by characterizing each building block, as visualized in Figure 6.1.

The *problem space* and the *algorithm space* remain the same as for MALAISE: the problems continue to be artificially generated auction instances, while the algorithm portfolio remains static and comprises heuristic algorithms for solving the WDP. Nevertheless, there are significant differences in other areas.

First of all, the only *features* needed for the prediction on a given instance are the execution time and social welfare of each algorithm on a sample of the instance. These features are termed low-knowledge, since no domain-knowledge is required to define them. The feature extraction is performed semi-statically through a process known as probing—taking a probe of algorithm performance. In contrast to similar approaches in the literature that run the algorithms for a short time and measure their performance, in PRAISE the algorithms are run to completion on a *sample* of the problem instance. The semi-static nature of a probing-based approach was explained in Section 2.3.2: the features are computed before actually solving the instance (like static features), but they do explore a part of the current instance's search space (similar to dynamic features).

The second notable difference to MALAISE concerns the *performance mapping* and, implicitly, the *selection mapping*. Performance models are built per algorithm by predicting the values of time and welfare of each algorithm on a given instance. The selection mapping then employs these performance models, together with simple hand-crafted rules, to select the best algorithm—the algorithm with the minimum predicted cost. The cost model, introduced in Section 5.1.1, considers both time and welfare objectives to quantitatively evaluate and compare algorithms. As with MALAISE, the selection is performed offline. Once the selection is finished, the selected algorithm is run to completion.

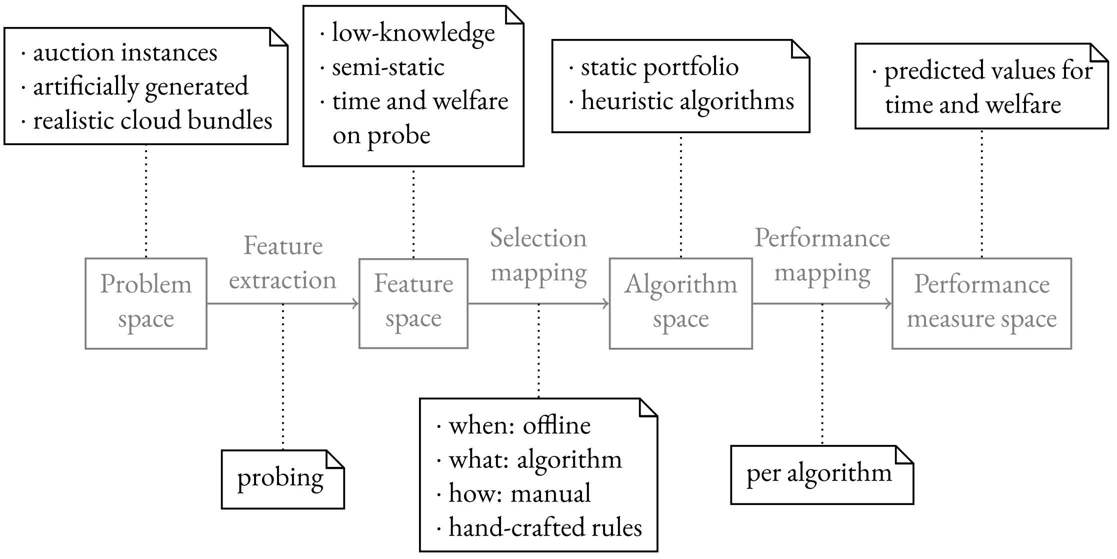


Figure 6.1: Overview of PRAISE approach for algorithm selection. Categorization and description of each building block (cf. Figure 2.5).

CONTRIBUTIONS

The PRAISE approach makes the following contributions to the field of algorithm selection:

- Same as MALAISE, it selects from a portfolio of *heuristic* algorithms for double combinatorial auctions.
- To that end, it builds individual performance models for each algorithm, for *both* time and welfare objectives.
- The features used in the prediction are obtained through probing, making the approach generic and transferable to other problem domains. No domain-specific features are employed, but the required knowledge is shifted onto deriving algorithm properties.

6.2 METHODOLOGY

This section describes the methodology for applying PRAISE on a single problem instance, as depicted in Figure 6.2.

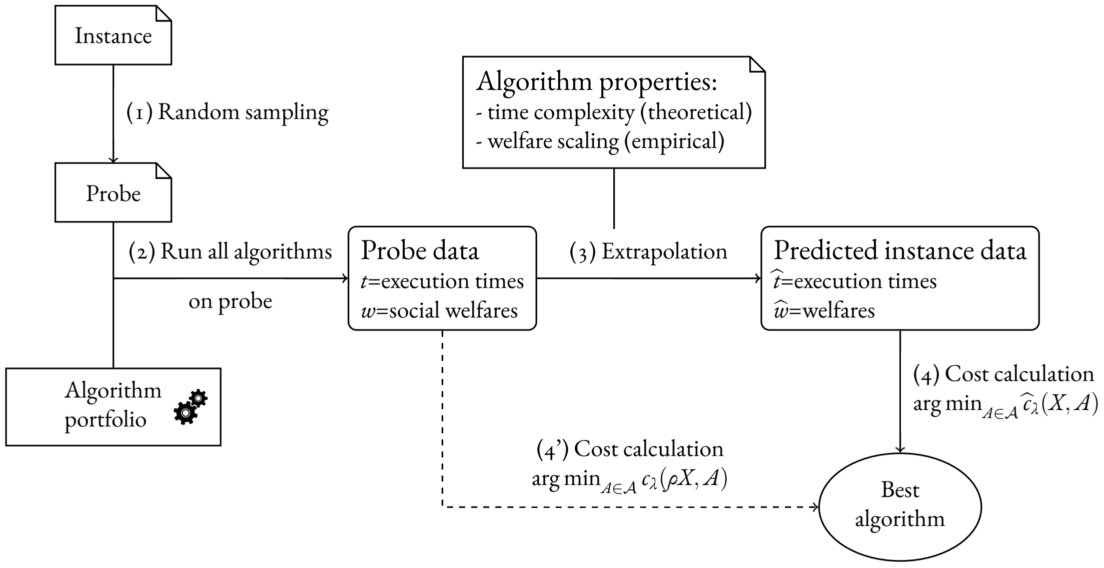


Figure 6.2: PRAISE methodology.

In step (1), a probe is taken from the problem instance through random sampling. The probe consists of a subset of the instance's bids and asks. Next, all the algorithms in the portfolio are (2) run on the probe, and the resulting welfares and runtimes are recorded. Using prior knowledge of the algorithms' properties, the probe data is (3) extrapolated to predict the execution time and welfare of each algorithm on the full instance. The algorithm properties characterize the scaling behavior for time and welfare with respect to problem size—the time complexity is theoretically proven, while the welfare scaling is derived empirically. Finally, in step (4), the predicted times and welfares are used to compute the predicted cost of each algorithm on the full instance. The algorithm with the minimum predicted cost is then selected as the best. Alternatively, the best algorithm on the probe can be selected as the best algorithm for the full instance (4').

Each step is discussed more extensively in the following.

6.2.1 PROBING

The probing step consists of taking a sample of the problem instance and then running all algorithms in the portfolio on the sample.

The idea is similar to other approaches (Beck & Freuder, 2004) where heuristic algorithms for the job shop scheduling problem are run for a short fixed time, and the best algorithm is chosen based on the solution quality at that point. However, such methods are aimed at selecting the fastest optimal algorithm, and are sensitive to how the solution quality changes over the running time.

In contrast, since the work presented thesis deals with heuristic algorithms, it considers both their runtime and solution quality (welfare). This is why both time and welfare are variable, and instead the sample size is fixed. A sample of the problem instance itself is thus taken, and the algorithms are run to completion on the sample.

The sampling technique used is simply random. This is the most appropriate technique for situations where not much information is available about the population, as in this case: PRAISE aims to reduce the domain expertise needed and increase applicability to other domains.

Formally, the sampling is defined by a parameter $\rho \in (0, 1)$, representing the sampling ratio, or the fraction of the full problem that is selected to be part of the sample. Given an auction instance $X \in \mathcal{X}$, consisting of n bids and m asks, a random sample is constructed by randomly selecting ρn bids and ρm asks. The sample problem instance is denoted by ρX.

For MALAISE, the feature extraction was performed in linear time, and could thus be neglected when evaluating the approach, since all the algorithms have a time complexity of at least $\mathcal{O}(n \log n)$.

In contrast, the time overhead of probing is non-negligible, proportional to the sample size, and dependent on the algorithms included in the portfolio.

For an instance X and a sampling ratio ρ, the time overhead τ can be defined as the sum of the execution times of all the algorithms in the portfolio on the sample ρX:

$$\tau(X, \rho) = \sum_{A \in \mathcal{A}} t(\rho X, A) \tag{6.1}$$

6.2.2 ALGORITHM PROPERTIES

Given the low-knowledge features, no expertise is required regarding the problem domain. Instead, the expertise is shifted to extracting the algorithm properties, increasing the generality of this approach. The only relevant properties concern the scaling of execution time and solution quality over problem size. They are discussed below, and summarized in Table 6.1.

For simplicity, it can be assumed that the number of bids and asks are in the same order of magnitude ($n \approx m$), and the number of resource types is much smaller than the number of bids ($l \lll n$). Then, without loss of generality, the problem size can be expressed through the number of bids n.

Table 6.1: Algorithm properties: scaling of time and welfare with problem size.

Algorithm	Time complexity	Welfare scaling
GREEDY1	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{0.86})$
GREEDY2	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{0.86})$
GREEDY3	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{0.87})$
GREEDY1S	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{0.96})$
HILL1	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^{0.85})$
HILL1S	$\mathcal{O}(n^2 \log n)$	$\mathcal{O}(n^{0.93})$
HILL2	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{1.03})$
HILL2S	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{1.03})$
SA	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{0.96})$
SAS	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^{0.96})$
CASANOVA	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
CASANOVAS	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$

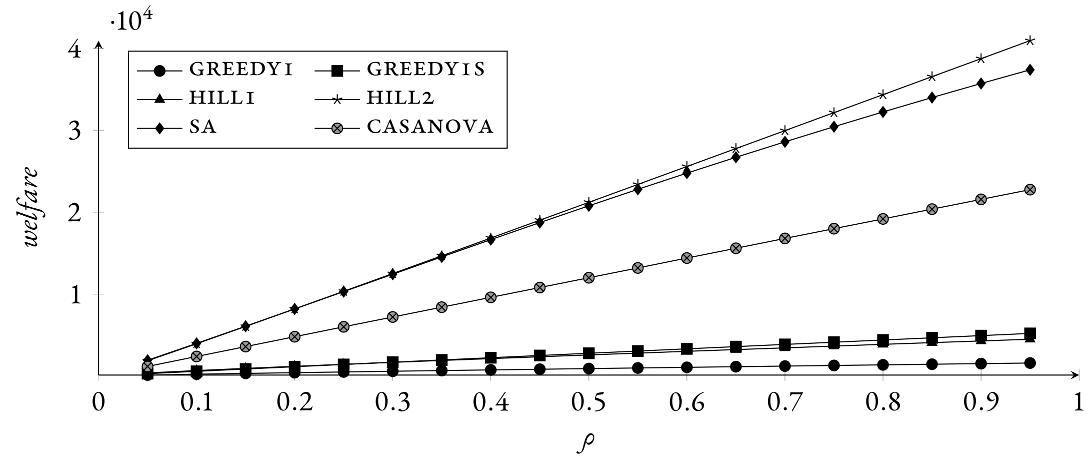
The time scaling over problem size is given by each algorithm's complexity class. Following a time complexity analysis, I derived each algorithm's asymptotic upper bound, as listed in Table 6.1. A few aspects of the analysis are discussed below.

For the greedy algorithms, the execution time is dominated by the sorting step (lines 4–5 in Algorithm 1), which has a complexity of $\mathcal{O}(n \log n)$ (Cormen et al., 2009). The rest of the algorithm is executed in linear time, $\mathcal{O}(n)$, since the **while** loop (lines 7–11) only goes through the sorted bid and ask lists and tries to match them, being executed at most $l(n + m)$ times. Thus the complexity of the greedy algorithms is $\mathcal{O}(n \log n)$. This can be similarly proven for the simulated annealing algorithms, where the cooling schedule has a $\mathcal{O}(1)$ number of steps, albeit a large number in the order of 10^4:

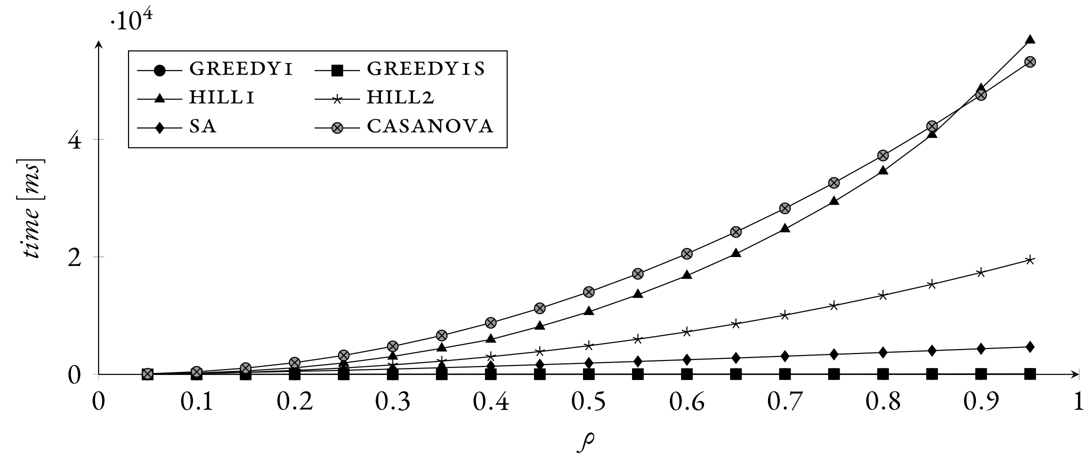
$$\text{maximum number of steps} = N_{it} \frac{\log T_{\min} - \log T_{\max}}{\log \alpha}. \tag{6.2}$$

For the HILL1(S) algorithms, the Master Theorem (Cormen et al., 2009) was employed, a method for analyzing the complexity of recursive algorithms.

Next, I investigated how the welfare computed by each algorithm scales with the problem size. This was derived empirically, since there is no theoretical basis for it. The algorithms were run on random samples of increasing size ($\rho = 0.05, 0.1, \ldots, 0.95$) for each instance in an artificially generated dataset. The average welfare over sampling ratio is plotted in Figure 6.3(a), for a subset of the algorithms in the portfolio. The plot suggests a near linear scaling, with



(a) Welfare scaling over problem size.



(b) Time scaling over problem size.

Figure 6.3: Welfare and time scaling over problem size, where the problem size is given by the sampling ratio ρ. Only 6 algorithms in the portfolio are shown.

different slopes per algorithm. The algorithms based on optimizing the ordering of bids and asks (all the greedy algorithms, HILL1(S)) have the worst scaling behavior. An interesting result is the difference between HILL2 and SA: the two curves grow together and diverge only after $\rho = 0.5$. This can be explained by the stochastic nature of simulated annealing, which accepts worse solutions with a probability that decreases with the temperature. The limits imposed on temperature and number of iterations speed up the search, but for large problems, they do not allow the algorithm to sufficiently explore the search space.

Given the measured welfare values over problem size, the precise welfare scaling was computed for each algorithm through curve fitting, using non-linear least squares. The obtained functions are shown in Table 6.1.

A similar investigation was performed for time, which validated empirically the theoretical complexity classes. Figure 6.3(b) shows how the execution time scales with the problem size—for readability reasons, only a subset of the algorithms are displayed. The plot highlights the significance of the algorithm properties, most visible when comparing the scaling curves of HILL1 and CASANOVA: up to a sample size of 0.85, HILL1 runs faster, after which point this flips, and HILL1 becomes slower than CASANOVA.

6.2.3 PREDICTION

In this section, steps (3) and (4) in the PRAISE workflow are described (cf. Figure 6.2), namely how the probing information and the algorithm properties are combined to predict algorithm performance, as well as how the best algorithm is selected.

During the probing, the time t and welfare w values on the sample are measured for each algorithm. Knowing how they scale with problem size (cf. Section 6.2.2), one can predict each algorithm's time $\widehat{t}$ and welfare $\widehat{w}$ values on the full instance through extrapolation.

Equation 6.3 defines the extrapolation for any function φ that describes a scaling behavior (e.g. $\varphi(n) = n \log n$). Given a sampling ratio ρ and the function value on the sample, $\varphi(\rho X)$, and assuming the function is invertible ($\exists \varphi^{-1}$, and is unique, such that $\varphi^{-1}(\varphi(x)) = x, \forall x$), the function value on the full instance, $\varphi(X)$, can be expressed as:

$$\varphi(X) = \varphi\left(\frac{\varphi^{-1}(\varphi(\rho X))}{\rho}\right) \tag{6.3}$$

Next, the cost model introduced in Section 5.1.1 is used to quantitatively compare algorithm performance with respect to both time and welfare. In this case, the predicted welfare and time values are used to compute the predicted cost, as shown in Equation 6.6. Besides the preference λ, the predicted cost of an algorithm A on a problem instance X also depends on the chosen sampling ratio ρ. The predicted costs of welfare and time are formulated below, in Equation 6.4 and Equation 6.5, respectively.

$$\widehat{c}_{\mathrm{w}}(X, A, \rho) = \frac{\widehat{w}_{\max}(X, \rho) - \widehat{w}(X, A, \rho)}{\widehat{w}_{\max}(X, \rho) - \widehat{w}_{\min}(X, \rho)} \tag{6.4}$$

$$\widehat{c}_{\mathrm{t}}(X, A, \rho) = \frac{\widehat{t}(X, A, \rho) - \widehat{t}_{\min}(X, \rho)}{\widehat{t}_{\max}(X, \rho) - \widehat{t}_{\min}(X, \rho)} \tag{6.5}$$

$$\widehat{c}_{\lambda}(X, A, \rho) = \sqrt{(\lambda \widehat{c}_{\mathrm{w}}\,(X, A, \rho))^2 + ((1-\lambda)\,\widehat{c}_{\mathrm{t}}\,(X, A, \rho))^2} \tag{6.6}$$

Finally, the algorithm with the minimum predicted cost is selected as best for the respective instance:

$$\mathrm{best}_{\lambda}(X, \rho) = \underset{A \in \mathcal{A}}{\arg\min}\, \widehat{c}_{\lambda}(X, A, \rho) \tag{6.7}$$

A simpler approach, that does not even require knowledge about the algorithms, would be to skip the extrapolation and assume that the best algorithm on the full instance is the same as the best algorithm on the sample—given by the minimum cost. Even though this ignores important differences in algorithm behavior, I include this approach in the evaluation—and name it PRAISE0—in order to visualize the impact of the knowledge of algorithm scaling behavior.

6.2.4 PREDICTION EVALUATION METRICS

The PRAISE approach can be evaluated by the same metrics as MALAISE (cf. Section 5.2.1). The accuracy measures how often the prediction is correct. The mean squared error *MSE* measures how bad the mispredictions are. The *RMSE*, or relative mean squared error, is used to compare the approach against other methods with respect to *MSE*. However, the *MSE* and *RMSE* metrics should be updated to include the non-negligible overhead of the probing phase.

Since the probing phase only affects the runtime of the full instance and not the welfare, I introduce a time cost of probing $\tilde{c}_t$ in Equation 6.8. This is defined as the time overhead τ (previously defined in Equation 6.1), normalized to the same time interval between the slowest and fastest algorithm in the portfolio. The time cost of probing is simply added to the time cost of each algorithm, resulting in an updated cost $\tilde{c}_\lambda$ in Equation 6.9.

$$\tilde{c}_t(X,\rho) = \frac{\tau(X,\rho)}{t_{\max}(X) - t_{\min}(X)} \tag{6.8}$$

$$\tilde{c}_\lambda(X,A,\rho) = \sqrt{\left(\lambda c_w\left(X,A\right)\right)^2 + \left(\left(1-\lambda\right)\left(c_t\left(X,A\right) + \tilde{c}_t\left(X,\rho\right)\right)\right)^2} \tag{6.9}$$

Note the subtle differences between the various costs introduced in this chapter. The cost $\tilde{c}_\lambda$ in this section uses the *real* welfare and time values of the selected algorithm, measured *after* the algorithm selection, to *evaluate* the algorithm performance. The predicted cost $\hat{c}_\lambda$ in Section 6.2.3 uses the *predicted* time and welfare values to *select* the best algorithm.

With the updated cost $\tilde{c}_\lambda$, one can reformulate the mean squared error $\widetilde{MSE}_\lambda$ in Equation 6.10, and $\widetilde{RMSE}_\lambda$ in Equation 6.11.

$$\widetilde{MSE}_\lambda(Y,\hat{Y},\rho) = \frac{1}{|Y|}\sum_{i=1}^{|Y|}\left(\tilde{c}_\lambda(X_i,\hat{Y}_i,\rho) - c_\lambda(X_i,Y_i)\right)^2 \tag{6.10}$$

$$\widetilde{RMSE}_\lambda(Y,\hat{Y},A^*,\rho) = \frac{\widetilde{MSE}_\lambda(Y,\hat{Y},\rho)}{MSE_\lambda(Y,A^*)} \tag{6.11}$$

6.3 EVALUATION

For consistency, the PRAISE approach was evaluated on the same dataset D3 as MALAISE. The dataset was artificially generated using CAGE, with the parameters described in Section 5.3.

6.3.1 SAMPLE SIZE STUDY

First, the effect of the chosen sampling ratio ρ on the prediction was investigated, in order to find the most appropriate value.

There are two interacting factors that affect the prediction: on the one hand, it is expected that the accuracy of the prediction increases with the sample size; on the other hand, a large sample also means higher time overhead. These factors are already encoded in the updated cost model $\tilde{c}_\lambda$, and correspondingly in the error $\widetilde{MSE}_\lambda$ and $\widetilde{RMSE}_\lambda$. As such, the optimal sampling ratio ρ_{best} is the one that minimizes the error:

$$\rho_{\text{best}} = \arg\min_{\rho} \widetilde{MSE}_{\lambda}(Y, \widehat{Y}, \rho). \tag{6.12}$$

From Equation 6.12, it also follows that the optimal sampling ratio is λ-dependent. Since the λ parameter indirectly controls the significance of the time overhead in the quality of the prediction, it is expected that the maximum acceptable sampling ratio increases with the λ values.

A range of acceptable sample sizes can be determined for each λ by considering the ratios for which the error relative to the single algorithm is below 1:

$$\rho \in [\rho_{\min}, \rho_{\max}] \text{ such that } \widetilde{RMSE}_{\lambda}(Y, \widehat{Y}, A^*, \rho) < 1. \tag{6.13}$$

Figure 6.4 exemplifies the sample size study for $\lambda = 0.5$ (equally balanced time and welfare), by plotting the $\widetilde{RMSE}$ over sampling ratio. The plot confirms the expected trend of prediction quality increasing with sample size (decreasing $\widetilde{RMSE}$) up to a certain point, after which the time overhead takes over, and the quality starts decreasing. This point indicates the optimal sampling ratio, in this case 0.15. Acceptable ρ values are in the interval $[0.1, 0.25]$, since with these values, PRAISE still improves upon the single algorithm approach, with $\widetilde{RMSE} < 1$.

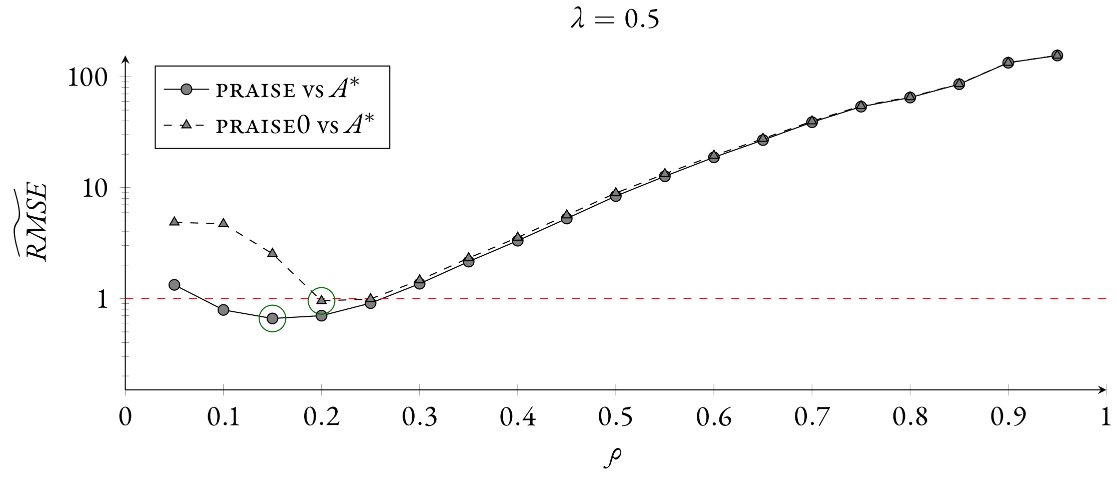


Figure 6.4: $\widetilde{RMSE}$ over sampling ratio ρ for $\lambda = 0.5$. Optimal ratios, ρ_{best} for PRAISE, and ρ^0_{best} for PRAISE0, are encircled in green. Acceptable ρ values are all ρ for which the $\widetilde{RMSE}$ curve is below 1, denoted by the dashed red line. For $\lambda = 0.5$, the best algorithm A^* is SAS.

This shows that with running only 20% of the problem size—and accounting for this runtime overhead in the error—I can deliver reasonable improvements.

I also included the results for PRAISE0, the probing-based approach that skips the extrapolation step and selects the best algorithm on the sample. The optimal sampling ratio in this case is $\rho^0_{\text{best}} = 0.2$, with $\rho = 0.25$ also acceptable. However, note that the improvements of PRAISE0 over the single algorithm are marginal, as the $\widetilde{RMSE}$ value is very close to 1.

A notable conclusion to be drawn from comparing PRAISE and PRAISE0 in Figure 6.4 is that using the algorithm properties to predict the algorithm performance on the full instance has an impact on the prediction quality, improving it considerably. This is especially visible for small sample sizes, where the time overhead is still manageable. For large sample sizes, PRAISE0 might be just as accurate as PRAISE, but the time overhead is too large for it to matter.

The same analysis is performed for all λ values, and the results are displayed in Table 6.2: the optimal ratio, as well as the range of acceptable ratios, for both PRAISE and PRAISE0. The results show that PRAISE can deliver good results for a wider range of sampling ratios—covering the lower end better than PRAISE0. This is also true for quality over λ values. The results in Table 6.2 reinforce the point that using the algorithm properties increases prediction quality.

Table 6.2: Optimal values and acceptable ranges of sampling ratios per λ. The superscript 0 denotes the PRAISE0 method, included here for comparison, which skips the extrapolation step. If there are no acceptable values, this is denoted by a horizontal bar.

λ	ρ^0_{best}	ρ_{best}	ρ^0_{min}	ρ^0_{max}	ρ_{min}	ρ_{max}
0.0	0.10	0.10	–	–	–	–
0.1	0.35	0.05	–	–	–	–
0.2	0.30	0.15	–	–	0.15	0.2
0.3	0.25	0.15	–	–	0.05	0.25
0.4	0.20	0.15	0.2	0.25	0.05	0.25
0.5	0.20	0.15	0.2	0.25	0.1	0.25
0.6	0.25	0.20	0.2	0.25	0.1	0.3
0.7	0.25	0.30	0.2	0.3	0.05	0.45
0.8	0.10	0.30	–	–	0.25	0.35
0.9	0.65	0.45	–	–	0.15	0.65
1.0	0.95	0.95	0.35	0.95	0.05	0.95

6.3.2 PREDICTION EVALUATION

In this section, the PRAISE approach is evaluated by looking at prediction accuracy, as well as comparing the cost error to other approaches: a single algorithm, as well as a random selection. The optimal sampling ratios found in Section 6.3.1 were used for each λ. Alternatively, the same value of ρ could be used over all λ values—a value inside the range of acceptable values for most λ, e.g., 0.25.

Figure 6.5 shows that the accuracy of my approach increases with λ. This can be explained by the fact that the impact of the time overhead decreases with λ, and thus larger sample sizes can be used. A larger sample is more representative of the problem instance, which means higher prediction accuracy.

The plot highlights one more time the usefulness of algorithm properties, as PRAISE0 is less accurate for all values of λ, except the extremes. For $\lambda = 0$, only the greedy algorithms are selected; as explained when evaluating MALAISE (cf. Section 5.3), the time differences between the greedy algorithms are essentially random; since they have the same time complexity, extrapolation does not aid in discriminating between their performance. At the other end, for $\lambda = 1$, the time overhead is inconsequential; therefore, any sample size can be used, with increasing accuracy; as the optimal ratio was determined to be 0.95, the sample is almost as big as the problem instance, therefore the knowledge of scaling behavior is irrelevant.

Next, the mispredictions are quantified by calculating the prediction error $\widetilde{MSE}$, which includes the time cost of probing (cf. Equation 6.10), and the approach is compared to a random selection, as well as a single algorithm, using the $\widetilde{RMSE}$ metric (cf. Equation 6.11). The results are plotted in Figure 6.6, for both PRAISE and PRAISE0. Note that an $\widetilde{RMSE}$ value

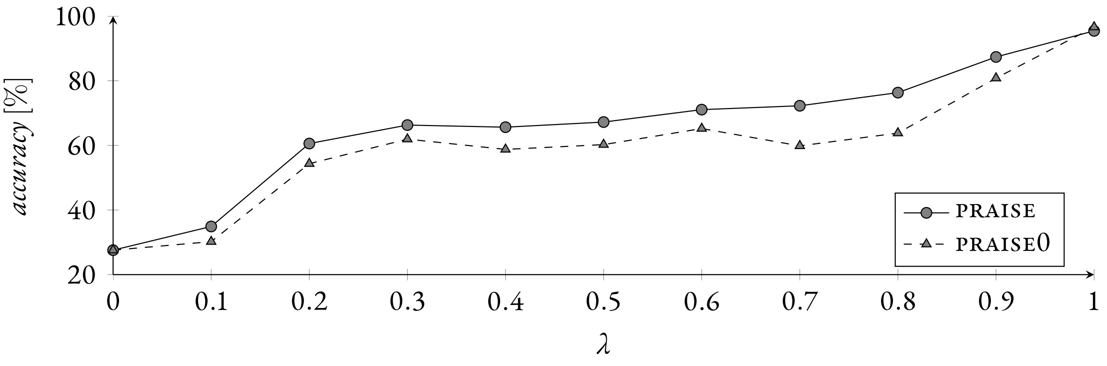


Figure 6.5: Accuracy of PRAISE for different λ values.

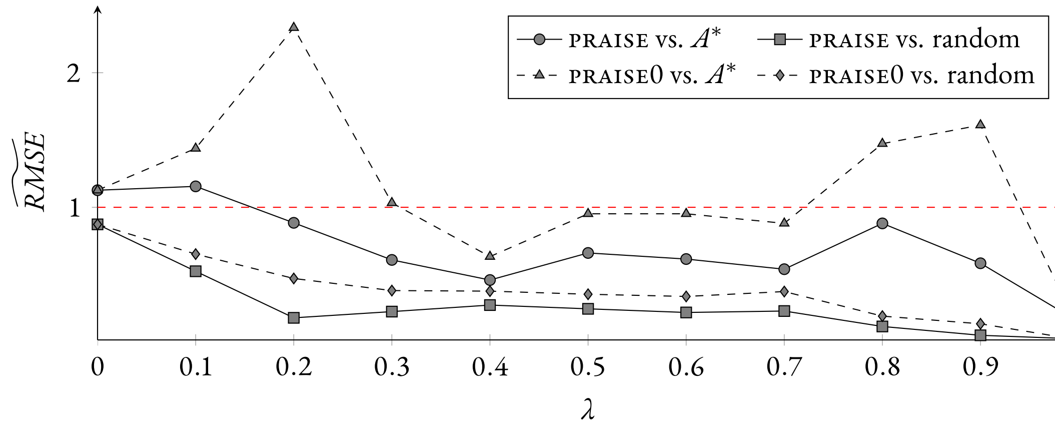


Figure 6.6: $\widetilde{RMSE}$ comparison of PRAISE to random selection and best pure algorithm for different λ values.

below 1 implies that the PRAISE approach is better than its counterpart in the comparison.

The comparison to the random selection shows that both PRAISE and PRAISE0 are always better than the random selection approach, up to 2 orders of magnitude. Moreover, the relative error decreases with increasing λ values.

Similarly, the PRAISE approaches are compared to 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. Figure 5.4). Figure 6.6 shows that PRAISE outperforms the single best algorithm for almost all λ values, with decreasing error over λ. The only exceptions are $\lambda = 0$ and $\lambda = 0.1$, where the time objective is either the only objective, or has a much higher importance than the welfare objective. This results in an equally high impact of the probing overhead, which cannot be amortized by prediction accuracy. Therefore, a single algorithm is preferable when the time objective is valued very highly.

In Figure 6.6, the impact of including knowledge of algorithm properties in the prediction can be observed more clearly: the PRAISE0 approach outperforms the single best algorithm only for mid-range λ values (between 0.4 and 0.7, where time and welfare are balanced), as well as for $\lambda = 1$, where the time overhead is not relevant. Furthermore, at the lower and upper end, PRAISE0 even exhibits two spikes where the relative error is unusually high compared to the single best algorithm, specifically at $\lambda = 0.2$ and $\lambda = 0.9$. Those are the points where extrapolation based on algorithm properties makes the biggest difference. Those are also the points where changing the weight of each objective effects big changes in the distribution of the best algorithms (cf. Figure 5.4), similar to an inflection point. For example, for $\lambda = 0.1$, the execution time is still important enough that the greedy algorithms are selected as best in ≈ 20% of the cases, even though they have poor solution quality. Then for $\lambda = 0.2$, the greedy algorithms are selected as best altogether is only ≈ 2.5% of the cases. Comparable circumstances are found at $\lambda = 0.9$, where the speed of simulated annealing algorithms is not sufficiently important anymore to outperform the slower, more accurate hill climbing algorithms; on the other hand, the slower CASANOVA(S) algorithms gain an advantage.

In summary, the evaluation showed that the proposed PRAISE approach accurately predicts the best algorithms for most values of λ—aided by the knowledge of algorithm properties—and outperforms the single best algorithms in these cases, even when including the time overhead of probing in the comparison.

The approach does not perform well for small λ values (high importance of time objective), where the time overhead renders the approach irrelevant, and a single algorithm is preferable.

6.3.3 OVERHEAD MITIGATION

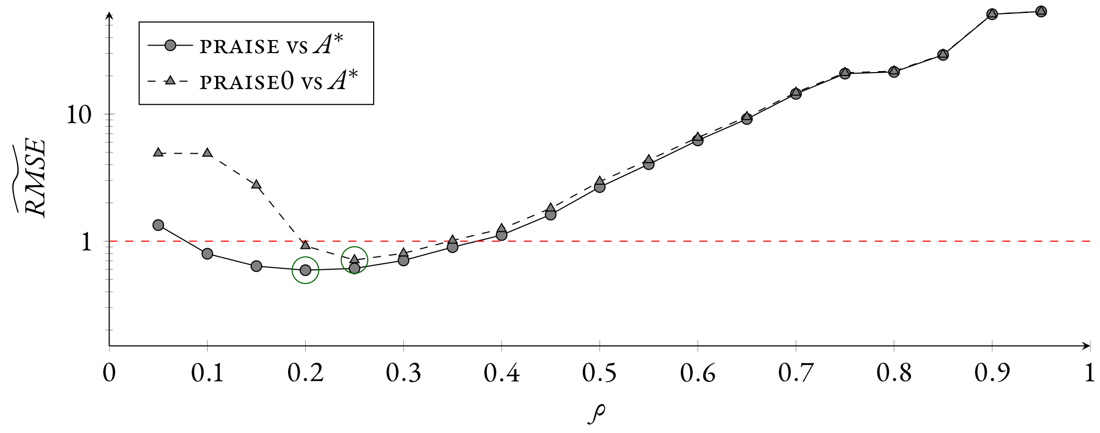
PRAISE yields reasonable results, even though the probing phase incurs a time overhead. Nevertheless, adding more algorithms to the portfolio increases this overhead, which could lead to a situation where PRAISE cannot improve upon a single algorithm. In this section, I explore ideas for reducing the overhead.

The first idea I propose is to parallelize the probing phase, by running the algorithms in parallel on the sample. This means that the overhead will be equal to the runtime of the slowest algorithm (cf. Equation 6.14), rather than the total runtime of the algorithms. The reasoning behind this idea is that running all the algorithms on the sample does not require a large amount of resources, be it memory (12 algorithms means that the memory required is within a factor of 12ρ of the memory required for the full instance) or CPUs (equal to the number of algorithms).

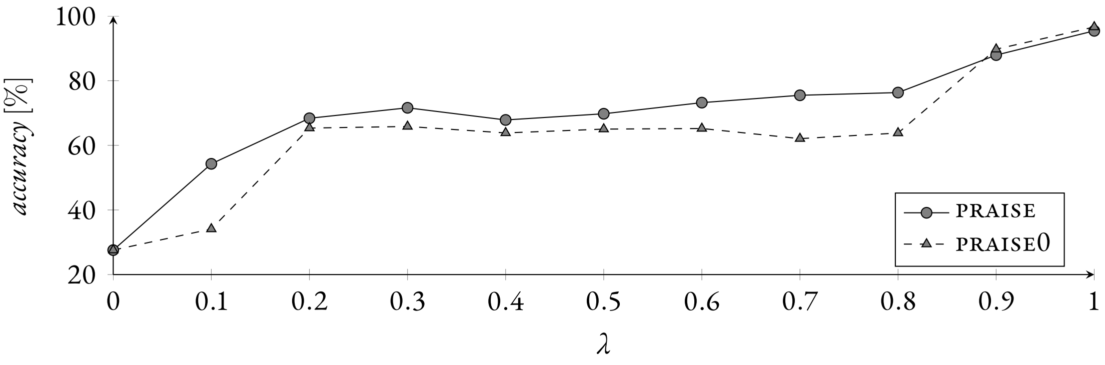
$$\tau\prime(X,\rho) = \max_{A\in\mathcal{A}} t(\rho X, A) \tag{6.14}$$

Figure 6.7 shows the effect that parallelizing the probing has on the sample size study and the prediction quality—measured by accuracy and $\widetilde{RMSE}$. Note that both PRAISE and PRAISE0 deliver better results—with more visible effects on PRAISE0, where the overhead was more problematic. However, the parallelization does not manage to completely overcome the impact of probing for $\lambda = 0$ and $\lambda = 0.1$.

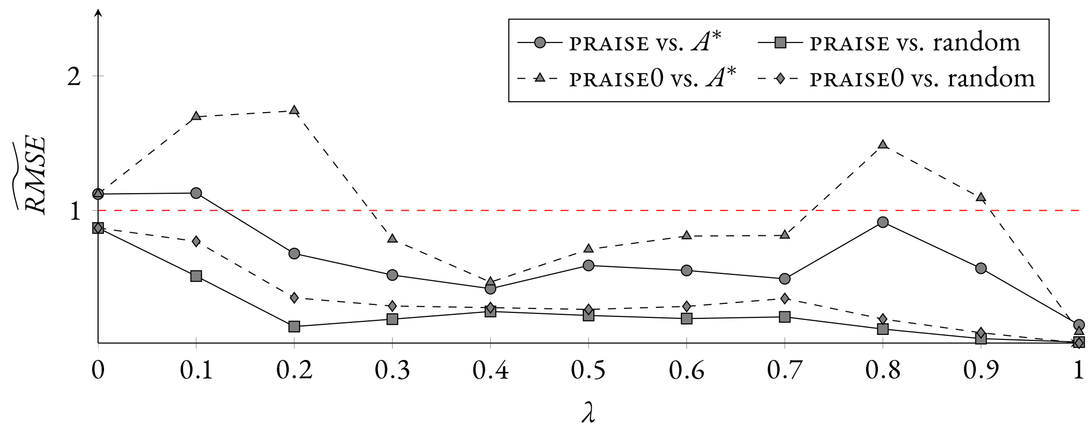
Another idea for improvement is to reduce the algorithm space in some cases, e.g. only select from the greedy algorithms for $\lambda = 0$, or exclude the CASANOVA(S) and HILL1(S) algo-



(a) $\widetilde{RMSE}$ over sampling ratio ρ for $\lambda = 0.5$.



(b) Accuracy for different λ values.



(c) $\widetilde{RMSE}$ comparison of PRAISE to random selection and best pure algorithm for different λ values.

Figure 6.7: Evaluation of PRAISE with parallel probing.

rithms for small values of λ, since they are too slow to be selected as best in those cases. Further analysis is required to evaluate the impact on the overhead and, implicitly, on $\widetilde{RMSE}$.

6.4 PRAISE VS. MALAISE

In this thesis, I proposed two approaches for algorithm selection for combinatorial auctions. I showed that they both yield good results, by improving significantly upon the single algorithm approach. This section discusses the differences between the two proposed approaches, and highlights the benefits and drawback of each approach.

Although both approaches perform offline selection of heuristic algorithms for combinatorial auctions—and consider both time and welfare objectives to assess the best algorithm—, they are very different otherwise. The key differences are highlighted in Table 6.3.

MALAISE uses supervised learning to train a portfolio-wide model that can predict the best algorithm for every given problem instance, based on domain-specific features extracted from the problem instance. In contrast, PRAISE extracts probing information (time and welfare of each algorithm on a sample of the problem instance) without any knowledge of the problem domain, and then uses this information to predict the time and welfare of each algorithm on the instance (per-algorithm models). This information is then used to select the best algorithm using hand-crafted rules—the algorithm with the minimum predicted cost. Even

Table 6.3: Comparison of the two proposed approaches for algorithm selection.

	MALAISE	PRAISE
Features	static high-knowledge domain-specific	semi-static low-knowledge probing information
Portfolio	static heuristics	static heuristics
Selection	offline supervised learning	offline hand-crafted-rules
Performance	per-portfolio best algorithm	per-algorithm predicted time and welfare

though no domain knowledge is required, the PRAISE approach still calls for expertise regarding the algorithms.

As a result, the approaches have different strengths. PRAISE is more generalizable due to the domain-independent features, and more robust to portfolio changes—when adding a new algorithm, only its performance model needs to be created. In the case of MALAISE, the portfolio-wide model needs to be re-trained after every addition or removal of algorithms from the portfolio. Moreover, individual models need to be trained for each λ value.

Nevertheless, once the training is done, the selection for MALAISE is fast, and delivers better results than PRAISE. This is mostly due to the time overhead introduced by the probing phase in PRAISE.

The advantages and disadvantages of each approach are summarized in Table 6.4.

Table 6.4: Advantages and disadvantages of proposed algorithm selection approaches.

MALAISE	PRAISE
– domain knowledge for features	+ no domain knowledge – knowledge of algorithm properties
– expensive training – retraining after portfolio changes – per-λ models need to be trained	+ no training + robust to portfolio changes
+ fast selection + works for all λ values	– time overhead of probing – does not work for small λ values

Reasoning draws a conclusion and makes us grant the conclusion, but does not make the conclusion certain, nor does it remove doubt so that the mind may rest on the intuition of truth, unless the mind discovers it by the path of experience.

Roger Bacon

7 Conclusion

This chapter provides an overview of the achievements of this thesis, and draws a few directions of possible future research.

7.1 SUMMARY

The overarching goal of this thesis was to improve the flexibility, scalability, and efficiency of cloud resource allocation by employing market-based mechanisms.

This was motivated by a discernible trend towards market-driven allocation in Cloud Computing, coupled with a number of shortcomings identified in the state of the art. More specifically, I addressed the issue of inflexible fixed-price models, the challenge of regulating variable demand and supply, and the lack of fine-grained control for cloud customers over the combinations of resources desired.

This work is a step towards understanding the strengths and challenges of implementing market concepts in the cloud resource allocation. I assessed the fundamental requirements of a market-based cloud resource allocation, but found that the most suitable approach to deal with the identified shortcomings—double combinatorial auction—is impracticable due to intractability, on one hand, and to input-dependent heuristics, on the other hand.

Then the achievements of this thesis can be summarized in one simple sentence:

Making combinatorial auctions more usable in practice through meta-heuristic approaches.

To accomplish this, contributions spanning over several research areas were made. First, a model was devised that formalizes the allocation problem as a double combinatorial auction. Since the optimal algorithm is $\mathcal{NP}$-hard, heuristic approaches for solving the combinatorial auction were investigated. A significant contribution of this thesis was to harmonize such existing approaches under a single governing problem formulation, and then systematically compare them over a broad spectrum of test scenarios. To make this possible, a flexible approach for generating realistic input data was proposed. This is the first work on artificial data generation that considers the two-sided aspect of the combinatorial auction, and generates multi-unit multi-good bundles. Another unique characteristic is the ability to generate data that resembles realistic cloud bundles, by relying on information extracted from public cloud traces.

The systematic evaluation reasserted the premise that heuristic algorithms for combinatorial auctions yield highly input-dependent results. More importantly, no algorithm was found to perform best in all cases, either with respect to solution quality, or execution speed.

For practical use, it was necessary to increase the robustness and reliability of heuristic algorithms. I proposed achieving this through a meta-heuristic approach termed *algorithm selection*, which selects the best heuristic algorithm to be used for each input. Then, the central contributions of this thesis are two different approaches for algorithm selection. This is the first work, to my knowledge, to apply algorithm selection to heuristic algorithms for combinatorial auctions, and as a result consider both time and solution quality in the selection.

The first approach, MALAISE, uses supervised learning to train a model that can predict, for each input, which algorithm will perform best on the given input. To quantitatively define what it means to *perform best*, a flexible cost model was proposed for comparing and scoring algorithms with respect to both execution time and solution quality; the importance of these two objectives can be changed depending on user preferences. The learning process first extracts features that could be predictive of algorithm performance, specifically engineered for the combinatorial auction domain. Using the proposed input data generator, this approach was evaluated, and showed to be up to 2 orders of magnitude better than using a single heuristic algorithm.

The second approach for algorithm selection, PRAISE, aims for a wider applicability, and as such is based on probing information rather than domain-specific features. For each input, I proposed probing the search space and predicting the best algorithm based on this information. The probing consists of running all the algorithms on a random sample of the input; then the results are combined with algorithm properties that define scaling behavior (determined theoretically when possible, or empirically), in order to predict each algorithm's performance on the full input. The same cost model is then employed to determine the best algorithm. The evaluation showed that this approach also improves upon the single heuristic algorithm approach, with one notable exception: when execution time is significantly more important than solution quality, the time overhead of probing cannot be overcome.

In conclusion, in this thesis I showed that heuristic algorithms for combinatorial auctions can be used reliably in practice through the proposed approaches for algorithm selection. These methods significantly outperform the single algorithm approach, with MALAISE yielding overall better results.

7.2 OUTLOOK

I believe that the work presented in this thesis can provide a strong foundation for future research in several directions.

Although the proposed model for the resource allocation problem as a combinatorial auction is generic enough, there may be some scenarios where the assumptions made are not valid or sufficient to capture certain characteristics. Future work should consider adding more complexity to the model and changing the auction mechanism accordingly. For example, more complex bidder preferences could be modeled using the XOR bidding language, allowing customers to express their interest in obtaining one of multiple possible bundles. Another promising direction is to investigate other payment rules and corresponding business models for the auctioneer, for example to ensure truthfulness of bidders and sellers.

There is immense research potential for the algorithm selection approaches. For the machine learning approach, future research should consider adding a feedback loop that enables re-training the models with incoming data. This is challenging due to the expensive training, which requires running all the algorithms on each training instance. As a result, it is necessary to purposefully and effectively select the instances which will be included in the training data (possibly based on similarity with existing data), and to find a balance with regards to when the models should be re-trained. Furthermore, other machine learning approaches could be considered. For example, regression can be used to predict individual algorithm performance, which can then inform the selection of the best algorithm.

More heuristic algorithms can be added to the existing portfolio (e.g., an evolutionary algorithm). This would require re-training in the machine learning-based approach, and characterizing the scaling behavior of the new algorithms in the probing-based approach. For the probing-based selection, a challenge related to adding more algorithms concerns the corresponding increase in probing overhead. Possibilities for parallelization should be investigated.

Another idea for future research is to explore other cost functions for determining the best algorithm with respect to both time and solution quality. Since execution time and solution quality are not directly comparable, in this thesis, this was dealt with by normalizing each objective before including them in the cost model. Another idea would be to express both objectives using a common denominator: money. That would require models for translating execution time and solution quality into monetary value.

On a different note, the input data generator can be extended with other pricing models or cloud traces, to widen the range of test cases.

Finally, although the proposed algorithm selection approaches target combinatorial auctions, they are generic enough to be applied to other areas as well. Investigating their applicability to other problem domains is a rich area of research, and it will require domain-specific feature engineering for the machine learning-based approach, as well as appropriate test data in both cases.

A
Algorithm Selection Survey

Kotthoff (2014) surveyed the existing research on algorithm selection and collected data[1] on yearly number of publication on the topic (cf. Figure A.1), types of algorithm portfolios used (cf. Figure A.2(a)), types of selection (cf. Figure A.2(b)), and features used (cf. Table A.1).

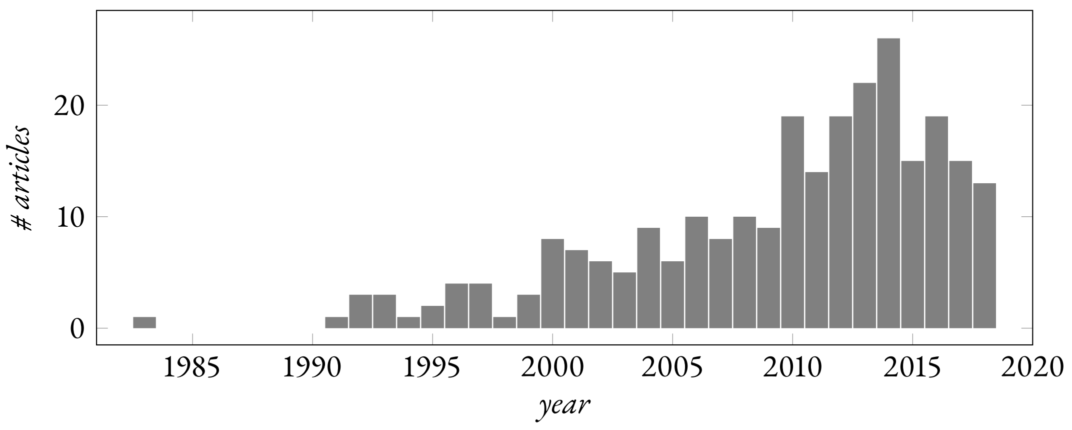
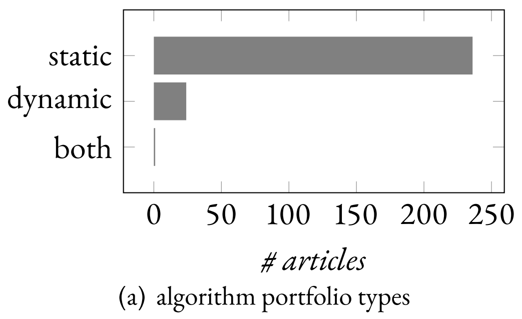


Figure A.1: Yearly evolution of the number of publications on algorithm selection.

[1] data available at `http://larskotthoff.github.io/assurvey/`



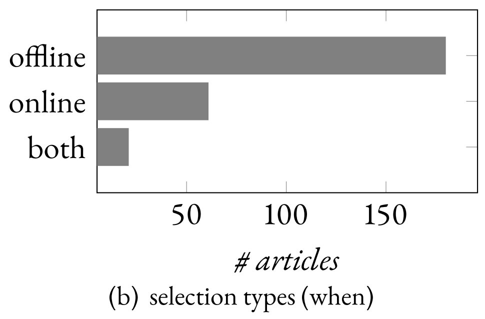


Figure A.2: Number of publications using the different types of algorithms portfolios (left) and the different selection approaches based on when the selection is done (right).

Table A.1: Classification of features used in state-of-the-art algorithm selection based on two criteria: domain knowledge needed and when (and how) are the features computed. Popularity of each type of features from a survey of 263 articles.

Features	**Knowledge**	**When & how**	**Usage (% articles)**
instance features	high	static	65 %
past performance	low	static	26 %
probing	low	semi-static	8.7 %
search statistics	low	dynamic	7.2 %
runtime performance	low	dynamic	2 %
problem domain features	high	static	2 %
algorithm features	low	static	1.1 %
algorithm parameters	low	static	0.8 %

Table A.2: Categorization of related work on algorithm selection for combinatorial auctions.

Citation	**Features**	**Portfolio**	**Selection**	**Performance model**
Leyton-Brown et al. (2003)	high-knowledge, static	static	offline	per-algorithm
Fitzgerald & O'Sullivan (2017)	low-knowledge, static	static	online	per-portfolio
Stern et al. (2010)	high-knowledge, static, dynamic	static	offline, online	per-portfolio

B
Pseudocode

This chapter is largely devoted to the pseudocode corresponding to the algorithms that prioritize sellers in the search process (denoted by the suffix -s).

Algorithm 8 Greedy algorithm with seller priority.

```
1: function GREEDY1S($n, m, l, b, r, a, s$)
2:     compute $f_g, f'_g, \forall g \in \mathcal{G}$ with method 1          ▷ relevance factors
3:     compute $d_i, \forall i \in \mathcal{U}$ and $d'_j, \forall j \in \mathcal{P}$      ▷ bid and ask densities
4:     sort bids descendingly by $d$
5:     sort asks ascendingly by $d'$
6:     $i \leftarrow 1; j \leftarrow 1$
7:     while $i \leq n$ and $j \leq m$ do
8:         if $r_{ig} \leq s_{jg}, \forall g \in \mathcal{G}$ and $b_i \geq a_j$ then   ▷ if ask j can satisfy bid i
9:             $x_i \leftarrow 1; y_{ij} \leftarrow 1$                    ▷ allocate resources offered by
                                                            seller j to bidder i
10:            $j \leftarrow j + 1$                               ▷ move to next seller
11:        $i \leftarrow i + 1$                                   ▷ move to next bidder
12:    return $(x, y)$
```

Algorithm 9 Function that returns the neighbor in the solution space of a given solution, by changing the ask ordering. Used in HILL1S.

```
1: function NEIGHBOR1S($x, y$)
2:     move ask $j$ to beginning of list                    ▷ $j \leftarrow$ critical $j + 1, m$
3:     return GREEDY1S($n, m, l, b, r, a, s$)$[6 : 12]$       ▷ apply on new ordering
                                                            lines 6–12 in Algorithm 8
```

Algorithm 10 Function that returns the neighbor in the solution space of a given solution, by toggling a random ask j. Used in HILL2S.

```
1: function NEIGHBOR2S($x, y$)
2:     $j \leftarrow \text{random}(1, m)$                                   ▷ randomly select ask
3:     if $y_{ij} = 0, \forall i \in \mathcal{U}$ then                         ▷ if ask j not already allocated
4:         for $i \leftarrow 1, n$ do                                     ▷ greedy-like search for bid
5:             if $x_i = 0$ then                                   ▷ if bid i not allocated
6:                 if $r_{ig} \leq s_{jg}, \forall g \in G$ and $b_i \geq a_j$ then     ▷ if ask j can satisfy bid i
7:                     $x_i \leftarrow 1; y_{ij} \leftarrow 1$                       ▷ match bid i and ask j
8:                     break                                   ▷ stop—match found
9:     return $(x, y)$                                              ▷ return neighboring solution
```

Algorithm 11 Function that returns the neighbor in the solution space of a given solution, by toggling a random x_i. Used in SA.

```
1: function NEIGHBOR_SA($x, y$)
2:     $i \leftarrow \text{random}(1, n)$                                   ▷ randomly select bid
3:     if $x_i = 1$ then                                             ▷ if bid i already allocated
4:         $x_i \leftarrow 0; y_{ij} \leftarrow 0, \forall j \in \mathcal{P}$                      ▷ deallocate bid i
5:     else                                                          ▷ if bid i not allocated
6:         for $j \leftarrow 1, m$ do                                     ▷ greedy-like search for ask
7:             if $y_{qj} = 0, \forall q \in U$ then                        ▷ if ask j not allocated
8:                 if $r_{ig} \leq s_{jg}, \forall g \in G$ and $b_i \geq a_j$ then     ▷ if ask j can satisfy bid i
9:                     $x_i \leftarrow 1; y_{ij} \leftarrow 1$                       ▷ match bid i and ask j
10:                    break                                   ▷ stop—match found
11:    return $(x, y)$                                              ▷ return neighboring solution
```

Algorithm 12 Function that returns the neighbor in the solution space of a given solution, by toggling a random ask *j*. Used in SAS.

```
1:  function NEIGHBOR_SAS($(x, y)$)
2:      $j \leftarrow$ random$(1, m)$                          ▷ randomly select ask
3:      if $\exists i \in \mathcal{U}, y_{ij} = 1$ then                ▷ if ask j already allocated
4:          $x_i \leftarrow 0; y_{ij} \leftarrow 0, \forall j \in \mathcal{P}$             ▷ deallocate ask j
5:      else                                              ▷ if ask j not allocated
6:          for $i \leftarrow 1, n$ do                              ▷ greedy-like search for bid
7:              if $x_i = 0$ then                             ▷ if bid i not allocated
8:                  if $r_{ig} \leq s_{jg}, \forall g \in G$ and $b_i \geq a_j$ then   ▷ if ask j can satisfy bid i
9:                      $x_i \leftarrow 1; y_{ij} \leftarrow 1$                 ▷ match bid i and ask j
10:                     break                             ▷ stop—match found
11:     return $(x, y)$                                     ▷ return neighboring solution
```

Algorithm 13 Casanova algorithm with seller priority.

```
1:  function CASANOVAS($n, m, l, b, r, a, s$)
2:      for $try \leftarrow 1, maxTries$ do                      ▷ restart search try times
3:          $(x, y) \leftarrow 0$                                 ▷ empty allocation
4:          sort bids descendingly by density
5:          sort asks ascendingly by score
6:          for $step \leftarrow 1, maxSteps$ do
7:              if $wp >$ rand$(0, 1)$ then                      ▷ with walk probability wp
8:                  $(x, y) \leftarrow$INSERT_S(random unallocated ask $j, x, y$)
9:              else if age(first unallocated ask) $>$ age(second unallocated ask) then
10:                 $(x, y) \leftarrow$INSERT_S(first unallocated ask, $x, y$)
11:             else if $np >$ rand$(0, 1)$ then                 ▷ with novelty probability np
12:                 $(x, y) \leftarrow$INSERT_S(second unallocated ask, $x, y$)
13:             else
14:                 $(x, y) \leftarrow$INSERT_S(first unallocated ask, $x, y$)
15:             if $step > \theta_r$ and no improvement in last $\theta_r/2$ steps then
16:                 break                                     ▷ soft restart
17:     return best $(x, y)$ found
```

Algorithm 14 Neighbor function for CASANOVAS algorithm. The function adds an unallocated ask and best matching bid to the current solution.

```
1:  function INSERT_S($j, x, y$)
2:      for $i \leftarrow 1, n$ do                                   ▷ greedy-like search for bid
3:          if $x_i = 0$ then                                         ▷ i not allocated
4:              if $r_{ig} \leq s_{jg}, \forall g \in \mathcal{G}$ and $b_i \geq a_j$ then      ▷ if ask j can satisfy bid i
5:                  $x_i \leftarrow 1; y_{ij} \leftarrow 1$                       ▷ match bid i and ask j
6:                  reset age($j$)
7:                  return $(x, y)$                                   ▷ match found
8:      for $i \leftarrow 1, n$ do                                   ▷ no match, restart search
9:          if $\exists q \in \mathcal{P}, y_{iq} = 1$ then                        ▷ i already allocated
10:             if $r_{ig} \leq s_{jg}, \forall g \in \mathcal{G}$ and $b_i \geq a_j$ then      ▷ if ask j can satisfy bid i
11:                 if $a_j < a_q$ then                               ▷ if ask j improves allocation
12:                     $x_i \leftarrow 1; y_{ij} \leftarrow 1$                   ▷ match bid i and ask j
13:                     $y_{iq} \leftarrow 0$                             ▷ undo allocation of ask q
14:                     reset age($j$)
15:                     return $(x, y)$                               ▷ match found
16:     return $(x, y)$                                               ▷ no match found
```

C
Datasets Parameters

Listing C.1: Parameters used to generate the D1 dataset (46656 instances).

```bash
#!/bin/bash
## this script assumes all models have 3 resource types
V_BIDS_N=1000
V_BIDS_MODEL="models/model-uniform3* models/model-hotspots3_8*
    models/model-part-000*0-of-*clustered*"
V_BIDS_BINNING_TYPE='regular'
V_BIDS_BINNING_COUNTS='16 8'
V_BIDS_DOMAIN='[128,128,128]'
V_ASKS_N=1000
V_ASKS_MODEL=${V_BIDS_MODEL}
V_ASKS_BINNING_TYPE=${V_BIDS_BINNING_TYPE}
V_ASKS_BINNING_COUNTS='8'
V_ASKS_DOMAIN=${V_BIDS_DOMAIN}
V_SLOPE='[1,1,1] [1,0.9,0.5] [1.1,1,0.9]'
V_FIXED='[0,0,0] [0,0,0.1]'
V_DIST_MEANS='0.0 0.1 0.5'
V_A_SIGMA='0.05 0.1 0.25'
V_B_SIGMA='0.05 0.1 0.25'
```

Listing C.2: Parameters used to generate the D2 dataset (11664 instances).

```
#!/bin/bash
## this script assumes all models have 4 resource types
V_BIDS_N=1000
V_BIDS_MODEL="models/model-uniform4* models/model-hotspots4_16*
    models/model-bitbrains*clustered*"
V_BIDS_BINNING_TYPE='regular'
V_BIDS_BINNING_COUNTS='16 8'
V_BIDS_DOMAIN='[128,128,128,128]'
V_ASKS_N=1000
V_ASKS_MODEL=${V_BIDS_MODEL}
V_ASKS_BINNING_TYPE=${V_BIDS_BINNING_TYPE}
V_ASKS_BINNING_COUNTS='8'
V_ASKS_DOMAIN=${V_BIDS_DOMAIN}
V_SLOPE='[1,1,1,1] [1,0.9,0.5,1] [1.1,1,0.9,1]'
V_FIXED='[0,0,0,0] [0,0,0.1,0]'
V_DIST_MEANS='0.0 0.1 0.5'
V_A_SIGMA='0.05 0.1 0.25'
V_B_SIGMA='0.05 0.1 0.25'
```

Listing C.3: Parameters used to generate the D3 dataset (69984 instances).

```
#!/bin/bash
## this script assumes all models have 3 resource types
V_BIDS_N=10000
V_BIDS_MODEL="models/model-uniform3* models/model-hotspots3_8*
    models/model-part-000*0-of-*clustered*"
V_BIDS_BINNING_TYPE='regular'
V_BIDS_BINNING_COUNTS='16 8 32'
V_BIDS_DOMAIN='[128,128,128]'
V_ASKS_N=10000
V_ASKS_MODEL=${V_BIDS_MODEL}
V_ASKS_BINNING_TYPE=${V_BIDS_BINNING_TYPE}
V_ASKS_BINNING_COUNTS='8'
V_ASKS_DOMAIN=${V_BIDS_DOMAIN}
V_SLOPE='[1,1,1] [1,0.9,0.5] [1.1,1,0.9]'
V_FIXED='[0,0,0] [0,0,0.1]'
V_DIST_MEANS='0.0 0.1 0.5'
V_A_SIGMA='0.05 0.1 0.25'
V_B_SIGMA='0.05 0.1 0.25'
```

References

Aarts, E., Korst, J., & Michiels, W. (2014). *Simulated Annealing*, (pp. 265–285). Springer US: Boston, MA. doi:10.1007/978-1-4614-6940-7_10.

Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A., & Tsafrir, D. (2013). Deconstructing Amazon EC2 spot instance pricing. *ACM Trans. Econ. Comput.*, 1(3), 16:1–16:20. doi:10.1145/2509413.2509416.

Amazon (2017). Amazon ec2 spot instaces. https://aws.amazon.com/ec2/spot/. (Accessed: October 7, 2019).

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Randy, K., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., & Zaharia, M. (2009). Above the clouds: A Berkeley view of Cloud Computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28(13), 2009.

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017). *Serverless Computing: Current Trends and Open Problems*, (pp. 1–20). Springer Singapore: Singapore. doi:10.1007/978-981-10-5026-8_1.

Beck, J. & Freuder, E. (2004). Simple rules for low-knowledge algorithm selection. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, (pp. 50–64). doi:10.1007/978-3-540-24664-0_4.

Ben-Yehuda, O. A., Ben-Yehuda, M., Schuster, A., & Tsafrir, D. (2014). The rise of RaaS: the resource-as-a-service cloud. *Communications of the ACM*, 57(7), 76–84. doi:10.1145/2627422.

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

Bitbrains IT Services Inc. (2014). GWA-T-12 fastStorage trace. http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains. Trace analysis by: Siqi Shen, Vincent van Beek, Alexandru Iosup (Accessed: October 7, 2019).

Borrett, J. E. & Tsang, E. P. K. (2009). Adaptive constraint satisfaction: The quickest first principle. In C. L. Mumford & L. C. Jain (Eds.), *Computational Intelligence: Collaboration, Fusion and Emergence* (pp. 203–230). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-01799-5_7.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12), 1695–1724. doi:10.1057/jors.2013.71.

Buyya, R., Yeo, C. S., & 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* (pp. 5–13).: IEEE. doi:10.1109/HPCC.2008.172.

Commission, I. S. O. E. et al. (2014). Information technology–Cloud Computing–overview and vocabulary. *International Standard*, 17788.

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

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition. ISBN: 0262033844, 9780262033848.

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

Deb, K. (2014). Multi-objective optimization. In E. K. Burke & G. Kendall (Eds.), *Search methodologies* (pp. 403–449). doi:10.1007/978-1-4614-6940-7_15.

Elsayed, S. A. M. & Michel, L. (2011). Synthesis of search algorithms from high-level cp models. In J. Lee (Ed.), *Principles and Practice of Constraint Programming – CP 2011* (pp. 256–270). Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 9783642237867.

Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, M., & Newling, T. (2004). *Patterns: service-oriented architecture and web services*. IBM Corporation, International Technical Support Organization.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems* (pp. 2962–2970).: Curran Associates, Inc. https://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning (Accessed: October 7, 2019).

Fitzgerald, T. & O'Sullivan, B. (2017). Analysing the effect of candidate selection and instance ordering in a realtime algorithm configuration system. In *Proceedings of the Symposium on Applied Computing* (pp. 1003–1008).: ACM. doi:10.1145/3019612.3019718.

Fujishima, Y., Leyton-Brown, K., & Shoham, Y. (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of the 16th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'99 (pp. 548–553). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. http://dl.acm.org/citation.cfm?id=1624218.1624297 (Accessed: October 7, 2019).

Gomes, C. P., Selman, B., & Crato, N. (1997). Heavy-tailed distributions in combinatorial search. In G. Smolka (Ed.), *Principles and Practice of Constraint Programming-CP97* (pp. 121–135). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/BFb0017434.

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

Graham, C., Ng, F., Singh, T., Gupta, N., Nag, S., & Roth, C. (2018). Forecast analysis: Public cloud services, worldwide, 2Q18 update. *Gartner Inc.* https://www.gartner.com/en/newsroom/press-releases/2018-09-12-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2019 (Accessed: October 7, 2019).

Grozev, N. & Buyya, R. (2014). Inter-cloud architectures and application brokering: Taxonomy and survey. *Software—Practice and Experience*, 44(3), 369–390. doi:10.1002/spe.2168.

Gudu, D. (2019a). ca-ingen: Combinatorial Auctions input GEnerator with realistic bundles for cloud resources. https://github.com/dianagudu/ca-ingen. (Accessed: October 7, 2019).

Gudu, D. (2019b). ca-portfolio: a portfolio of heuristic algorithms for double combinatorial auctions. https://github.com/dianagudu/ca-portfolio. (Accessed: October 7, 2019).

Gudu, D., Hardt, M., & Streit, A. (2016). On MAS-based, Scalable Resource Allocation in Large-scale, Dynamic Environments. In *2016 Intl IEEE Conferences on 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)* (pp. 567–574).: IEEE. doi:10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0097.

Gudu, D., Hardt, M., & Streit, A. (2018a). Combinatorial Auction Algorithm Selection for Cloud Resource Allocation Using Machine Learning. In M. Aldinucci, L. Padovani, & M. Torquati (Eds.), *Euro-Par 2018: Parallel Processing* (pp. 378–391). Cham: Springer International Publishing. doi:10.1007/978-3-319-96983-1_27.

Gudu, D., Hardt, M., & Streit, A. (2019). A Unified Comparative Study of Heuristic Algorithms for Double Combinatorial Auctions: Locality-constrained Resource Allocation Problems. In J. van den Herik & A. P. Rocha (Eds.), *Agents and Artificial Intelligence* (pp. 3–22). Cham: Springer International Publishing. doi:10.1007/978-3-030-05453-3_1.

Gudu, D., Zachmann, G., Hardt, M., & Streit, A. (2018b). 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 - Volume 1: ICAART* (pp. 58–69).: INSTICC SciTePress. doi:10.5220/0006593900580069.

Guzek, M., Gniewek, A., Bouvry, P., Musial, J., & Blazewicz, J. (2015). Cloud brokering: Current practices and upcoming challenges. *IEEE Cloud Computing*, 2(2), 40–47. doi:10.1109/MCC.2015.32.

Hermann, W. (2016). Deutsche Börse Cloud Exchange gibt auf. *Computerwoche*. https://www.computerwoche.de/a/deutsche-boerse-cloud-exchange-gibt-auf,3223201 (Accessed: October 7, 2019).

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

Hoos, H. H. & Boutilier, C. (2000). Solving combinatorial auctions using stochastic local search. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence* (pp. 22–29).: AAAI Press. http://dl.acm.org/citation.cfm?id=647288.721095.

IBM (2015). Ilog cplex 12.6.3. http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud. (Accessed: October 7, 2019).

Jrad, F. (2014). *A service broker for Intercloud computing*. PhD thesis, Karlsruhe, Karlsruher Institut für Technologie (KIT). urn:nbn:de:swb:90-423243.

Kadioglu, S., Malitsky, Y., Sellmann, M., & Tierney, K. (2010). Isac –instance-specific algorithm configuration. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence* (pp. 751–756). Amsterdam, The Netherlands, The Netherlands: IOS Press. http://dl.acm.org/citation.cfm?id=1860967.1861114.

Khnaser, E., Leong, L., Toombs, D., Richard, S., Fritsch, J., Waite, A., Delory, P., Clayton, T., Siegfried, G., Meinardi, M., Murray, G., Simpson, N., & Davis, K. (2018). 2019 Planning Guide for Cloud Computing. *Gartner Inc.* https://www.gartner.com/doc/3891095/-planning-guide-cloud-computing (Accessed: October 7, 2019).

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.

Kothari, A., Sandholm, T., & Suri, S. (2004). Solving combinatorial exchanges: Optimality via a few partial bids. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04 (pp. 1418–1419). Washington, DC, USA: IEEE Computer Society. doi:10.1109/AAMAS.2004.248.

Kotthoff, L. (2014). Algorithm selection for combinatorial search problems: A survey. *AI Magazine*, 35(3), 48–60. doi:10.1609/aimag.v35i3.2460.

Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: A survey. In C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, & D. Pedreschi (Eds.), *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach* (pp. 149–190). Springer. doi:10.1007/978-3-319-50137-6_7.

Lai, K. (2005). Markets are dead, long live markets. *SIGecom Exchanges*, 5(4), 1–10. doi:10.1145/1120717.1120719.

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

Lehmann, D., Oćallaghan, L. I., & Shoham, Y. (2002). Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM (JACM)*, 49(5), 577–602. doi:10.1145/585265.585266.

Lemke, C., Budka, M., & Gabrys, B. (2015). Metalearning: a survey of trends and technologies. *Artificial Intelligence Review*, 44(1), 117–130. doi:10.1007/s10462-013-9406-y.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., & Shoham, Y. (2003). Boosting as a metaphor for algorithm design. In F. Rossi (Ed.), *Principles and Practice of Constraint Programming – CP 2003* (pp. 899–903). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-540-45193-8_75.

Leyton-Brown, K., Nudelman, E., & Shoham, Y. (2002). Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In P. Van Hentenryck (Ed.), *Principles and Practice of Constraint Programming - CP 2002* (pp. 556–572). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/3-540-46135-3_37.

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

Lucking-Reiley, D. (2000). Vickrey auctions in practice: From nineteenth-century philately to twenty-first-century e-commerce. *Journal of Economic Perspectives*, 14(3), 183–192. doi:10.1257/jep.14.3.183.

Luenberger, D. G. & Ye, Y. (2015). *Linear and nonlinear programming*, volume 228. Springer. ISBN: 9780387745039, 9780387745022, 9781441945044, doi:10.1007/978-3-319-18842-3.

Manvi, S. S. & Shyam, G. K. (2014). Resource management for infrastructure as a service (IaaS) in Cloud Computing: A survey. *Journal of network and computer applications*, 41, 424–440. doi:10.1016/j.jnca.2013.10.004.

Marler, R. T. & Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6), 369–395. doi:10.1007/s00158-003-0368-6.

Marshall, D. (2007). Understanding full virtualization, paravirtualization, and hardware assist. *VMWare White Paper*, (pp.1́7). https://www.vmware.com/de/techpapers/2007/understanding-full-virtualization-paravirtualizat-1008.html (Accessed: October 7, 2019.

Mashayekhy, L., Nejad, M. M., & Grosu, D. (2014). A two-sided market mechanism for trading big data computing commodities. In *Big Data (Big Data), 2014 IEEE International Conference on* (pp. 153–158).: IEEE. doi:10.1109/BigData.2014.7004225.

McAfee, R. P. & McMillan, J. (1987). Auctions and bidding. *Journal of Economic Literature*, 25(2), 699–738. http://www.jstor.org/stable/2726107.

Mell, P. & Grance, T. (2011). The NIST definition of Cloud Computing. doi:10.6028/NIST.SP.800-145.

Myerson, R. B. (1981). Optimal auction design. *Mathematics of Operations Research*, 6(1), 58–73. doi:10.1287/moor.6.1.58.

Myerson, R. B. & 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.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1), 48–49. https://www.pnas.org/content/36/1/48.

Nejad, M. M., Mashayekhy, L., & Grosu, D. (2015a). 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.

Nejad, M. M., Mashayekhy, L., & Grosu, D. (2015b). 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.

Nemhauser, G. L. & Wolsey, L. A. (1988). *Integer and Combinatorial Optimization*. New York, NY, USA: Wiley-Interscience. ISBN: 047182819X.

Neumann, D. G. (2007). *Market Engineering: A structured design process for electronic markets*. Universitätsverlag Karlsruhe, Karlsruhe. ISBN: 9783866440654.

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

Nisan, N., Roughgarden, T., Tardos, E., & Vazirani, V. V. (2007b). *Algorithmic Game Theory*. New York, NY, USA: Cambridge University Press. ISBN; 0521872820.

Padberg, M. & 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.

Parkes, D. C. (2001). *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, Philadelphia, PA, USA. ISBN: 0493131213.

Parkhill, D. F. (1966). *Challenge of the computer utility*. Addison-Wesley Publishing Company. ISBN: 0240507177.

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

Reiss, C., Wilkes, J., & Hellerstein, J. L. (2011). *Google cluster-usage traces: format + schema*. Technical report, Google Inc., Mountain View, CA, USA. Revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

Reiter, S. (1977). Information and performance in the (new) welfare economics. *The American Economic Review*, 67(1), 226–234. http://www.jstor.org/stable/1815908.

Rice, J. R. (1976). The algorithm selection problem. volume 15 of *Advances in Computers* (pp. 65–118). Elsevier. doi:10.1016/S0065-2458(08)60520-3.

Roberts, M. & Howe, A. E. (2006). Directing a portfolio with learning. In *AAAI 2006 Workshop on Learning for Search* (pp. 129–135).

Russell, S. & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition. ISBN: 0136042597, 9780136042594.

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.

Satterthwaite, M. (1993). The bayesian theory of the k-double auction. *The Double Auction Market: Institutions, Theories, and Evidence*, (pp. 99–123). doi:0.4324/9780429492532.

Satterthwaite, M. A. & Williams, S. R. (2002). The optimality of a simple market mechanism. *Econometrica*, 70(5), 1841–1863. http://www.jstor.org/stable/3082022.

Schnizler, B., Neumann, D., Veit, D., & 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.

Shen, S., v. Beek, V., & Iosup, A. (2015). Statistical characterization of business-critical workloads hosted in cloud datacenters. In *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (pp. 465–474). doi:10.1109/CCGrid.2015.60.

Shoham, Y. & Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations*. Cambridge University Press. ISBN: 0521899435.

Smith, D. & Anderson, E. (2018). Hype cycle for cloud computing, 2018. *Gartner Inc.* https://www.gartner.com/en/documents/3884671 (Accessed: October 7, 2019).

Smith, V. L. (1982). Microeconomic systems as an experimental science. *The American Economic Review*, 72(5), 923–955. http://www.jstor.org/stable/1812014.

Stergiou, K. (2008). Heuristics for dynamically adapting propagation. In *Proceedings of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial Intelligence* (pp. 485–489). Amsterdam, The Netherlands, The Netherlands: IOS Press. http://dl.acm.org/citation.cfm?id=1567281.1567388.

Stern, D., Samulowitz, H., Herbrich, R., Graepel, T., Pulina, L., & Tacchella, A. (2010). Collaborative expert portfolio management. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA*. http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1857.

Stokely, M., Winget, J., Keyes, E., Grimes, C., & Yolken, B. (2009). Using a market economy to provision compute resources across planet-wide clusters. In *Proceedings of the 2009 IEEE International Symposium on Parallel and Distributed Processing*, IPDPS '09 (pp. 1–8). Washington, DC, USA: IEEE Computer Society. doi:10.1109/IPDPS.2009.5160966.

Toosi, A. N., Khodadadi, F., & Buyya, R. (2016a). SipaaS: Spot instance pricing as a service framework and its implementation in openstack. *Concurrency and Computation: Practice and Experience*, 28(13), 3672–3690. doi:10.1002/cpe.3749.

Toosi, A. N., Vanmechelen, K., Khodadadi, F., & Buyya, R. (2016b). An auction mechanism for cloud spot markets. *ACM Trans. Auton. Adapt. Syst.*, 11(1), 2:1–2:33. doi:10.1145/2843945.

Vickrey, W. (1961). Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1), 8–37. doi:10.1111/j.1540-6261.1961.tb02789.x.

von Neumann, J., Morgenstern, O., & Rubinstein, A. (1944). *Theory of Games and Economic Behavior (60th Anniversary Commemorative Edition)*. Princeton University Press. ISBN: 9780691130613, http://www.jstor.org/stable/j.ctt1r2gkx.

Watzl, J. (2014). *A framework for exchange-based trading of cloud computing commodities*. PhD thesis, Ludwig Maximilian University of Munich (LMU). urn:nbn:de:bvb:19-168702.

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

Wolpert, D. H. & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67–82. doi:10.1109/4235.585893.

Wu, L. & Buyya, R. (2012). Service level agreement (sla) in utility computing systems. In V. Cardellini, E. Casalicchio, K. R. L. J. Castelo Branco, J. C. Estrella, & F. J. Monaco (Eds.), *Performance and dependability in service computing: Concepts, techniques and research directions* (pp. 1–25). IGI Global. doi:10.4018/978-1-60960-794-4.ch001.

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In A. Cimatti & R. Sebastiani (Eds.), *Theory and Applications of Satisfiability Testing – SAT 2012* (pp. 228–241). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-31612-8_18.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2007). Satzilla-07: the design and analysis of an algorithm portfolio for sat. In *International Conference on Principles and Practice of Constraint Programming* (pp. 712–727).: Springer. doi:10.1007/978-3-540-74970-7_50.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). Hydra-MIP: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)* (pp. 16–30). https://www.cs.ubc.ca/~hoos/Publ/XuEtAl11.pdf (Accessed: October 7, 2019).

Yun, X. & Epstein, S. L. (2012). Learning algorithm portfolios for parallel execution. In Y. Hamadi & M. Schoenauer (Eds.), *Learning and Intelligent Optimization* (pp. 323–338). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-34413-8_23.

Zhang, M., Ranjan, R., Haller, A., Georgakopoulos, D., Menzel, M., & Nepal, S. (2012). An ontology-based system for cloud infrastructure services' discovery. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)* (pp. 524–530).: IEEE.

Zurel, E. & Nisan, N. (2001). An efficient approximate allocation algorithm for combinatorial auctions. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, EC '01 (pp. 125–136). New York, NY, USA: ACM. doi:10.1145/501158.501172.