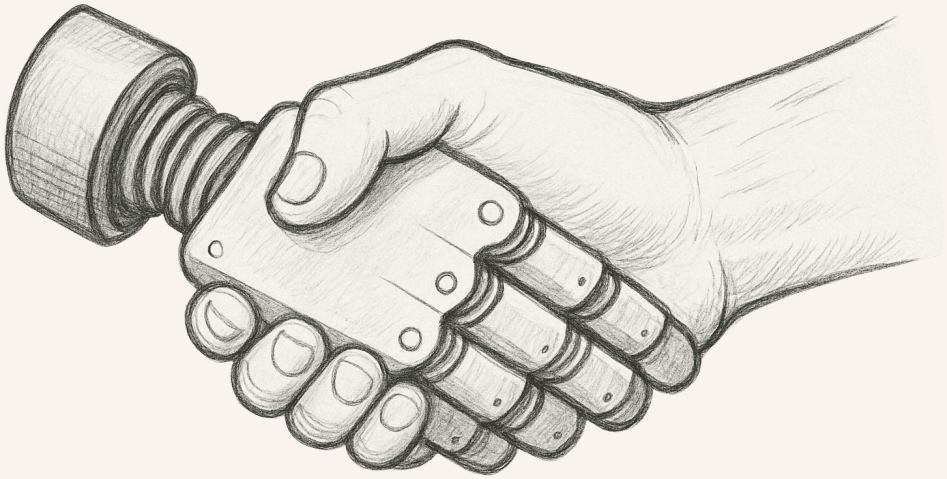
On the Effectiveness of Modeling Uncertain Constraint-Based Utility Functions with Quadratic Polynomials

With Applications in Autonomous Negotiations



Valdimar Miguel Þórsson

On the Effectiveness of Modeling Uncertain Constraint-Based Utility Functions With Quadratic Polynomials

With Applications in Autonomous Negotiations

by

Valdimar Miguel Þórsson

Daily Supervisor:	Catholijn M. Jonker
Responsible Supervisor:	Alexander Heinlein
External Committee member:	Robbert Fokkink
Project Duration:	January, 2025 - September, 2025
Faculty:	Institute of Applied Mathematics, TU Delft

TU Delft

Preface

The past two years have demanded more perseverance than I ever knew I had. Despite this, I am grateful for those challenging moments and the choices I made along the way. The hard times were more than balanced by the happy experiences I had during this journey. I will always cherish the memories I created here and the new relationships I formed. I am grateful for the time I spent with both of my football teams, as well as the many tennis games and burger-and-movie nights I enjoyed with my newly made friends.

This project has provided me with an extraordinary experience as an applied mathematician: I have literally applied mathematics. I did this in the context of autonomous negotiations, a topic I had not encountered before, and I am especially grateful to my daily supervisor, Catholijn, for introducing me to it. Exploring the field of Autonomous Negotiations has been extremely rewarding, and your guidance made this journey even more enjoyable. I would also like to thank my responsible supervisor, Alexander. Thank you for your assistance and for being available as my supervisor even though this field is as new to you as it is to me. Specifically I would like to thank you for providing me with new perspectives which, as it turned out, shaped the results. Lastly, I want to extend my thanks to Robbert Fokkink for accepting a place on my thesis committee and for the time you invested in this project.

I want to thank my family and friends back home. Special thanks to my mom, dad, and my siblings, who always support me in everything I do.

Finally, I'd like to thank my biggest supporter, my partner and my best friend, Karen. You have made this whole experience worth it.

Valdimar Miguel Þórsson
Delft, October 2025

Abstract

The curse of dimensionality poses a fundamental challenge in autonomous negotiations: as the number of issues and their interdependencies increase, exhaustive evaluation of the outcome space quickly becomes infeasible. This thesis addresses this problem by introducing a surrogate-based method that approximates uncertain hypercubic constraint-based utility functions with quadratic polynomials. An autonomous negotiation agent can then search for high-utility outcomes in this surrogate model. The research objective was to investigate how efficiently an autonomous negotiation agent can identify high-utility bids with this approach, and how this approach compares to linear approximations and established benchmark agents.

The main contributions of this thesis are threefold. First, it introduces a probabilistic complexity measure for these hypercubic functions, capturing how parameters such as dimensionality, constraint width, the number of constraints, and the number of issues interact to shape the function's complexity. Second, it develops a novel agent that leverages a regression model with quadratic basis functions to construct a surrogate model of a hypercubic constraint-based utility function. Third, it evaluates the agent through extensive experiments, demonstrating how performance scales with complexity. Following the steps outlined in this thesis, the performance of surrogate models can be directly compared.

The results demonstrate that the surrogate-based method is a promising approach, as the agent constructed in this thesis outperforms the agents from the 2014 Automated Negotiating Agent Competition which used similar scenarios as those considered in this thesis. These agents all have in common that they directly search the utility function as opposed to a surrogate model of it. Furthermore, the results indicate that simple basis functions, such as quadratic ones, enable the agent to reach the global maximum of its utility function in low-complexity hypercubic cases, with performance scaling reasonably well up to medium complexity. Beyond this point, however, performance deteriorates rapidly, clearly signaling the need for more expressive surrogate models.

Contents

Preface i

Abstract ii

Nomenclature vii

1 Introduction 1
- 1.1 Terminology . . . 2
  - 1.1.1 Negotiation domain . . . 2
  - 1.1.2 Negotiation protocol . . . 2
  - 1.1.3 Preference profiles . . . 3
- 1.2 Utility functions . . . 3
  - 1.2.1 Hypercubic constraint-based utility functions . . . 5
- 1.3 Problem definition . . . 6
- 1.4 Motivation . . . 7
- 1.5 Research Objective . . . 8
- 1.6 Thesis outline . . . 9

2 Literature review 10
- 2.1 Mechanism design for nonlinear utility functions . . . 10
  - 2.1.1 The Bidding-Based Protocol . . . 11
  - 2.1.2 Extensions of the Bidding-Based Protocol . . . 11
  - 2.1.3 Decomposing the utility function . . . 14
- 2.2 Agent design for nonlinear utility functions . . . 17
  - 2.2.1 Approximating the utility function . . . 17
  - 2.2.2 The 2014 Automated Negotiating Agent Competition . . . 19
- 2.3 Research Gap . . . 23

3 Mathematical background 25
- 3.1 The complexities of hypercubic functions . . . 25
  - 3.1.1 The dimensionality of a constraint . . . 29
  - 3.1.2 The width of a constraint . . . 29
  - 3.1.3 The number of issues . . . 30
  - 3.1.4 The number of constraints . . . 31
  - 3.1.5 Quantifying the complexity of a constraint-based utility function . . . 33
- 3.2 Sampling . . . 37
  - 3.2.1 Geometrics of n-dimensional spaces . . . 37
  - 3.2.2 Manhattan distance between constraints . . . 39
  - 3.2.3 Sampling evenly . . . 41
- 3.3 Regression . . . 43
  - 3.3.1 Maximum likelihood and least squares . . . 44
  - 3.3.2 Linear basis functions . . . 45
  - 3.3.3 Quadratic basis functions . . . 46
  - 3.3.4 Underfitting, Overfitting, and Regularization . . . 46
- 3.4 Optimization . . . 48

3.4.1 First-order methods . . . 49
3.4.2 Second-order methods . . . 49
3.5 Conclusion . . . 53

4 Method **55**
4.1 Sampling . . . 56
4.1.1 Where to sample . . . 56
4.2 Regression . . . 60
4.2.1 Making A symmetric . . . 60
4.2.2 Constructing the normal equations . . . 61
4.2.3 Extracting the parameters . . . 61
4.3 Optimization . . . 62
4.3.1 When A is negative (semi)definite . . . 62
4.3.2 If A is not Negative (semi)definite . . . 63
4.4 Conclusion . . . 64
4.4.1 Bidding strategy . . . 64
4.4.2 Opponent model . . . 64
4.4.3 Acceptance strategy . . . 65

5 Experiments **66**
5.1 Experiment overview . . . 66
5.2 Experimental setup . . . 66
5.2.1 Negotiation scenarios . . . 66
5.3 Agent configuration . . . 67
5.3.1 The Cubes . . . 67
5.3.2 The manhattan distance . . . 68
5.3.3 Exploration and exploitation . . . 68
5.3.4 Sample sizes and regularization parameters . . . 68
5.3.5 Number of initializations . . . 70
5.4 Summary . . . 71

6 Results **73**
6.1 How the method scales with the complexity parameters . . . 73
6.1.1 The dimensionality of the constraints . . . 73
6.1.2 The width of the constraints . . . 75
6.1.3 The number of constraints . . . 76
6.1.4 The number of issues . . . 78
6.2 How the method scales with complexity . . . 79
6.3 Summary . . . 81

7 Discussion and Conclusion **82**
7.1 On how the methods scale with complexity . . . 82
7.1.1 The complexity of the agent . . . 82
7.1.2 The complexity of the functions . . . 83
7.2 Why does Quadratic outperform Linear? . . . 85
7.3 Conclusion . . . 86
7.3.1 Research Subquestion 1 . . . 86
7.3.2 Research Subquestion 2 . . . 87
7.3.3 Main Research Question . . . 89
7.4 Limitations and future work . . . 89
7.4.1 Model expressiveness and tractability . . . 89

7.4.2 Negotiation scenarios . . . 89
7.4.3 Complexity measure . . . 90
7.4.4 Sampling . . . 90
7.5 Contributions . . . 90
7.5.1 Basis for future work . . . 90
7.5.2 Complexity measure . . . 91

References **92**

A Additional information **97**
A.1 Genius . . . 97

B Proofs **98**
B.1 proof of Lemma 2 . . . 98
B.2 Proof of Lemma 3.(2) . . . 98
B.3 Proof of Theorem 4 . . . 99
B.4 Proof of Lemma 4 . . . 99
B.5 Generalization of Theorem 4 . . . 100

C Additional results **102**
C.1 Additional results . . . 102
C.1.1 Linearization vs L-BFGS-B . . . 102
C.1.2 L-BFGS-B initializations . . . 104
C.1.3 Cross effects . . . 105
C.2 Deploying QuadrApprox against ANAC 14 agents . . . 106

D Declaration on AI use **108**

Acronyms

ANAC Automated Negotiating Agent Competition. 17, 19, 23, 82, 89, 97, 106, 107

BBP Bidding-Based Protocol. 11–14, 16

BCGO box constrained global optimization. 48–50, 52, 88

BFGS Broyden-Fletcher-Goldfard-Shanno method. 51, 52

CD Coordinate Descent. 21

DaC Divide-and-Conquer. 15

DAOP Decomposable Alternative Offers Protocol. 15

GGD Generalized Gaussian Distribution. 17, 18

L-BFGS limited-memory BFGS. 52, 83

L-BFGS-B limited-memory BFGS with box constraints. 49, 52–54, 63–65, 67, 70–72, 83, 88, 103, 104

MSE mean squared error. 47, 73, 79–81, 83, 84, 88

MUL Marginal Utility Loss. 15

NG Number Generator. 57

OSD Overall Satisfaction Degree. 14

RNG Random Number Generator. 57

RP Representative Protocol. 12, 13

SAOP Stacked Alternating Offers Protocol. 2, 3, 10, 15, 66

TAP Threshold Adjusting Protocol. 12

Notation

Symbol	Definition
I_i	The ith issue under negotiation
c_k	The kth constraint
γ_k	The dimensionality of constraint c_k
Γ_k	The set of issues that define constraint c_k
d	Number of issues/dimensionality
n	Sample size
p	Number of weights in a regression model
ℓ	Cube side lengths
u	Utility function
s	Number of initializations of optimization
$[d]$	The index set $\{1, ..., d\}$

1 Introduction

Negotiation is a process where a joint decision is made by two or more parties who begin by expressing conflicting demands and then move towards agreement by a process of concession making [35]. Following this definition, the study of negotiations has applications in a much larger domain than just *negotiating*, such as procurement [13], supply chain management [57], resource allocation [2], and even communication between Mars rovers [16]. In fact, designing a vehicle can even be thought of as a negotiation [36]. In recent years, there has been a growing interest in the design of automated negotiators, i.e., autonomous computer systems, referred to as *agents*, capable of negotiating on behalf of humans.

There are several benefits to delegating a negotiation to an agent. Agents can work much faster than humans, which helps reduce the cognitive load involved. This becomes especially important in negotiations over multiple interdependent issues. In addition, agents are not subject to psychological biases and are expected to produce consistent outcomes (except in cases where stochastic behavior is intentionally programmed or the agent is deliberately designed to act unpredictably [8]).

During a typical automated negotiation, two or more agents propose various resolutions, also called *bids*, to the issues being negotiated. Any bid proposed by an agent should reflect its preferences — or the preferences of the human(s) the agent represents. Ultimately, all agents either reach an agreement or fail to do so before a termination criterion is met.

To reach an agreement, agents must be able to find bids that all negotiating agents are willing to accept. However, in negotiations over multiple interdependent issues, even finding a bid that the agent itself would accept can become difficult. Exhaustively evaluating each possible bid can become intractable due to the curse of dimensionality, requiring them to conduct a strategic search. *The aim of this thesis is to explore previous efforts in working with this problem, and to contribute to the existing literature with a new method aimed at finding bids that the agent itself deems good, and to evaluate and report on its performance.*

This chapter begins by covering the relevant terminology from the field of autonomous negotiations, defining the negotiation scenario and its building blocks. This is followed by an introduction into modeling outcome preferences of a negotiation with utility functions, where the constraint-based utility function — the class of functions considered for this thesis — is introduced. This is followed by a discussion of the mathematical challenges involved in working with such functions, along with the motivation behind the chosen approach to address them. This chapter then concludes on the research question and the thesis outline.

1.1. Terminology

This section covers the relevant terminology from the field of autonomous negotiations that will be referred to in this thesis. This covers the negotiation domain, the negotiation protocol, and preference profiles. Together, the three terms make up the *negotiation scenario*.

Autonomous negotiations are first and foremost categorized by the number of negotiating agents and the number of issues to be negotiated on. When there are two negotiating agents, the negotiation is referred to as a *bilateral negotiation*; while if there are more than two agents, it is referred to as *multilateral negotiation*. Similarly, when there is a single issue being negotiated on, the negotiation is referred to as a *single-issue negotiation*; while if there are more than a single issue, it is referred to as a *multi-issue negotiation*.

Assumption 1:

This thesis is exclusively focused on *bilateral multi-issue negotiations*.

1.1.1. Negotiation domain

A possible resolution to the issues under negotiation is referred to as a *contract* and is of the form $\mathbf{x} = (x_1, ..., x_d)$, where $x_i \in I_i$ for all $i = 1, ..., d$ where I_i denotes i-th issue domain. When an agent proposes a contract to its opponent as a solution to the negotiation, the contract is referred to as a *bid*. The set of all possible bids in a given negotiation is called the *negotiation domain* and is denoted by

$$\Omega := \prod_{i=1}^{d} I_i.$$

The size of the negotiation domain — or the number of possible bids — is denoted by $|\Omega|$, and the size of the ith issue domain — or the number of possible values the ith issue can take — is denoted by $|I_i|$.

For instance, in a negotiation between a car salesman and a potential buyer on whether a car should be high tier or low tier, and whether it should include a warranty or not, the negotiation domain would be the set

$$\Omega = \{(\text{high}, \text{warranty}); (\text{high}, \text{no warranty}); (\text{low}, \text{warranty}); (\text{low}, \text{no warranty})\}.$$

The size of this domain is $|\Omega| = 4$, and the sizes of the issue domains are $|I_1| = 2$ and $|I_2| = 2$.

Assumption 2:

The negotiation domains used in this thesis are integer domains where any two issues I_i and I_j have the same domain size. That is, In a negotiation over d issues, $|I_i| = |I_j|$ for all $i, j = 1, ..., d$ and $i \neq j$. Thus, $|I|$ will be used to denote *the* issue domain size.

1.1.2. Negotiation protocol

The negotiation protocol defines the structure of the negotiation mechanism, specifically which actions agents are permitted to take and at what times, as well as the termination criteria.

According to the Stacked Alternating Offers Protocol (SAOP) [6], turns are taken in a clockwise manner, where each agent gets a single action per round. One of the negotiating agents starts the negotiation with an bid that is immediately observed by all agents. The next agent in line is then allowed to either accept the bid, override the bid with a counteroffer, or terminate the

negotiation. A turn finishes when all agents have performed one action. This process is repeated until either a unanimous agreement is reached, a deadline is met, or one of the negotiating parties terminates the negotiation. The deadline is prespecified as either a time limit or a limit on the number of rounds.

Assumption 3:

The negotiation protocol exclusively used in this thesis is the SAOP.

1.1.3. Preference profiles

Each negotiating agent has their own preference profile, which is an ordering of the agent's preferences of the possible bids. The preference profile is characterized by three preference relations: $\succeq, \succ$, and $\sim$, which ensure that the agent has a way of choosing between any two contracts.

For any two contracts $\mathbf{x}'$ and $\mathbf{x}$, an agent is said to *weakly prefer* contract $\mathbf{x}'$ over $\mathbf{x}$ if $\mathbf{x}' \succeq \mathbf{x}$. If $\mathbf{x} \not\succeq \mathbf{x}'$ also holds, the agent is said to *strongly prefer* contract $\mathbf{x}'$ over $\mathbf{x}$, denoted as $\mathbf{x}' \succ \mathbf{x}$. However, if $\mathbf{x}' \succeq \mathbf{x}$ and $\mathbf{x} \succeq \mathbf{x}'$ both hold, then the agent is said to be indifferent to the two contracts, denoted as $\mathbf{x}' \sim \mathbf{x}$.

In some cases, an agent's preference profile will also specify a *reservation value* which is "as low as the agent will go", so to speak. That is, the set of all contracts that a rational agent would not offer nor accept has a theoretical least upper bound, measured by the preference relations, which depends on this value.

1.2. Utility functions

There are six constraints, *the axioms of utility theory* [66], that any preference relation is required to obey: *orderability, transitivity, continuity, substitutability, monotonicity*, and *decomposability*. These are all reasonable assumptions that ensure that the preference relation is consistent, and the first three ensure rationality. For instance, the transitivity constraints state that given any three choices, if an agent prefers the first one to the second one and prefers the second one to the third one, then the agent must prefer the first one to the third one.

Assumption 4:

An autonomous negotiation agent's preferences obey the axioms of utility theory.

Side note:

The rationality assumption in negotiations is first proposed by co-founders of the Harvard Negotiation Project, Fisher and Ury, in their book *Getting to Yes* [20]. The book was a groundbreaking treatise on negotiations at the time, notably introducing the concept of BATNA: the Best Alternative To a Negotiated Agreement, which is related to the reservation value in the field of autonomous negotiations. However, Kahneman, who, from years in psychology, knew that, in his words, "It is self-evident that people are neither fully rational nor completely selfish, and that their tastes are anything but stable" [42]. Through decades of research with the economist Amos Tversky, the pair would later launch the field of *behavioral economics* — which Kahneman received a Nobel prize for 6 years after the passing of Tversky — by showing in his book *Thinking, Fast and Slow* [42], that *man* is in fact irrational. For instance, Chris Voss, the FBI's chief international hostage and kidnapping

negotiator from 2003 to 2007, notes in his memoir *Never Split the Difference* [75] that after he and his colleagues had studied *getting to yes*, they found that its methods were not working in the field. The reason: hostage takers do not act rationally.

In their 1944 book *Theory of Games and Economic Behavior [74]*, who many would say is the defining book for game theory, Von Neumann and Morgenstern derived from these six axioms the following consequence (for the proof, see [74])

- **Existence of Utility Function**: If an agent's preferences obey the axioms of utility, then there exists a function u such that $u(\mathbf{x}') > u(\mathbf{x})$ if and only if $\mathbf{x}'$ is preferred to $\mathbf{x}$, and $u(\mathbf{x}') = u(\mathbf{x})$ if and only if the agent is indifferent between $\mathbf{x}'$ and $\mathbf{x}$. That is,

$$u(\mathbf{x}') > u(\mathbf{x}) \iff \mathbf{x}' \succ \mathbf{x} \text{ and } u(\mathbf{x}') = u(\mathbf{x}) \iff \mathbf{x}' \sim \mathbf{x}.$$

Side note:

"Many economists will feel that we are assuming far too much... We have practically defined numerical utility as being that thing for which the calculus of mathematical expectations is legitimate" - [74], § 3.1.1 p.16 and § 3.7.1 p. 28

This consequence, along with Assumption 4, merits the formulation of an agent's preferences by means of a utility function. These functions can then be optimized to find highly preferable contracts, as opposed to searching for them through exhaustive enumeration. Constructing these functions is a whole field in itself. Some notable literature covering the formulation of utility functions includes utility theory, e.g. [74, 66, 10]; risk analysis, e.g. [10, 11]; and the work of Keeney and Raiffa [43]. Notably, however, there are alternatives to formulating preferences, such as graph-based, which can also be manipulated to find good bids [65].

A negotiating agent's utility function, as previously mentioned, represents its preferences over a set of issues being negotiated. Utility functions can be either *cardinal* or *ordinal*. For an ordinal utility function u, the magnitude of the difference between $u(\mathbf{x}')$ and $u(\mathbf{x})$, for any two contracts $\mathbf{x}'$ and $\mathbf{x}$, has no meaningful interpretation. Whether $u(\mathbf{x}') > u(\mathbf{x})$ or $u(\mathbf{x}') < u(\mathbf{x})$ bears the only meaning. These functions do not measure the *intensity* of preferences. Cardinal utility functions, on the other hand, do measure the intensity of preferences. For instance,

$$0 < u(\mathbf{x}) - u(\mathbf{x}') = 2[u(\mathbf{x}) - u(\mathbf{x}'')] \tag{1.1}$$

implies that the preference of $\mathbf{x}$ over $\mathbf{x}'$ is twice as strong as the preference of $\mathbf{x}$ over $\mathbf{x}''$. However, in the case of ordinal utility functions, Expression (1.1) only implies that $\mathbf{x}$ is preferred both over $\mathbf{x}'$ and $\mathbf{x}''$

Cardinal utility functions are commonly constructed to map to the unit interval; that is, $u : \Omega \mapsto [0, 1]$. This is to ensure comparability across outcomes and to simplify interpretation, where 0 represents the least preferred and 1 the most preferred option.

When the negotiation domain is intractably large, formulating the preferences in terms of a cardinal utility function and optimizing that function is much more efficient than exhaustively evaluating each outcome. Humans tend to simplify the structure of their preferences and prefer to negotiate one issue at a time [71], treating each issue as an independent contributor to the total utility. Under these independence assumptions, the utility of a bid on d issues can be

formulated as a linear additive function of the form

$$u(\mathbf{x}) = \sum_{i=1}^{d} w_i e_i(x_i) \tag{1.2}$$

where e_i are the evaluation functions [35, 41], each acting as a utility function for the corresponding issue, and w_i is the corresponding weight. Functions of the form (1.2) can be optimized in linear time with respect to d by simply optimizing each e_i individually.

However, when issues are interdependent, the utility contribution of some issues may be conditioned on the value of others. Consider, for example, the car example from Section 1.1.1: if a car is of a low tier, perhaps a warranty is unnecessary, and vice versa. As functions of the form (1.2) do not capture interaction effects, they cannot be used in negotiations involving multiple interdependent issues, as nonlinearities need to be introduced.

1.2.1. Hypercubic constraint-based utility functions

The class of utility functions that will be considered for this thesis is constraint-based utility functions [37, 31]. Having a constraint-based utility function, each agent has their own unique set of constraints C, where each constraint $c_k \in C$ represents a region in the negotiation domain and an associated utility value. The dimensionality of a region represented by a constraint thereby constitutes the level of issue interdependency associated with that constraint. An agent's utility for a contract is then defined as the sum of the utilities of all the constraints it satisfies. That is, the utility function takes the form:

$$u(\mathbf{x}) = \sum_{c_k \in C} \omega(c_k, \mathbf{x}) \tag{1.3}$$

where $w(\cdot)$ is defined by the shape of the constraint, i.e., the shape of the region represented by the constraint. These can theoretically be any shape, with the most prominent in the literature reviewed for this thesis being cube-shaped, also referred to as hypercubic constraints. These are formally defined in Chapter 3.

Assumption 5:

This thesis is exclusively focused on hypercubic constraints.

Informally, a hypercubic constraint-based utility function is the same as placing some number of hypercubes into the negotiation domain and assigning to each one a utility. Then, a contract's utility is the sum of the utilities of the hypercubes it intersects with. Figure 1.1 illustrates this with an example.

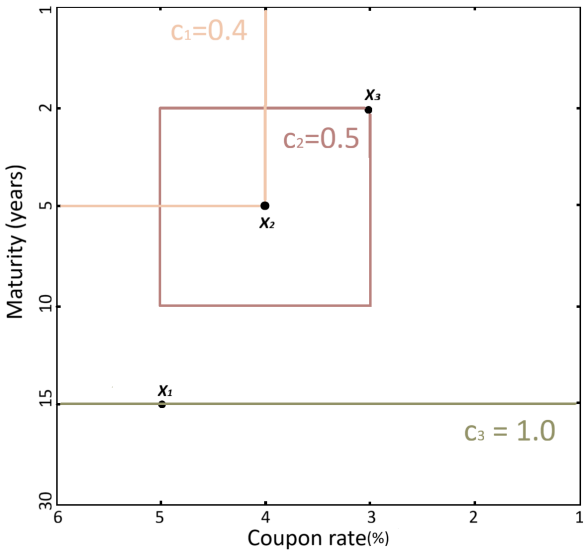


Figure 1.1: A pension fund is negotiating with the government over the purchase of a custom-designed bond product. The negotiation focuses on two key features: the *coupon rate* and the *maturity period*. Given its long-term liabilities, the fund places a high value on long maturities as they align well with its financial obligations. As a result, any bond with a 15-30-year maturity is considered ideal, regardless of the coupon rate. This is captured by constraint c_3, with an outcome such as $\mathbf{x}_1$ yielding the maximum utility of 1.0. In contrast, short- and mid-term bonds with low coupon rates are generally undesirable. However, short-term bonds with high coupon rates, while not preferred, may still be acceptable. These are represented by constraint c_1. Meanwhile, mid-range combinations of coupon rates and maturities are considered reasonably attractive and are captured by constraint c_2. For instance, an outcome like $\mathbf{x}_3$ might yield a utility of 0.5 under this constraint. Certain outcomes satisfy multiple constraints. For example, an offer that fulfills both c_1 and c_2, such as $\mathbf{x}_2$, may be particularly appealing, resulting in a combined utility of 0.9.

1.3. Problem definition

A problem that arises in autonomous negotiations is that the search space grows exponentially with the number of issues. This is a concrete manifestation of *the curse of dimensionality*, which makes exhaustively evaluating each possible outcome intractable.

Consider, for example, the pharmaceutical sector, where companies commonly procure tens or hundreds of products from various suppliers [45]. Suppose a company is interested in procuring 25 products from a vendor, where the unit quantity of each product can be negotiated. Even if each product can be configured by choosing one of 10 possible unit quantities, the company has a total of 10^{25} possible options to evaluate. Even attempting to evaluate 1% of such a space would require more computational resources than are feasible with modern hardware.

A prominent approach to circumvent this problem is to frame the negotiators' preferences in terms of a utility function and then to optimize that function. A large body of literature covers this process with linear additive functions of the form (1.2), which can be optimized in linear time. For the pharmaceutical company, this means finding the global optimum in milliseconds on modern hardware, or even in a few seconds with a pen and paper. However, once interdependencies are introduced into the function through nonlinearities, a combinatorial explosion occurs, as each issue can no longer be optimized independently.

Negotiation scenarios with large domains and interdependencies in preferences commonly occur in various fields, including e-commerce, Manufacturing and production planning, vehicle design, logistics and supply chains, Energy systems, resource allocation, and more. This, in turn,

requires agents to negotiate simultaneously on each issue, creating a combinatorial explosion. The particular class of utility functions that is commonly used to model these scenarios is constraint-based functions. Of particular interest in this thesis is the hypercubic function, which was the most prominent constraint-based function in the literature reviewed for this thesis.

Optimizing these functions to find high-utility contracts is a combinatorial optimization problem which are known to be NP-hard. These functions can, however, sometimes present structures that can be manipulated; mainly, constraints being grouped. To direct this thesis towards a more general approach, the final assumption of this chapter is as follows:

Assumption 6:

The agent knows the parameters of its utility function but not its analytical form.

This assumption means that the utility function is treated as *uncertain* [66]: the agent can query it but cannot observe the underlying structure, such as the positions of the individual hypercubes and their sizes. What is known are the high-level parameters of the function, which are the topic of Section 3.1. For instance, even if the function were as simple as $u(x) = x^2$, the agent would need to infer its behavior by evaluating individual values of x.

The theoretical groundwork outlined in this chapter leads directly to a core practical problem:

> *How can an agent identify high-utility contracts in a negotiation domain when endowed with an uncertain hypercubic constraint-based utility function?*

1.4. Motivation

Of the existing approaches that were reviewed for this thesis, most of them apply some optimization algorithm directly to the utility function in an attempt to find "good bids", however they are defined. However, not many of them take the approach of first attempting to simplify the utility function by approximating it, creating, in a sense, a surrogate utility function that is then optimized.

A complex utility function can be approximated by a simpler one, which, in turn, is easier to optimize. One such approach, developed by Hindriks, Jonker, and Tykhonov [35], provided the initial inspiration for this work. The method, discussed in more detail in Section 2.2, involves linearizing the utility function by averaging out interdependencies, resulting in an approximation similar in form to Equation (1.2), which, as noted earlier, can be easily optimized in linear time. *This section of the literature — specifically, the optimization of surrogate utility functions in autonomous negotiation — is sparse, and it is the section this thesis aims to contribute to.*

When taking the approach of optimizing a surrogate model, there is a certain balance that needs to be considered:

> *An overly accurate approximation may replicate the original function's complexity, providing little benefit in terms of optimization. On the other hand, an overly simplistic approximation may lose too much information, making the optimization irrelevant.*

This thesis extends the linearization idea of Hindriks, Jonker, and Tykhonov to constraint-based utility functions and to introduce second-order interactions. This is done with quadratic polynomials, which can be viewed as decomposing higher-dimensional dependencies into pairwise interactions. Quadratic polynomials strike a balance between expressiveness and tractability, which makes them suitable for the task at hand.

The appeal of quadratic functions can be understood by considering Bézout's theorem from algebraic geometry, which describes the number of intersections of polynomial curves.

Theorem 1: ***Bézout's theorem*** **[21]**

If C and D are complex projective (algebraic) curves with no common components, then

$$\sum_{P \in C \cap D} i(C \cap D, P) = (degC)(degD)$$

where $i(C \cap D, P)$ is the intersection multiplicity of C and D at point P.

Now, finding the optima of an d-variate polynomial of degree k can be done by setting the d univariate partial derivatives to zero, or

$$\begin{cases} \frac{\partial \hat{u}}{\partial x_1} = 0 \\ \frac{\partial \hat{u}}{\partial x_2} = 0 \\ \vdots \\ \frac{\partial \hat{u}}{\partial x_d} = 0. \end{cases}$$

In the case of cubic polynomials, these are quadratic equations, but linear in the case of quadratic polynomials. By Bézout's theorem, this means that linear polynomials do not have an optimum, and that quadratic polynomials have at most one optimum. For cubic polynomials, however, the system becomes quadratic, and Bézout's theorem implies up to 2^d possible optima. Thus, restricting to degree-2 polynomials gives a *unimodal*[1] surrogate for the original utility functions, avoiding an exponential explosion in local optima. As quadratic polynomials are smooth and unimodal, the stationary point is unique. When the polynomial is concave, this point is a global maximum. In the case where it is not concave, second-order optimization algorithms can be used, which are ideal for quadratic functions.

1.5. Research Objective

The objective of this thesis is to explore how efficiently an agent can identify high-utility bids by searching within a quadratic surrogate model of its utility function, which is of a hypercubic constraint-based form. Since these utility functions can range from very simple to highly complex — as will be discussed in Section 3.1 — it is necessary to evaluate the agent's performance relative to the complexity of the underlying utility function. Accordingly, the following question summarizes the central research objective of this thesis:

Main Research Question:

How does the efficiency of an autonomous negotiation agent in finding high-utility bids using quadratic surrogate models of uncertain hypercubic constraint-based utility functions scale with increasing utility function complexity, and how does it compare to using linear surrogate models?

This question breaks down into the following subquestions:

[1] A unimodal function has at most a single optimum. The quadratic polynomial does not technically have a single optimum when it is indefinite, but rather a single flat region of optima.

Research Subquestion 1:

How can the complexity of a hypercubic constraint-based utility functions be measured?

Research Subquestion 2:

How can quadratic polynomials be effectively constructed and used as surrogates for uncertain hypercubic constraint-based utility functions in negotiations?

1.6. Thesis outline

The structure of this thesis is as follows:

Chapter 1: Introduction
Introduces the relevant terminology from the field of autonomous negotiations, with a focus on utility functions, particularly hypercubic constraint-based utility functions. It then examines the mathematical difficulties these functions pose in a negotiation setting, particularly those stemming from the curse of dimensionality. Finally, the research objective is outlined.

Chapter 2: Literature review
Covers the ways researchers have approached the challenges of using nonlinear utility functions in autonomous negotiations, from both the mechanism design and agent design perspectives.

Chapter 3: Mathematical Background
Addresses Subquestion 1 and Subquestion 2 and the relevant mathematical considerations for answering the Main Research Question. These include quantifying the complexity of the utility function, determining how and where to sample, and how to construct and optimize the surrogate.

Chapter 4: Method
The considerations and conclusions from Chapter 3 are implemented into a functioning autonomous negotiation agent.

Chapter 5: Experiments
The experimental setup designed to address the Main Research Question is presented, and the agent's parameters are calibrated accordingly.

Chapter 6: Results
The results of the numerical experiments described in Chapter 5, conducted with the agent introduced in Chapter 4, are presented and discussed at a high level.

Chapter 7: Discussion and Conclusion
The results presented in Chapter 6 are interpreted in light of the mathematical foundations established in Chapter 3. This chapter provides direct answers to the research questions and outlines the methodological limitations of the study and potential directions for future work.

Appendices
The appendices include information on the simulation environment used for experiments, the agents performance in a real negotiation setting, proofs, and additional figures that were not included in the main chapters.

2

Literature review

Research on automated negotiations, where utility functions capture preferences with interdependencies between issues, has primarily concentrated on two areas [38]: mechanism design and agent design. This chapter reviews both strands of the literature that inform this thesis, addressing each perspective in turn, and concludes with a discussion on the research gap that this work aims to fill.

The mechanism side of the literature — or the protocols — starts with the *Bidding-Based Protocol* proposed by Ito, Hattori, and Klein [37], and expands from there into protocols built upon that one. Finally, the section concludes by touching on protocols based on decomposing the utility function. The agent side of the literature — or the negotiation strategies — introduces the *WAID* method [35], a method based on the *Generalized Gaussian Distribution* [28], and then covers some of the agents from the 2014 *Automated Negotiating Agent Competition*.

2.1. Mechanism design for nonlinear utility functions

Protocols are problem-solving tools, with each one serving its own purpose. A widely adopted way of handling complexities associated with nonlinear utility functions in autonomous negotiations is to design protocols that facilitate them. While the aim of the *Stacked Alternating Offers Protocol (SAOP)* might be to replicate the standard way of negotiating, the *Hybrid Secure Protocol* [24] aims to ensure privacy. The *mediator based protocols* [37, 31, 24, 23, 44] aim to facilitate nonlinear utility functions, while the Vickrey–Clarke–Groves mechanism [66] aims for socially optimal solutions by enforcing honesty. In their work Marsa-Maestre et al. [55], compiled a handbook with guidelines on how to select the most appropriate protocol for a given negotiation problem, and more recently, the same authors proposed a machine learning-based method for choosing the most effective protocol for a given problem [5]. In this section, various protocols that are designed to facilitate nonlinear utility functions are covered.

Many of the protocols covered in this section are structured around the constraint-based utility functions mentioned in Section 1.2.1, and two central ideas to facilitate them: directing the search for bids to regions where a deal is likely to be struck, and the decomposition of the negotiation domain or the utility functions themselves.

In their paper *Negotiating Complex Contracts*, Klein et al. [44] present what is most likely the first negotiation protocols that are specific to nonlinear utility functions. The preferences of the negotiating agents are described using an influence matrix H, where each cell quantifies the impact by the presence of a given pair of issues i and j, on the total utility. The total utility of

a contract is then the sum of the cell values for every issue pair present in the contract, or

$$U = \sum_i \sum_j H_{ij} S_i S_j$$

where S_i takes values 0 or 1, denoting the presence or absence of issue i, respectively. In their protocol, a mediator guides the negotiation by proposing a random contract to the negotiating agents which the agents then votes to either accept or reject. Contracts accepted by both agents are then mutated by flipping one issue bit, and contracts rejected by either or both agents are replaced with a mutation of the last accepted one. This process iterates until the utility values stabilize.

2.1.1. The Bidding-Based Protocol

In [37], Ito, Hattori, and Klein proposed a Bidding-Based Protocol (BBP) with a mediator. In their work, they consider the hypercubic function described in Section 1.2.1 as a utility function. The protocol proceeds in four stages:

- *Sampling*: Each agent draws a fixed number of uniformly random samples from their utility space.
- *Adjusting*: Each agent searches for a local optimum in the neighborhood of each sample using simulated annealing.
- *Bidding*: Each agent evaluates the utility of the contract identified during the adjusting step. For contracts that yield a utility exceeding the reservation value, the agent constructs a bid consisting of all contracts within the corresponding region that achieve the same utility level.
- *Deal Identification*: The mediator determines the final bid by identifying all combinations of contracts — one from each agent — that are mutually consistent, meaning their corresponding contract regions overlap. To find the overlap that maximizes social welfare, the mediator employs a breadth-first search with branch cutting.

Theoretically, this approach is guaranteed to find the socially optimal contract under certain conditions. If each agent exhaustively samples its entire utility space and has a reservation value of zero, the resulting bids will fully represent their utility functions. With complete information, the mediator can then perform an exhaustive search over all bid combinations to identify the outcome that maximizes social welfare. However, assuming that k bids are submitted by each of N agents, the number of bid combinations that the mediator has to consider is N^k. The authors mention that the maximum number of bids that their system could handle was around 6.4×10^6. This scalability bottleneck affects the optimality of the contracts identified, and the failure rate — or the proportion of experiments that did not result in a deal — was high. This insight led them to extend their protocol with the *Iterative Narrowing Protocol*, discussed in the upcoming section.

2.1.2. Extensions of the Bidding-Based Protocol

The Iterative Narrowing Protocol

The Iterative Narrowing Protocol [33] extends the BBP by having agents iteratively refine their bids to narrow down the negotiation space. Instead of submitting all bids at once, agents generate structured bids in three successive rounds, each round designed to focus more precisely on promising subregions of the utility space. The bid generation process proceeds in the following way:

- *Cluster Bidding:* Each agent groups the constraints of their utility function into clusters. A single bid is then created per cluster, covering the region defined by the intersection of constraints within that cluster.
- *Widest-Constraint Bidding:* The mediator identifies all overlaps (intersections) between the cluster bids submitted in the first round using exhaustive search.
- *Peak Bidding:* The third round is the same in spirit to the bidding step of the BBP, but now it is applied to the refined overlaps of round two.

While still constrained by the total combination limit of 6.4×10^6, the three-phase structure allows agents to explore the space more efficiently by focusing computational effort where it is most likely to yield a successful deal.

With a similar experimental setup as that of [37], they showed that the effectiveness of the protocol is particularly pronounced in scenarios where the agents' utility functions exhibit clustered structures. For such cases, the failure rate is significantly improved compared to the BBP while achieving similar optimality rates.

This demonstrates the importance of focused search. Although the space for the mediator to search is now considerably smaller, the exhaustive search still puts a strong limitation on the computational cost for finding an optimal deal. To combat this, Fujita and Ito proposed the *Threshold Adjusting Protocol (TAP)*.

Threshold Adjusting Protocol

The TAP [22] extends the BBP by making the deal identification step incremental, where the agents incrementally reveal more and more of their utility spaces to the mediator. Fujita and Ito define a threshold as a certain utility value and the revealed area for agent i corresponding to that threshold, simply as

$$\text{Revealed area}_i = \{\text{all } \mathbf{x} \text{ such that } u_i(\mathbf{x}) > \text{Threshold}\}.$$

The authors do not explain how they identify the revealed area. This threshold is lowered in each round, revealing more of the agent's utility space, like land slowly surfacing as a tide recedes. Each round is then the same as in the BBP with the exception of the mediator having to search the space revealed in that round as opposed to the entire space. As a consequence, the agreements found by this protocol tend to have a higher utility.

Under similar conditions, their results were comparable to those of [37] and [33] in terms of maximizing social welfare. A key distinction, however, was that the TAP imposed no limit on the number of bids an agent could submit. The authors, however, do not report failure rates in their evaluation. Furthermore, as the authors do not explain how they identify the revealed area, this approach seems to be limited to cases where identifying the revealed area is straightforward, which is not the case with most nonlinear functions.

Representative protocols

In [24], Fujita, Ito, and Klein note that the TAP and the BBP scale poorly with the number of agents, and the computational complexity for finding the solution was too large. Furthermore, they note, the agents' preferences are not completely concealed. Therefore, they proposed the *Representative Protocol (RP)* [24], an extension of the TAP.

The RP nominates some number of agents to essentially do the mediator's job. The protocol proceeds in three steps. First, all agents share with the mediator the amount of their utility space they are willing to reveal. Those who are most keen on sharing get nominated as representatives.

Next, the elected representatives employ breadth-first search with branch cutting on the revealed space and propose potential agreements to the remaining agents. Finally, the non-representative agents respond to the proposed agreements from the representative agents. If any agent rejects the proposed agreement, step two is repeated.

In their experiments, they compared the representative protocol with the BBP. They found that their setup significantly reduces computational complexity by limiting exhaustive negotiation to a smaller group, making it scalable with respect to the number of agents, and far outperformed the BBP in terms of social optimality and failure rates.

They later proposed the *distributed* RP [25], noting that there were two main issues with their original RP. Firstly, agents have to reveal *some* private information. Secondly, scalability for the complexity of the agent's utility function isn't very good. They note that, if the utility function has narrower constraints, the utility space becomes extremely nonlinear, making it very difficult for the representative protocol to reach an optimal agreement point. A narrow constraint is a constraint defined by a hypercube with short side lengths. This is covered in detail in Section 3.1.

In the distributed RP, the mediator divides the utility space between some prespecified number of representative agents, each responsible for searching their allocated portion of the space using local search algorithms, allowing parallel computation. They do, however, not report on how the utility space is divided among the mediators. To evaluate the total utility of a given state securely, agents share encrypted pieces of their utility values with a subset of mediators. These mediators then use Shamir's secret sharing scheme [68] to collaboratively reconstruct the total utility without revealing individual values.

They experimented with constraint-based utility functions with narrow and cone-shaped constraints, and four mediators. Although they did not compare their method to their previous one, and did not report failure rates, they did claim their approach drastically reduced computational complexity while reaching socially optimal outcomes and completely concealing the agent's preferences.

Heuristically guided search

In [54], Marsa-Maestre et al. also note that as the number of issues under consideration increases, the difference between having wide or narrow constraints becomes more relevant. Though the BBP may work in scenarios defined using wide constraints, Marsa-Maestre et al. show that their performance decreases drastically in highly nonlinear scenarios defined using narrow constraints.

They proposed three alternative search methods to simulated annealing for each agent in the adjusting step of the BBP: a probabilistic greedy search, an approach based on integer programming, and a search based on finding maximum weight independent sets within the constraint space of the agents. In addition, they propose a probabilistic search in the mediator instead of an exhaustive search. To achieve this, they define a *bid quality factor*, a measure that combines a bid's utility and its viability, or the likelihood of resulting in a deal. They define the measure as

$$Q_c = u_c^{\alpha} \cdot v_c^{1-\alpha}$$

where u_c is the utility of a bid, v_c is the cardinality of the set of contracts which match the bid, and $\alpha \in [0, 1]$ is a tunable parameter which models the risk attitude of the agent. The mediator's bid combination selection is then biased so that bids with a higher quality factor have a higher probability of being selected for bid combinations.

However, they note, since high-utility regions are narrower, it is more likely that a single shot of the algorithm yields no solution. In these cases, it would be desirable that the agents would be able to *learn* from previous interactions to find bids that are more likely to reach an agreement. For this to be possible, they propose two processes: one through which the mediator gives feedback to the negotiating agents, and another through which the agents use this feedback in generating bids. They therefore propose two additional steps for the BBP: a *feedback* step and an *adjusted sampling* step. In the feedback step, the mediator requests that the agents relax some of their bids to lead the negotiations to zones in the contract space where higher joint gains can be achieved. In the adjusted sampling step, the agents compute a new set of bids, taking into account the feedback provided by the mediator.

In their experiments, they compared their proposed approaches to the BBP but with narrower constraints. Their results showed their approaches lowered the scalability problem of [37] by achieving both significantly better optimality and failure rates. These results emphasize how a simple heuristic can greatly improve the search by narrowing it down to important regions of the space.

In [52], Lopez-Carmona et al.proposed the *NegoExplorer*, a non-mediated bilateral protocol based on offers in the form of regions of the negotiation space. The mechanism is recursive in the sense that when agents agree on a region proposed by an agent, a new negotiation on regions of a lower size is performed within the previously agreed region.

They define the Overall Satisfaction Degree (OSD) of a region R for an agent a, as an estimate of the overall utility of the region for that agent. Let $S^R = \{\mathbf{x}_k \in R | k = 1, ..., n\}$ be the set of n contracts in R, δ the agent's reservation value, and S^R_δ the subset of R containing the i contracts that satisfy $u_a(\mathbf{x}) \geq \delta$. An agent a computes the OSD of a region R by means of the following formula:

$$OSD(a, R) = \gamma \cdot \frac{i}{n} + (1 - \gamma) \cdot \frac{\sum_{\mathbf{x} \in S^R_\delta} (u_a(\mathbf{x}) - \delta)}{i \cdot (1 - \delta)}$$

The parameter γ weights the ratio of the number of contracts above the reservation value, and the normalized overall utility surplus for contracts above the reservation value. That is, when $\gamma = 1$, the OSD depends only on the number of contracts with a utility value above the utility threshold, and when $\gamma = 0$, the OSD depends only on the utility values of those contracts. The idea behind this definition of OSD is that an agent can consider separately both the probability that a random contract in R falls above δ, and the utility of the acceptable contracts. The OSD is then used by the negotiating agents to both assess their opponent's bids and to formulate their own.

In addition, the protocol allows for a *suggestion* action where the agents suggest a move of a region in the direction of its center of mass. The center of mass is found by assigning more mass to those samples from within the region that are above the reservation value δ. Notably, NegoExplorer had an almost 0% failure rate over all experiments.

2.1.3. Decomposing the utility function

As opposed to working directly with the utility function, as has been the focus of this chapter so far, some effort has been put into working with a simplified version of it. This section covers the literature concerning the mechanism side of that approach.

The Decomposable Alternative Offers Protocol

In [50], Li, Hadfi, and Ito propose the *Decomposable Alternative Offers Protocol (DAOP)*, an adaptation of the SAOP incorporating the Divide-and-Conquer (DaC) paradigm. DAOP allows the division of the complete negotiation domain into several sub-domains that the agents negotiate over.

DAOP begins by asking the agents to group the issues into k clusters, where k is pre-specified, based on the interdependency relationships between the issues. The agents then negotiate over each cluster separately. Once agreements are reached over all clusters, the DAOP combines these sub-agreements into complete agreements.

To determine the clustering of the issues, the interdependency among them is measured in terms of Marginal Utility Loss (MUL). Formally, define the set of d issues $\mathcal{D} = \{1, ..., d\}$. For any subset $S \subset \mathcal{D}$, define the sub-domains $\Omega_S = \Pi_{j \in S} I_j$ where I_j is the jth issue domain. Finally, the MUL, $\frac{\Delta u}{\Delta S}$ for any given subset of issues S is defined as

$$\frac{\Delta u}{\Delta S} = \max_{\Omega_{-S}} [\max_{\Omega_S} u(S, -S) - \min_{\Omega_S} u(S, -S)]$$

where u is the utility function of the agent and $-S = \mathcal{D}/S$. A lower MUL indicates that the corresponding cluster retains more of the global utility information, making it a more effective grouping.

The functions used for evaluating the complete bids differ from the ones used for sub-bids. To infer the utility functions for evaluating sub-bids, the original utility function must be decomposed. The authors demonstrate this decomposition for linear additive functions and for an exponential function of a sum. However, they do not address how to decompose utility functions that include interaction terms between issues.

In their experiments, they endowed two agents with quasi-similar exponential utility functions, which lead the agents to arrive at the same clusters as those that minimized their individual MUL. The paper does not explore how clustering should be handled when agents' individual MUL lead to conflicting groupings.

The authors conduct experiments with six issues and two agents, comparing the DAOP with the SAOP. The results did show that incorporating the DaC algorithm significantly improves both social welfare and time efficiency.

Hierarchical model

In [78], Zhang and Klein propose a hierarchical negotiation model where a constraint-based utility function is assumed to have a natural hierarchy based on abstraction with domain knowledge. Within each level, issues are clustered together where the issues that belong to the same cluster are highly interdependent, while the level of dependency across different clusters is much lower. The more of such constraints and the more important they are, the stronger the interdependencies among these issues are.

They propose a protocol that adopts this hierarchical representation, where agents iteratively submit meta-level information about issue dependencies. The mediator clusters interdependent issues into decision groups and constructs a negotiation agenda, represented as a directed acyclic graph, to guide the negotiation order. At each level, agents submit preferred bids, and the mediator selects common choices by identifying intersections. The process repeats across levels, with earlier decisions constraining later ones, and includes backtracking if negotiations stall or fail to meet utility thresholds.

Their results showed that this structure effectively reduces the negotiation space and accounts for interdependencies among issues. It is, however, restricted to negotiations using constraint-based utilities that are assumed to have a natural hierarchy. These hierarchical structures might, however, be viewed as latent variables, giving rise to dimension reduction techniques.

Hypergraphs

Noting that the constraint-based utility functions can be viewed as linear in terms of the constraints, as they are essentially a sum of constraints (recall Expression (1.3)), Robu, Somefun, and Poutré adopt a hypergraph representation of these functions in [65]. This has since become a popular way of formulating preferences or decomposing constraint-based utility functions. For more information on hypergraphs in autonomous negotiations, see e.g. [15, 72, 30, 73].

In [23], Fujita, Ito, and Klein apply the concept of issue grouping to the BBP by adopting a graph-based representation of the utility functions. They define a measure for the interdependency rate between any two issues in a constraint-based utility function by either the number of constraints inter-relating them, the number of terms in these constraints, the utility values of these constraints, or the product of the number of terms of these constraints and their utility values. Their protocol extends the BBP by first having each agent construct an interdependency graph based on its utility function, which it then submits to the mediator. The interdependency graph has nodes representing issues and edges representing interdependency between them, with the weight of an edge representing the interdependency rate between the issues. This is depicted in Figure 2.1.

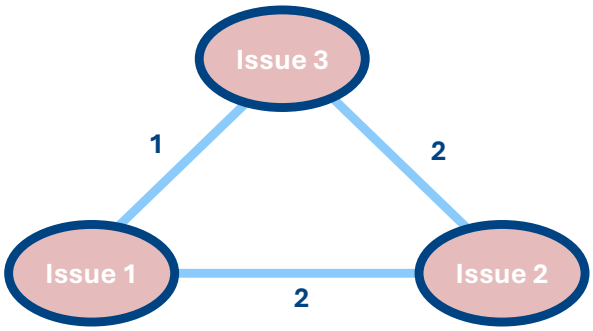


Figure 2.1: An example of an interdependency graph where two constraints inter-relate issues 1 and 2, two constraints inter-relate issues 2 and 3, and one constraint inter-relates issues 1 and 3.

The mediator then constructs a social interdependency graph by summing the weights of each issue across all agents' graphs and attempts to find an optimal issue-grouping using simulated annealing. Next, the mediator submits these issue groupings to the agents, who then go through the sampling, adjusting, and bidding steps of the BBP. However, in the bidding step, before submitting their bid, the agents divide them into sub-bids for each issue-group and determine their valuations for each sub-bid. Finally, the mediator identifies the final contract by finding all the combinations of bids, one from each agent, that specify overlapping contract regions.

In their experiments, they compared their method, using all measures of interdependency, with the BBP, as well as the approach of [54] using Maximum Weight Independent Set. They found similar results in terms of optimality for all methods. The main differences in their results were in the failure rate with their method and that of [54] having nearly a 0% failure rate in all experiments, while it quickly reached 100% in the BBP. They do, however, show that their

method is highly sensitive to the number of issue groups, imposing a trade-off between failure rates and optimality rates. While failure rates decrease as the number of issue groups increases, so does the optimality rate.

2.2. Agent design for nonlinear utility functions

The basis of a negotiation agent's decision-making model is the strategies it employs during the encounter, captured in the agent function [66]. A negotiation agent is typically equipped with multiple strategies that collectively guide the agent through the negotiation. The agent design can thus have multiple attributes where each one can be tailored to any specific scenario, ranging anywhere from simple random bidders to reinforcement learning agents endowed with advanced learning techniques.

In [7], Baarslag et al. propose a component-based architecture for a negotiation agent called BOA. Their architecture summarizes all negotiation strategies into three distinct components:

- *Bidding Strategy (B)*: Determines the appropriate bids to be made given the current state of the negotiation.
- *Opponent Model (O)*: A learning technique aimed at constructing a model of the opponent's preferences.
- *Acceptance Strategy (A)*: Determines whether the bid that the opponent proposes is acceptable or not.

These three components should not only function well individually but also in harmony. For instance, the bidding strategy must balance exploration and exploitation, thereby enhancing the opponent model, which in turn informs the acceptance strategy that a better bid than the current one might be expected later on. Additionally, the bidding strategy could potentially bias its exploration towards regions of the domain that the opponent model has deemed likely to yield an agreement.

This thesis primarily focuses on the bidding strategy, or how well a quadratic polynomial serves as a surrogate for the original utility function, with the sole purpose of finding high utility bids. For this reason, the literature surrounding the other two components will not be covered. For the interested reader, [7] and [8] are good starting points.

In this section, various ways that agents can be designed to work with nonlinear utility functions with interdependencies are covered. This includes agents that simplify their utility function as well as contestants from the 2014 Automated Negotiating Agent Competition (ANAC), which all directly optimize their utility function.

2.2.1. Approximating the utility function

Generalized Gaussian Distribution as a basis function

In [28], Hadfi and Ito explore using the Generalized Gaussian Distribution (GGD)

$$g(x;\rho,\mu,\beta) = \frac{\rho}{2\beta\Gamma(\frac{1}{\rho})} e^{-\left(\frac{|x-\mu|}{\beta}\right)^{\rho}}, \tag{2.1}$$

for approximating constraint-based utility functions. The proposed approximation yields a compact and concave form that unifies many forms of constraint-based utility functions (cubic, bell, conic, etc.).

The parameters of Expression (2.1), they note, can be tuned so that the resulting expression represents a multitude of geometric shapes that could approximate these constraints. Indeed, for the cubic, conic, and bell-shaped constraints, Figure 2.2 shows configurations of the parameters ρ, β, and μ in two dimensions that result in the GGD taking on similar shapes.

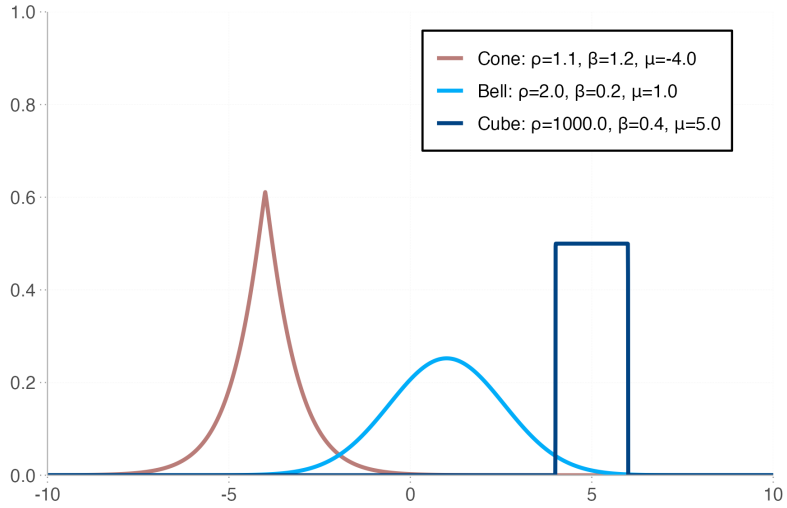


Figure 2.2: GGD with different values for ρ. Figure is adapted from [28]

For the general case of n-dimensions, the authors come up with the following parametrization of Expression (2.1) to represent the constraints:

$$f(x; \rho, \beta, \gamma, \delta, \mu, \zeta) = \gamma + \beta e^{\delta - |\zeta x - \mu|^{\rho}}. \tag{2.2}$$

This particular parametrization allows for tuning of the parameters to not only approximate the shapes of the constraints but also their size. Expression (2.2) can serve as a basis function in a regression model for approximating a constraint-based utility function of the form (1.3). The resulting approximation then takes the form

$$u(\mathbf{x}) = \sum_{j=1}^{m} w_j f_j(\mathbf{x}, \cdot).$$

Although they do not experiment with this approach in a negotiation setting, they do evaluate how good the approximation is with cubic and conic constraints. They do that by generating random contract points within the constraint being approximated, and then check if the same contracts fall within the concavity of f. Additionally, they compare the volume of the constraint being approximated and that of f. Their results show that the approximation can be very accurate for cubic constraints, while for conic constraints, the accuracy depends heavily on parameter tuning. Their experiments are, however, limited to three-dimensional constraints.

Linearizing the utility function

Hindriks, Jonker, and Tykhonov proposed the WAID method [35], a weighted averaging method for decomposing utility functions with interdependencies, transforming them to functions without such dependencies. These functions will then meet the input requirements of efficient algorithms designed for linear utility functions.

The WAID method, as the authors note, is inspired by the following observations: there are some bids that are not acceptable to agents or are too optimistic to be an outcome of the

negotiation. In effect, it is possible to indicate an expected region of utility of the outcome, and in real-life cases, a fairly accurate approximation can be obtained for utility functions that are "calm".

The central idea is to transform a nonlinear utility function $u(x_1, ..., x_d)$ into a linear approximation $u'(x_1, ..., x_d)$ of the form (1.2). This is done by approximating each of the evaluation functions $e_i(x_1, ..., x_d)$ of u by functions $e'_i(x_i)$ where the influence of the values of other issues $x_j, j \neq i$, on the associated value $e_i(x_1, ..., x_d)$ have been eliminated. To do this, they average out the influence of other issues on a particular issue.

Their method proceeds in the following four steps:

1. *Estimate the utility of an expected outcome*. This estimate is referred to as the "m-point" and is used to define a region of the utility space where the actual outcome is expected to be. They used 0.75 as a reference point.
2. *Select a type of weighting function*, ψ. This weighting function associates a weight with each point in the utility space. The closer the utility of that point is to the m-point, the higher the weight, making the approximation more accurate around the m-point.
3. *Approximate of the utility function*. To approximate the utility function, they weigh each evaluation function of the original utility function with the weighting function from step 2 and then integrate out all but one variable:

$$e'_i(x_i) = \int_V \psi_i(x_1, ..., x_d) e_i(x_1, ..., x_d) dV, \tag{2.3}$$

   where V is volume corresponding to some $d - 1$ of the d dimensions. The result of this step is a linear additive function of the form (1.2).
4. *Perform an analysis of the difference of the original and approximated utility space*. When a new bid is sent to or received from the opponent, the agent must check the associated utility in the original utility function to ensure that the bid is not worse than the current utility acceptance level. To assess the range of the error for any given utility level, they measure the difference $|u(x_1, ..., x_d) - u'(x_1, ..., x_d)|$.

The authors note that the technique is only applicable to utility functions that can be modeled by polynomial functions of modest power.

The authors later collaborated in [34] where they implemented the WAID method in a negotiation algorithm and analyzed the risk of a bad negotiation outcome when using an approximation of the utility function. To do that, they presented a checking procedure to control the risk of erroneous bids, which is implemented in step 4 above. That is, every time a bid $(x_1, ..., x_)$ is to be proposed to an opponent, the algorithm first queries $|u(x_1, ..., x_d) - u'(x_1, ..., x_d)| > \delta$ for some pre-specified δ. If this query is true, a new search for a bid is started. By implementing this checking procedure, they show that a trade-off can be made between computational cost, which increases for smaller δ, and approximation accuracy, which also increases for smaller δ.

2.2.2. The 2014 Automated Negotiating Agent Competition

The Automated Negotiating Agent Competition (ANAC) focuses specifically on the design of practical negotiation strategies. In particular, the overall aim of the competition is to advance the state-of-the-art in the area of autonomous negotiations, with an emphasis on the development of successful automated negotiators in realistic environments with incomplete information [4].

In 2014, the competition was held where agents negotiated with constraint-based utility functions of varying complexity, as well as dealing with large-scale outcome spaces. The complexity of

the utility functions was defined in terms of their entropy, which has inspired the complexity measure that will be used for this thesis. Furthermore, agents were not allowed to access the analytical form of their utility functions directly; therefore, they needed to explore the outcome space in a strategic manner to generate their bids as opposed to manipulating their analytical structure. As these agents are designed for scenarios similar to those considered in this thesis, this section provides an overview of some of the agents and their performance. In appendix C.2, the final version of this thesis's agent will compete against some of these agents using the same scenarios as in the competition itself.

GANGSTER: Genetic algorithm

De Jonge and Sierra entered the competition with their agent GANGSTER [17], which stands for Genetic Algorithm NeGotiator Subject To alternating offERs. As the name suggests, their agent searches the large-scale bid space using a genetic algorithm. This algorithm iteratively combines randomly sampled contracts in such a way that the highest utility contracts "stay alive".

Specifically, their agent samples 120 random contracts, selects the 10 best ones, and then constructs 110 new contracts by pairing each of these 10 best ones using a specific cross-over strategy. This strategy assigns a value to each issue that has an even probability of being from either of the two contracts in the corresponding pair. Then, out of the new 120 contracts (which contain the 10 best ones from before), they select the 10 best ones and repeat the process.

This process is carried out twice: once for contracts sampled randomly from the entire space and again for contracts selected to be within a certain distance of the opponent's previous bid, measured by the Manhattan distance. This distance decreases as the deadline moves closer, essentially acting as a concession strategy. Combining these two thereby balances exploration and exploitation.

They define the *aspiration level* as the time-dependent function $m_1(t)$ and the maximum distance as the time-dependent function $m_2(t)$. Then, for the set X of the 20 contracts found — the 10 from the global search and the 10 for the local search — the contract to be bid is chosen based on these functions. In particular, they define the subset

$$Y = \{\mathbf{x} \in X | u(\mathbf{x}) \geq m_1(t) \wedge \text{dist}_t(\mathbf{x}) \leq m_2(t)\}$$

where u is the utility function and $\text{dist}_t(x)$ is the lowest Manhattan distance between $\mathbf{x}$ and any bid made by the opponent. The bid to be made is then defined as the contract $\mathbf{x}^* \in Y$ with the highest Manhattan distance between itself and any bid made by the agent. This was believed to increase the chance of the bid being accepted by the opponent while also ensuring a high utility for the agent.

Out of the top five agents, GANGSTER performed best in terms of average individual utility and shared the first prize with AgentM in terms of average utilitarian social welfare.

AgentM: Simulated Annealing algorithm

Niimi and Ito entered the competition with AgentM [61], a compromiser agent that performs bidding with the goal of improving both its own utility and its opponent's utility.

The agent constructs three bids: a bid found using Simulated Annealing, the highest utility bid offered by the opposing agent, and a frequency bid. The frequency bid is obtained by keeping a log of the frequency of the values proposed by the opponent for each issue, and then choosing the most frequent issue values for each issue. This is depicted in Figure 2.3.

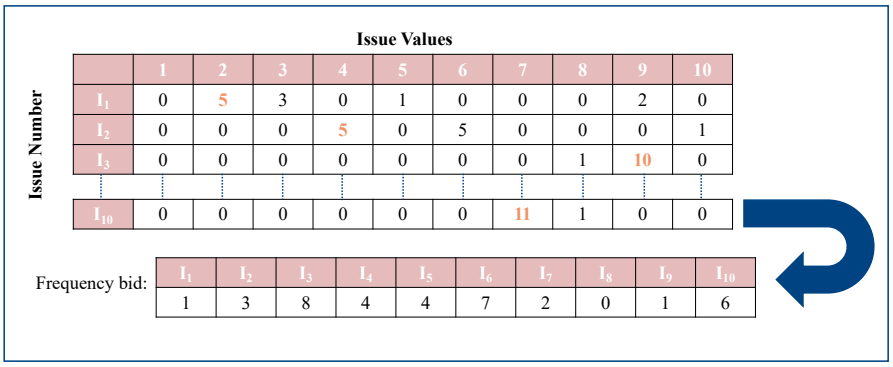


Figure 2.3: The frequency bid is obtained by counting the occurrences of issue values the opponent has proposed. Figure is adapted from [61]

After constructing these three bids, the agent tries out combinations of them using the bid found using Simulated Annealing as a base bid, and proposes the best one. When presented with an opponent's bid, the agent then simply accepts if it is higher than the worst of its own bids so far.

Out of the top five agents, AgentM placed second in terms of individual utility, and did best in terms of the average distance from the Pareto frontier, the average Nash distance, and, as mentioned before, shared the first prize with GANGSTER in terms of average utilitarian social welfare.

G2A: Greedy Coordinate descent algorithm

Szöllősi-Nagy, Festen, and Skarżyńska entered the competition with G2A [70], an agent that uses a greedy variation on the Coordinate Descent (CD) method in order to find high Social Welfare outcomes.

Their CD algorithm performs line search by maximizing the utility function over a single issue at a time with the others held fixed, and then iterates until a stopping condition. They used a greedy CD strategy in the sense that it optimizes over the issue that gives the best improvement of utility first. The initial bids are chosen based on the negotiation phase. In the opening phase of the negotiation, they are uniformly sampled from the entire domain. During the middle game phase however, they are chosen from the opponent's bid history. They do that as an attempt to trace a path between the opponent's bids and bids that are good for G2A.

To choose which issue to maximize over next, the issue selection strategy iteratively loops through all d issues and evaluates for each one

$$x_i^{r+1} \in \underset{x_i}{\operatorname{argmax}}\, u(x_1^r, ..., x_{i-1}^r, x_i, x_{i+1}^r, ..., x_d^r)$$
$$x_j^{r+1} = x_j^r, \forall j \neq i$$

and then chooses the issue giving the highest utility and makes the corresponding change before repeating the loop. This process is then terminated when utility values stabilize or some number of iterations has been reached.

They use a simple opponent model in order to prevent the agent from making bids that are much better for the opponent than itself. Their opponent model is similar to the frequency bid table used by AgentM. They keep a log of the frequency of issue values proposed by the opponent and compare it to the expected frequency of issue values if each option were chosen uniformly at random.

G2A did not reach the overall top five but placed fifth in the final round in terms of individual utility and sixth in terms of social welfare.

WhaleAgent: Simulated Annealing and Hill Climbing algorithm

Sato and Ito entered the competition with WhaleAgent [67], an agent that searches the bid space using a combination of simulated annealing and hill climbing, and a combination of a hardheaded strategy and a conceder strategy to set a utility threshold for deciding whether to accept or propose a bid.

The agent begins by choosing the best bid out of the bids offered by the opponent and uses that as a starting point for hill climbing. If the bid found using hill climbing has a higher utility than the current threshold, the agent proposes that bid. If not, the agent initializes simulated annealing from a random starting point and then compares the bid found with the same threshold. This is repeated until a bid is found that has a higher utility than the threshold.

The agent initially sets the utility threshold using a hardheaded strategy, which dynamically updates the threshold based on the best bid found using simulated annealing, as well as time, and decreases it slightly as the deadline approaches. The agent then switches to a conceder strategy if the opponent proposes the same bid twice. The conceder strategy lowers the threshold at a faster pace than the hardheaded strategy.

Out of the top five, WhaleAgent placed third in terms of individual utility and second in terms of all social welfare measures.

BraveCat: Iterative deepening algorithm

Zafari and Nassiri-Mofakham entered the competition with BraveCat [77], an agent that uses iterative deepening search to find bids, which allows it to overcome the limitations imposed by the amount of memory needed in large domains.

For its bidding strategy, BraveCat sets a utility threshold that is gradually lowered over time, but shows a form of randomness and awards nice moves by the opponent. They define a nice move by the opponent as one that is better than the previous best bid proposed by the opponent, in terms of BraveCat's utility function. If the opponent makes a nice move, the utility threshold is lowered for the next bid search.

The authors note that when deciding on a bid to propose to the opponent, the bidding strategy of the agent could easily find the bids with utility values equal to or greater than the target utility value of the agent if storing all bids in memory was feasible. However, due to limitations imposed by the huge amount of memory needed, doing so would not be possible. To overcome the challenge of finding good bids, they adopt an iterative deepening search. Their search algorithm iteratively samples randomly from the bid space and compares the utilities of the bids found to a threshold, which decreases as the number of iterations increases. Then, among all bids that have a higher utility than the threshold defined in the bidding strategy, the bid to be offered is chosen based on Euclidean distance from previous opponent bids.

For a candidate bid $\mathbf{x}$, they compute the Euclidean distance, $N^i_{\mathbf{x}}$, from $\mathbf{x}$ and the last $i = 1, ..., 100$ bids made by the opponent, and multiply that with the time, $M^i_{\mathbf{x}}$, since the opponent offered the corresponding bid:

$$F^i_{\mathbf{x}} = N^i_{\mathbf{x}} \times M^i_{\mathbf{x}}, \quad i = 1, ..., 100.$$

This measure is then normalized as $P^i_{\mathbf{x}} = \frac{1}{F^i_{\mathbf{x}}}$, and the candidate bid that gets offered is the one

maximizing the estimated opponent utility:

$$u^{OP}(\mathbf{x}) = \sum_{i=1}^{100}(1 - 0.3t_i)P^i_{\mathbf{x}}, \quad 0 \leq t \leq 1$$

where t_i is the time at which the last ith bid was received from the opponent, and $(1 - 0.3t_i)$ represents the assumed concession rate of the opponent.

BraveCat did not reach the top five but placed ninth in terms of individual utility and seventh in terms of social welfare.

2.3. Research Gap

This review has examined two complementary dimensions of the literature on automated negotiation and the facilitation of nonlinear utility functions: the design of mechanisms and the design of agents.

On the mechanism design side, the mediator-based approaches of Klein et al. [44] and the bidding-based framework proposed by Ito, Hattori, and Klein [37] laid a foundation for future work. While theoretically effective, these approaches encountered severe scalability challenges. Successive extensions sought to reduce computational overhead and improve robustness. These extensions showed that when exhaustive search becomes intractable, a focused search becomes necessary.

Considerable effort has been invested in designing protocols where simplification of utility functions is carried out before bid search. These have shown how structural simplification can greatly improve efficiency. The common assumption here is, however, that agents have access to the analytical form of their utility functions, that a mediator is present to guide the search, and/or that the functions exhibit structures that can be manipulated by grouping constraints together or decomposing them based on various heuristics.

On the agent design side, research has explored a variety of strategies. With a few exceptions, existing approaches attempt to search directly in the original utility space. The WAID method linearizes utility functions but is limited to relatively simple and continuous utility functions. Hadfi and Ito [28] explored the use of generalized Gaussian distributions for approximating constraint-based utility functions. Their method was, however, highly sensitive to parameter tuning, and experiments were not conducted in a negotiation setting. Their results did, however, show that using the generalized Gaussian distribution to approximate constraint-based utility functions is a promising approach.

The 2014 ANAC further showcased practical approaches. However, all agents searched directly in the utility function. Despite their diversity, most of the agents prioritized bid generation and concession strategies over sophisticated opponent modeling. Erez and Zuckerman [19] reported that intricate learning algorithms offered limited practical benefit in scenarios where a negotiation is not repeated with data from previous rounds being available. Nonetheless, all agents used some form of a heuristic in order to direct their bid search towards zones where a deal was likely to be struck, and a high-utility bid might be found.

Although much of the reviewed literature focuses on finding socially optimal bids as opposed to high-utility bids, as will be the goal in this thesis, and often involves searching directly within the utility function, it still offers valuable insights. After all, both tasks involve searching for a needle in a haystack. Techniques that prove effective in identifying socially optimal bids can often be adapted, at least in part, to support the search for high-utility bids. Indeed,

the considerations that shaped the methods discussed in this chapter are also relevant to the problem addressed in this thesis. Chapter 4 presents the proposed method, which incorporates many of these considerations, either directly or in adapted form.

The research objective of this thesis is to explore how well an autonomous negotiator performs in finding high-utility bids in a surrogate model of its utility function, when it is modeled as a quadratic polynomial. Despite the rich body of literature surrounding function approximations, applying it in an agent's bidding strategy remains limited. As noted earlier, the most common approach in the literature reviewed for this thesis is to directly search the utility function, or to manipulate structural properties it might have. The only answer to this thesis's research question found in the literature was that the generalized Gaussian distribution might work quite well, and that a linearization might perhaps not. These two methods are similar to a quadratic polynomial in their own way. A linearization represents a polynomial of a single degree lower than the quadratic polynomial, and the generalized Gaussian distribution gives a concave approximation. The quadratic polynomial is unimodal, meaning that although it is not always concave, it does have at most a single optimum. What all these methods do have in common though is that they can be implemented as basis functions in regression models.

Exploring the use of regression models as surrogates for constraint-based utility functions remains to be done. This work furthermore opens space for future research as there are countless other basis functions to explore, such as Fourier-based functions, higher-order polynomials, the generalized Gaussian distribution, or even a combination of these functions.

3

Mathematical background

Before constructing the agent, several mathematical considerations need to be made. These are the topics of this chapter.

First and foremost, in order to answer the Main Research Question, a way of quantifying the complexity of a constraint-based utility function must be established. This is the topic of the first section.

Second, the sampling step cannot be a simple uniform sample over the entire domain due to the curse of dimensionality and the difference in how expressive a quadratic polynomial is and the utility function. These considerations raise the question of how the sampling step can aid in the function approximation, which is the topic of the second section. Sampling will be performed in regions of the domain, and the shapes and placements of these regions will be discussed, as well as the sample distribution.

Third, once the sampling step has been established, the modeling of the utility function with quadratic polynomials is addressed. This is the topic of the third section. This will be done via quadratic regression, where quadratic polynomials are used as basis functions in a regression model that is linear in the parameters. An analytical formula for the parameters is derived through maximum likelihood and least squares, and the regression model with both linear and quadratic basis functions is described. The section then concludes with a discussion on the bias-variance trade-off.

Fourth, searching the quadratic surrogate for a high utility bid is addressed. Even though quadratic polynomials are trivial to optimize over a closed domain in low dimensions, doing so in a high dimensional setting is a completely different story. However, as quadratic polynomials possess various desirable trades for optimization, such as being smooth and unimodal, second-order optimization methods that rely on the Hessian are ideal for the task. These are the topics of the fourth section.

Finally, in the fifth section, the discussion of this chapter will come to a conclusion, which is addressed in the method chapter following this one.

3.1. The complexities of hypercubic functions

In a hypercubic constraint-based utility function in d dimensions, each constraint c_k is a hypercube of dimensionality $\gamma_k \leq d$ with *widths* $\mathbf{v}_k = (v_{ki})_{i \in \Gamma_k}$ where $\Gamma_k \subseteq [d]$ is the index set representing the γ_k issues that *define* c_k (i.e., the γ_k axes to which the sides of the constraint

are parallel), and is thus denoted as $c_k(\gamma_k, \mathbf{v}_k)$. The widths should not be confused with the *side length*, as it refers to the number of contracts along the corresponding axis that the constraint spans. In this way, a constraint is formally defined as

Definition 1:

A *hypercubic constraint* in a negotiation domain $\Omega = \prod_{i=1}^{d} I_i$ is defined as:

$$c_k(\gamma_k, \mathbf{v}_k) := \prod_{i \notin \Gamma_k} [0, |I|) \times \prod_{i \in \Gamma_k} \left[x_{ki}^{\min},\ x_{ki}^{\min} + v_{ki}\right) = \prod_{i=1}^{d} L_{ki} \tag{3.1}$$

where x_{ki}^{min} is some integer in $[0, |I_i| - v_{ki} - 1]$ for all $i = 1, ..., d$. Each interval in (3.1) specifies the valid values of the corresponding issue required to satisfy the constraint, and is denoted by L_{ki}.

A contract $\mathbf{x} = (x_1, ..., x_n)$ then satisfies constraint $c_k(\gamma_k, \mathbf{v}_k)$ if and only if $x_i \in c_k(\gamma_k, \mathbf{v}_k)$ for all i.

Side note:

The former product over all $i \notin \Gamma_k$ in (3.1) contains the entire corresponding issue domains as their value is irrelevant as to whether the contract satisfies it or not. The latter product, therefore, represents the hypercube itself, while the whole expression represents the constraint.

When satisfied, each constraint has an associated utility value which is equal to some constant a_k, i.e., the *height* of the cube, which is zero if $c_k(\gamma_k, \mathbf{v}_k)$ is not satisfied. Thus, the hypercubic constraint-based utility function, which will from now on be referred to as simply a hypercubic function, is formally defined as

Definition 2:

A *hypercubic function* u_C is a function of the form

$$u(\mathbf{x}) = \sum_{c_k \in C} \omega(c_k, \mathbf{x}) \tag{3.2}$$

where

$$w(c_k, \mathbf{x}) = \begin{cases} a_k & \text{if} \quad \mathbf{x} \in c_k \\ 0 & \text{otherwise} \end{cases}$$

and c_k is a hypercubic constraint for all $k = 1, ..., |C|$.

This definition results in a bumpy, nonlinear utility function with high points where many constraints are satisfied. Figure 1.1 shows a top view of a hypercubic function along with an example explaining it, and Figure 3.1 shows a hypercubic function over a domain of two issues with multiple constraints and their heights representing their added utility values.

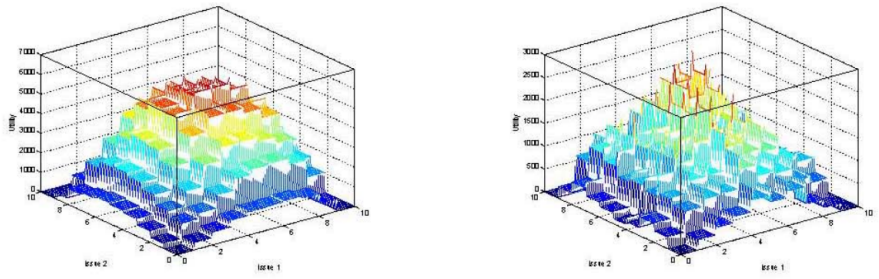


Figure 3.1: A 3d view of a hypercubic function.

In [29], Hadfi and Ito define the parameters that could affect the complexity of a general constraint-based utility function as the number of issues in the negotiation domain, the number of constraints that define the function, the distribution of the number of issues involved in a constraint, and the domain size and type of the issues themselves. However, another parameter commonly used for the specific case of hypercubic constraints is the constraint width [25, 54], or the $\mathbf{v}_k$ parameter.

Although other parameters can be considered, such as the number of groups; that is, assuming the constraints are grouped together in the domain, how many groups are there; or the proportion of disjoint constraints, the parameters considered for this thesis are:

- m: The number of constraints that define the function
- γ: The dimensionalities of the constraints
- v: The widths of the constraints
- d: The number of issues in the domain

where $\gamma := (\gamma_k)_{k=1}^m$ and $v := (\mathbf{v}_k)_{k=1}^m$. A hypercubic function can then be parametrized by (m, γ, v, d)-tuples, which can be used to generate these functions. This will be done to generate results later, but the following definition will come in handy in this section.

Definition 3:

A *uniformly generated hypercubic function* in a d-dimensional negotiation domain Ω is a hypercubic function parametrized by (m, γ, v, d), where:

- For each $k = 1, ..., m$ and each $i = 1, ..., d$

$$x_{ki}^{\min} \sim U\big(0,\ |I_i| - v_{ki} - 1\big)$$

  are independent uniform random variables.
- For each $k = 1, ..., m$, Γ_k is is chosen uniformly at random among all γ_k-sized subsets of the index set $[d]$.

In other words, the placements of the constraints and which issues define them are uniform random.

As can be seen deduced from earlier discussion and Figures 3.1 and 1.1, two or more constraints can *intersect*, which essentially creates a new one having their combined utility. Formally,

Definition 4:

Let $c_1(\gamma_1, \mathbf{v}_1)$ and $c_2(\gamma_2, \mathbf{v}_2)$ be two constraints in a hypercubic function. Then they are said to intersect if

$$[x_{1i}^{min}, x_{1i}^{min} + v_{1i}) \cap [x_{2i}^{min}, x_{2i}^{min} + v_{2i}) \neq \emptyset \; \forall i \in \Gamma_1 \cap \Gamma_2.$$

Their intersection constitutes an *intersection constraint* which *contains them*, and is referred to as a

- *Type 1 constraint* if
$$\Gamma_1 \cap \Gamma_2 = \emptyset$$
- *Type 2 constraint* if
$$\Gamma_1 \cap \Gamma_2 \neq \emptyset$$

A constraint that is not an intersection constraint is referred to as a *base constraint*.

Definition 4 is illustrated in Figure 3.2.

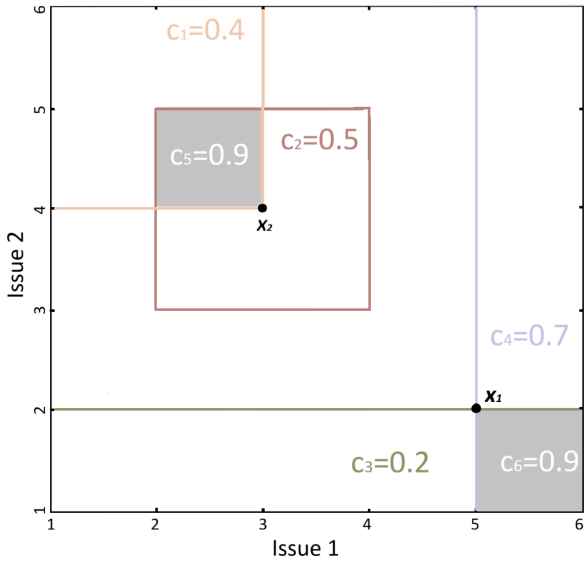


Figure 3.2: A negotiation domain with four base constraints: c_1 and c_2 (both with $\gamma = 2$ and $v = 3$), and c_3 and c_4 (both with $\gamma = 1$ and $v = 2$), and two intersection constraints: c_5 and c_6, along with their corresponding utilities. Constraints $c_1 = [1, 3] \times [4, 6]$ and $c_2 = [2, 4] \times [3, 5]$ intersect to create constraint $c_5 = [2, 3] \times [4, 5]$ which has their combined utility of 0.9. Constraint c_5 is an intersection constraint of type 2 as the intervals defining constraints c_1 and c_2 intersect; i.e. $\Gamma_1 = \{1, 2\} = \Gamma_2$. On the other hand, constraints $c_3 = [1, 6] \times [1, 2]$ and $c_4 = [5, 6] \times [1, 6]$ intersect to create constraint $c_6 = [5, 6] \times [1, 2]$ which has their combined utility of 0.9. Constraint c_6 is an intersection constraint of type 1 as the intervals defining constraints c_3 and c_4 do not intersect; i.e. $\Gamma_3 = \{2\}$ while $\Gamma_4 = \{1\}$ so $\Gamma_3 \cap \Gamma_4 = \emptyset$.

Side note:

An intersection constraint can be viewed as the intersection between two constraints, with one of them being a base constraint. Indeed, all intersection constraints are intersections between base constraints, but any subset of constraints that an intersection constraint consists of is itself a constraint that intersects with base constraints.
For example, if $c_5 = c_1 \cap c_2 \cap c_3$ where c_1, c_2 and c_3 are base constraints, then the intersection of any pair, say c_1 and c_2, constitutes an intersection constraint. That is, $c_4 = c_1 \cap c_2 \neq \emptyset$,

so that c_5 can equivalently be written as

$$c_5 = c_3 \cap c_4.$$

Furthermore, for a utility function u_C and a base constraint c_k, it holds that $c_k \in C$. However, if c_k is an intersection constraint, then $c_k \notin C$ but

$$c_k = \bigcap_{i \in S \subseteq [m]} c_i, \quad \text{where } c_i \in C \text{ for all } i$$

and S is the index set of base constraints that make up c_k.

Moving forward, the following assumption is made:

Assumption 7:

In this thesis, γ_k and $\mathbf{v}_k$ are assumed to be fixed across all m base constraints in a uniformly generated hypercubic function. Specifically, for any pair of constraints $c_k(\gamma_k, \mathbf{v}_k), c_j(\gamma_j, \mathbf{v}_j) \in C$,

$$\gamma_k = \gamma_j = \gamma \quad \text{and} \quad v_{ki} = v_{ji} = v \quad \text{for all } i = 1, \ldots, \gamma,$$

where $\gamma \leq d$ and $v \leq |I|$. Hence, γ and v are treated as constants rather than tuples, representing respectively the *dimensionality* and the *width* of each constraint in C.

Each of these four complexity parameters is dependent on the other, meaning that changing one of them will create a change in the complexity produced by the others. This, and their individual contributions to complexity, is the topic of this section.

3.1.1. The dimensionality of a constraint

The γ parameter, or the number of issues involved in a constraint, dictates the level of interdependency between issues. A higher γ means a constraint is represented by a higher dimensional hypercube. By assumption 7, all m base constraints in a utility function have the same γ and so a higher value of γ means that the hypercubic function has a higher dimensional structure.

Consider a constraint with $\gamma = 1$. That is, there is only a single issue involved in this constraint, and all but one interval in Expression (3.1) are the entire corresponding issue domain. Then, by uniformly randomly picking any contract from the space, the probability of it satisfying this constraint is simply the probability of the sample's value for that issue being inside the single interval defined by the constraint. Once γ increases, this probability decreases exponentially as each issue value is independent, as per Definition 3. This is summarized in Theorem 2 in the upcoming subsection.

3.1.2. The width of a constraint

The width of a base constraint, v, indicates the lengths of the γ intervals that define it. Thus, a constraint c_k with $\gamma = j \leq d$ and width v has an associated utility value which is obtained by bids where $x_i \in L_{ki}$ for all $i \in \Gamma_k$. The interplay between γ and v is generalized in the following theorem:

Theorem 2:

Let $c_k(\gamma_k, v_k)$ be one of m base constraint from a uniformly generated hypercubic function. Then, for a uniformly randomly sampled contract $\mathbf{x} \in \Omega$, the following holds:

$$\mathbb{P}(\mathbf{x} \in c_k) = \left(\frac{v}{|I|}\right)^{\gamma}.$$

Proof: As per Assumption 2, each $x_i \in [0, |I|)$. As L_{ki} is placed in $[0, |I|)$ at uniform random, the probability that $x_i \in L_{ki}$ is clearly $\frac{|L_{ki}|}{|I|}$. As per Assumption 7, $|L_{ki}|$ is constant for all i so this becomes $\frac{v}{|I|}$. By Definition 3, each x_i are independent so $\mathbb{P}(x_i \in L_{ki} \wedge x_j \in L_j) = \mathbb{P}(x_i \in L_1)\mathbb{P}(x_j \in L_j)$ where $i \neq j$. Finally as $L_r = [0, |I|)$ for all $r \notin \Gamma_k$,

$$\mathbb{P}(\mathbf{x} \in c_k) = \left(\frac{v}{|I|}\right)^{\gamma}$$

□

Corollary 1:

Let $c_k(\gamma_k, \mathbf{v}_k)$ be any constraint from a uniformly generated hypercubic function. Then, for a uniformly randomly sampled contract $\mathbf{x} \in \Omega$, the following holds:

$$\mathbb{P}(\mathbf{x} \in c_k) = \prod_{i \in \Gamma_k} \frac{v_{ki}}{|I|}$$

Consider again the three constraints in Figure 1.1. These constraints showcase both the difference between a wide constraint and a narrow constraint, as well as the interplay between γ and v. Constraints $c1$ and c_2 both have width $v = 3$ and an associated $\gamma = 2$. This means that the two pairs of intervals that define them capture three integer values each, and that both issues need to be inside their corresponding interval for either constraint to be satisfied. On the other hand, constraint $c3$ has width 2 and an associated $\gamma = 1$. By Theorem 2, a random sample has a $\left(\frac{3}{6}\right)^2$ probability of satisfying constraints c_1 or c_2 and a probability of $\frac{2}{6}$ of satisfying c_3.

3.1.3. The number of issues

It goes without saying that the more issues there are, the greater the search space becomes, and hence its complexity. That is, when the search space grows, the difficulty of finding an exact point grows exponentially. Exhaustively searching the integer domain of $[0, 9]^2$ for the point $(1, 1)$ is very easy, but finding $(1, 1, ..., 1)$ in $[0, 9]^{50}$ is far fetched. However, in terms of constraint-based utility functions, the complexity stemming from the size of the space is not as prominent in itself but rather the potential it creates for the other parameters.

As hypercubic constraints are defined as intervals, satisfying a constraint boils down to finding the correct interval for the correct issue. Consider, for example, a utility function consisting of a single constraint in the $[0, 9]^{50}$ domain, where the constraint is defined as the interval $[0, 2]$ for issue 2. Now the complexity associated with the high dimensionality is essentially a greater memory requirement as the other 49 issues become irrelevant. Indeed, by Theorem 2, simply picking a random point in the domain would yield a 0.3 probability of satisfying this constraint, irrespective of the dimensionality of the space.

Now, circling back to the first paragraph: if the constraint in the example above was not simply a 3-value interval for a single issue, but rather for all 50 issues, then the probability becomes

$0.3^{50} \approx 1.4 \times 10^{-26}$. That is, as d grows, the potential for γ to grow increases. As will be shown in the upcoming section, the number of constraints m can also grow the potential complexity for both γ and v, hinting that the induced complexity of d is highly dependent upon m.

3.1.4. The number of constraints

At a first glance, more constraints might seem like something associated with less complexity: the more constraints, the more likely that a random sample satisfies one of them. Although that statement is true, the complexity arises with the additional structure created from more constraints; specifically when two or more of them intersect, essentially creating a new — often more complex — constraint with a combined utility of those intersecting.

Now, the more constraints there are, the more of them can intersect, and the higher the probability of two or more of them intersecting. As an intersection constraints has a combined utility of the constraints that it consists of, the global maximum of a hypercubic function is generally at an intersection constraint. Furthermore, an intersection constraint is generally more complex than the base constraints. To see this, consider the following lemma:

Lemma 1:

Let $c_3 = (\gamma_3, \mathbf{v}_3)$ denote the intersection of any two constraints $c_1(\gamma_1, \mathbf{v}_1)$ and $c_2(\gamma_2, \mathbf{v}_2)$ in a hypercubic function. Then, for γ_3 and each v_{3j}, where $j \in \Gamma_3$, the following conditions hold:

(1). $\max(v_{1j} + v_{2j} - |I|, 1) \leq v_{3j} \leq \min(v_{1i}, v_{2i})$

(2). $\max(\gamma_1, \gamma_2) \leq \gamma_3 \leq \min(\gamma_1 + \gamma_2, d)$

Proof:

1. Consider any interval $[x_{1i}^{min}, x_{1i}^{min} + v_{1i}) \cap [x_{2i}^{min}, x_{2i}^{min} + v_{2i}) = [x_{3i}^{min}, x_{3i}^{min} + v_{3i})$ where $i \in \Gamma_3$. If this interval is empty then the two constraints did not intersect which is a contradiction. As $[x_{3i}^{min}, x_{3i}^{min} + v_{3j})$ is nonempty, $v_{3j} > \min(v_{1j}, v_{2j})$ cannot hold. Thus $v_{3j} \leq \min(v_{1j}, v_{2j})$ must hold.

   If $v_{1i} + v_{2i} > |I|$, then v_{3i} must be at least $v_{1i} + v_{2i} - |I|$. If $v_{1i} + v_{2i} \leq |I|$ then $v_{1i} + v_{2i} - |I| \leq 0$. However, as the two intervals do intersect, $\max(1, v_{1i} + v_{2i} - |I|) \leq v_{3j}$ must hold.

2. As $\gamma_3 = \gamma_1 + \gamma_2 - |\Gamma_3|$, $\gamma_3 = \gamma_1 + \gamma_2$ is obvious when c_3 is a type 1 constraint. As c_3 is made up of c_1 and c_2, $\gamma_3 > \gamma_1 + \gamma_2$ can clearly not hold.

   In case c_3 is a type 2 constraint, $\gamma_3 = \gamma_1 + \gamma_2 - |\Gamma_3|$ takes it's lowest value when $|\Gamma_3|$ is as big as it can get. This is clearly when $|\Gamma_3| = \min(\gamma_1, \gamma_2)$. Thus, $\gamma_3 = \gamma_1 + \gamma_2 - \min(\gamma_1, \gamma_2)$ is the lowest attainable value for γ_3, or equivalently $\gamma \geq \max(\gamma_1, \gamma_2)$.

□

In other words, if a hypercube is thought to be more "complex" when it is of a higher dimensionality and with smaller widths, an intersection constraint is never of less complexity than the constraints that it consists of.

Side note:

Even though the m base constraints are generated under Assumption 7, intersection constraints can be any shape and size dictated by Lemma 1. This can be seen in Figure 3.2. Both intersection constraints c_5 and c_6 have smaller widths than the constraints that they

contain, and c_6 has a $\gamma = 2$ while the constraints that it contains both have $\gamma = 1$.

Following Lemma 1, the following can easily be derived:

Corollary 2:

A type 1 constraint $c_3(\gamma_3, \mathbf{v}_3)$ that is made up of the constraints $c_1(\gamma_1, \mathbf{v}_1)$ and $c_2(\gamma_2, \mathbf{v}_2)$ has $\gamma_3 = \gamma_1 + \gamma_2$ and $\mathbf{v}_3 = (\mathbf{v}_1, \mathbf{v}_2)$.

By definition 3, intersection constraints occur randomly. Therefore, a hypercubic function with $m = 100$ can have a vastly more constraints than the 100 base constraints. This number is bounded by the size of the superset of the 100 base constraints, and the domain size. That is, if a utility function is constructed to have $m = 2$, these constraints might intersect to create a third one. When a $m = 3$, there can be at most 7 constraints. This was generalized in the following theorem:

Theorem 3:

Let K be the total number of unique constraints in a hypercubic function $u = (m, \gamma, v, d)$ in a negotiation domain of size $|\Omega| = d \times |I|$. Then the following holds:

$$K \leq \min(2^m, M)$$

where

$$M = \sum_{i=\gamma}^{d} \binom{d}{i} |I|^i \tag{3.3}$$

Proof: Clearly K cannot be greater than 2^m as it corresponds to the size of the superset constructed from m sets. However, as the domain is bounded, there cannot be m unique constraints in it if m is greater than M. Indeed, for each possible way of choosing a constraint $(\mathbf{v}_k, \gamma_k)$, there are

$$\prod_{i=1}^{\gamma} (|I| - v_{ki} + 1)$$

ways to place the lower bounds of each $\mathbf{v}_k$. That is, there are

$$\binom{d}{\gamma_k} \prod_{i=1}^{\gamma} (|I| - v_{ki} + 1) \leq \binom{d}{\gamma_k} |I|^{\gamma_k} \tag{3.4}$$

ways to place the lower bound of each $\mathbf{v}_k$, where the upper bound in Expression (3.4) is derived from Lemma 1. Now, For any set of base constraints, the attainable range for an intersection constraint's γ_k is $[\gamma, ..., d]$. Therefore, for any attainable γ_k, Expression (3.4) provides an upper bound on the ways to place the lower bound of each $\mathbf{v}_k$, resulting in a total of

$$\sum_{i=\gamma}^{d} \binom{d}{i} |I|^i$$

attainable unique constraints, which is exactly M in Expression (3.3). □

When $m = 100, \gamma = 10, v = 2$ and $d = 50$, the actual number of constraints has an upper bound of approximately 10×10^{30}, and each of these constraints will most likely have higher values of γ and lower values of v than the base constraints, as per Lemma 1. Note that this is an upper bound. This number occurs with an extremely low probability, as will be shown shortly.

3.1.5. Quantifying the complexity of a constraint-based utility function

In [29], Hadfi and Ito defined the complexity of a constraint-based utility function using the theoretical notion of entropy:

$$H(\pi) = -\sum_{j=1}^{m} \pi_j \log \pi_j$$

where $\pi = (\pi_1, ..., \pi_m)$ is the distribution of the number of issues per constraint. However, this does not capture the complexities imposed by narrowing the constraints. Furthermore, simply viewing the distribution π gives little insight into the scaling of complexity with increasing dimension of constraints, or γ.

To quantify the complexity of a hypercubic constraint-based utility function, this thesis adopts the approach of considering the difficulty of randomly sampling individual constraints. That is, how expressive a model needs to be to accurately approximate the utility function, based on the theoretical performance of the simplest possible model: a uniform random sample. To do this, the uniformly generated hypercubic function will be viewed from a probabilistic perspective and entropy will then be used to quantify complexity. The probability used for the entropy is based on Theorem 2 for all constraints in the function. To do this, the probability of a particular intersection occurring in a uniformly generated hypercubic function is needed. To derive this, the following lemma is needed:

Lemma 2:

For any two constraints $c_1(\gamma_1, \mathbf{v}_1)$ and $c_2(\gamma_2, \mathbf{v}_2)$ in a uniformly generated hypercubic function, the random variable

$$X \;:=\; |\Gamma_1 \cap \Gamma_2|$$

is hypergeometrically distributed with parameters (d, γ_1, γ_2). That is, for every integer n,

$$\mathbb{P}(X = n) \;=\; \frac{\binom{\gamma_1}{n}\binom{d-\gamma_1}{\gamma_2-n}}{\binom{d}{\gamma_2}}$$

where $\max\{0, \gamma_1 + \gamma_2 - d\} \leq n \leq \min\{\gamma_1, \gamma_2\}$ and $\mathbb{P}(X = n) = 0$ otherwise.

Proof: See Appendix B.1 □

From Lemma 2, the following can easily be derived:

Corollary 3:

The probability that any two constraints $c_1(\gamma_1, \mathbf{v}_1)$ and $c_2(\gamma_2, \mathbf{v}_2)$ in a uniformly generated hypercubic function intersect to form a type 1 constraint is

$$\mathbb{P}(X = 0) \;=\; \frac{\binom{d-\gamma_1}{\gamma_2}}{\binom{d}{\gamma_2}}.$$

Now that the probability of a type 1 intersection has been derived, the probability of a type 2 intersection is needed to complete the picture. To derive the probability of a type 2 intersection, the probabilities surrounding the intresection of two subintervals of the same interval is needed:

Lemma 3:

Let $v_1, v_2, |I|, v_3 \in \mathbb{N}$ with $v_3 \leq \min(v_1, v_2)$ and $v_1, v_2 \leq |I|$. Furthermore, let $L_1 = [x_1^{\min}, x_1^{\min} + v_1)$ and $L_2 = [x_2^{\min}, x_2^{\min} + v_2)$ be two intervals placed independently and uniformly at random in $[0, |I|)$. Then,

(1)
$$\mathbb{P}(L_1 \cap L_2 = \emptyset) = \frac{(|I| - v_1 - v_2 + 1)(|I| - v_1 - v_2 + 2)}{(|I| - v_1 + 1)(|I| - v_2 + 1)}.$$

(2)
$$\mathbb{P}\left(|I_1 \cap I_2| = v_3\right) = \frac{1}{(|I| - v_1 + 1)(|I| - v_2 + 1)} \sum_{\delta \in D_{v_3}} N(\delta),$$

where

- If $v_3 < \min(v_1, v_2)$, then $D_{v_3} = \{\delta = |v_1 - v_3|\}$ and
$$N(\delta) = 2 \cdot \max\left(0, \min(|I| - v_1, |I| - v_2 - \delta) + 1\right).$$
- If $v_3 = v_2 \leq v_1$, then $D_{v_3} = \{0, 1, \ldots, v_1 - v_2\}$ and
$$N(\delta) = \max\left(0, \min(|I| - v_1, |I| - v_2 - \delta) + 1\right).$$

Proof: (proof of (2). is in Appendix B.2). An interval of length v_1 in $[0, |I|)$ can start at any of $|I| - v_1 + 1$ positions, and similarly an interval of length v_2 can start at any of $|I| - v_2 + 1$ positions. Thus there are $(|I| - v_1 + 1)(|I| - v_2 + 1)$ equally likely pairs of starting positions.

The intervals L_1 and L_2 are disjoint if and only if one ends before the other starts. There are $|I| - v_1 - v_2 + 1$ possible gaps between them, and for each such gap there are two orderings (either L_1 is to the left of L_2 or vice versa), plus the boundary case where they exactly touch. Altogether this yields $(|I| - v_1 - v_2 + 1)(|I| - v_1 - v_2 + 2)$ disjoint placements. Dividing disjoint cases by total cases gives the stated probability. □

The discussion of this section has lead to the following Theorem:

Theorem 4:

The probability that any two constraints $c_1(\gamma_1, \mathbf{v}_1)$ and $c_2(\gamma_2, \mathbf{v}_2)$ selected from a uniformly generated hypercubic function at uniform random intersect to create the intersection constraint $c_3(\gamma_3, \mathbf{v}_3)$is

$$\mathbb{P}\big(\Gamma_1 \cap \Gamma_2 = \Gamma_3\big) \prod_{j \in (\Gamma_1 \cap \Gamma_2)}^{\gamma_3} \mathbb{P}\Big(\big[x_{1j}^{\min},\ x_{1j}^{\min} + v_{1j}\big] \cap \big[x_{2j}^{\min},\ x_{2j}^{\min} + v_{2j}\big] = v_{3j}\Big) \tag{3.5}$$

where

$$\mathbb{P}(\Gamma_1 \cap \Gamma_2 = \Gamma_3) \ = \ \frac{\binom{\gamma_1}{\gamma_3}\binom{d-\gamma_1}{\gamma_2-\gamma_3}}{\binom{d}{\gamma_3}\binom{d}{\gamma_2}}$$

and

$$\mathbb{P}\Big(\big[x_{1j}^{\min},\ x_{1j}^{\min} + v_{1j}\big] \cap \big[x_{2j}^{\min},\ x_{2j}^{\min} + v_{2j}\big] = v_{3j}\Big) = \frac{1}{(|I| - v_{1j} + 1)(|I| - v_{2j} + 1)} \sum_{\delta \in D_{v_{3j}}} N(\delta) \tag{3.6}$$

where

- If $v_{3j} < \min(v_{1j}, v_{2j})$, then $D_{v_{3j}} = \{\delta = |v_{1j} - v_{3j}|\}$ and

$$N(\delta) = 2 \cdot \max\left(0, \min(|I| - v_{1j}, |I| - v_{2j} - \delta) + 1\right).$$

- If $v_{3j} = v_{2j} \leq v_{1j}$, then $D_{v_{3j}} = \{0, 1, \ldots, v_{1j} - v_{2j}\}$ and

$$N(\delta) = \max\left(0, \min(|I| - v_{1j}, |I| - v_{2j} - \delta) + 1\right).$$

Proof: See Appendix B.3. □

This can be generalized even further to capture the probability of an intersection between any pair of constraints. This generalization is not needed for this thesis but is laid out in Appendix B.5 along with a sketch of a proof.

Now that the probability of two constraints intersecting to create any specific one has been derived, as well as all possible configurations of intersection constraints, the complexity of a uniformly generated hypercubic function can be formally defined as:

Definition 5:

A uniformly generated hypercubic function with parameters (m, d, v, γ) has a complexity defined as

$$\mathcal{C} = -\log\left[\frac{1}{m}\left(\frac{v}{|I|}\right)^{\gamma} + \frac{1}{K}A\right]$$

where the term A is defined recursively as

$$A = m \cdot \frac{1}{2}\sum_{c_3 \in S}\mathbb{P}(b, c_3)\left[p_3 + \frac{3}{2}\sum_{c_4 \in S}\mathbb{P}(c_3, c_4)\left(p_4 + \left(\frac{3}{2}\right)^2\sum_{c_5 \in S}\mathbb{P}(c_4, c_5)\left(\cdots\right.\right.\right.$$
$$\left.\left.\left.\cdots + \left(\frac{3}{2}\right)^k\sum_{c_{k+3} \in S}\mathbb{P}(c_{k+2}, c_{k+3}) \cdot p_{k+3}\right)\right)\right].$$

Here:

- S is the set of all attainable intersection constraint configurations according to Lemma 1,
- $\mathbb{P}(c_j, c_k)$ is the probability that constraint c_j intersects with a base constraint to form constraint c_k, as measured by Theorem 4. Here, b represents a base constraint, hence the first $\mathbb{P}(b, c_3)$.
- Each p_j denotes the probability of satisfying constraint c_j with a single uniform random sample, measured by Corollary 1.
- And the recursion depth k is determined by the upper bound K as described by Theorem 3:

$$m\sum_{i=1}^{k}\left(\frac{3}{2}\right)^i = K \iff k = \frac{\ln\left(\frac{K/m+3}{2}\right)}{\ln\left(\frac{3}{2}\right)} - 1$$

where k is derived from closed form solution to geometric series.

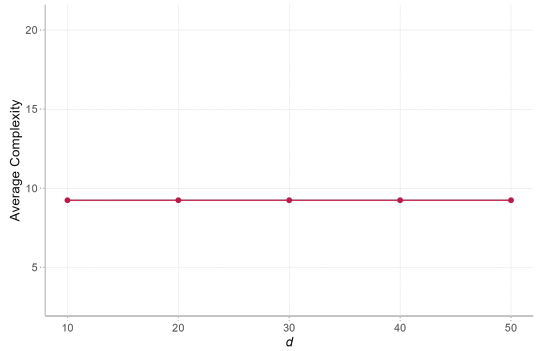
Intuition. This measure is designed to approximate the probability that a uniformly sampled contract $\mathbf{x}$ satisfies at least one of the m base constraints or any of the feasible intersection constraints, weighted by both their likelihood of occurrence and their contribution to complexity. Any constraint c_k's contribution to complexity is then measured by the corresponding p_k term. Since each base constraint is guaranteed to be part of the function, the first term is derived

directly from Theorem 2 and scaled by m, the total number of base constraints. The second term is weighted by the total number of potential intersection constraints, where A acts as a recursively defined weighted sum over them.

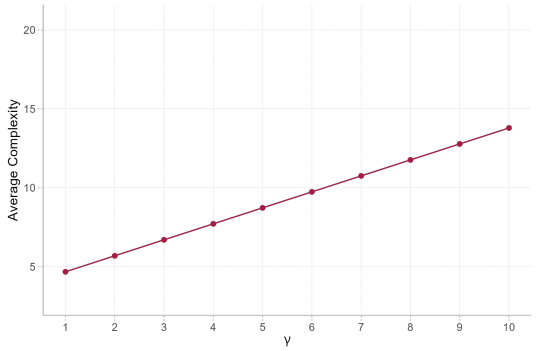
The intuition behind A is grounded in the side note following Definition 4 in Section 3.1, and is motivated by the following ideas:

1. There are m base constraints, and each pair of them is equally likely to intersect. The term $m \cdot \frac{1}{2} \sum_{c_3 \in S} \mathbb{P}(b, c_3)$ captures the probability, aggregated over all base constraints, that an intersection with another base constraint yields a valid constraint c_3.
2. This probability is used to weigh the p_3 term, which measures the relative constraint c_3's contribution to complexity.
3. Once a constraint like c_3 has been formed by intersecting two base constraints, the probability that it further intersects with another base constraint to form c_4 is weighted by a factor of $\frac{3}{2}$, reflecting the increasing number of constraints in the system and the combinatorial growth of interaction terms at each recursive depth.
4. This recursive process continues, with each new intersection constraint being weighted accordingly and multiplied by its corresponding p_j, until the total number of constraints reaches the predefined upper bound K.

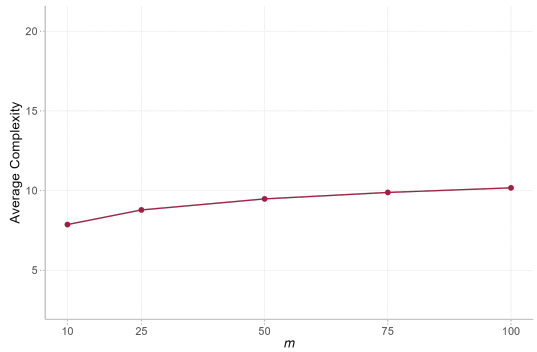
Figures 3.3 illustrates the individual effects each parameter has on this measure. As will be shown in Chapter 6, these figures are consistent with the agent's performance.



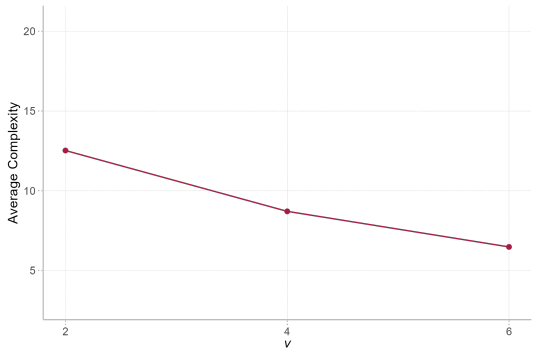
(a) Number of issues



(b) Dimensionality of constraints



(c) Number of constraints



(d) Widths of consraints

Figure 3.3: Each complexity parameter fixed and average over the others with configurations $d \times \gamma \times m \times v$ where $d = \{10, 20, 30, 40, 50\}, \gamma = [1, 10], m = \{10, 25, 50, 75, 100\}$ and $v = \{2, 4, 6\}$, corresponding to the configurations used in this thesis' main experiments.

3.2. Sampling

In one of his essays *Glow, Big Glowworm* [27], Gould recounts a visit to the Waitomo Glowworm Caves near Lake Rotorua in New Zealand. Thousands of glowworms lit up these dark caves, as if the cave ceiling had transformed into a star-strewn night sky. However, due to the glowworms' territorial behavior, they have evolved to leave an even spacing between each other, unlike the stars in the sky, which display gaps and clusters relative to the observer's position.

The stars can be thought of as points drawn from a uniform random distribution, whereas the glowworms are not themselves sampled uniformly; instead, their pairwise distances are approximately uniform. In many cases, a sampling method more resembling the glowworms of the Waitomo caves is preferred over the randomness of the stars. For instance, quasi-Monte Carlo integration relies on (t, m, s)-nets, which are point sets that are carefully constructed to minimize *discrepancy*, a measure of deviation from a perfectly even coverage of the space [60].

In this thesis, there are two main motivations for a structured sampling procedure. The first motivation is related to function approximation, specifically, how much of the domain it should cover. The sampling procedure must neither under-represent nor overextend over the domain. As the approximation will be unimodal, while the utility function may exhibit multiple local optima, overextending the sample may result in a great loss of information. This motivates partitioning the domain into subspaces to sample from within. The discussion of Section 3.2.1 concludes that these subspaces should be cube-shaped.

The second motivation is related to the curse of dimensionality. As the number of issues increases, a combinatorial explosion occurs in the possible values that the issues can jointly take. That is, any blind spot apparent in the data will exponentially increase in volume as the dimension increases, making it unrealistic to sample the entire space in high dimensions. This problem motivates the need for both sampling that *evenly* covers the space, resembling the Waitomo caves, and the need for the search to be biased towards regions of the space where the chances of finding a high-utility bid, or a socially optimal bid, are higher. Biasing the search is a well-established approach in the literature, with numerous heuristics and related techniques having been explored, as discussed in Chapter 2. Furthermore, an additional benefit of a sample that evenly covers the space is that it promotes consistency across approximations, preventing any single approximation from performing better or worse than the others by chance.

3.2.1. Geometrics of n-dimensional spaces

The importance of the geometry of sampling increases with the dimensionality of the domain [48]. As dimensionality increases, the behavior of most shapes changes in unintuitive but significant ways. As integer negotiation domains are the focus of this thesis, this section examines the geometry of three fundamental shapes in $\mathbb{R}^n$: the hypersimplex, the hypersphere, and the hypercube. Although many geometric shapes exist in $\mathbb{R}^n$, these three are the most commonly used in practical sampling applications, each exhibiting distinct scaling properties that, in turn, affect the sample sets. This will then be discussed in terms of an integer negotiation domain.

The hypersimplex

The n-dimensional *hypersimplex* consists of $n + 1$ points that are all equidistant. This is the maximum number of such points that can exist in $\mathbb{R}^n$ (and in $\mathbb{N}^n$) with this property. An edge connects every pair of vertices, every triple forms an equilateral triangle, every quadruple a regular tetrahedron, and so on. More generally, the n-dimensional hypersimplex is bounded by $\binom{n+1}{k+1}$ hypersimplices of dimension k. A hypersimplex with sidelengths $a = 2$ is referred to as a unit hypersimpex.

A shown in [48], the volume of an n-dimensional hypersimplex in $\mathbb{R}^n$ with edge length a is given by

$$V_n = \frac{1}{n!}\sqrt{\frac{n+1}{2^n}}\,a^n.$$

Clearly, the volume decreases rapidly toward zero as the dimension n increases due to the factorial in the denominator, regardless of the edge length a. In other words, the geometry of the hypersimplex effectively collapses inward.

The hypersphere

The n-dimensional *hypersphere* consists of all points that have a maximal distance of R from its center. The 2-dimensional hypersphere is the common sphere; the 3-dimensional hypersphere is the ball. A hypersphere with $R = 1$ is referred to as a unit hypersphere.

As shown in [48], for even-numbered dimensions $n = 2p$, the volume of the hypersphere in $\mathbb{R}^n$ is given by

$$V_{2p} = \frac{R^{2p}\gamma^p}{p!},$$

and for odd-numbered dimensions $n = 2p + 1$, the volume is given by

$$V_{2p+1} = \frac{R^{2p+1}2^{p+1}\gamma^p}{1 \cdot 3 \cdot \ldots \cdot (2p+1)}.$$

As n increases, the volume — as with the hypersimplex — decreases rapidly towards zero due to the factorial in the denominator, regardless of the radius R.

The hypercube

The n-dimensional *hypercube* is a geometric shape with edge length ℓ, where each vertex has coordinates composed of either a or $a + \ell$, where a is some number, in every dimension. Thus, the bordering faces of the hypercube are orthogonal to each other or parallel. When $\ell = 1$, the hypercube is referred to as a unit hypercube.

The volume of a n-dimensional hypercube in $\mathbb{R}^n$ with sidelength ℓ is simply given by $V_n = \ell^n$. As n increases, the volume decreases to zero when $0 \leq \ell < 1$, stays the same when $\ell = 1$, and blows up exponentially when $\ell > 1$.

In terms of an integer negotiation domain $[0, k]^d$, these shapes occupy a proportion $(\frac{V_n}{k})^d$ of the domain, where V_n is only growing with n in the hypercubic case. In that case, this becomes $(\frac{\ell}{k})^d$ when the cube has sidelengths ℓ. This proportion shrinks with d, but as the numerator grows at the same exponential rate as the denominator, this happens at a much slower pace than in the case of the hypersphere and the hypersimplex. That is, when sampling from within a geometric shape in increasing dimension, the proportion of the domain occupied by a hypersphere or a hypersimplex allows the sample only a glimpse of the domain. At the same time, the hypercube does a much better job at maintaining its proportional size.

To illustrate this phenomenon, consider a hypersphere of radius 1 and its enclosing hypercube with side length 2. Although the hypersphere touches all faces of the hypercube, whose volume rapidly grows with increasing dimension, the proportion of the hypercube's volume occupied by the hypersphere rapidly diminishes with increasing dimension. The same goes for the hypersimplex. The case of the hypersphere is depicted in Figure 3.4 where, for dimensions greater than 10, the volume of the hypersphere becomes negligible relative to that of the hypercube.

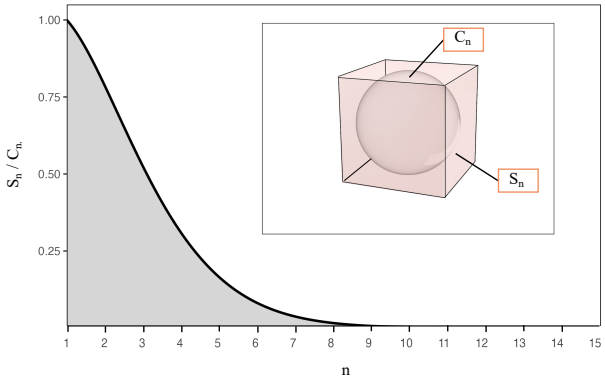


Figure 3.4: Ratio of the volumes of the unit hypersphere and the embedding hypercube of sidelength 2 up to 14 dimensions. Figure is adapted from [48]

Side note:

To avoid ambiguity, a cube will from here on out refer to a cube-shaped region of the domain, while a hypercube will refer to a hypercubic constraint.

3.2.2. Manhattan distance between constraints

Consider sampling from a cube with sidelengths of 2 in the $[1, 10]^3$ domain. In order to cover the entire domain, 125 mutually exclusive cubes will need to be placed in this domain to be sampled from. However, if the domain is now increased to be of 10 dimensions, or $[0, 10]^{10}$, this number becomes just under 10 million cubes. Thus, although the cube size grows at the same exponential rate as the domain, the sizes of the issue domains being larger prevent the cube size from keeping up in higher dimensions. Efficiently searching for bids, therefore, requires narrowing down the search space to promising regions. Alternatively, the cube size can be increased. This, however, is not a good strategy, as it will lead to greater information loss, as previously discussed. Therefore, strategically placing the cubes must be done.

Previous approaches to narrowing down the search space have taken various forms, as has been discussed in Section 1.6. As noted by Erez and Zuckerman [19], an intricate opponent learning is only suitable for scenarios where data from previous negotiations is available, whereas a heuristic is more suitable for standalone negotiations.

The heuristic that will be used in this thesis is based on the consideration that if a bid was made, the particular contract is "important" in the sense that either agent would accept it. It is fair to assume that a contract with close proximity to the proposed one might also be accepted. The more complex the utility function is, the closer this proximity would need to be. Thus, a distance-based heuristic will be implemented, where the distance from some previous bids will be exploited. This can be done to find bids that are good in terms of individual utility by considering the agent's own bids, as well as bids that are more socially optimal by also considering the opponent's bids. However, the key challenge lies in choosing an appropriate distance metric that best captures the similarity in utility.

De Jonge and Sierra [17] suggest using the Manhattan distance for hypercubic constraints, due to the nature of the two. Indeed, if a bid $\mathbf{x}$ satisfies a constraint c_k, then for the γ issues $x_i \in \mathbf{x}$

that the constraint is defined by, each one defines an interval $[x_i^{min}, x_i^{max}]$ of length v. Moving a sample cube a Manhattan distance of k thereby means choosing some $\ell \leq k$ issues and adding or subtracting to them a combined value of k. Furthermore:

Lemma 4:

Let $v_1, v_2, |I|, v_3 \in \mathbb{N}$ with $v_3 \leq \min(v_1, v_2)$ and $v_1, v_2 \leq |I|$. Furthermore, let $L_1 = [x_1^{\min}, x_1^{\min} + v_1)$ and $L_2 = [x_2^{\min}, x_2^{\min} + v_2)$ be two intervals such that $L_1 \cap L_2 \neq \emptyset$ and with all valid placements (x_1^{min}, x_2^{min}) being equally likely. Then

$$P((L_1 \pm 1) \cap L_2 = \emptyset | L_1 \cap L_2 \neq \emptyset) = \frac{N_{end}}{N_{int}}$$

where

$$N_{\text{int}} = \sum_{\delta=-v_1+1}^{v_2-1} \max\{0, \min(|I| - v_2, |I| - v_1 - \delta) - \max(0, -\delta) + 1\},$$

and

$$N_{\text{end}} = \sum_{\delta=-v_1+1}^{v_2-1} \mathbf{1}_{\text{escape}}(\delta) \max\{0, \min(|I| - v_2, |I| - v_1 - \delta) - \max(0, -\delta) + 1\},$$

with the indicator

$$\mathbf{1}_{\text{escape}}(\delta) = \begin{cases} 1, & \text{if } [x_1^{min}+1, x_1^{min}+v_1+1) \cap [x_1^{min}+\delta, x_1^{min}+\delta+v_2) = \emptyset \\ & \quad \text{or } [x_1^{min}-1, x_1^{min}+v_1-1) \cap [x_1^{min}+\delta, x_1^{min}+\delta+v_2) = \emptyset, \\ 0, & \text{otherwise.} \end{cases}$$

Proof: See Appendix B.4 □

Theorem 5:

Let C be a d dimensional cube in $\Omega = \prod_{i=1}^{d}$ and let $c_k(\gamma_k, v)$ be a base constraint in Ω. Given that $C \cap c_k \neq \emptyset$, the probability of $C \cap c_k = \emptyset$ after shifting k uniformly randomly picked sides of C by a factor of ± 1, the probability that $C \cap c_k = \emptyset$ is:

$$\left[\frac{\gamma_k N_{end}}{d N_{int}}\right]^k$$

where N_{end} and N_{int} are defined as in Lemma 4.

Proof: For each shift, the probability that the selected side is in Γ_k is $\frac{\gamma_k}{d}$. If the shifted side did intersect with c_k then the probability that the corresponding pair stops intersecting after the shift is $\frac{N_{end}}{N_{int}}$ by Lemma 4. As Γ_k is picked at uniform random, and so are the k sides that are shifted, these events are independent and the resulting probability becomes

$$\left[\frac{\gamma_k N_{end}}{d N_{int}}\right]^k$$

□

Side note:

In Theorem 5, k is equivalent to the Manhattan distance that the cube is moved.

3.2.3. Sampling evenly

Perhaps the most straightforward approach to achieving an even coverage in a domain is to construct a grid where some N equidistant points are placed on each dimension. While this method is adequate in low dimensional domains, it becomes impractical as dimensions increase due to the curse of dimensionality. Consider e.g., a 2-dimensional 3 by 3 grid consisting of 9 points. Extending this grid to 3 dimensions would require $3^3 = 27$ points, and continuing this way to, say, 25 dimensions, would require hundreds of billions of points. The uniform sample, as mentioned at the beginning of this section, displays clumpiness which can result in unnecessary computational efforts and large unsampled gaps in the domain. For these reason, as well as those mentioned at the beginning of this section, there is a need for more efficient and intelligent sampling strategies.

Discrepancy [60] is a measure that quantifies how evenly a sample covers a space, or the deviation of a sample from being evenly distributed. Formally, let $I_s := [0,1]^s$ be the closed s dimensional unit cube and $P = x_1, ..., x_N \in I_s$ be a sample of points from I_s. Then, let $B \subseteq I_s$ and $A(B,P)$ be a counting function that indicates the number n with $1 \leq n \leq N$ for which $x_n \in B$. If $\mathcal{B}$ is then a nonempty family of Lebesgue-measurable subsets of I_s, the general notion of discrepancy of the sample P is given by

$$D_N(B;P) = \sup_{B \in \mathcal{B}} \left| \frac{A(B;P)}{N} - V_s(B) \right|,$$

where $V_s(B)$ denotes the s-dimensional volume of B. Various variations of discrepancy exist, each specified by the specific family $\mathcal{B}$. Two of the most common ones are star discrepancy and extreme discrepancy.

Definition 6:

The *star discrepancy* $D_N(P)$ of the sample $P = x_1, ..., x_N \in I_s$ is defined by $D_N(P) = D_N(\mathcal{J}^*; P)$, where $\mathcal{J}^*$ is the family of all subintervals of I_s of the form $\gamma_{i=1}^{s}[0, u_i)$ where $0 < u_i \leq 1$ for all i.

Definition 7:

The *extreme discrepancy* $D_N(P)$ of the sample $P = x_1, ..., x_N \in I_s$ is defined by $D_N(P) = DN(\mathcal{J}; P)$, where $\mathcal{J}$ is the family of all subintervals of I_s of the form $\gamma_{i=1}^{s}[u_i, v_i)$ where $0 \leq u_i < v_i \leq 1$ for all i.

The desirable notion in this connection is that of a *low-discrepancy sequence*, which is informally defined as a sequence S of elements of I_s, for which $D_N^*(S)$ or $D_N(S)$ is of the order $N^{-1} \log^s N$ for all $N \geq 1$ [3]. Low-discrepancy sampling schemes have been shown to yield faster convergence than uniform random sampling in a variety of applications, including numerical integration [60], physics-informed neural networks [76], deep learning [56], optimization [9, 46], and option pricing [32]

Various schemes exist with the aim of achieving low discrepancy of the sample, e.g., Jitterred sampling and Latin hypercube sampling [47], where the domain is subdivided into equally sized subdomains which are then sampled from. A more general concept of stratification was

developed by Sobol in [69], that resulted in the so-called (t, m, s)-nets and (t, s)-sequences. To formally define (t, m, s)-nets and (t, s)-sequences, the notion of the elementary interval is required. A subinterval E of the form

$$E := \gamma_{j=1}^{s} \left[\frac{a_j}{b^{\ell_j}}, \frac{a_j + 1}{b^{\ell_j}} \right) \subseteq [0, 1)^s$$

where $0 \leq aj < b^{\ell_j}$ and $0 \leq \ell_j$ are integers, is called an *elementary interval in base b*. Consequently the volume of E is

$$V_s(E) = \gamma_{j=1}^{s} \frac{1}{b^{\ell_j}} = b^{-\sum_{j=1}^{s} \ell_j}.$$

Figure 3.5 shows the structure of all elementary intervals with volume $V_2(E) = \frac{1}{16}$ in base $b = 2$ for dimension $s = 2$.

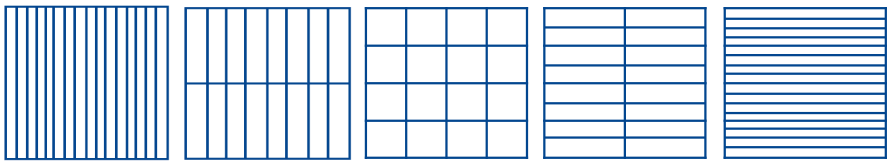


Figure 3.5: All elementary intervals in base $b = 2$ and dimension $s = 2$ with volume $V_2(E) = \frac{1}{16}$. Figure is adapted from [47].

Now, formally,

Definition 8:

Let $b \geq 2$ and $0 \leq t \leq m$ be integers. Then, a (t, m, s)-net P in base b is a point set of b^m points in I_s such that $A(E, P) = b^t$ for every elementary interval E in base b with $V_b(E) = b^{t-m}$.

As a consequence, the star discrepancy $D^*_{b^m}(E, P) = 0$ where the supremum is over every elementary interval E in base b with $V_s(E) = b^{t-m}$. Finally,

Definition 9:

Let $b \geq 2$ and $t \geq 0$ be integers. An infinite sequence $x_1, x_2, \ldots$ of points in I_s is called a (t, s)-sequence in base b if for all integers $k \geq 0$ and $m > t$ the point set consisting of the x_n with $kb^m < n \leq (k + 1)b^m$ is a (t, m, s)-net in base b.

Consequently, the first b^m points of a (t, s)-sequence form a (t, m, s)-net. In [60], it is shown that for the star discrepancy of the first b^m points in a (t, s)-sequence S in even base b with $s \geq 2$, is of the order $N^{-1} \log^s N$ making it a low-discrepancy sequence. A deterministic constructions of (t, m, s)-nets is can thus be done by selecting the first b^m points of (t, s)-sequences in even base. Furthermore, selecting the first k points, where $k \in (b^{m-1}, \ldots, b^m)$, yields a $(t, m-1, s)$-net, augmented by $k - b^{m-1}$ additional points. These extra points are distributed in such a way that the complete set of b^m points forms a (t, m, s)-net. In other words, the sample continues to evenly cover the space but is no longer guaranteed to be a low-discrepancy sample as the added points may appear somewhat irregular. Figure 3.6 shows 300 points drawn from the uniform random distribution and the first 300 points of a $(t, 2)$-sequence in base $b = 2$, or a Sobol sequence.

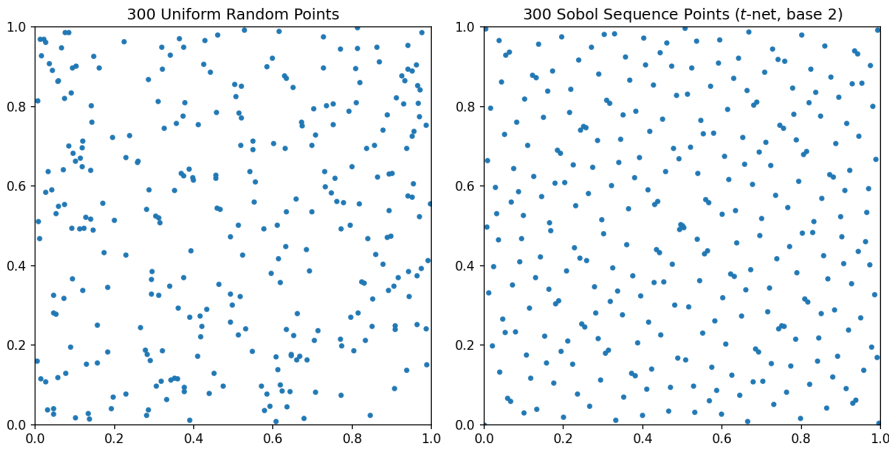


Figure 3.6: Example of 300 points generated using Sobol sequence(Right) and Uniform random distribution(Left). The first $2^8 = 256$ points of the Sobol sequence form a $(t, 8, 2)$-net which evenly covers the domain, while the remaining 44 points can be seen clustered at seemingly random (e.g. at (0.5,0.5)).

Constructing (t, s)-sequences is beyond the scope of this thesis but a detailed discussion is given in Sobol book [69] for the case of $b = 2$, which has lead to the case of $b = 2$ being called Sobol points, and a generalization for $b \geq 3$ in [59].

This thesis adopts a sampling scheme based on Sobol points. This is because Sobol sequence generation is computationally efficient, especially in high dimensions, as numbers are already represented in base 2 on digital computers [63]. Although sampling schemes based on low-discrepancy sequences have been shown to outperform those based on uniform random sampling, comparative studies among different low-discrepancy sequences have demonstrated marginal differences in their performance in low dimensions (e.g., [76, 9]), while Sobol sequences have been shown to outperform in higher dimensions (e.g., [9]).

3.3. Regression

In this thesis, the agent's utility function will be approximated using quadratic regression, trained on samples from the original utility function. Regression models are a natural choice as the sample data includes both features and labels; that is, the data is of the form

$$\{\mathbf{x}_i, y_i\}_{i=1}^{n},$$

where $\mathbf{x}_i$ is the d-dimensional vector representing the bid corresponding to the ith sample, while $y_i = u(\mathbf{x}_i)$ is the utility value of that bid.

Regression models that are linear in the parameters can be expressed in the form

$$\hat{u}(\mathbf{x}_i) = w_0 + \sum_{j=1}^{m-1} w_j \phi_j(\mathbf{x}_i), \tag{3.7}$$

where $\phi_j : \mathbb{R}^d \to \mathbb{R}$ are fixed or parameterized basis functions, and $w_j \in \mathbb{R}$ are coefficients, or *weights*, and w_0 is the so called *bias* (not to be confused with the bias in a statistical sense) [12]. These parameters can be estimated from the data, which is usually done through maximum likelihood and least squares [1]. There are alternative methods for obtaining a point estimate of the parameters. The main appeal of this approach is that it can be shown to be the best estimator asymptotically in terms of its rate of convergence as the sample size increases [26].

These models can be either linear or nonlinear in $\mathbf{x}$ through the basis functions, but are linear in the parameters w_j. The number $m-1$ of basis functions depends on the type of functions used, the dimensionality of the input, and the complexity of the functions. For example, if ϕ_j are polynomials up to degree k, the number $m-1$ will represent all combinations of d variables raised to powers that sum to at most k.

The key distinction between these models lies in the choice of basis functions $\{\phi_j\}$, which ultimately determine the approximation accuracy. These basis functions must be carefully selected for the approximation to align with the task at hand: to be a surrogate for the original utility function.

This section begins with a derivation of the least squares solution for estimating the model parameters. After that, a discussion on linear and quadratic basis functions is given, followed by a discussion on the bias-variance trade-off: the trade-off between selecting models of varying complexity.

3.3.1. Maximum likelihood and least squares

Deriving the parameters of model (3.7) will be done in this thesis via maximum likelihood and least squares. To do that, the following assumption needs to be made:

Assumption 8:

The target variable y_i is assumed to be given by the deterministic function $\hat{u}$ in (3.7), with additive Gaussian noise. That is,

$$y_i = \hat{u}(\mathbf{x}_i) + \epsilon_i = w_0 + \sum_{j=1}^{m-1} w_j \phi_j(\mathbf{x}_i) + \epsilon_i$$

where $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ are independent and identically distributed.

There are $m+1$ parameters that can be derived from model (3.7): the bias w_0, the $m-1$ weights w_j, and the variance, σ^2. However, when the goal is to make predictions — as is the case in this thesis — only the bias and the weights need to be learned. By letting $\mathbf{w} = [w_0, w_1, ..., w_{m-1}]^T$ include the bias and then defining the *design matrix* as

$$\mathbf{\Phi} = \begin{bmatrix} 1 & \phi_1(\mathbf{x}_1) & \cdots & \phi_{m-1}(\mathbf{x}_1) \\ 1 & \phi_1(\mathbf{x}_2) & \cdots & \phi_{m-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(\mathbf{x}_n) & \cdots & \phi_{m-1}(\mathbf{x}_n) \end{bmatrix}, \tag{3.8}$$

$\hat{u}$ can be expressed in matrix form as

$$\hat{u}(\mathbf{x}_i) = \mathbf{\Phi}_i \mathbf{w}$$

where $\mathbf{\Phi}_i$ is the ith row of $\mathbf{\Phi}$.

Now, due to the added Gaussian noise, the target vector $\mathbf{y} = \mathbf{\Phi w} + \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is the additive noise vector, is assumed to be distributed as

$$\mathbf{y}|\mathbf{x}, \boldsymbol{\theta} \sim \mathcal{N}(\mathbf{y}|\mathbf{\Phi w}, \sigma^2)$$

where $\boldsymbol{\theta} = (w_0, \mathbf{w}, \sigma^2)$ are all model parameters. This results in the log-likelihood function

$$\begin{aligned} \ln p(\mathbf{y}|\mathbf{x}, \mathbf{w}, \boldsymbol{\sigma}^2) &= \sum_{i=1}^{n} \ln \mathcal{N}\left(y_i|\boldsymbol{\Phi}_i\mathbf{w}, \sigma^2\right) \\ &= \frac{N}{2}\ln\sigma^2 - \frac{N}{2}\ln 2\pi - \sigma^2 RSS(\mathbf{w}) \end{aligned} \tag{3.9}$$

where

$$RSS(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \boldsymbol{\Phi}_i\mathbf{w})^2 \tag{3.10}$$

is the least squares error function [12]. If $\boldsymbol{\Phi}$ is of full rank, the Hessian

$$\frac{\partial^2}{\partial \mathbf{w}^2} RSS(\mathbf{w}) = \boldsymbol{\Phi}^T\boldsymbol{\Phi}$$

is positive definite, since for any $\mathbf{v} > 0$,

$$\mathbf{v}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi})\mathbf{v} = (\boldsymbol{\Phi}\mathbf{v})^T(\boldsymbol{\Phi}\mathbf{v}) = \|\boldsymbol{\Phi}\mathbf{v}\|^2 > 0. \tag{3.11}$$

Hence, when $\boldsymbol{\Phi}$ is full rank, the least squares error function has a unique global minimum [58]. It is noteworthy that the matrix can only be of full rank if it is overdetermined, i.e., when $n \geq m$. A discussion on the matrix $\boldsymbol{\Phi}$ being singular, ill-conditioned, or not full rank is given in Section 4.2.

Taking the partial derivative of Expression (3.9) with respect to $\mathbf{w}$ and equating to zero yields the global minimum of $RSS(\mathbf{w})$, or the so called *normal equations* for the least squares problem:

$$\hat{\mathbf{w}}_{ML} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\mathbf{y} = \boldsymbol{\Phi}^\dagger\mathbf{y}, \tag{3.12}$$

where $\boldsymbol{\Phi}^\dagger$ is the (left) pseudo inverse of the matrix $\boldsymbol{\Phi}$ [1].

3.3.2. Linear basis functions

The simplest form the Equation (3.7) can take is when the basis functions ϕ_j are the identity functions for their corresponding feature variable, i.e.

$$\phi_j(\mathbf{x}_i) = x_{ij}$$

where x_{ij} is the value of the jth issue in the ith sample. In this case, the design matrix is simply the data matrix, resulting in a total number of parameters equal to $d + 1$, which is the number of issues plus the bias. This corresponds to *linear regression*, which is widely used due to its simplicity and interpretability.

The key property of the model is that $\hat{u}$ is assumed to be an affine linear function of the input, taking the form

$$\hat{u}(\mathbf{x}_i) = w_0 + \sum_{j=1}^{d} w_j x_{ij} \tag{3.13}$$

which can be written in matrix form simply as

$$\hat{u}(\mathbf{x}_i) = w_0 + \mathbf{x}_i\mathbf{w}.$$

This means that the model forms a hyperplane, giving $\hat{u}$ a constant slope along each dimension. Therefore, this model cannot represent curvature or interaction effects between dimensions. Furthermore, since the hyperplane is constructed within a bounded region of the domain, the maximum is located at a vertex, meaning the model will never capture internal optima.

3.3.3. Quadratic basis functions

To address the limited expressiveness of the linear basis functions, polynomials introduce higher powers of the variables as well as interaction terms. In the quadratic case, the basis functions can be any combination of one or two issues, or

$$\phi_j(\mathbf{x}_i) \in \{x_{i1}, .., x_{id}, x_{i1}^2, ..., x_{id}^2, x_{i1}x_{i2}, ..., x_{id-1}x_{id}\},$$

making the total number of parameters equal to $2d + \frac{d(d-1)}{2} + 2$. This results in $\hat{u}$ taking the form

$$\hat{u}(\mathbf{x}_i) = \left(\sum_{j=1}^{d}\sum_{k=1}^{d} a_{jk}x_{ij}x_{ik}\right) + \left(\sum_{j=1}^{d} b_j x_{ij}\right) + w_0 \tag{3.14}$$

which can be written in matrix form as

$$\hat{u}(\mathbf{x}_i) = \mathbf{\Phi}_i\mathbf{w} = \mathbf{x}_i A\mathbf{x}_i^T + \mathbf{b}\mathbf{x}_i + w_o$$

where A is a $d \times d$ matrix with entries a_{jk} and $\mathbf{b}$ is a column vector with entries b_j. This corresponds to *quadratic regression*.

This means that the model forms a flexible hypersurface, capable of representing curvature and interaction effects between any two dimensions. Furthermore, the model is not restricted to exhibiting only optima on vertices as

$$\frac{\partial\hat{u}}{\partial\mathbf{x}_i} = 2A\mathbf{x}_i + \mathbf{b} = 0 \Rightarrow \mathbf{x}_i^* = \frac{1}{2}A^{-1}\mathbf{b}$$

can take values anywhere in the region. Whether $\mathbf{x}_i^*$ is a maximum, minimum, or a saddle point depends on the definiteness of A. This is discussed in Section 4.2.

3.3.4. Underfitting, Overfitting, and Regularization

When a high-utility bid is found in the quadratic surrogate, it is important that it is also a high-utility bid in the actual utility function. That is, it is important that the regression model *generalizes* well over the unsampled bids in the corresponding cube. In their book *Deep Learning* [26], Goodfellow et al. write that the factors determining how well a learning algorithm performs are its ability to

1. make the training error small, and to
2. make the difference between the training and generalization error small.

Training error refers to the average error obtained over the sampled bids, while generalization error refers to the error over the unsampled bids. They identify these two aspects as reflecting *the* fundamental trade-off in machine learning. Regarding the former factor, if the model is too simple, it cannot capture the structure of the training data, resulting in a high training error. This problem is referred to as *underfitting*. Regarding the latter factor, if the model is overly flexible, it may adapt too closely to the training examples, failing to generalize and resulting in a large difference between training and generalization performance. This problem is referred to as *overfitting*. The likelihood of either problem can be reduced primarily in two ways: by increasing the amount of training data available and by adjusting the model's *capacity*, i.e., the model's complexity.

The relationship between overfitting, underfitting, and capacity is closely tied to the relationship between the bias and variance of the estimated model parameters. Indeed, it can be shown that

for the estimated parameters $\hat{\theta}$ and the true parameters[1] θ, it holds that

$$\begin{aligned} MSE(\hat{\theta}) = \mathbb{E}_D[(\theta - \hat{\theta})^2] &= \left(\mathbb{E}_D[\theta] - \hat{\theta}\right)^2 + \mathbb{E}_D\left[(\theta - \mathbb{E}_D(\theta))^2\right] \\ &= Bias(\hat{\theta})^2 + Var(\hat{\theta}) \end{aligned} \tag{3.15}$$

where MSE stands for mean squared error. Bias and variance measure two different sources of error in an estimator [58]. Bias measures the expected deviation from the true value of the parameter. Models with a low capacity relative to the sample size and complexity of the underlying function will exhibit higher bias and are likely to struggle in fitting the training set, resulting in an underfit. Variance, on the other hand, provides a measure of the deviation from the expected estimator value that any particular sampling of the data is likely to cause. Models with a high capacity relative to the sample size and complexity of the underlying function will have higher variance and are likely to overfit the training set. Desirable estimators are those that manage to keep both their bias and variance somewhat low. In general, models perform best when their capacity and sample size are appropriate for the complexity of the underlying function. This is depicted in Figure 3.7.

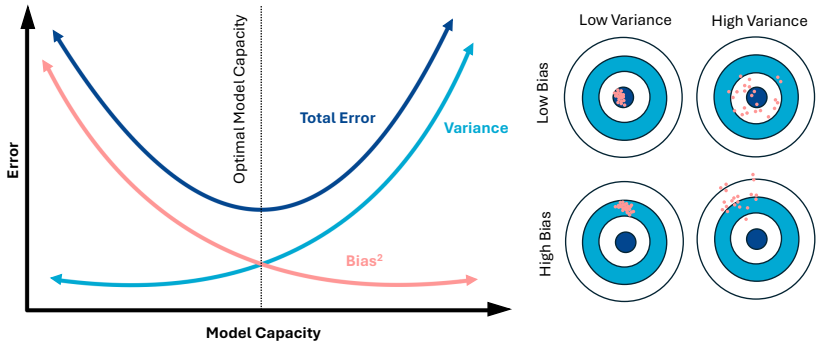


Figure 3.7: The best performing models are those that have appropriate capacity with respect to the sample size and the complexity of the underlying function. When this is the case, bias and variance are both low. Figure is adapted from [58].

In the context of this thesis, the MSE is in terms of $\hat{\mathbf{w}}_{ML}$, which is determined by the normal equations. Affecting those can be done by increasing the sample size and by adding a regularization term.

When the sample size is increased, the model has access to a broader and more representative view of the underlying data-generating distribution. This generally improves its ability to approximate the true structure and reduces the variance of its predictions. In cases where the entire domain and its labels are available for training, the model will achieve the best performance permitted by its capacity, thereby reducing the variance to zero. In this case, the MSE is solely the bias squared term of Equation (3.15).

Regularization is a technique often used to control for overfitting. This can be achieved by adding a penalty to the error function, which discourages weights from becoming too large. Adding a regularization term will essentially dampen the capacity of the model, thereby adding to the bias, but with the goal of subtracting even more from the variance. In terms of the least

[1] Assuming an underlying data generating distribution $\mathcal{D}$, θ is the *true* parameter and the expectations in Equation (3.15) are w.r.t. $p(\mathcal{D}|\theta)$ [58].

squares error function (3.10), this takes the form

$$\frac{1}{2}\sum_{i=1}^{n}(y_i - \mathbf{\Phi}_i\mathbf{w})^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{3.16}$$

where λ is the regularization parameter, which indicates the relative importance of the regularization term when compared with the least squares error function. Regularizing with $\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$ is referred to as ℓ_2 regularization, as $\mathbf{w}^T\mathbf{w} = \|\mathbf{w}\|_2^2$, or otherwise *weight decay*, as it encourages weight values to decay towards zero unless supported by the data. Other regularization methods include ℓ_1 and elastic net, which balances the effects of ℓ_1 and ℓ_2. However, adding a quadratic regularizer, such as in (3.16), to the least squares error function has the advantage that the resulting regularized least squares error function remains quadratic as a function of $\mathbf{w}$. Additionally, $\hat{\mathbf{w}}_{ML}$ is guaranteed to have a unique closed-form solution, even if $\mathbf{\Phi}$ is not full rank. To see this, recall from derivation (3.11) that if $\mathbf{\Phi}$ is full rank, then the Hessian $\mathbf{\Phi}^T\mathbf{\Phi}$ is positive definite and $\hat{\mathbf{w}}_{ML}$ is unique. However, taking the partial derivative of the regularized least squares error function (3.16) with respect to $\mathbf{w}$ and equating to zero yields

$$\hat{\mathbf{w}}_{ML} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y} \tag{3.17}$$

which is unique. Indeed, $\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I$ is positive definite as for any matrix B, it holds that

$$\mathbf{v}^T(B^TB)\mathbf{v} = \|B\mathbf{v}\|_2^2 \geq 0 \Rightarrow \mathbf{v}^T(B^TB + \lambda I)\mathbf{v} = \|B\mathbf{v}\|_2^2 + \lambda\|\mathbf{v}\|_2^2 > 0$$

when $\lambda > 0$. As $\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I$ is positive definite, it follows from derivation (3.11) that $\hat{\mathbf{w}}_{ML}$ is unique.

As the hypercubic functions can range from very simple to extremely complex, while a quadratic polynomial is always relatively simple, the model will inevitably underfit as the complexity increases. As a regression model with linear basis functions has a lower capacity than a model with quadratic basis functions, the bias of this model will be higher when trained on the hypercubic functions. Furthermore, as the quadratic regression model is essentially a linear regression model with added capacity, the quadratic model should perform better than the linear model. If this does not happen, the quadratic model has very likely been overfit. As discussed in this section, this issue can be prevented by increasing the sample size and incorporating a regularization term. This approach will be adopted in this thesis, with the specific values of the sample size and regularization term being derived in Chapter 6.

3.4. Optimization

Once a surrogate has been constructed within a cube, the next step is to search for its global maximum. These problems are often referred to as *box constrained global optimization (BCGO) problems* [18].

When the function being optimized is linear, BCGO problems become trivial as the function can be optimized over each axis, one at a time. Additionally, when the function is quadratic and concave, the global maximum has the analytical formula

$$x^* = -\frac{1}{2}A^{-1}b. \tag{3.18}$$

However, when the function is multivariate and quadratic, and is not concave, the problem does not have a polynomial time $(1-\epsilon)$ approximation algorithm [18].

This section explores ways to tackle this problem, beginning with a discussion on solving linear BCGO problems, and then quadratic BCGO problems.

Side note:

Optimization algorithms are typically discussed in terms of minimizing an objective function. This tradition will be maintained in this chapter. However, a minimization problem can be easily transformed into a maximization problem by simply negating the objective function.

3.4.1. First-order methods

First-order optimization methods solely rely on the gradient of the objective function [58]. They have the advantage that the gradient is cheap to compute and to store, but they do not model the curvature of the space, and hence they can be slow to converge. However, when optimizing a linear additive function of the form (1.2), they are ideal. Indeed, in the case of linear regression, the basis functions are simply the identity function and the function takes the form of (3.13). In this case, the entire function can be optimized by optimizing each element of the sum individually.

Now, assuming that all $0 \geq x_{ij}$, as is the case for this thesis, this optimization problem simply boils down to checking the sign of the weights w_j. If w_j is positive, $w_j x_{ij}$ takes its maximum value when x_{ij} is as large as can be, and vice versa when w_j is negative. When w_j is zero, any value can be chosen. Thus, in a cube-shaped domain, the global maximum is obtained at one of the domain's corners or along one of its edges.

Geometrically, the surrogate utility function is a hyperplane with a constant gradient equal to the weight vector. This means the direction of steepest ascent is fixed across the domain, and the global optimum is found by moving along this direction until a boundary is reached.

3.4.2. Second-order methods

Second-order optimization methods incorporate the second-order derivatives, which will often yield faster convergence [58]. This is because second-order derivatives encode the rate of change of the gradient in each direction. Thus, when the surrogate is a quadratic function — a smooth function involving interaction terms between any two variables — using second-order optimization methods is ideal.

An alternative approach, applicable when the stationary point $\frac{1}{2}A^{-1}b$ is not a global maximum, is to linearize the surrogate by eliminating the interaction terms. This reduces the problem to the trivial variable-wise optimization previously discussed. Although this strategy does not guarantee convergence to the global maximum, it will converge to a point located at a corner or vertex of the domain. Since the maximum of the original surrogate is also attained at a corner or vertex, this approach is worth exploring. In Chapter 4, this idea is investigated with linearization similar to that of the WAID method.

This section concludes on the L-BFGS-B method, which is a limited-memory version of the most popular *quasi-Newton method*, which has been adapted for BCGO problems. Quasi-Newton methods are much more computationally efficient than global optimization methods and metaheuristics, such as simulated annealing, but at the cost of being local in the sense that they do not guarantee a global optimum with a single initialization. Increasing the chances of finding the global optimum can, however, be drastically improved by choosing various initialization points for the method as opposed to a single shot. However, once the surrogate has been constrained to its domain, i.e., a cube, some methods fall short as they are meant for unconstrained convex minimization problems.

Newton's method

The classic second-order method is Newton's method [58]. Newton's method is derived through a second-order Taylor series approximation of the objective function (in this case, the quadratic regression model). It uses a trade-off between the first- and second-order derivatives to *descend* towards the minimum, with a balance between directions that are sufficiently steep but do not have drastically changing gradients. A Newton update is of the form

$$\mathbf{x}^*_{t+1} = \mathbf{x}^*_t - \epsilon_t H^{-1}(\mathbf{x}^*_t)\nabla\hat{u}(\mathbf{x}^*_t)$$

where $H(\mathbf{x}^*_t)$ is the Hessian of the objective function evaluated at $\mathbf{x}^*_t$, ϵ_t is the learning rate at iteration t, and $\nabla\hat{u}(\mathbf{x}^*_t)$ is the gradient of the objective function $\hat{u}$ evaluated at $\mathbf{x}^*_t$.

Side note:

If H were the identity function, Newton's method would be the steepest descent method.

In terms of the quadratic model, the Hessian is simply the weight matrix A and the gradient is $2A\mathbf{x} + b$. This results in the Newton update

$$\begin{aligned}\mathbf{x}^*_{t+1} &= \mathbf{x}^*_t - \epsilon_t \left(A^{-1}(2A\mathbf{x}^*_t + b\mathbf{x}^*_t)\right) \\ &= -\frac{1}{2}A^{-1}b\end{aligned}$$

when $\epsilon_t = \frac{1}{4}$. Notice that this is Expression (3.18). That is, this update approximates the objective function with a convex quadratic function (a bowl) and moves towards the bottom of it at a rate of ϵ_t [1]. When the objective function is actually a bowl, the bottom can be found in a single step.

In the special cases of a convex quadratic objective function, Newton's method can find the unique global minimum in a single step. The drawbacks of Newton's method, however, are the requirement that the Hessian be positive definite and the need to evaluate the inverse Hessian at every iteration, which is computationally costly. Newton's method is designed explicitly for convex quadratic functions with positive-definite Hessian matrices. In the context of this thesis, Newton's method can be applied when the weight matrix A is negative (semi)definite, in which case the regression model is concave. In this case, by setting the learning rate to $\frac{1}{2}$ and negating the objective function, Newton's method gives the global optimum $\frac{1}{2}A^{-1}b$ in a single iteration. However, when A is not negative (semi)definite, the classical Newton's method cannot be used. This is where quasi-Newton methods come in.

Quasi-Newton methods

Quasi-Newton methods are among the most sophisticated analytically for solving unconstrained optimization problems [53], with some of them also being applied to constrained optimization problems, including BCGO problems. For general nonlinear, not just convex, objective functions, quasi-Newton methods have been shown to locally converge superlinearly [53, 62].

Quasi-Newton methods are similar at their core to Newton's method, with the exception that they iteratively build up an approximation of the Hessian as opposed to using the true one. At each step, they approximate the Hessian using information gleaned from the gradient vector, which circumvents the cost associated with evaluating the inverse Hessian matrix at every iteration. The difference between the methods in the quasi-Newton family is then in how this approximation is performed.

Denote by G_t the approximation of A^{-1} in the tth step. This is typically initialized as the identity matrix, which amounts to the steepest descent method (see side note on previous page). Then, in each iteration, G_t is updated from G_{t-1} with low-rank updates derived from the Matrix Inversion Lemma [1]. Thus, a quasi-Newton update is as follows:

$$\mathbf{x}_{t+1}^* = \mathbf{x}_t^* - G_t[\nabla \hat{u}(\mathbf{x}_t^*)].$$

Discussing how G_t is constructed is outside the scope of this thesis, but a detailed discussion can be found in [53]. Intuitively, multiplying the Hessian by the parameter change is an approximation to the gradient change. Therefore, multiplication of the inverse Hessian approximation G_{t+1} with the gradient change is an approximation to the parameter change. The goal is thus to find a symmetric matrix G_{t+1} satisfying the *quasi-Newton condition*, or

$$\mathbf{x}_{t+1} - \mathbf{x}_t = G_{t+1}[\nabla \hat{u}(\mathbf{x}_{t+1}) - \nabla \hat{u}(\mathbf{x}_t)],$$

which represents an underdetermined system of equations with an infinite number of solutions. Among these, the Broyden-Fletcher-Goldfard-Shanno method (BFGS) chooses the closest symmetric G_{t+1} to the current G_t [62]. The BFGS method is the most popular quasi-Newton method [58], and a limited-memory variation of it will be used in this thesis.

The L-BFGS method

The BFGS method chooses the closest symmetric G_{t+1} to the current G_t by posing a minimization objective function $\|G_{t+1} - G_t\|_{F_w}$ in the form of a weighted Frobenius norm — hence the F_w subscript — subject to the quasi-Newton condition. In other words, the algorithm finds a G_{t+1} satisfying

$$\begin{aligned} &\text{Minimize}_{G_{t+1}} && \|G_{t+1} - G_t\|_{F_w} \\ &\text{subject to:} && \mathbf{x}_{t+1} - \mathbf{x}_t = G_{t+1}\big[\nabla \hat{u}(\mathbf{x}_{t+1}) - \nabla \hat{u}(\mathbf{x}_t)\big], \\ & && G_{t+1}^\top = G_{t+1}. \end{aligned} \tag{3.19}$$

Side note:

Using different norms in Problem (3.19) leads to a different variation of the quasi-Newton method [1].

Problem (3.19) is a quadratic optimization problem with linear constraints. Optimization problems where there are linear equality constraints and a quadratic objective function sometimes have a closed-form solution. This is because it is not uncommon that the equality constraints can be discarded along with their corresponding variables, leading to an unconstrained, quadratic optimization problem that can be defined in terms of the remaining variables [1]. One notable closed-form solution to such a problem is the normal equations (3.12) as a closed-form solution to the least squares error function.

The closed-form solution to Problem (3.19), or the BFGS update to G_t, exists, and is as follows: (see [62] for a detailed derivation)

$$G_{t+1} = (I - \Delta_t q_t v_t^T) G_t (I - \Delta_t v_t q_t^T) + \Delta_t q_t q_t^T. \tag{3.20}$$

where $\Delta_t = 1/(q_t^T v_t)$ and

$$q_t = \mathbf{x}_{t+1} - \mathbf{x}_t; \quad v_t = \nabla \hat{u}(\mathbf{x}_{t+1}) - \nabla \hat{u}(\mathbf{x}_t).$$

Intuitively, the update in (3.20) can be seen as removing and then replacing curvature information along q_t and v_t. The projection term

$$(I - \Delta_t q_t v_t^\top) G_t (I - \Delta_t v_t q_t^\top)$$

annihilates the component of the old approximation G_t in the direction of v_t, ensuring that outdated curvature information in that direction is discarded. The corrective term

$$\Delta_t q_t q_t^\top$$

then enforces the secant condition $G_{t+1} v_t = q_t$, thereby injecting the new curvature information consistent with the observed change in gradient. In this way, (3.20) preserves symmetry and positive definiteness when $q_t^\top v_t > 0$.

The drawbacks of the BFGS method are its need to carry over G_t, a matrix of size $\mathcal{O}(d^2)$, from each iteration to the next. However, the limited-memory BFGS (L-BFGS) reduces this memory requirement from $\mathcal{O}(d^2)$ to $\mathcal{O}(d)$ by never explicitly building G_t. L-BFGS only stores the m most recent (q_t, v_t)-pairs and reconstructs the effect of G_t on a vector by recursively applying the BFGS update (3.20) to those stored pairs [1].

The L-BFGS-B method

The L-BFGS-B algorithm, originally proposed by Byrd et al. [14] as a limited-memory quasi-Newton algorithm for solving large nonlinear optimization problems with simple bounds on the variables, extends L-BFGS to handle box constraints. That is, for a function $\hat{u}$ of d variables, the algorithm seeks to solve

$$\begin{aligned} &\text{Minimize}_{\mathbf{x}} && \hat{u}(\mathbf{x}) \\ &\text{subject to:} && l \leq \mathbf{x} \leq u \end{aligned} \tag{3.21}$$

where l and u are vectors of length d, representing lower and upper bounds on the variables, respectively. The original paper is a 22-page detailed description of the algorithm. The following summarizes the algorithm at a level sufficient for understanding its main ideas, while avoiding unnecessary technical details. The interested reader is referred to the original work (see [14]).

At each iteration t, an L-BFGS approximation G_t to the *Hessian* is updated. Indeed, a novel feature of the algorithm is that the update Equation (3.20) is represented in a compact form that is efficient for BCGO problems [14], and is modified to approximate the Hessian and not the inverse Hessian. G_t is then used to define a quadratic model $m_t(\mathbf{x})$ of the objective function $\hat{u}$. This is done using a second-degree Taylor series approximation at the current iterate $\mathbf{x}_t$:

$$m_t(\mathbf{x}) = \hat{u}(\mathbf{x}_t) + \nabla \hat{u}(\mathbf{x}_t)^T (\mathbf{x} - \mathbf{x}_t) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_t)^T G_t (\mathbf{x} - \mathbf{x}_t),$$

where the gradient $\nabla \hat{u}$ is provided by the user. Since the objective function used in this thesis is itself quadratic, the quadratic model m_t will coincide with the true function, especially as t increases. In this sense, the algorithm leverages the problem's structure to its fullest extent.

A gradient projection method is then performed to distinguish between *active* variables, i.e., variables that will be held at their bounds, and *free* variables. To do this, the generalized Cauchy point $\mathbf{x}^c$ is computed, which is defined as the first local minimizer of the univariate, piecewise quadratic

$$q_t(t) = m_t(x(t))$$

where $x(t)$ is the piecewise linear path obtained by projecting the steepest descent direction onto the $[l, u]$–bounds of (3.21). Then for each $i = 1, ..., d$, the variables that satisfy either

$x_i = l_i$ or $x_i = u_i$ at $\mathbf{x}^c$ constitute the *active set* $\mathcal{A}(\mathbf{x}^c)$ of variables to be held at their bounds. From there, the minimization problem

$$\text{Minimize}_{\mathbf{x}} \ \{m_t(\mathbf{x}) : x_i = x_i^c, \ \forall i \in \mathcal{A}(\mathbf{x}^c)\} \tag{3.22}$$

$$\text{subject to: } l_i \leq x_i \leq u_i, \ \forall i \notin \mathcal{A}(\mathbf{x}^c) \tag{3.23}$$

is approximately solved using a direct dual method using Lagrange multipliers, treading the active bounds (3.22) as equality constraints and ignoring the free bounds (3.23). The resulting approximate minimizer is then truncated to satisfy the (3.23) bounds. The search direction is then defined to be the vector connecting the current iterate to this approximate minimizer. Finally, a line search is performed along the search direction, which leads to $\mathbf{x}_{t+1}$.

The storage requirements of L-BFGS-B can be adjusted by the user through the recursion parameter m introduced earlier. In practice, the algorithm uses roughly $(12 + 2m)d$ storage locations. Since relatively small values of m are recommended ($3 \leq m \leq 20$) [79], the method remains applicable for very large-scale problems. The computational cost per iteration is also modest, ranging from approximately $4md + d$ multiplications when no bounds are active, up to $m^2 d$ multiplications in the case where all variables lie at their bounds.

3.5. Conclusion

This chapter answers Subquestion 1 by constructing a measure of complexity based on the theoretical notion of entropy. Furthermore, this chapter lays the mathematical foundation for answering Subquestion 2 by addressing the considerations needed to effectively use quadratic polynomials as surrogates for uncertain hypercubic utility functions. To implement them in an autonomous negotiation agent is the topic of the upcoming chapter. Once this agent has been constructed, the Main Research Question can finally be answered.

To ensure that the answers to all research questions are presented in one place rather than scattered throughout the thesis, the answer to Subquestion 1 is included together with the answers to Subquestion 2 and the Main Research Question in Section 7.3.

This chapter looked at the difficulties of sampling in high-dimensional spaces. It started by examining d-dimensional geometry to explain why hypercubic subspaces are useful for dividing up the negotiation domain. As dimensionality grows, many geometric shapes lose their effectiveness for sampling because their volume shrinks too quickly. Hypercubes, on the other hand, keep a stable structure that scales more reliably, making them a better choice for defining local sampling regions. The chapter then highlighted the role of strategic sampling, introducing a bid-based heuristic designed to steer samples toward areas where high-utility offers have been seen in the past. Finally, it discussed the need for even, consistent coverage within each sampling region and presented Sobol sequences as a way to generate low-discrepancy samples. Compared to uniform random sampling, Sobol sequences help maintain consistency and reduce variance.

The discussion then turned to regression models that are linear in the parameters as a way to construct a quadratic surrogate model of the utility function. By sampling from the utility function, these models can be constructed with quadratic or linear basis functions through the normal equations. The normal equations are a closed form solution to the minimization of the least squares objective function, resulting in the parameters needed to construct the model.

To search the surrogate, there can be a closed form solution to the stationary point. However, sometimes this point is not a global maximum in which case an optimization is needed. As there is no polynomial time $(1 - \epsilon)$ approximation algorithm for quadratic polynomials with

box constraints, this thesis adopts the L-BFGS-B algorithm. The L-BFGS-B algorithm is a second-order algorithm and is specifically designed for box-constraints.

Together, the components discussed in this chapter form the mathematical backbone of the agent that will be constructed in Chapter 4. The next chapter presents the method, where the considerations addressed in this chapter are implemented step-by-step and integrated into a negotiation strategy.

4

Method

This chapter introduces the proposed implementation of the discussion in Chapter 3 as a way to enable negotiation agents to identify high-utility bids in negotiation scenarios with hypercubic utility functions. A high-level version of this implementation is as follows:

1. Sample evenly from within some M cube-shaped regions of the domain, which are located based on the Manhattan distance from some exploitation points.
2. Construct a piecewise surrogate model for the hypercubic utility function by performing M quadratic regressions, each one fitted locally within a distinct cube.
3. Optimize the quadratic surrogate model to find its maximum value.
4. Compare the M maximum values found and choose the best.

The remainder of this chapter outlines each step in detail. To begin with, the notation used in this chapter is laid out.

Notation

The algorithms presented in this section have the following conventions regarding indexing and data structure access:

- Subscripts (e.g., x_i) denote the ith element of a list or array.
- Square brackets (e.g., $x[j]$) are used to access the value at index j of an array or list.
- If an element x_i is itself an array or a list, nested square brackets are used to access its components; that is, $x_i[j]$ refers to the jth element of the ith entry.
- All indexing is **1-based**, unless otherwise specified (i.e., the first element has index 1).

For example, given the list of pairs

$$\texttt{cubeBounds} = \big[[0,1],\ [3,4],\ [5,6],\ [3,4]\big],$$

then

$$\texttt{cubeBounds}_3 = [5,6] \quad \text{and} \quad \texttt{cubeBounds}_3[1] = 5.$$

That is, the subscript refers to the third element of `cubeBounds`, and the square brackets access the first element of that sublist.

Side note:

All configurations of individual parameters are discussed in Chapter 5.

4.1. Sampling

During the agent's initialization, the agent will construct a d-dimensional cube with side lengths ℓ and fill it with the first n Sobol points. The agent then stores these samples, along with their utility values and the bounds of the cube. It does so by calling Function 1: `createCube`.

Function 1 `createCube`(d, ℓ, n)

Require: Dimensions of domain d, cube side lengths ℓ, sample size n.
Ensure: A sample with features in `cubeBids` and labels in `cubeUtils`.

1: `Sobol` $\leftarrow$ Sobol sequence of dimension d
2: **for** $i \leftarrow 1$ to d **do**
3: `cubeBounds`$_i \leftarrow [0, 0 + \ell]$
4: **for** $j \leftarrow 1$ to n **do**
5: `cubeBids`$_i[j] \leftarrow$ round (`Sobol`$_i[j] \cdot \ell$) to the nearest integer
6: **end for**
7: **end for**
8: `cubeUtils` $\leftarrow u$(`cubeBids`)

As the Sobol points are defined in the unit cube, each point needs to be multiplied by ℓ to grow the cube to the desired size. As will be discussed in the upcoming subsection, this cube is then shifted around the domain, which requires its bounds to be known; hence, line 3. Thereafter, each of the d entries of each of the n samples is rounded to the nearest integer.

Sobol sequences are included in many software packages; in this thesis, the Java package *FinMath* was utilized. Regarding the parameters for generating the sequence, only the sample size $n = b^m$ can be tuned. This is done in Chapter 5. In terms of a (t, m, s)-net, the other parameters are predetermined: s represents the number of issues under negotiation, and the smallest possible t for given s and b is found by construction, not by evaluation, meaning that no analytical formula for it exists. That is, t depends on a nondeterministic algebraic structure based on whether a certain set of matrices, which are constructed from direction numbers and primitive polynomials, has full rank. So for any given m, b, and s; t can take on multiple values. Further information, as well as tables with some obtained values for t for Sobol points, can be found in [39].

4.1.1. Where to sample

Once the agent has explored the first cube it creates, it moves on to the next one, and from there to the next one, and so on. This subsection explores how to construct these additional cubes and where to place them.

Reducing time by using a single cube

Reusing the originally constructed cube from Function 1 throughout the entire negotiation can save considerable time. To see this, consider two scenarios: calling Function 1 M times, once for each cube to be sampled from, and calling Function 1 once and then shifting the same sample M times around the domain. The latter approach is ideal when the structure of the sample should not change, as is the case when using a Sobol sample.

Generating a Sobol sample of size n for M randomly placed cubes means performing M Number Generator (NG) calls (line 1 in Function 1) and $M \cdot d$ Random Number Generator (RNG) calls (placement of cube's side $j, j = 1, ..., d$), and $M \cdot n \cdot d$ additions and multiplications (scaling and shifting each sample). However, this can also be done using a single NG call, $M \cdot d$ RNG calls, $M \cdot n \cdot d$ additions, and only $n \cdot d$ multiplications, by shifting the same cube around instead of creating M new ones. Indeed, calling Function 1 once and then shifting that cube around reduces the M NG calls to a single one and requires only a single call to line 5, thereby reducing the $M \cdot n \cdot d$ multiplications to $n \cdot d$. Then each cube can be randomly shifted with $M \cdot d$ RNG calls and $M \cdot n \cdot d$ additions, or shifts. Now, when d, n and/or M are large, this can mean a difference of getting a few extra rounds from the negotiation without affecting performance.

Figure 4.1 illustrates this. The yellow line corresponds to the shifting method, where one million Sobol samples were drawn, scaled to fit the $[0, 2]^d$ cube, and then shifted sequentially through a total of 10 cubes by adding a uniform random number to each of the d entries of each sample with the range set to fit the $[0, 9]^d$ domain. This was done for $d = 1, ..., 50$, corresponding to the dimensions considered for this thesis. For the resampling method — the blue line — 1 million fresh samples were drawn from each cube. The claim is that any difference between the methods should stem from the additional NG calls and multiplications, as well as the added numbers generated in NG calls as the dimension increases.

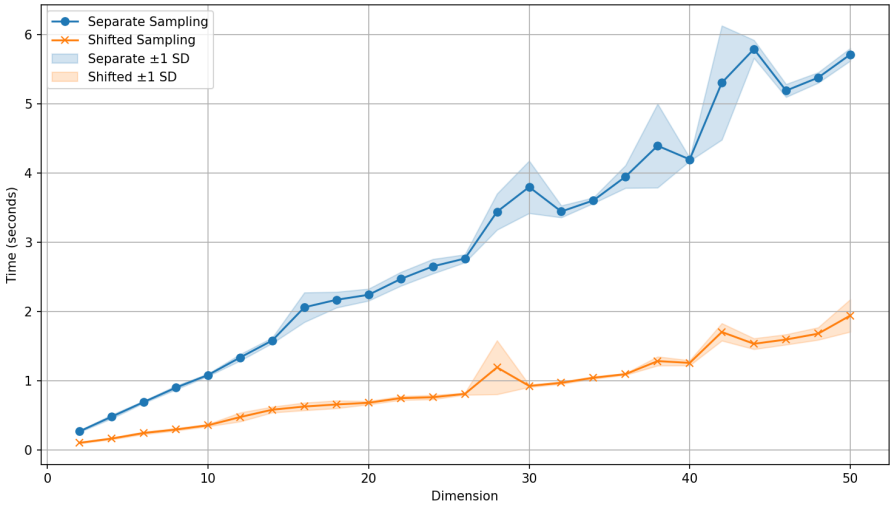


Figure 4.1: The difference in time between sampling 10 cubes with sidelength 2 in increasing dimension vs sampling once and shifting that sample. The figure was generated using Python.

Figure 4.1 shows a clear difference in the time requirement for the two methods. In this particular example, this difference results in the shifting method being around three times faster in 50 dimensions. Reusing and shifting samples can also be employed as part of a parallelization scheme, where each cube is assigned a copy of the base sample and shifted independently. Before shifting the cube, however, the direction of the shift must be considered. That is, whether the shift should exploit previous knowledge or not, and if so, how.

Balancing exploration and exploitation

Once the agent finds a high-utility bid, the region around that bid should be explored as it might contain other good — or even better — bids. Perhaps the constraint satisfied by this bid intersects with another constraint, which then leads to a higher utility.

As the agent knows that this constraint yields a good utility, it could simply stay within the

region of this constraint throughout the entire negotiation. When an agent takes the action that it currently believes to be optimal at each step, it is referred to as a greedy agent [66]. The problem with a greedy agent is that it will focus all its efforts on the first high-utility bid it finds, which may prevent it from finding other, better bids. To solve this problem, the agent should only be greedy some of the time. That is, it must make a trade-off between exploiting what it knows to be good and exploring unknown regions of the domain in hopes of finding something better.

The exploration-exploitation trade-off arises in many AI tasks, most notably in reinforcement learning [66]. One solid approach to this trade-off is the ϵ-greedy strategy. This approach is particularly convenient when exploring large spaces within a given time interval, as ϵ can be made a function of time. Taking this approach, the agent explores with probability $1 - \epsilon$ and exploits with probability ϵ.

This thesis adopts the ϵ-greedy strategy to balance exploration and exploitation with a time-dependent ϵ. That is, before choosing where to sample, the agent calls Function 2: `epsilon`, which indicates how many of the M cubes should be used for exploitation and how many for exploration.

Function 2 `epsilon`(M, t, B)

Require: Number of cubes M, current time t, maximum exploitation ration B.
Ensure: The number of cubes to use for exploitation.
1: $\epsilon \leftarrow g(t)$
2: return(Round $(B \cdot M \cdot \epsilon)$ to the nearest integer)

The time variable t in Function 2 is normalized so that $t = 0$ at the start of the negotiation and $t = 1$ at its end. The exploitation parameter is then defined as $\epsilon = g(t)$, where $g \mapsto [0, 1]$ is increasing with t, which encourages the agent to exploit more as the negotiation progresses. Once $g(t)$ takes its maximum value, or at the end of the negotiation, the B parameter caps the exploitation ratio. That is, the agent never spends more than a proportion corresponding to B of its effort on exploitation in a single round. In principle, the rate of increase can be implemented as any monotonically increasing function of time, and B can be tuned to the utility function complexity. If the utility function is very complex, then perhaps the agent should start exploiting sooner. B can also be made a function, in which case it might make the exploitation ratio higher if it happens to find a high-utility bid in scenarios where it doesn't expect to find one.

Exploiting

There are multiple scenarios that can be exploited. In a negotiation, an agent might exploit its opponent, e.g., by sampling close to where the opponent frequently bids [61, 70], or it might exploit itself by sampling close to where its own bids have been. In this thesis, four specific bids will be exploited:

Exploitation bids

1. The **best** bid the *agent* has made
2. The **last** bid the *agent* made
3. The **best** bid the *opponent* has made

4. The **last** bid the *opponent* made

where "best" is in terms of the agent's own utility function.

When the agent places its cube to exploit exploitation bids 1 or 2, it can't simply place it at the same spot it was when these bids were obtained, as this would simply lead to the same bid. This would have no added benefit as the goal is to find a new and better bid. This is where the discussion of Section 3.2.2 comes in. The agent places a cube's corners at the selected bid and then randomly shifts a subset of the cube's sides by one unit. According to Theorem 5, the probability of falling outside a base constraint after k such independent shifts is given by

$$\left[\frac{\gamma N_{end}}{dN_{int}}\right]^k$$

where N_{int} and N_{end} are described in Lemma 4. Since all constraints are either base constraints or intersections of base constraints, the number of shifts can be adjusted to control this probability. If the agent desires a probability p of leaving any base constraint satisfied by the exploitation bid, this shift behavior must be tuned accordingly. The agent may perform this tuning dynamically, using its prediction accuracy as a guide. A drop in predictive accuracy may indicate a high γ, a low v, or both—since this probability clearly decreases with increasing d, increases with γ, and decreasing with v. In such cases, the agent may be motivated to exploit bids that are closer to the current exploitation points. By Assumption 7, the agent can tune this distance based on Theorem 5 to have some particular probability of falling of a base constraint. Still, it is assumed not to know whether a prediction accuracy is good or bad as it has no access to experiences outside the current encounter. For this reason, the agent will use a time-dependent distance throughout a negotiation that is initialized based on Theorem 5. It does so by calling Function 3: `setDist`.

Function 3 `setDist(dist,` t`)`

Require: Initialized Manhattan distance `dist`, current time t.
Ensure: The Manhattan distance to place its cube from the exploitation bid.
1: return (Round $h(t)$·`dist` to the nearest integer)

As in Function 2, the time variable t is normalized to be between $[0, 1]$. The Manhattan distance is then scaled by $h(t)$ where $h \mapsto [0, 1]$ which ensures that the probability of the cube dropping of the constraint lowers with time. This scaling factor can be chosen to be any decreasing function of time.

The shiftCube function

To summarize this section, Function 4: `shiftCube` describes how the agent goes about placing the next cube to be sampled from.

When the agent explores, it places the `cubeBounds` list at a random location in the domain (line 4) and then moves the bids to fit this cube (line 6). When it exploits, it places each entry of the corresponding exploitation bid at the upper or lower bounds of each `cubeBounds` array (line 15). This must be done with caution, as the bounds must not go outside the domain. At this point, the constraint that was satisfied by the bid is guaranteed to be inside the cube. Then, based on the output of `setDist(dist,`t`)`, which indicates the Manhattan distance to move the cube, the agent chooses some random `cubeBound` arrays and shifts each one of them by 1 so that the corresponding bid entry exits the cube (lines 16-18). The agent then shifts the corresponding

Function 4 `shiftCube`(`mode`, `cubeBounds`, `cubeBids`, `dist`, t)

Require: Which exploitation bid number `mode`, current cube `cubeBounds`, features `cubeBids`, Initialized Manhattan distance `dist`, current time t.
Ensure: Features in `cubeBids` and labels in `cubeUtils` for regression.

1: **if** `mode`$== 0$ **then**
2: **for** $i \leftarrow 1$ to d **do**
3: $r \leftarrow$ uniform random integer between $-$`cubeBounds`$_i[1]$ and $|I|-$`cubeBounds`$_i[2]$
4: `cubeBounds`$_i \leftarrow$ `cubeBounds`$_i + r$
5: **for** $j \leftarrow 1$ to n **do**
6: `cubeBids`$_i[j] \leftarrow$ `cubeBids`$_i$[j] $+ r$
7: **end for**
8: **end for**
9: `cubeUtils` $\leftarrow u$(`cubeBids`)
10: Break
11: **end if**
12: $b \leftarrow$ Exploitation bid `mode`
13: **for** $i \leftarrow 1$ to d **do**
14: Fit `cubeBounds`$_i$ around $b[i]$
15: `dims` $\leftarrow$ uniform random subset of $[1, ..., d]$ of size `setDist`(`dist`,t)
16: **if** $i \in$ `dims` **then**
17: `cubeBounds`$_i \pm 1$
18: **end if**
19: Shift `cubeBids`$_i$ between `cubeBounds`$_i$
20: **end for**
21: `cubeUtils` $\leftarrow u$(`cubeBids`)

entries of the sample by adding the difference between `cubeBounds` before and after they were fit around the bid (line 19). Then, by Theorem 5, the probability of the constraint having left the cube is $\left[\frac{\gamma}{d}\left(\frac{N_{end}}{N_{int}}\right)\right]^{\texttt{dist}}$.

In every round of a negotiation, this function is called M times. Each time, the agent performs regression over the samples and then searches the regression model for high-utility bids. This is the topic of the upcoming sections.

4.2. Regression

The utility function's surrogate model is of the form (3.14), which will be restated here:

$$\hat{u}(\mathbf{x}_i) = \left(\sum_{j=1}^{d}\sum_{k=1}^{d} a_{jk}x_{ij}x_{ik}\right) + \left(\sum_{j=1}^{d} b_j x_{ij}\right) + w_0. \tag{4.1}$$

To derive the parameters of this expression, the design matrix (3.8) is constructed with each row consisting of all possible unary and binary combinations of all variables. Before doing so, however, A must be forced to be symmetric, as that reduces the number of parameters by half.

4.2.1. Making A symmetric

A should by default be symmetric, but due to round-off errors, this can sometimes not be the case. To see this, consider Expression (4.1): the first sum involves two sets of all interaction

terms, one that goes on the lower triangle of A and the other that goes to the upper triangle. By ensuring A is symmetric, only one of these triangles needs to be learned.

This is a simple procedure. By assuming A is symmetric, the first bracket in (4.1) can be rewritten by noting that if $A = A^T$,

$$\left(\sum_{j=1}^{d}\sum_{k=1}^{d} a_{jk}x_{ij}x_{ik}\right) = \sum_{j=1}^{d} a_{jj}x_{ij}^2 + \sum_{j=1}^{d}\sum_{k=j+1}^{d} 2a_{jk}x_{ij}x_{ik}.$$

By doing this, the design matrix becomes

$$\mathbf{\Phi} = \begin{bmatrix} 1 & x_{11} & \dots & x_{1d} & x_{11}^2 & \dots & x_{1d}^2 & 2x_{11}x_{12} & \dots & 2x_{1d-1}x_{1d} \\ 1 & x_{21} & \dots & x_{2d} & x_{21}^2 & \dots & x_{2d}^2 & 2x_{21}x_{22} & \dots & 2x_{2d-1}x_{2d} \\ \vdots & \dots & \vdots & \vdots & \dots & \vdots & \vdots & \dots & \vdots & \\ 1 & x_{n1} & \dots & x_{nd} & x_{n1}^2 & \dots & x_{nd}^2 & 2x_{n1}x_{n2} & \dots & 2x_{nd-1}x_{nd} \end{bmatrix}$$

which has $p = d(d+1)/2$ columns, or weights that need to be learned, as opposed to the $d(d+1)$ weights that would have needed to be learned if each interaction term was double-counted.

4.2.2. Constructing the normal equations

After the design matrix has been constructed, A and b are extracted from the regularized normal equations (3.17), which will be restated here:

$$\hat{\mathbf{w}}_{ML} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}.$$

There are several additional benefits to incorporating a regularization term, as discussed in Section 3.3.4. The regularization term decreases the likelihood of overfitting and ensures that $\hat{\mathbf{w}}_{ML}$ is unique by forcing $\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I$ to be positive definite. Furthermore, as $\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I$ is also symmetric, $(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}$ is guaranteed to exist, even if $\mathbf{\Phi}^T\mathbf{\Phi}$ happens to be singular.

Both due to potential numerical issues and since $(\mathbf{\Phi}^\top\mathbf{\Phi} + \lambda I)^{-1}$ is symmetric and positive definite, the inverse should be derived through Cholesky decomposition. This is done by first factorizing

$$(\mathbf{\Phi}^\top\mathbf{\Phi} + \lambda I) = LL^\top,$$

where L is a lower triangular matrix with positive diagonal entries. The inverse can then be obtained by solving

$$(\mathbf{\Phi}^\top\mathbf{\Phi} + \lambda I)X = I$$

in two steps: first solving $LY = I$ via forward substitution, and then $L^\top X = Y$ via back substitution. The resulting matrix X is the desired inverse. Compared to a general-purpose LU or QR decomposition, this procedure is both faster — requiring only $\frac{1}{3}p^3$ operations — and more numerically stable for symmetric positive definite systems.

4.2.3. Extracting the parameters

Once $\hat{\mathbf{w}}$ has been derived, A, b and w_0 must be extracted so the surrogate model (4.1) can be constructed, or they used to derive $-\frac{1}{2}A^{-1}b$ in case it is a global maximum. This is one of the topics covered in the upcoming section.

Extracting the parameters from $\hat{\mathbf{w}}_{ML}$ is easy, as the first term corresponds to w_0, the following d terms to the entries of b, and the last $d + \binom{d}{2}$ terms to the entries of A. This should be obvious by looking at the design matrix (**??**). This is done in Function 5: `extractFeatures`, which the agent calls after calling Function 4.

Function 5 `extractFeatures(cubeBids, cubeUtils)`

Require: Features `cubeBids`, labels `cubeUtils`.
Ensure: Weights matrices A and b and bias w_0 for Quadratic surrogate.

Construct $\mathbf{\Phi}$ from `cubeBids`
Derive $(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}$ using Cholesky
$\hat{\mathbf{w}}_{ML} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$ where $\mathbf{y}$ =`cubeUtils`
$w_0 \leftarrow \hat{\mathbf{w}}_{ML}[1]$
$b \leftarrow \hat{\mathbf{w}}_{ML}[2 : d+1]$
k = 1 + d + d
for $i = 1$ to d **do**
    $A[i,i] \leftarrow \hat{\mathbf{w}}_{ML}[1 + d + i]$
    for $j = i + 1$ to d **do**
        $A[i,j]$ and $A[j,i] \leftarrow \hat{\mathbf{w}}_{ML}[k]$
        $k \leftarrow k + 1$
    end for
end for

4.3. Optimization

As mentioned in Section 3.4, the difficulty of optimizing the surrogate in search of a maximum depends on its shape. As derived in Section 3.3.3, the stationary point of the surrogate is at

$$-\frac{1}{2}A^{-1}b.$$

Whether this stationary point is a maximum, minimum, or a saddle point depends on the definiteness of A. Indeed, by the second derivative test of multivariate calculus, a function $\hat{u} : \mathbb{R}^n \mapsto \mathbb{R}$ is concave if and only if $\nabla^2 f$ is negative definite for all x. In the case of the quadratic surrogate

$$\hat{u}(\mathbf{x} = \mathbf{x}_i^T A\mathbf{x}_i + b\mathbf{x}_i + w_0,$$

$\nabla^2\hat{u} = 2A$ which is negative definite when A is negative definite. Thus, when A is negative definite, the surrogate model is concave and takes its maximum value at $-\frac{1}{2}A^{-1}b$. When A is negative semidefinite, a small (negative) regularization parameter can be added to its diagonal, shifting its eigenvalues slightly down, making it negative definite.

Thus, there are two scenarios to consider:

- when A is negative (semi)definite, and
- when A is not negative (semi)definite.

4.3.1. When A is negative (semi)definite

If A is negative definite, all of its eigenvalues are less than zero, and A is invertible. Furthermore, the global maximum of the surrogate model will be at $-\frac{1}{2}A^{-1}b$. To check if A is negative definite, the agent utilizes the *spectral theorem* [1]. By the spectral theorem, as A is symmetric with real entries, there exists a diagonalization of the form $A = A = Q\Lambda Q^\top$ where Q is orthogonal and Λ is a diagonal matrix containing the eigenvalues of A on its diagonal. This decomposition is referred to as an *eigendecomposition*, and can be used to check if A is negative (semi)definite or not by checking the values of Λ. Furthermore, if A is negative definite, then A^{-1} exists and can be derived by simply inverting the entries on the diagonal of Λ. This is because $A^{-1} = (Q\Lambda Q^\top)^{-1} = Q\Lambda^{-1}Q^\top$ as Q is orthogonal. The agent checks whether A is negative definite and derives its inverse by calling Function 6: `isNegativeDefinite`.

Function 6 `isNegativeDefinite`(A)

Require: Weights matrix A.
Ensure: A^{-1} if A is negative (semi)definite.
1: $A \leftarrow \frac{1}{2}(A + A^\top) - \lambda I$
2: Derive $Q\Lambda Q^\top$ through eigendecomposition of A
3: **if** $diag(\Lambda) > 0$ for all entries **then**
4: $\quad A^{-1} \leftarrow Q\Lambda^{-1}Q^\top$
5: **else**
6: $\quad A^{-1} \leftarrow$`NULL`
7: **end if**

Function 6 returns A^{-1} as well as the boolean *is A negative definite?*. If A is not negative definite, then the inverse is useless and possibly ill-conditioned, and it is not returned. In either case, whether A is negative semidefinite or negative definite, adding a small negative regularization term $Q\Lambda Q^\top - \lambda I$ is, however, done (line 1). Not only does this ensure A is negative definite (and thereby invertible) in case of semidefiniteness, but it also provides more numerical stability. As the regularizer shifts the eigenvalues μ_i down to $\mu_i - \lambda$, a negative semidefinite matrix becomes negative definite and those $\mu_i \approx 0$ are shifted slightly further from 0 which makes the condition number

$$\kappa(A) = \frac{\max_i(\mu_i + \lambda)}{\min_i(\mu_i + \lambda)}$$

slightly better. If Function 6 returns True, the agent simply derives $-\frac{1}{2}A^{-1}b$, stores it, and moves on to the next cube by calling Function 4 again. However, as $-\frac{1}{2}A^{-1}b$ may be outside the domain, the agent needs to project it back into the cube. This is handled in Function 7, which is the topic of the upcoming section.

4.3.2. If A is not Negative (semi)definite

In the case that $isNSD$ returns False, the surrogate model's global maximum point will not be stationary, but will lie at one of the corners or along one of the axes of the cube. In this case, an optimization algorithm is required. As discussed in Section 3.4, the algorithm that will be used is the L-BFGS-B.

As the global maximum will be at a corner or along one of the axes — just as is the case with a linear model — a linearization of the surrogate is worth exploring. To do this, interaction terms must be eliminated, which can be done by dropping them altogether or by adjusting the corresponding variable to the interaction effect, inspired by Hindriks, Jonker, and Tykhonov WAID method [35]. The variable was adjusted using a weighted average. This was experimented with and is reported on in Appendix C.1. The result of this experiment showed that a single initialization of L-BFGS-B outperforms linearization. As discussed in Section 3.4, initializing L-BFGS-B multiple times will increase its performance. Since a single initialization outperformed linearization, L-BFGS-B will be used solely with multiple initializations. The particular number of initializations is discussed in Chapter 5.

Once Function 5 has been called, the agent calls Function 7: `findBid`, which either constructs a quadratic surrogate and optimizes it or returns $-\frac{1}{2}A^{-1}b$ in case it is a global maximum.

As the objective is to maximize the quadratic function while L-BFGS-B is a minimization algorithm, the quadratic function is minimized (line 8). Lastly, line 4 ensures that x^* is within

Function 7 `findBid`(A, b, w_0, s)

Require: Weight matrices A, b, bias w_0, number of L-BFGS-B initializations s.
Ensure: A bid x^*.

```
1: if !is.NULL(isNegativeDefinite(A)) then
2:     x* = −½A⁻¹b
3:     for i = 1 = 1 to d do
4:         x*_i = min(max(x*_i, cubeBounds_i[1]), cubeBounds_i[2])
5:     end for
6:     Break
7: else
8:     objective← − (xAx^T + bx + w_0)
9:     x* ←NULL
10:    inits← s random points in cubeBounds
11:    for i = 1 to s do
12:        candit= argmax (L-BFGS-B(init[i], objective))
13:        if u(candit) > u(x*) then
14:            x* = candit
15:        end if
16:    end for
17: end if
```

the cube in case $-\frac{1}{2}A^{-1}b$ is outside it, by clamping each entry to the closest point on the cube's boundary in terms of Euclidean distance.

4.4. Conclusion

To conclude this chapter, *QuadrApprox*, short for *Quadr*atic *Approx*imation, will be described in terms of the BOA architecture. QuadrApprox is the name of the agent that utilizes the method drawn out in this chapter as its bidding strategy, and when put up against other autonomous negotiation agents, QuadrApprox employs an acceptance strategy and an opponent model, which are described in this section. However, as the aim of this thesis is to analyze QuadrApprox's bidding strategy, the acceptance strategy and opponent model are simple and implemented without any motivation. These are described here solely for the experiments in Appendix C.2.

4.4.1. Bidding strategy

For the bidding strategy, the agent follows the four steps outlined at the beginning of this chapter: sample, approximate, optimize, and check. This was the topic of this chapter and is summarized in Function 8.

The threshold in line 2 is a decreasing function of time, which indicates the utility threshold that the agent will not go below at time t. Function 8 is an answer to Subquestion 2, which is discussed in Chapter 7.3. To answer the Main Research Question, experiments will need to be conducted with this bidding strategy. This is the topic of the upcoming chapter. First, the agent's opponent model and bidding strategy will be laid out.

4.4.2. Opponent model

The agent's opponent model will be to include exploitation bids 3 and 4 in `exploitBids` in Function 8. This ensures that the agent samples bids in close proximity to areas where the opponent is likely to accept. Based on the logic from Section 3.2.2, if the opponent has previously

Function 8 `makeBid`(B, M, `exploitBids`, `cubeBids`, `cubeBounds`, s, `dist`)

Require: Exploitation threshold B, number of cubes M, which exploitation bids to use `exploitBids`, features `cubeBids`, current cube `cubeBounds`, number of L-BFGS-B initializations s, initialized Manhattan distance `dist`.
Ensure: A bid x^*.

```
x* ←NULL
while u(x*) <threshold(t) do
    exploitCubes ← uniform random subset of [1, ..., M] of size epsilon(M, t, B)
    for i = 1 to M do
        if i ∈ exploitCubes then
            mode ← uniform random number in exploitBids
        else
            mode ← 0
        end if
        cubeBids, cubeUtils←shiftCube(mode, cubeBounds, cubeBids, dist, t)
        A, b, w_0 ← extractFeatures(cubeBids,cubeUtils)
        candidate ← findBid(A, b, w_0, s)
        if u(candidate) > u(x*) then
            x* ← candidate
        end if
    end for
end while
Propose bid x* to opponent
```

proposed a bid, then proposing a bid at a small Manhattan distance from it increases the likelihood of acceptance.

4.4.3. Acceptance strategy

When receiving bids, the agent should have a way to decide whether to accept the bid or make a counteroffer. It does so using Function 9: `acceptBid`, which is an adaptation of the acceptance strategy used by AgentM [61]. That is, the agent will only accept bids that are greater than or equal in utility to the average utility of its own best and its own worst bids.

Function 9 `acceptBid`($\mathbf{x}$)

Require: A bid from opponent $\mathbf{x}$, best bid $\mathbf{x}_{best}$, worst bid $\mathbf{x}_{worst}$.
Ensure: Boolean `accept`.

```
if u(x) ≥ average(u(x_best), u(x_worst)) then
    accept ← True
else
    accept ← False
end if
```

5 Experiments

The research question will be addressed through experiments. To assess how the method performs with increasing preference complexity, it is necessary to conduct numerous experiments and then combine their results. In this chapter, the experimental setup for conducting this assessment will be described.

5.1. Experiment overview

For the main experiments, the agent will take part in bilateral SAOP negotiations against a clone of itself with the threshold in line 2 of Function 8 set to zero for all t. This will be done using agents that model the utility function using linear and quadratic regression. The two negotiating agents will have separate preferences but of the same complexity, and after each negotiation, both agents log the bids they made and the corresponding utility. As each negotiation will essentially have two versions of the agent, there will be twice as much data. That is, the bidding strategy from Section 4.4 will be experimented with. This will provide clear insight into how efficiently an agent finds high-utility bids when it searches the surrogates instead of the utility function itself.

5.2. Experimental setup

Experiments will be carried out in Genius, an automated negotiation simulator that is discussed in Appendix A.1, with the hypercubic utility functions described in Chapter 3. The agent's performance will be assessed for increasing complexity of these functions, measured as described in Section 3.1. This will be done by randomly generating utility functions from (d, m, γ, v) profiles with varying values of their elements. The particular values are discussed in the upcoming subsection. The results will then be displayed and analyzed in terms of the agent's ability to find high-utility bids over the range of each parameter, as well as over the range of the complexity measure described in Section 3.1, which will provide a definite answer to the Main Research Question

5.2.1. Negotiation scenarios

To answer the research question, a sufficiently broad complexity range must be experimented with to determine where the method performs optimally and where it does not perform at all. To ensure this, the (d, m, γ, v) tuples used to generate the utility functions need to be set to reflect this range.

Side note:

Recall Assumption 7

The number of issues, or the d parameter, will be in $\{10, 20, 30, 40, 50\}$, and each issue can take any of 10 values. This yields domains of sizes that range from 10^{10} to 10^{50}. At the lower end of this spectrum, domains of size 10^{10} are still, in principle, exhaustively enumerable. As the domain size increases toward 10^{50}, exhaustive search is no longer an option, not even theoretically. The number of possible bids corresponds to the number of atoms on Earth.

The dimensionality of the constraints, or the γ parameter, will range from 1 to 10. This is motivated similarly to the domain sizes. For all constraints, both original and intersection constraints, the minimum γ values are those of the base constraints, as per Lemma 1. As quadratic functions capture pairwise interdependencies, constraints constructed from the lower end of this spectrum should be feasible for a quadratic function to capture. As γ increases towards 10, the level of interdependencies in the function will be far beyond what a quadratic model can be expected to capture.

The width of the constraints, or the v parameter, will be in $\{2, 4, 6\}$. As the effects of v and γ on complexity are highly dependent on each other, a value of $v = 6$ for all base constraints and a value of $\gamma = 1$ should represent constraints that a quadratic function can capture. However, when v is reduced to 2, a relatively high value of γ is expected to make the function too complex for a quadratic model to capture.

The number of constraints, or the m parameter, will be in $\{10, 25, 50, 75, 100\}$. This range corresponds to a maximum of 1013 intersection constraints and a maximum of $\approx 10^{30}$ intersection constraints, as per Theorem 3. However, these are upper bounds, and the actual number will most likely be considerably lower. When there are at most 1023 total constraints, corresponding to $m = 10$, and if they are derived from low-complexity base constraints, it should be feasible for the model to find high-utility bids. However, when the function consists of $m = 100$, potentially high-complexity, base constraints, and the maximum number of intersection constraints is $\approx 10^{29}$, it should be extremely unlikely that the model performs well.

By focusing on these ranges, the experiments in this thesis stress-test the method under "highly possible", "heavy but possible", and "completely intractable" conditions. This helps illustrate where more capacity is needed in the model.

5.3. Agent configuration

Before conducting these experiments, the agent's parameters must be specified. These are the cube sizes, the number of cubes, the Manhattan distance, the functions g and h from Functions 2 and 3, the sample sizes and regularization parameters, and the number of L-BFGS-B initializations.

5.3.1. The Cubes

To ensure that there will also be samples inside the cube and its edges and not only at its corners, ℓ must be at least equal to 2. If this is not done, the quadratic model cannot have an internal optimum, resulting in the model essentially acting like a linear model, with its maximum value occurring at a corner of the cube.

For this thesis, the cube's side lengths ℓ were set to 2. The reason for this is twofold. First, the number of bids contained in a cube with side lengths ℓ is $(\ell + 1)^d$ or 3^d when $s = 2$, of which

$3^d - 2^d$ will be internal points. This also means that increasing the side lengths by 1 when d is already large will greatly increase this number. However, a cube that is too large will result in too significant an information loss, as discussed in Section 3.2. Second, a side length of 2 ensures that 30% of each issue domain is occupied by each cube, which constitutes more than the smallest constraint (having width $v = 2$) will do. This is not to say the cube occupies 30% of the domain, though, as the curse of dimensionality makes that proportion shrink. However, as per Theorem 2, the probability of a cube with side lengths $\ell = 2$ capturing a constraint is always relatively high.

The number of cubes M placed in each round of negotiation was set to 100. A natural question might be whether this biases the results — after all, 100 cubes cover a much smaller proportion of a domain of size 10^{50} than of one of size 10^{10}. The short answer is: not really. As shown in Theorem 2, the probability of a randomly placed cube intersecting a constraint is independent of the overall domain size. The results will confirm this. The choice of $M = 100$ was therefore made for simplicity. It's large enough that the agent has a good chance of finding a high-utility bid, but still small enough to keep the runtime of Function 8 reasonable. Increasing M further was not expected to significantly improve the results, since the agent is ultimately limited by its own approximation capacity. That said, Function 4 is trivial to parallelize, so in a practical implementation, M could be scaled up according to available computational resources and negotiation time limits.

5.3.2. The manhattan distance

The Manhattan distance, or the `dist` variable in Function 8, was initialized to be equal to d, corresponding to a single shift along each axis. The h function in Function 3 was set to $(1 - t)$, which ensures the agent's risk aversion increases linearly as the negotiation progresses. That is, `dist` decreases from d to 0 over the course of the negotiation.

5.3.3. Exploration and exploitation

To prevent the agent from abandoning exploration entirely, ϵ was capped at $\epsilon = 0.8$ by setting $B = 0.8$ in Function 2. Thus, at all times, the agent spends at least 20% of its effort exploring. The g function in Function 2 was set to t^2 so the agent only explores in the beginning of the negotiation but quickly starts exploiting what it finds.

5.3.4. Sample sizes and regularization parameters

Even though there is a regularization term which ensures that $\hat{\mathbf{w}}_{ML}$ is unique, it is still a good measure to have the sample size large *enough*. As the design matrix contains quadratic terms, in the absence of a regularization term, collinearities can occur even if the samples are not collinear and $n \geq p$. For example, the design matrix might have the linear vectors $v_2 = (1, 0, 1, \mathbf{0})$, $v_2 = (1, 1, 0, \mathbf{0})$, and $v_3 = (1, 0, 0, \mathbf{0})$, where $\mathbf{0}$ indicates the rest of the vector is all zeros. These vectors are not collinear; however, the interaction term $v_1 v_2$ is equal to $(1, 0, 0, \mathbf{0}) = v_3$. This can especially arise in integer domains where each variable has a small number of values to take. When the regularization term is very small, vectors can be *approximately collinear*. It is therefore a good measure to have $n \gg p$.

As mentioned in Section 3.3, the main appeal of deriving the parameters of the regression model via maximum likelihood and least squares is that it can be shown to be the best estimator asymptotically in terms of its rate of convergence as the sample size increases. That is, the model should perform better as the sample size increases. Although sampling is cheap, deriving $\hat{\mathbf{w}}_{ML}$ can become expensive when the sample size gets very large. In an autonomous negotiation

with time constraints, a trade-off must therefore be made between the computational cost and accuracy. On the other hand, the marginal benefit of increasing the sample size rapidly decreases as the sample size grows. That is, the difference in performance when a sample size n is increased to $n + r$ is much greater when n is small than when n is large.

One rough heuristic that is sometimes advocated is that the number of data points should be no less than some multiple of the number of adaptive parameters in the model [12]. This approach will be adopted with the particular multiple derived through experiments. Using the complexity measure from Section 3.1, a "good" sample size for each level of complexity was computed.

This was done using leave-one-out cross-validation with regularization parameters

$$\lambda \in \{10^3, 10^2, 10, 10^{-1}, 10^{-2}, 10^{-4}, 10^{-6}\}$$

and over sample sizes ranging from small up to the entire domain. To achieve this, smaller domains were required than those used in the main experiments. Then, the training and generalization errors were compared by measuring their gap, or the absolute difference between them. The lowest sample size that resulted in a gap within a 5% range of the lowest gap obtained for that complexity level was then stored, along with the regularization parameter used to obtain it. Going by this gap is motivated by the discussion in Section 3.3.4.

Additionally, the regularization parameter drops sharply from a large value to nearly zero as the complexity of the utility function increases. This behavior is expected: once the utility function becomes more complex than what a simple quadratic model can effectively capture, there is less need to penalize complexity, and the model no longer benefits from being regularized. In fact, the optimal regularization parameter quickly settles at 10^{-6}, corresponding to a complexity level of approximately 6. Therefore, a fixed regularization parameter of 10^{-6} will be used for all experiments with complexity > 6. This can be seen in Figure 5.1

The domain sizes used for these experiments were used to compute the parameter multiple to be used in the main experiments. That is, a quadratic regression needs $d(d+1)/2$ parameters in a d-dimensional domain. If a sample size of $10,000$ was found for a 10-dimensional domain, then the desired parameter multiple is $\frac{10,000}{55} \approx 182$. However, using smaller domains leads to the complexity that was experimented with only being in the range of $[0, 12]$ while the main experiments will be conducted over the range of $[2.5, 20]$. As they seemed to be linearly increasing — and for simplicity — a linear regression was performed on the data and used to extrapolate to the higher complexity levels. Figure 5.1 depicts the resulting sample sizes as multiples of the parameters needed in the corresponding domain size.

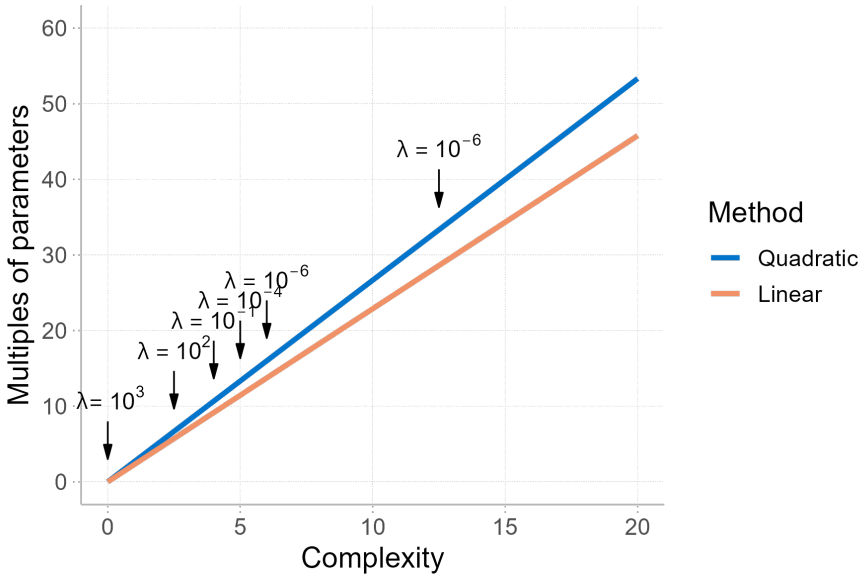


Figure 5.1: The sample sizes to be used as a multiple of model parameters along with the regularization terms. For the linear model, the regularization parameters converged to 10^{-6} by complexity level 5, but are omitted from the plot to avoid visual clutter.

To motivate this approach as opposed to considering domain size — as would be more typical when tuning a sample size — consider the *meaning* of the complexity measure. A hypercubic function of a certain complexity should not require more samples even if the domain size increases. Indeed, the complexity measure represents the probability that the sample satisfies a constraint, and this probability already accounts for the domain size. Furthermore, by Theorem 2, the probability of a random sample satisfying any given constraint does not depend on domain size. Thus, the domain size should be considered only as a complexity parameter, rather than the size of the search space.

Side note:

As these experiments tested the generalization error of both methods, this experiment also provided a glimpse of the results to come. Indeed, in most cases, the optimal sample size corresponded to a lower error for the quadratic model than the linear model.

5.3.5. Number of initializations

The experiment mentioned in Section 4.3.2 (see Appendix C.1.1) shows that L-BFGS-B clearly outperforms linearization when optimizing a quadratic function and will therefore be used for that task. However, as L-BFGS-B is a local optimizer that is being used for maximizing high-dimensional non-concave functions, multiple starts are necessary. However, increasing the initializations comes with a computational cost that needs to be weighed against its benefit.

To gain a better insight into the benefit of adding more initializations, $1,000$ random quadratic functions were generated in 50 dimensions, and each one was optimized using 500 initializations of L-BFGS-B. This dimensionality was chosen as it is the maximum considered in this thesis, and because higher dimensions naturally require more initializations. For each initialization, the best performance up until, and including, the current one was stored. Figure 5.2 shows the average marginal difference between the performance of 500 initializations and n initializations where $n = 1, ..., 499$, with a zoomed-in window on indices 10 to 50 and 150 to 200.

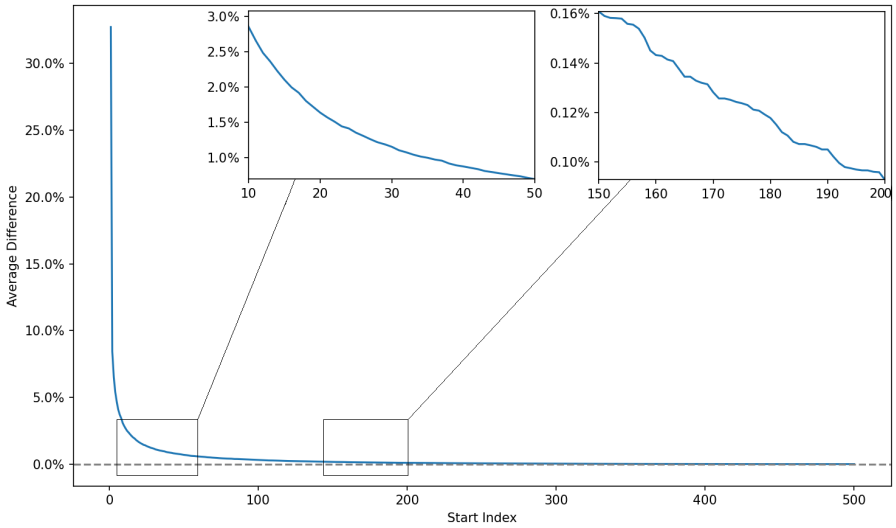


Figure 5.2: The maximum value found with n initializations divided by that found after 500 initializations of L-BFGS-B in the same function plotted as a percentage difference.

As expected, the marginal improvement decreases rapidly with each additional initialization. The average difference in performance between using 1 and 500 initializations is just over 30%, while the improvement from 35 to 500 initializations is under 1%, and from 200 to 500 is under 0.1%. This demonstrates a classic "more is better" trade-off: more initializations yield better results, but with diminishing returns and increased computational cost. To validate this in practice, the agent was experimented with using $1, 10, 35$, and 200 initializations. Results (presented in Appendix C.1.2) confirm the trend shown in Figure 5.2. Specifically, the difference in average utility between using 35 and 200 initializations was approximately 1%, favoring 200 initializations, but this came at more than double the runtime.

Given the low difference between the two, obtaining twice as many bids was deemed more important for QuadrApprox than achieving a slightly better optimization performance. However, to answer the Main Research Question, 200 initializations were used, as the marginal improvement beyond this point of approximately 0.1% when compared to 500 initializations did not justify the added computational complexity. Consequently, the final implementation of `QuadrApprox` uses 35 initializations per optimization, while the experimental results in Chapter 6 are based on 200. Lastly, the recursion parameter m was set to 10, which is the default value in most libraries.

5.4. Summary

The following bullet points summarize the experimental setup:

- Each issue domain has size $|I| = 10$.
- The number of issues are $d \in \{10, 20, 30, 40, 50\}$.
- The number of issues per base constraint is $\gamma \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.
- The number of base constraints is $m \in \{10, 25, 50, 75, 100\}$.
- The width of an base constraint is $v \in \{2, 4, 6\}$.
- Each cube will have side lengths $\ell = 2$.
- The number of cubes is $M = 100$.

- The Manhattan distance for exploiting is initialized at d and decreases at the rate of $(1-t)$.
- The number of cubes to use for exploitation is $0.8 \cdot M \cdot t^2$.
- The sample size and the regularization parameters can be seen in Figure 5.1
- 200 L-BFGS-B initializations will be used for each surrogate in the main experiments, but 35 initializations when QuadrApprox is deployed, to get more rounds in. This is done with the recursion parameter $m = 10$.

6

Results

This chapter presents the results of the experiments conducted to derive an answer to the Main Research Question.

This chapter is divided into two main sections. The first focuses on the individual effects of the four complexity parameters on the performance of an agent using surrogate models, as well as the cross-effects between them. Specifically, it examines the agent's efficiency in finding high-utility bids within the utility function with respect to each parameter and its cross-effects.

The second section evaluates model performance with respect to complexity as measured by Definition 5. This is done considering both the agent's ability to find high-utility bids and in terms of mean squared error (MSE). MSE is introduced only in this final section because the Main Research Question concerns the agent's performance in negotiation contexts, where finding high-utility bids is more relevant. Nonetheless, MSE is included to provide a more complete picture.

6.1. How the method scales with the complexity parameters

In this section, the results from the experiments will be showcased in terms of the individual effects of the four complexity parameters, d, m, v and γ, and their cross-effects. This is done by fixing one or two parameters and averaging the utility obtained over the entire range of the parameters while holding one or two of them fixed. This averaging allows to isolate and observe the impact of each parameter, as well as each parameter pair, on its own.

Since the ranges of each complexity parameter are structured to span from very simple to very complex cases, this averaging will result in the individual utility values being rather low when plotted against individual parameters. However, the focus here is not as much on the individual utility values as it is on the shapes of the resulting curves. The utility values become more meaningful when plotted against the overall complexity measure in the next section. In this context, a flat curve indicates that a parameter has little individual influence on the agent's performance, while a steeper curve suggests a stronger individual effect. A cross-effect indicates that increasing complexity through one parameter increases the induced complexity of the other.

6.1.1. The dimensionality of the constraints

The γ parameter, or the dimensionality of the constraints, dictates the level of interdependency between issues. As quadratic functions capture interdependencies between pairs of issues, a

high γ value was expected to substantially reduce the accuracy of the approximation, making it difficult for the agent to find high-utility bids. Conversely, for low values of γ (i.e., $\gamma = 1$ or 2), the agent was expected to perform reasonably well. Figure 6.1 shows the agent's performance in finding high-utility bids using both linear and quadratic surrogate models of the utility function across the entire range of γ.

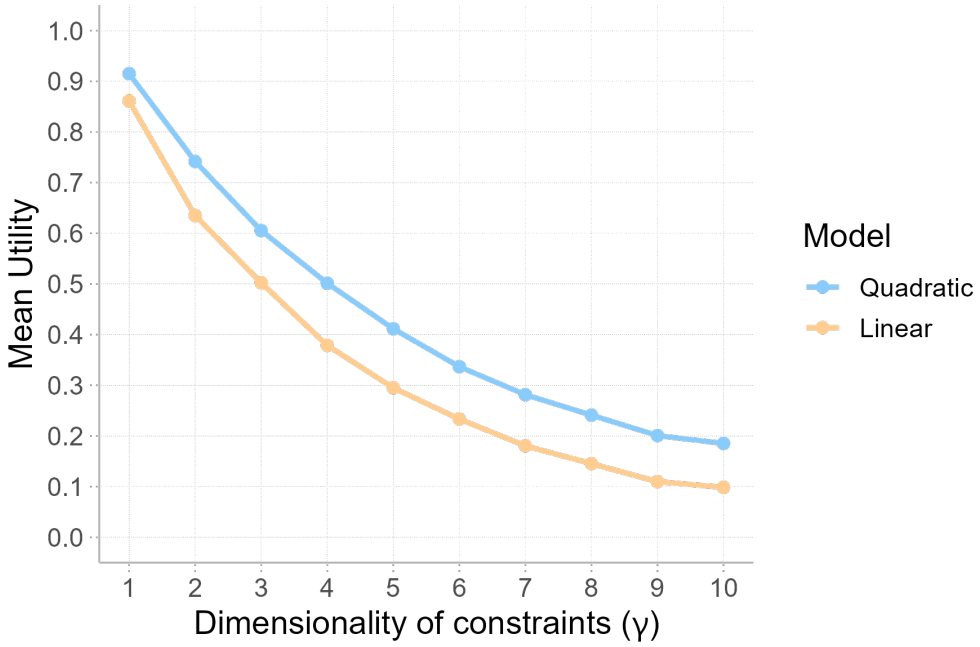


Figure 6.1: The individual effect of γ on the performance of the agent while using linear and quadratic approximations.

The steep decline showcased in Figure 6.1 suggests that increasing γ produces a great negative effect on the performance of both methods. This is consistent with Figure 3.3, which shows that γ has the greatest effect on the complexity measure of all parameters.

Not surprisingly, the quadratic model outperforms the linear model over the entire axis. Both the linear and the quadratic models do, however, perform very well when γ is low. When $\gamma = 1$, both models are consistently finding bids with utility around 0.9. When $\gamma = 2$, the performance of both models reduces quite drastically, from 0.91 to 0.74 in the quadratic case, and from 0.87 to 0.63 in the linear case. These numbers are not bad, but a reduction of this significance is particularly surprising for the quadratic model, as quadratic functions capture pairwise interdependencies while linear functions do not. However, the performance of the linear model does reduce more than that of the quadratic one. As γ increases beyond $\gamma = 2$, both models' performance reduces in a seemingly exponential manner, with the difference between the two stabilizing around $\gamma = 4$.

Figures 6.2 presents a heatmap of average utilities obtained for all combinations of parameter values between γ and v. The marginal rate of change is then plotted above and to the right of the heatmap, illustrating the cross-effects by showing how utility changes along one parameter while the other is held constant. When the rate of change in utility of one parameter, say γ, varies significantly across different fixed values of the other parameter, such as v, the complexity induced by γ is highly dependent on v. Conversely, if the rate of change remains relatively constant across all fixed values of v, the complexity induced by γ is independent of v. When the

induced complexity from γ depends on the value of v and vice versa, cross-effects are present.

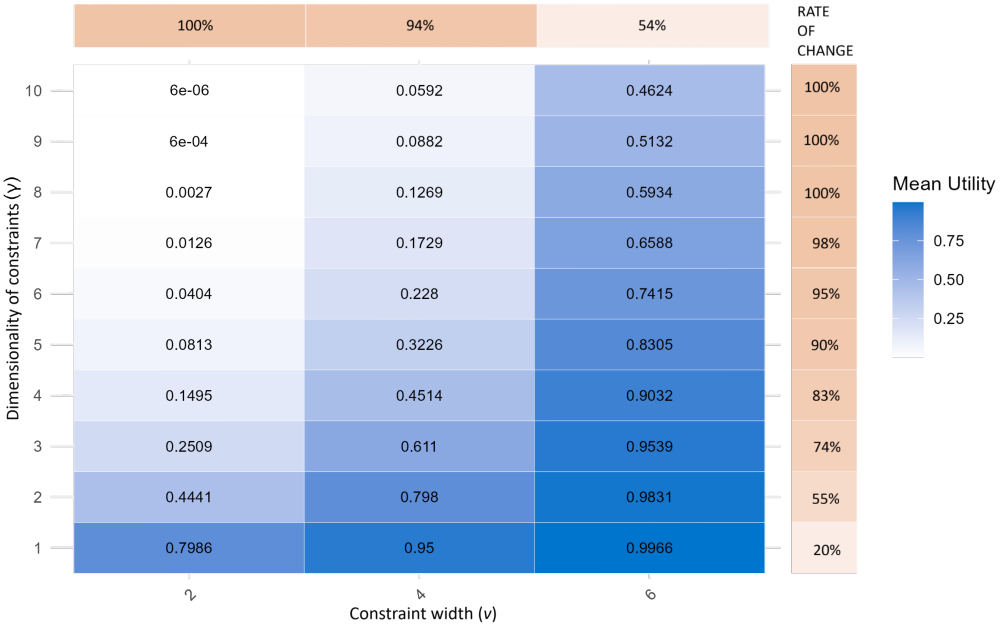


Figure 6.2: Average utilities obtained while fixing v and γ and averaging over the other parameters. Rate of change on right (top) side indicates the change in utility over the corresponding row (column)

In Figure 6.2, the cross-effects are particularly pronounced. Specifically, when $v = 2$ and $v = 4$, the utility decreases by around 100% over the range of γ, while decreasing by just over 50% when $v = 6$. Furthermore, when fixing γ, the marginal rate of change in v varies between $20 - 100\%$, indicating that v might be more dependent on γ than vice versa. Additionally, the difference between the highest obtained utility, corresponding to $v = 6$ and $\gamma = 1$, and the lowest, corresponding to $v = 2$ and $\gamma = 10$, is essentially the entire utility range $[0, 1]$. This clearly demonstrates how the agent's performance is significantly impacted when a low value of v is coupled with a high value of γ.

Similarly, Figure C.4 shows that when d is fixed, this range is $63 - 88\%$ with a notably sharp increase between $d = 10$ and $d = 20$, meaning that the induced complexity of γ is somewhat dependent on the value of d but not nearly as much as on v. However, when fixing γ, the marginal rate of change in d ranges from 4% to 68%, indicating a very strong dependence.

In Figure C.6, the complexity induced by γ can be seen to be somewhat independent of the value of m. In contrast, when fixing γ, the marginal rate of change in m ranges from 11% to 44%. As a result, this represents an asymmetric dependency rather than a true cross-effect between the two parameters.

6.1.2. The width of the constraints

Recall that the hypercubic constraints are defined as sets of intervals. The v parameter, or the width of the constraints, indicates the lengths of these intervals. When these intervals are shorter, meaning the constraints are narrower, the corresponding hypercubes are smaller, and the function becomes "wilder". Due to the limited expressiveness of a quadratic function, setting $v = 2$ was expected to decrease significantly the agent's performance, as opposed to $v = 4$ or 6, especially when coupled with a high γ. Figure 6.3 shows the agent's performance in finding high-utility bids using both linear and quadratic surrogate models of the utility function across the entire range of v.

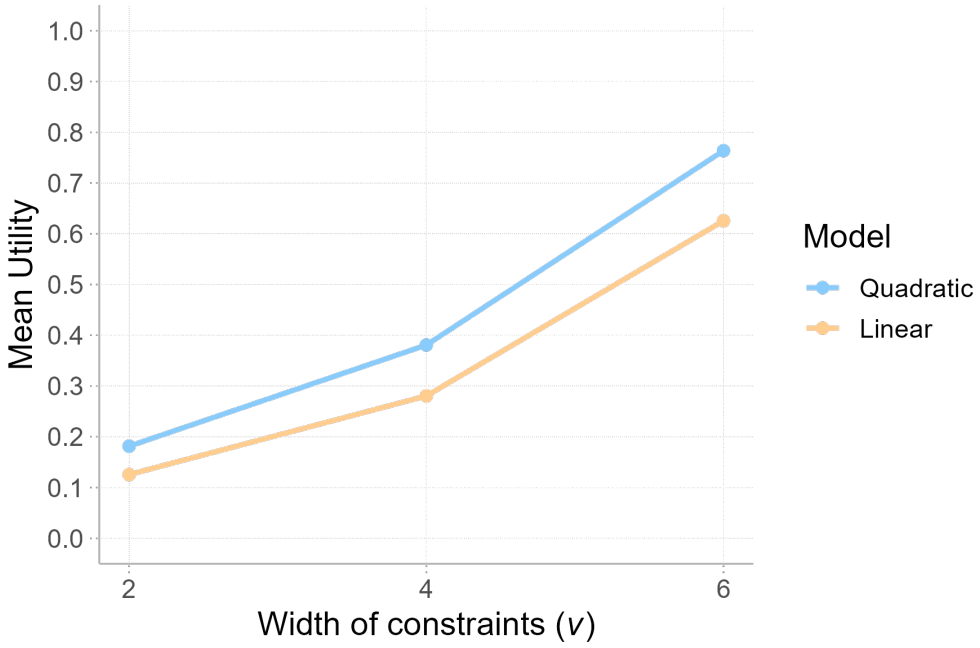


Figure 6.3: The individual effect of v, the width of the constraints, on the performance of the agent while using linear and quadratic approximations.

The steep incline that can be seen in Figure 6.3 suggests that the individual effects of decreasing v produce a great negative effect on the performance of both models. Furthermore, these effects are very much in line with Figure 3.3.

When constraints are narrow, or have width $v = 2$, both models perform roughly the same, reaching a very low average utility of around 0.2. This average then shows an apparent increase with increasing v, reaching a high average of 0.77 for the quadratic model and 0.62 for the linear model. The gap between the two methods is slightly increasing over the entire axis, indicating that their performance scales similarly with respect to constraint width, with the quadratic model perhaps scaling slightly better.

When coupled with γ, the cross-effects in Figure 6.2 are very clear, as previously discussed. When coupled with m, cross-effects are also quite pronounced, although not as much as with γ. Figure 6.4 illustrates this.

Figure 6.4 shows that, when m is fixed, the rate of change for v ranges from $70 - 81\%$, which indicates a rather small dependence compared to that of γ. On the other hand, when fixing v, the rate of change for m ranges from $20 - 50\%$, indicating that the induced complexity of m is somewhat highly dependent on v.

In Figure C.5, the complexity induced by v can be seen somewhat independent of the value of d, ranging between $73 - 78\%$ when d is fixed. When v is fixed, d shows a range of $10 - 27\%$, which indicates a slight but noticeable dependence. Thus, only an asymmetric dependency exists between d and v, where v can be seen to be independent of d.

6.1.3. The number of constraints

The m parameter, or the number of constraints, is not quite true to its name. As per Theorem 3, the potential number of interaction constraints when $m = 100$ can become $\approx 10^{30}$ in the scenarios considered in this thesis, with all of these constraints having equal or greater complexity than the

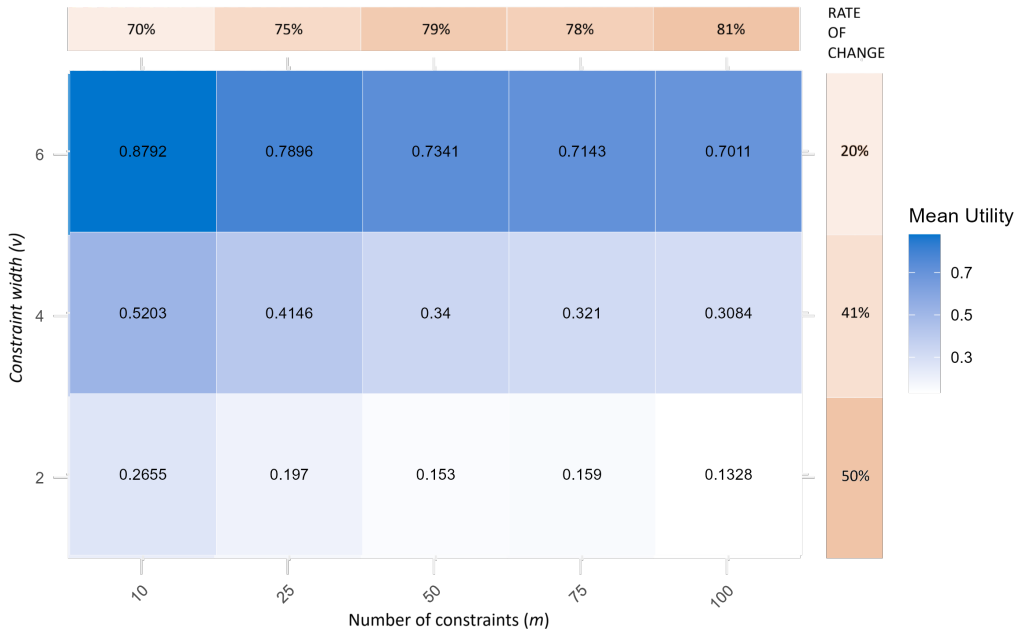


Figure 6.4: Average utilities obtained while fixing m and v and averaging over the other parameters. Rate of change on right (top) side indicates the change in utility over the corresponding row (column)

original ones. Furthermore, as these intersections are the peaks of the function, more constraints mean a wilder, more complex function. As $m = 10$ corresponds to at most 1023 constraints while $m = 100$ corresponds to at most $\approx 10^{29}$, the agent's performance was expected to decrease over this range. Figure 6.5 shows the agent's performance in finding high-utility bids using both linear and quadratic surrogate models of the utility function across the entire range of m.

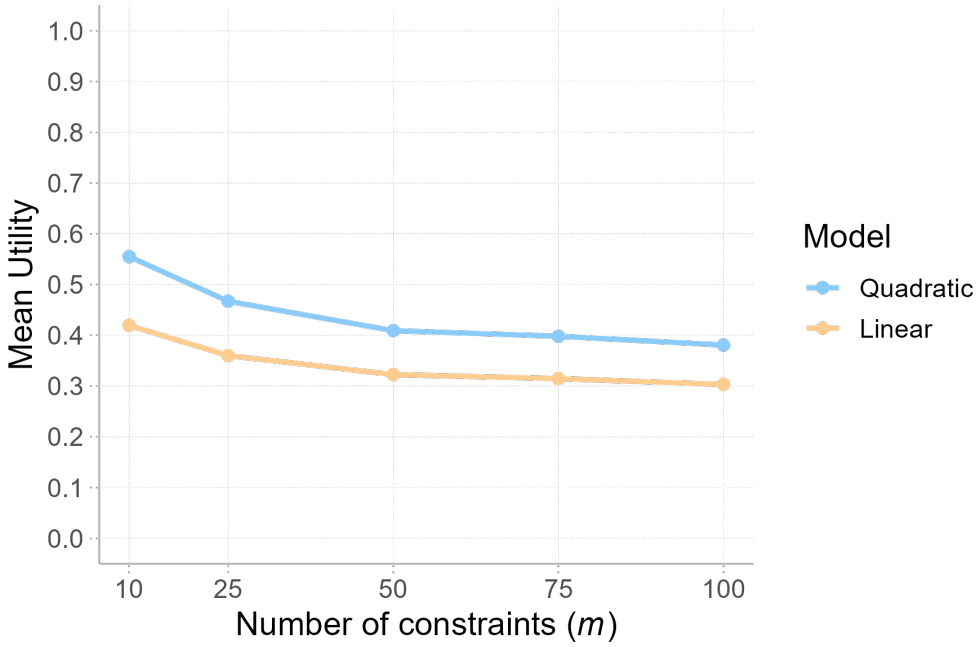


Figure 6.5: The individual effect of m, the number of constraints, on the performance of the agent while using linear and quadratic approximations.

The effects of the number of constraints can be seen delivering a small but consistent decrease in the average utility for both methods. Furthermore, as the utility obtained in the lower range of m is not very high, similar to v and γ, the m parameter is likely highly dependent on the other parameters and does not significantly increase complexity on its own. This is in line with Figure 3.3, which shows that m does have some individual effects but not quite of the same caliber as γ and v.

Due to the consistent utility value of around 0.4 over the x axis, this should be representative of the average utility obtained across all experiments. Although this might seem rather low, this, as previously mentioned, is obtained by averaging over the other parameters. As previous results suggest, a high γ and low v are moving this average down. If γ were cut off at a lower number than 10, these lines would shift upwards. This can be clearly seen in Figures 6.4 and 6.2, where fixing $\gamma = 1$ or $v = 6$ results in utilities in the ranges of $0.87 - 0.98$ and $0.7 - 0.87$ over all values of m, respectively.

Indeed, it has been shown that the induced complexity of m is highly dependent on v and γ. Furthermore, Figure 6.6 illustrates the relationship between m and d.

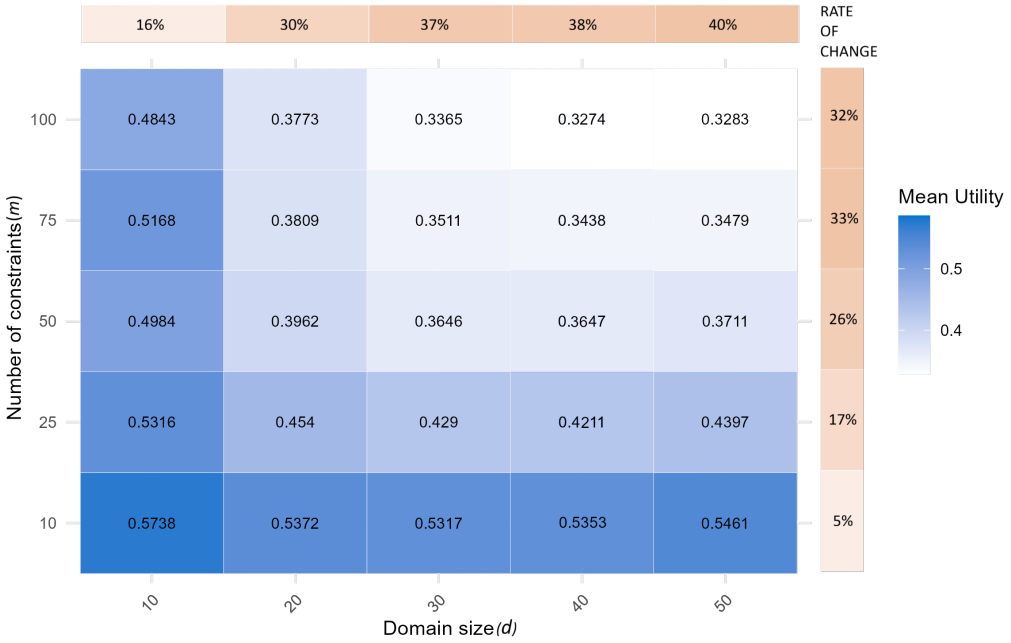


Figure 6.6: Average utilities obtained while fixing d and m and averaging over the other parameters. Rate of change on right (top) side indicates the change in utility over the corresponding row (column)

Figure 6.6 confirms that, although the effects on performance produced by m alone are quite minimal, the induced complexity of m does highly depend on all other parameters. Additionally, the induced complexity of d can also be seen to highly depend on m, indicating the presence of cross-effects.

6.1.4. The number of issues

The d parameter, or the number of issues, indicates the domain size. As each issue domain has size 10, the domain size is $|\Omega| = 10^d$. As mentioned in Section 3.1, the domain size was not expected to affect the agent's performance a lot on its own, but rather that it would produce more strain on the hardware. Figure 6.7 shows the agent's performance in finding high-utility bids using both linear and quadratic surrogate models of the utility function across the entire

range of d. When averaged over the other parameters, the effect of domain size itself is minimal.

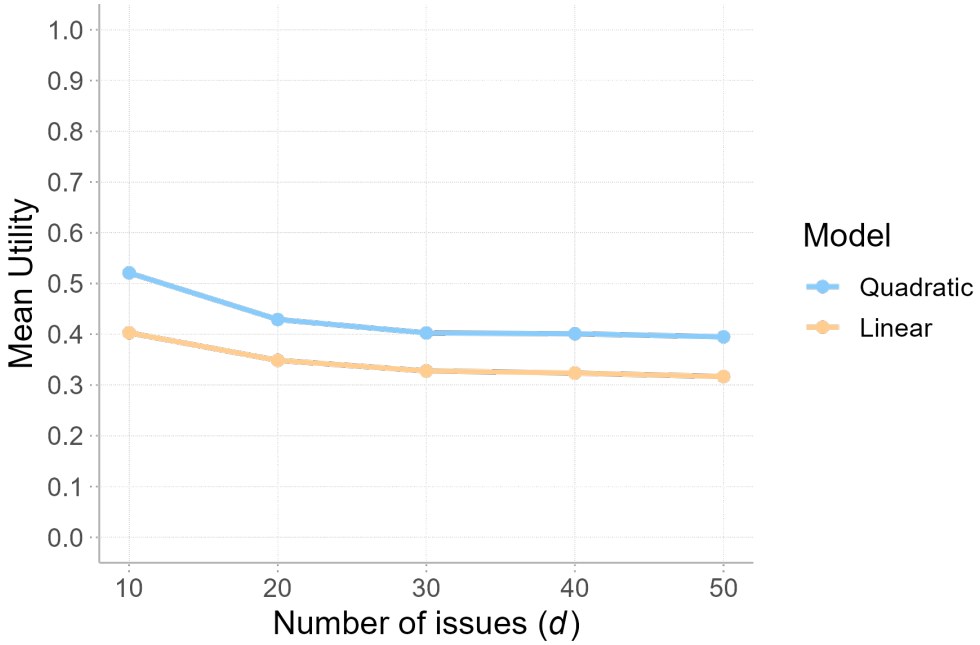


Figure 6.7: The individual effect of d, the number of issues on the performance of the agent while using linear and quadratic approximations.

Although there is a fairly sharp decrease between $d = 10$ and $d = 20$, the curve is somewhat stable thereafter. This is in line with Figure 3.3, which suggested that d would have practically no individual effects on complexity. This does not come as a surprise, as the effects from the domain size should mostly be visible when varying two or more variables. As discussed in Section 3.1, the complexity produced by d is mostly through creating potential for the other parameters.

As has previously been illustrated, the induced complexity of d highly depends on all the other parameters. Although m and d showed rather minimal individual effects, Figure 6.6 shows that there is a presence of cross-effects between them. Figure C.4 furthermore shows a presence of cross–effects between d and γ. However, only an asymmetric dependency exists between d and v, where v was shown to be independent of d.

6.2. How the method scales with complexity

Finally, the top part of Figure 6.8 shows the agent's performance in finding high-utility bids using both linear and quadratic surrogate models of the utility function across the entire range of the complexity measure derived in Section 3.1. The bottom part depicts the MSE obtained over the experiments. As can be seen in Figure 6.8, the agent performs very well with low complexity functions, but this performance rapidly decays through the medium complexity functions, and finally reaches an average utility of 0 for complexities above ≈ 14. The agent using the quadratic model can be seen outperforming the linear model over the entire axis (excluding the part where both models average a utility of 0 or 1). Furthermore, the MSE obtained is consistently higher for the linear model over the entire axis.

Comparing the top and bottom plots in Figure 6.8, the MSE obtained in the experiments follows a similar pattern to the agent's performance. For the quadratic model, an MSE between 0.015

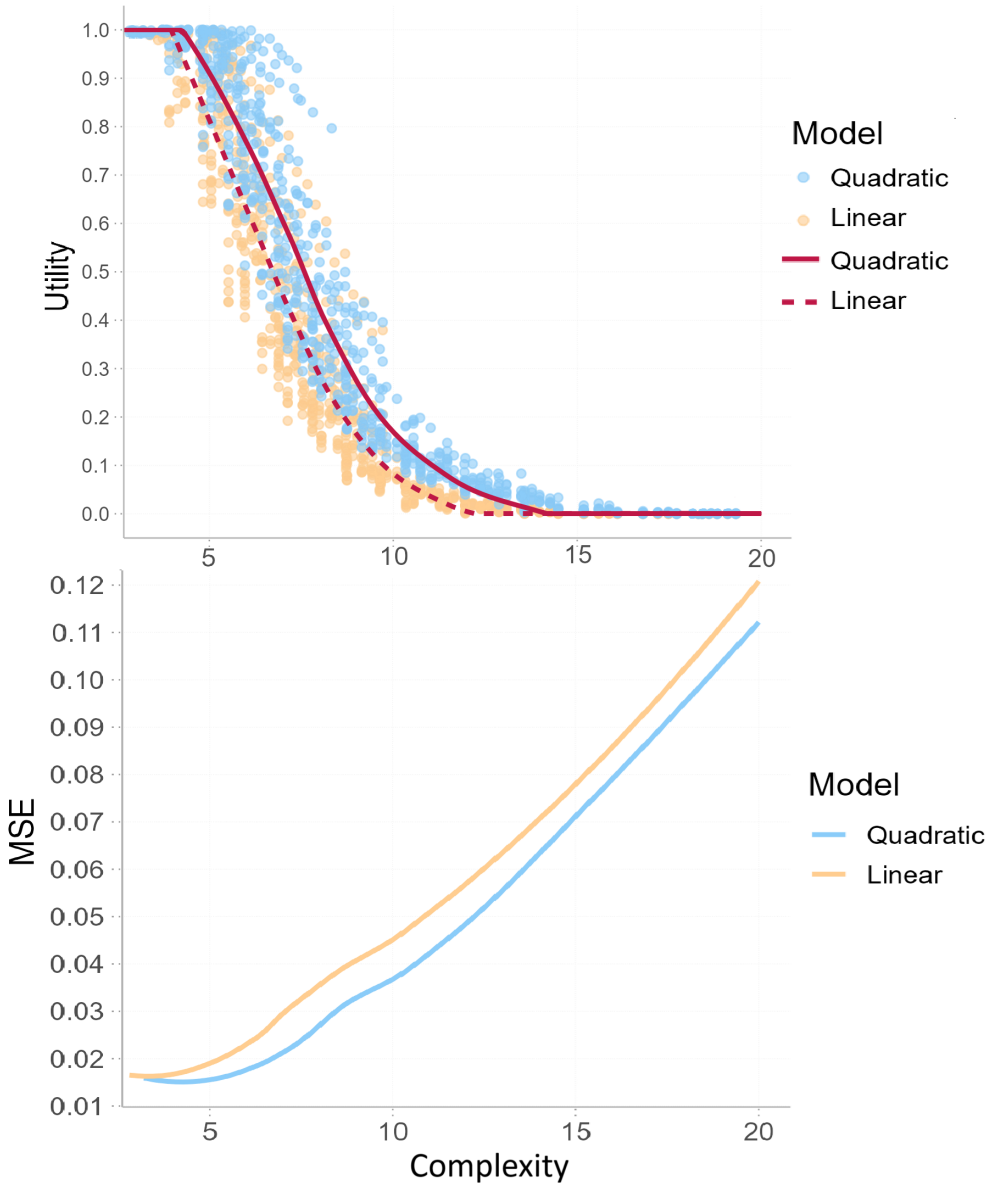


Figure 6.8: Final results showing the performance of the agent, both in terms of MSE and average utility, while using linear and quadratic approximations plotted against the complexity measure.

and 0.060 corresponds to finding bids with utility in the range 0.0–1.0, whereas for the linear model this relationship holds only for an MSE between 0.015 and 0.040. This is summarized in Table 6.1.

Table 6.1: Performance categories of agent using quadratic and linear surrogate models.

		Quadratic model		Linear model	
Performance	**Utility**	**MSE**	**Complexity**	**MSE**	**Complexity**
Good	$[0.7, 1.0]$	$[0.015, 0.018]$	$[2.5, 6.5]$	$[0.015, 0.020]$	$[2.5, 6.0]$
Mediocre	$[0.5, 0.7]$	$[0.018, 0.022]$	$[6.5, 7.5]$	$[0.020, 0.025]$	$[6.0, 7.0]$
Bad	$[0.2, 0.5]$	$[0.022, 0.033]$	$[8, 9.5]$	$[0.025, 0.037]$	$[7.0, 8.5]$
None	≈ 0	> 0.060	> 14.0	> 0.040	> 12.0

6.3. Summary

The agent, which used quadratic regression to model its hypercubic utility function, performed consistently better than when using linear regression, both when examining individual complexity parameters and in terms of the complexity measure derived in Section 3.1. Furthermore, the effects of the parameters on the agent's performance are consistent with the discussion of Section 3.1.

Overall, among the four parameters, γ and v show the strongest individual effects on the agent's performance, as well as the strongest cross-effects. The other parameters, m and d, appear to be more dependent on interactions with other parameters, showing minimal individual effects. The presence of cross-effects was observed between all pairs of parameters except γ and m, and v and d. In these cases, asymmetric dependencies were observed with the induced complexity of m depending on γ but not vice versa, and the induced complexity of d depending on v but not vice versa. This is summarized in Figure 6.9, where the effect of one parameter on another is measured as the absolute increase in the marginal rate of change for the corresponding pair.

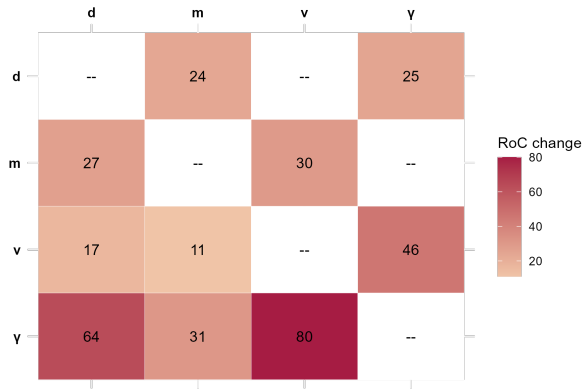


Figure 6.9: Summary of asymmetric dependencies between complexity parameters, expressed as the absolute size of the rate-of-change ranges. Each entry indicates the change induced in the column parameter by fixing the row parameter, quantifying how much the induced complexity of the column parameter depends on the value of the row parameter.

To conclude this chapter, the model's performance was plotted against the complexity measure presented in Section 3.1. In terms of this measure, the short answer to the research question is that the model does not scale very well with complexity. However, it does outperform the linear model on all accounts, which strongly suggests that a higher-order polynomial would outperform a quadratic one. For the quadratic model, an MSE between 0.015 and 0.060 corresponds to finding bids with utility in the range $0.0 - 1.0$, whereas for the linear model this relationship holds only for an MSE between 0.015 and 0.040. This is summarized in Table 6.1. To give a more intuitive idea of what the values of the *complexity* column of Table 6.1 actually represent, Table 6.2 provides examples of parameter settings that result in both models performing good, mediocre, bad, and not at all.

Table 6.2: Parameters set to the complexity ranges in Table 6.1

Performance	γ	v	m	d
Good	2	6	25	20
Mediocre	3	4	50	30
Bad	4	4	75	40
None	6	2	100	50

7

Discussion and Conclusion

The results presented in Chapter 6 demonstrate that searching for high-utility bids in surrogate models constructed from hypercubic utility functions offer a promising approach. When these surrogate models are quadratic polynomials, the agent can efficiently find medium to high-utility bids in functions of complexity $\mathcal{C} \in [0, 7.5]$ where $\mathcal{C}$ is the complexity measure from Definition 5.

QuadrApprox, the agent developed in this thesis, outperforms the top-performing agents from ANAC 2014 on the medium-complexity utility functions included in Genius — the automated negotiation simulator used for the experiments in this work (see Appendix A.1 for further details). Notably, QuadrApprox achieves this by a substantial margin, further highlighting the potential and effectiveness of the proposed surrogate-based approach.

In this chapter, the results presented in Chapter 6 are discussed from the viewpoint of the complexity parameters and how they affect the agent's performance. Thereafter, direct answers to the three research questions are provided along with a discussion on this work's limitation and potential avenues for future research.

7.1. On how the methods scale with complexity

7.1.1. The complexity of the agent

As utility functions become more complex, larger sample sizes are generally required to achieve accurate approximations. While the contribution of the domain's dimensionality — or the number of issues — alone to utility function complexity is relatively small, the computational complexity of quadratic regression models grows by a factor of $\mathcal{O}(d^2)$. To assess the practical applicability of linear and quadratic surrogate models, it is therefore important to analyze their computational cost and memory requirements alongside the sample sizes needed for reliable performance.

Despite the curse of dimensionality affecting the required sample size in high dimensions, linear and quadratic regression remain practical approaches within moderate dimensions such as those considered in this thesis. For example, the most complex utility function considered in this thesis exists in $d = 50$ dimensions and is defined by $m = 100$ constraints, each with widths $v = 2$ and containing $\gamma = 10$ issues. Let p denote the number of features in the design matrix $\Phi \in \mathbb{R}^{N \times p}$. For linear regression, $p = d$, whereas for quadratic regression, $p = \frac{d(d+1)}{2}$. According to Figure 5.1, a sample size of around $N = \frac{50 \cdot 50 \cdot 51}{2} = 63{,}750$ is required for the quadratic model to approximate this function. Performing linear or quadratic regression on up to tens of

thousands of samples is well within the capabilities of modern hardware. This can, however, be a limitation in an autonomous negotiation where there is a time limit.

Let $N = 63{,}750$ denote the number of sampled points and $p = \frac{d(d+1)}{2}$ denote the number of parameters in quadratic regression. Then, in $d = 50$ dimensions, the design matrix Φ is of size $63{,}750 \times 1{,}275$, which is about 25 times larger than what is required for linear regression when $d = 50$. Now, assuming 4-byte floats, storing Φ requires $N \times p \times 4$ bytes, or approximately

- 13 MB for linear regression, and
- 325 MB for quadratic regression.

When $d = 100$, however, this goes to approximately 5 GB for quadratic regression, and to approximately 50 TB when $d = 1{,}000$.

Once Φ is constructed, the maximum likelihood estimator is computed as

$$\hat{\mathbf{w}}_{ML} = (\mathbf{\Phi}^\top \mathbf{\Phi} + I)^{-1} \mathbf{\Phi}^\top \mathbf{y},$$

where forming $\mathbf{\Phi}^\top \mathbf{\Phi}$ requires $\mathcal{O}(Np^2)$ operations, and inverting the resulting $p \times p$ matrix requires $\mathcal{O}(p^3)$. For quadratic regression in $d = 50$ and with $N = 63{,}750$, this totals around $\mathcal{O}(10^{11})$ operations, which translates to $\mathcal{O}(4 \times 10^9)$ in the linear case. Both are certainly feasible for modern machines, but, as previously noted, can pose a limitation in an autonomous negotiation setting.

As mentioned in Section 3.4.2, the complexity and memory requirements from an L-BFGS-B iteration are both of the order $\mathcal{O}(d)$. Furthermore, the method has been shown to converge superlinearly, making one initialization relatively cheap. Lastly, L-BFGS-B has been shown to be enjoy parallelism [64]. On a single machine, the speed benefits of L-BFGS come from using the approximated second-order information (modeling the interactions between variables) [49].

7.1.2. The complexity of the functions

The experiments clearly show that the complexity of hypercubic utility functions has a decisive impact on the performance of agents using quadratic surrogate models. Among the four parameters studied, the dimensionality γ emerged as the most dominant driver of complexity.

The experimental results indicate that the performance of an autonomous negotiation agent using quadratic surrogate models decreases sharply as the dimensionality of the constraints, γ, increases. For low values of γ, particularly $\gamma = 1$ and $\gamma = 2$, the quadratic model performs relatively well. This is expected, as $\gamma = 2$ corresponds directly to the maximum order of interdependencies that a quadratic model can express. However, as γ increases beyond 2, performance decreases sharply. By $\gamma = 10$, the quadratic approximation effectively collapses many-way interdependencies into only pairwise ones, creating a severe mismatch between the utility function's structure and the model's representational capacity. This finding is reflected both in the agent's ability to find high-utility bids and in the mean squared error (MSE) analysis.

In order to imitate this, Figure 7.1 gives the MSE of a quadratic regression over polynomials of degrees ranging from 1 through 10 — corresponding to the configurations of γ considered for this thesis.

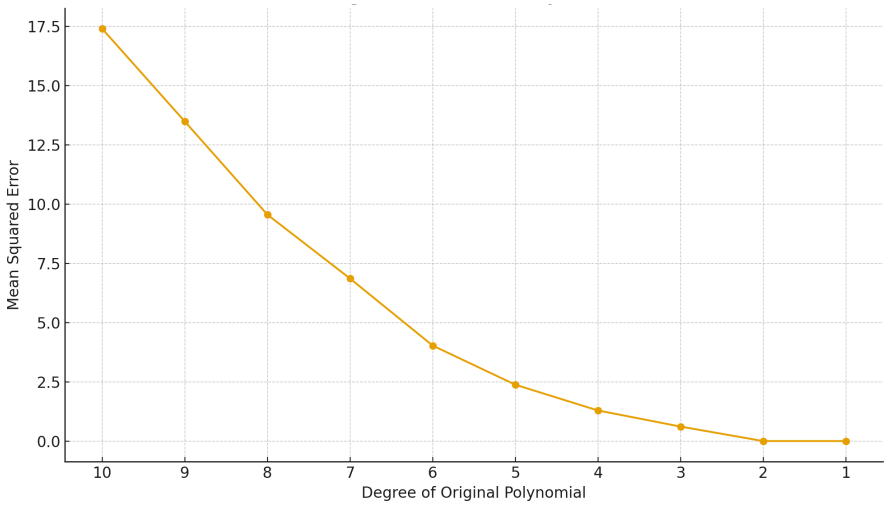


Figure 7.1: How the performance of a quadratic regression over varying degrees of polynomial scales, as measured by MSE.

As can be seen in the figure, the shape of the MSE is very similar to that of Figure 6.1. This consistency suggests that model performance is tightly coupled with the representational limitations imposed by dimensionality.

Constraint width v interacts strongly with γ. Narrow constraints produce highly irregular, "peaky" utility landscapes. By Theorem 2, the probability of a random sample satisfying a base constraint is $\left(\frac{v}{|I|}\right)^{\gamma}$ which in turn requires the agent to perform use very large sample sizes when γ is high and/or v is low. Assuming the sample size is large enough, the quadratic model still needs to capture this very wild, potentially high-dimensional, structure. The results mirror this: with $v = 2$ and $\gamma = 10$ and averaged over all other parameters, the agent finds bids of average utility ≈ 1, while this average is ≈ 0 when $v = 6$ and $\gamma = 1$.

By framing the hypercubic constraints in a probabilistic context and defining the complexity of a constraint as the probability that a uniform random sample satisfies it, Theorem 2 shows the clear interplay between v and γ and how they contribute to complexity both individually and through cross-effects. Further cross-effects were detected through empirical results. These were observed between d and m, between d and γ, and between m and v.

The effects from the number of issues d are minimal when viewing d in isolation but d contributes significantly to complexity through γ. As Corollary 3 shows, higher d increases the probability of type 1 intersections, where constraints combine to form a new one with their combined dimensionality. In practice, this means that large d values magnify the impact of both γ and m: more issues create more opportunities for constraints to intersect, that in all cases, as per Theorem 2, forms constraints that are more complex than those that make them. Figure 6.9 further emphasizes this interplay.

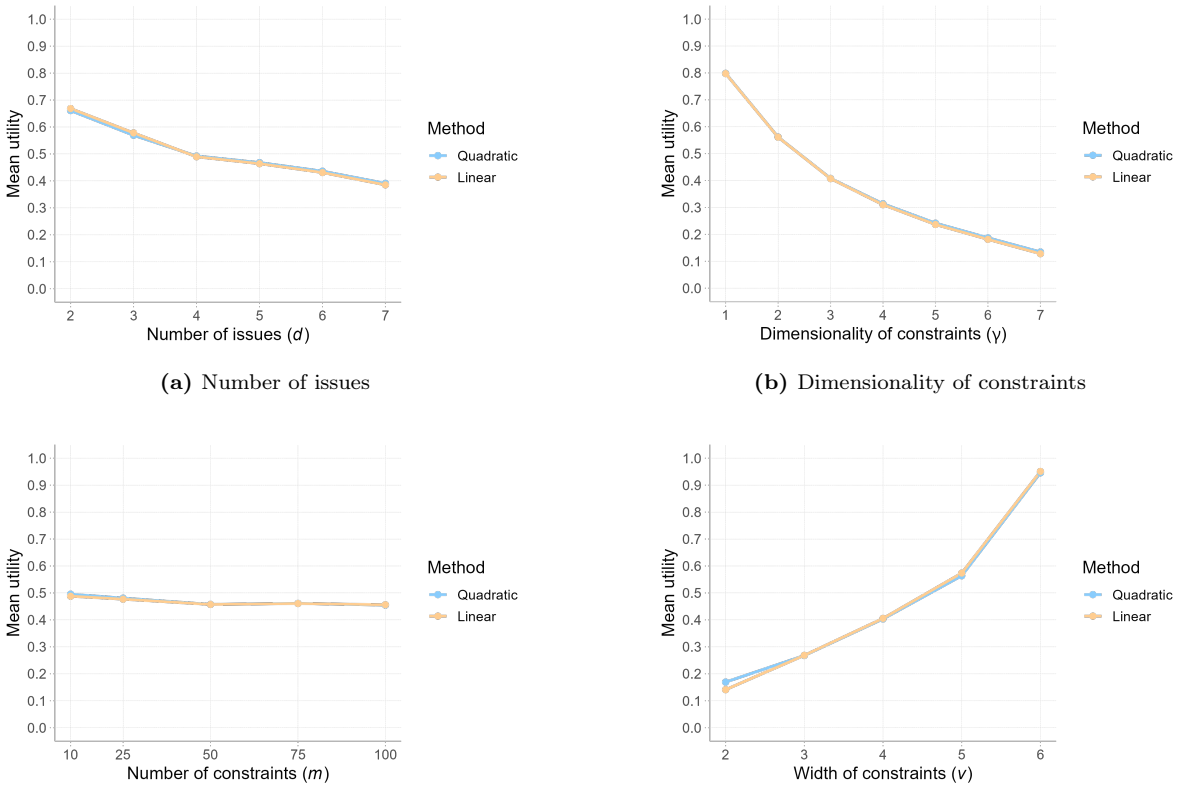
Notbly, the induced complexity of m depends relatively highly on the value of γ while γ is somewhat independent of m. The number of constraints m plays a subtler role. On its own, m does not induce as much complexity as γ or v, but its effects compound when intersections are considered. More constraints increase the likelihood of intersection constraints, many of which inherit higher dimensionalities and narrower widths. These intersections often define the global maximum of the utility function, making the function more rugged and challenging for quadratic surrogates to capture.

Taken together, these results highlight a layered structure of complexity. γ is the primary source, with v acting as its closest partner through strong cross-effects. m and d exert their influence by fueling intersections, which in turn amplify the effects of γ and v. This hierarchy explains why surrogate performance deteriorates most dramatically under combinations of high γ and low v, or when large m and d values provide fertile ground for intersection-driven complexity.

7.2. Why does Quadratic outperform Linear?

The agent consistently performs better in all experiments when using the quadratic model than when using the linear one. As the quadratic model contains the linear model in the b and w_0 terms, it should theoretically perform at least as good as the linear. If it does not, there is a high likelihood that the quadratic model has been overfit, as its expressiveness, or capacity, is higher than that of the linear one.

The main difference between the quadratic and the linear model is the quadratic's ability to capture optima that occur inside the cube. When it does so, the A matrix becomes negative definite, indicating that the surrogate model is concave. When this happens, the agent can find the surrogate model's global maximum by its closed form solution $-\frac{1}{2}A^{-1}b$. To verify this claim, a second round of experiments was run where, whenever Function 6 returns `True`, the cube is skipped. That is, when the quadratic model is concave, the agent simply discards the current cube and moves on to the next one. This was done in smaller domains as domain size should not affect this behavior. Figures in 7.2 depict the agent's performance.



(a) Number of issues

(b) Dimensionality of constraints

(c) Number of constraints

(d) Width of constraints

Figure 7.2: Performance of the agent using linear and quadratic model without allowing the quadratic model to be concave.

As can be seen in these figures, the two models perform just about exactly the same. This absolutely highlights the added value of going from the linear model to the quadratic model. This also strongly suggests that a higher-order polynomial would outperform the quadratic one in modeling the utility function. However, by Bezout's theorem (Theorem 1), a higher-order polynomial will have multiple local optima which could pose some limitations.

7.3. Conclusion

This research sought to answer the following research question:

Main Research Question:

How does the efficiency of an autonomous negotiation agent in finding high-utility bids using quadratic surrogate models of uncertain hypercubic constraint-based utility functions scale with increasing utility function complexity, and how does it compare to using linear surrogate models?

To answer this question, the complexity of a hypercubic utility function was quantified and the effects on the performance of both the quadratic and linear surrogates was experimented with over several utility functions, each of a particular level of complexity. However, before that could be done, several considerations needed to be made, and those were formed as sub-questions. Those are the subjects of the upcoming subsections, and the main research question will then be answered.

7.3.1. Research Subquestion 1

To answer the main Research Question, a way to measure the complexity of hypercubic constraint-based utility functions needed to be established. This sparked the following Subquestion:

Research Subquestion 1:

How can the complexity of a hypercubic constraint-based utility functions be measured?

Previous efforts in quantifying the complexity of utility functions had been done by Hadfi and Ito where they did so using entropy. Their approach, however, does not capture all relevant parameters for hypercubic constraints.

This thesis defines four parameters for measuring complexity:

- m: The number of constraints that define the function
- γ: The dimensionalities of the constraints
- v: The widths of the constraints
- d: The number of issues in the domain

Extending Hadfi's and Ito's idea of using entropy, this thesis measures complexity as the model's need for expressiveness to accurately capture the utility function. To do that, the probability of a uniform random sample satisfying a base constraint in a uniformly generated hypercubic function is derived to be

$$\left(\frac{v}{|I|}\right)^{\gamma}$$

Then, accounting for all possible intersection constraints and their probability of occurring,

complexity of a uniformly generated hypercubic function is defined as

$$-\log\left[\frac{1}{m}\left(\frac{v}{|I|}\right)^{\gamma}+\frac{1}{K}A\right]$$

where A is described in Definition 5. The intuition behind this is provided as a paragraph in Section 3.1.5.

7.3.2. Research Subquestion 2

After defining a measure to compare results, deciding how to construct and use a quadratic polynomial as a surrogate needed to be established. That is, the first question that needed to be answered was:

Research Subquestion 2:

How can quadratic polynomials be effectively constructed and used as surrogates for uncertain hypercubic constraint-based utility functions in negotiations?

As the analytical form of the utility functions was considered unavailable — hence the "uncertain" part of the research question — it was clear that this would be a sampling based approach. Thus, this question can be broken down into three parts:

Research Subquestion 2:

1. How can the sampling procedure aid the function approximation and in finding high utility bids?
2. How can a quadratic polynomial be constructed from samples of a hypercubic function
3. How can a global maximum of a quadratic polynomial in a closed high dimensional hypercube searched for.

Sampling

The sampling procedure needed to address two main problems: the difference in expressiveness of the quadratic polynomial to be constructed from the samples, and that of the utility function; and the curse of dimensionality.

The hypercubic function can be extremely wild, while a quadratic polynomial is rather calm in comparison. Indeed, the hypercubic function can exhibit multiple local optima and exist over multiple (and various) dimensions, while the quadratic polynomial is unimodal. Thus, the polynomial can not be defined over the entire domain. To address this issue, sampling should to be performed over subregions of the domain, and those should be defined as cubes. A local quadratic polynomial should then be constructed within each cube. Furthermore, by performing multiple local approximations, parallelism is trivial.

To further aid in the function approximation, the sampling should be performed using low-discrepancy sampling schemes. The particular one used in this thesis were Sobol samples. Sobol samples have been shown to stabilize model performance at a lower sample size than uniform samples, and they are deterministic which both ensures consistency in the approximation, and can be exploited by shifting the cubes being sampled from. Shifting the cube can save considerable time which can be valuable in autonomous negotiations.

To derive the optimal sample size, the MSE can be measured for various sample sizes over various complexity levels and for each one the sample size yielding the best results can be compared to the dimension of the corresponding domain. The dimension determines the number of parameters needed in the approximation method, and the optimal (or a good) sample size can be chosen as a parameter multiple which can then be used for any domain size. In this thesis, the best results were defined in terms of the sample size that were within a 5% range of the sample size that minimized the absolute difference between generalization and training error.

Due to the curse of dimensionality, the entire domain cannot be split into cubes which can then all be used for sampling. A $[0, 9]^{50}$ domain — a domain size considered for this thesis — can fit approximately 10^{26} cubes with the side lengths used in this thesis, or $\ell = 2$. For this reason, these cubes needed to be strategically placed. The literature contains various ways of directing a search for a "good bid" in autonomous negotiations. The method that was adopted in this thesis was to place the cubes both at random and at a close distance to where previous high utility contracts had been found, with distance defined in terms of the Manhattan distance. To balance between exploration and exploitation, an ϵ-greedy strategy was employed.

Regression

Quadratic polynomials can be constructed from samples of a hypercubic function using regression models with quadratic basis functions. These models can be linear in the parameters, taking the form

$$\hat{u}(\mathbf{x}_i) = w_0 + \sum_{j=1}^{m-1} w_j \phi_j(\mathbf{x}_i) \tag{7.1}$$

where the ϕ_j are the basis functions, corresponding to all combinations of d variables raised to powers that sum to at most 2, and each w_j corresponds to a weight.

To derive the parameters of Expression (7.1), minimizing the least-squares error function yields a closed form solution:

$$\hat{\mathbf{w}}_{ML} = (\mathbf{\Phi}^T \mathbf{\Phi} + \lambda I)^{-1} \mathbf{\Phi}^T \mathbf{y} = \mathbf{\Phi}^\dagger \mathbf{y}.$$

where $\mathbf{\Phi}$ is the design matrix, constructed by setting $\mathbf{\Phi}_{ij} = \phi_{ij}(\mathbf{x}_{ij})$, and λ is a regularization parameter. Then for any contract $\mathbf{x}$, it's value in the surrogate model is $\mathbf{x}\hat{\mathbf{w}}_{ML}$ where $\mathbf{x}$ is considered a row vector and $\hat{\mathbf{w}}_{ML}$ a column vector.

Optimization

Finally, a method for searching the quadratic surrogate model had to be established. Maximizing a non-concave quadratic function inside a cube is referred to as a bound-constrained global optimization (BCGO). Since no known polynomial-time $(1 - \epsilon)$ approximation algorithm exists for this class of non-convex optimization problems, second-order methods were explored.

Second-order methods incorporate second-order derivatives of the objective function. A particular class of second-order methods represents algorithms that are among the most sophisticated analytically for solving unconstrained optimization problems, with some of them also being applied to constrained optimization problems, including BCGO problems. These are the quasi-Newton methods. The most popular one, and the one used in this thesis, is the Limited-memory Broyden-Fletcher-Goldfarb-Shanno algorithm with Box constraints (L-BFGS-B). As detailed in Section 3.4.2, L-BFGS-B is a scalable, memory-efficient variant of quasi-Newton methods adapted to handle BCGO problems. It offers a complexity and memory footprint of $\mathcal{O}(d)$ where d is the dimensionality of the domain, while also demonstrating superlinear convergence rates for smooth functions. This makes each local initialization relatively inexpensive in high dimensions while still highly effective.

7.3.3. Main Research Question

Table 6.1 depicts values for the complexity measure $\mathcal{C}$ derived in this thesis, which correspond to when the agent's efficiency in finding high-utility bids is good, mediocre, bad, and none at all. Furthermore, Figure 6.8 illustrates this visually by showing the average utility of the bids that the agent could find.

An agent following the steps outlined in this thesis will perform very well with low complexity functions, but this performance rapidly decays through the medium complexity functions, and finally reaches an average utility of 0 for complexities above $\mathcal{C} = 14$. The agent using the quadratic model can be seen performing consistently better than when using the linear model over the entire complexity axis (excluding the part where both models average a utility of 0 or 1).

The overall aim of the Automated Negotiating Agent Competition (ANAC) is to advance the state-of-the-art in the area of autonomous negotiations, with an emphasis on the development of successful automated negotiators in realistic environments with incomplete information. In 2014, agents competed in similar scenarios as those considered in this thesis. When the method developed in this thesis was implemented in an autonomous negotiation agent, named **QuadrApprox**, the agent outperformed the top-performing ANAC 14 competitors in terms of individual utility, and by "winning" in a vast majority of the times. This is illustrated in Appendix C.2

7.4. Limitations and future work

7.4.1. Model expressiveness and tractability

This work focuses on modeling hypercubic functions with quadratic polynomials, using regression models that are linear in the parameters but with quadratic basis functions. While quadratic basis functions were the focus of this thesis, many alternatives could be explored.

Because the surrogate models are locally quadratic, they cannot capture highly disjoint, non-smooth, or high-dimensional utility surfaces. Nevertheless, quadratic basis functions have been shown here to outperform linear ones, suggesting that higher-order polynomials or other nonlinear basis functions may yield further improvements.

Future work should therefore investigate alternative basis functions. The framework developed in this thesis can readily accommodate such extensions, provided that the optimization procedure is adapted to the particular basis function used. Furthermore, the tractability of the model must be kept in mind. This depends highly on the size of the design matrix which has as many columns as there are basis functions. Although the quadratic model is fairly tractable, it can be slow when implemented on a run of the mill computer on cpu, which might be limiting in an autonomous negotiation.

7.4.2. Negotiation scenarios

This thesis is limited to hypercubic constraint-based utility functions and integer domains with equal-length issue domains, where each base constraint has the same structure. In these functions, each constraint is defined as a Cartesian product of intervals over a subset of issues, forming an axis-aligned hyperrectangle (or "hypercube") in the discrete space.

However, constraint-based utility functions can be constructed with constraints taking almost any shape. Although the hypercubic ones are most prominent in the literature reviewed for this thesis, other common ones are *Bell constraints [52]* and *Cone-shaped constraints [25]*. These

are defined in a way that the proximity to the center of the constraint determines the utility, which may provide a more natural representation in continuous domains. The hypercubic representation therefore constitutes a simplification which falls short of representing the general case of constraint-based utility modeling. Extending this work to broader classes of constraint shapes and domains is an important avenue for future research, both to better reflect real-world scenarios and to test the robustness of the surrogate-based appraoch.

The assumptions of equal-sized issue domains and structurally identical base constraints were introduced to reduce variability in experiments. Relaxing these assumptions can be done while the discussion of Section 3.1 remains relevant, but adapting the framework to non-hypercubic shapes would require redefining the parameters and revisiting that discussion. For example, extending the framework to domains with varying issue sizes or heterogeneous base constraints can be done using Corollary 1 which in turn allows the complexity measure to be redefined.

7.4.3. Complexity measure

The complexity measure can be reconstructed where A is derived through the probability of a particular constraint c_k existing in the function, as opposed to the probability that two constraints intersect to create c_k. This would allow each attainable intersection constraint to be weighted directly by its likelihood of being present in the function, rather than by tracing its lineage through successive intersections. Although this might not make the measure more accurate, it would certainly be a cleaner, more complete approach. This would require the following theorem to be finished and proved:

> *A utility function $u = (\gamma, v, n, m)$ that is uniformly constructed contains an intersection constraint $c_k = (\gamma_k, v_k)$ with probability...*

7.4.4. Sampling

The curse of dimensionality causes the negotiation domain to grow exponentially with the number of issues, which makes strategic sampling essential. In local approximation methods such as the one used in this thesis, the placement of these samples becomes crucial. Although exploration is necessary, the exploitation strategy used here could be improved.

In this thesis, the agent used a Manhattan-distance heuristic to place its samples at regions it deemed likely to contain high-utility bids. Although it has been noted in the literature that this might be the more appropriate way to identify high utility regions as the negotiation is not repeated, more advanced measures, or even learning-based sampling strategies, could yield better performance.

The literature on pattern recognition and active learning offers many possible extensions. Implementing to this thesis's method an advanced learning technique would undoubtedly make the agent's performance a bit better. For example, a Bayesian approach could be adopted in which the agent maintains a prior over the utility function and updates it with each new sample or round of negotiation.

7.5. Contributions

7.5.1. Basis for future work

This thesis identifies a research gap; mainly, searching a surrogate of the utility function in autonomous negotiations as opposed to the utility function itself being an uncommon approach which, as shown in this thesis, has a great potential. This research contributes to the literature

by laying the ground for filling this gap, providing a mathematical background and a method that are suitable for linear-in-parameters regression models, and the considerations needed to implement them. Thus, implementing this thesis's method with any basis function can be done by following the steps it lays out, with a careful consideration to the balance so often mentioned: between approximation accuracy and search complexity. This thesis represents an example of how this can be done.

7.5.2. Complexity measure

By following the steps of this thesis, various combinations of basis functions and optimization algorithms can be experimented with. To compare the results between such experiments, this thesis suggests the following measure to quantify a complexity of hypercubic constraint-based utility functions.

$$\mathcal{C} = -\log\left[\frac{1}{m}\left(\frac{v}{|I|}\right)^{\gamma} + \frac{1}{K}A\right]$$

where A is defined in Definition 5. Experiments can then be conducted over various configurations of the following parameters

- m: The number of constraints that define the function
- γ: The dimensionalities of the constraints
- v: The widths of the constraints
- d: The number of issues in the domain

When performance is then viewed against this measure, two approaches — such as was done with the linear and quadratic basis functions in this thesis — can be compared.

References

[1] Charu C Aggarwal, Lagerstrom-Fife Aggarwal, and Lagerstrom-Fife. *Linear algebra and optimization for machine learning*. Vol. 156. Springer, 2020.

[2] Bo An et al. "Automated negotiation with decommitment for dynamic resource allocation in cloud computing". In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*. 2010, pp. 981–988.

[3] Emanouil Atanassov and Sofiya Ivanovska. "On the use of Sobol'sequence for high dimensional simulation". In: *International Conference on Computational Science*. Springer. 2022, pp. 646–652.

[4] Reyhan Aydoğan et al. "A Baseline for Nonlinear Bilateral Negotiations: The full results of the agents competing in ANAC 2014". In: *Intelligent Computational Systems: A Multi-Disciplinary Perspective*. Bentham Science Publishers, 2017, pp. 93–121.

[5] Reyhan Aydoğan et al. "A machine learning approach for mechanism selection in complex negotiations". In: *Journal of Systems Science and Systems Engineering* 27.2 (2018), pp. 134–155.

[6] Reyhan Aydoğan et al. "Alternating offers protocols for multilateral negotiation". In: *Modern approaches to agent-based complex automated negotiation*. Springer, 2017, pp. 153–167.

[7] Tim Baarslag et al. "Decoupling negotiating agents to explore the space of negotiation strategies". In: *Novel insights in agent-based complex automated negotiation*. Springer, 2014, pp. 61–83.

[8] Tim Baarslag et al. "Learning about the opponent in automated bilateral negotiation: a comprehensive survey of opponent modeling techniques". In: *Autonomous Agents and Multi-Agent Systems* 30.5 (2016), pp. 849–898.

[9] Waqas Haider Bangyal et al. "Comparative analysis of low discrepancy sequence-based initialization approaches using population-based algorithms for solving the global optimization problems". In: *Applied Sciences* 11.16 (2021), p. 7591.

[10] Emilio Barucci, Claudio Fontana, et al. *Financial markets theory*. Springer, 2017.

[11] Tim Bedford and Roger Cooke. *Probabilistic risk analysis: foundations and methods*. Cambridge University Press, 2001.

[12] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[13] Andrew Byde et al. "AutONA: A system for automated multiple 1-1 negotiation". In: *Proceedings of the 4th ACM conference on Electronic commerce*. 2003, pp. 198–199.

[14] Richard H Byrd et al. "A limited memory algorithm for bound constrained optimization". In: *SIAM Journal on scientific computing* 16.5 (1995), pp. 1190–1208.

[15] Jianlong Cai, Jieyu Zhan, and Yuncheng Jiang. "CP-nets-based user preference learning in automated negotiation through completion and correction". In: *Knowledge and Information Systems* 65.9 (2023), pp. 3567–3590.

[16] Bradley J Clement and Anthony C Barrett. "Continual coordination through shared activities". In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. 2003, pp. 57–64.

[17] Dave De Jonge and Carles Sierra. "GANGSTER: an automated negotiator applying genetic algorithms". In: *Recent advances in agent-based complex automated negotiation*. Springer, 2016, pp. 225–234.

[18] Etienne De Klerk. "The complexity of optimizing over a simplex, hypercube or sphere: a short survey". In: *Central European Journal of Operations Research* 16.2 (2008), pp. 111–125.

[19] Eden Shalom Erez and Inon Zuckerman. "Dona—a domain-based negotiation agent". In: *Recent Advances in Agent-based Complex Automated Negotiation*. Springer, 2016, pp. 261–271.

[20] Roger Fisher and William L. Ury. *Getting to yes: Negotiating agreement without giving in*. Houghton Mifflin, 1981.

[21] Stephanie Fitchett. "Bezout's Theorem: A Taste of Algebraic Geometry". In: *Florida Atlantic University Honors College* ().

[22] Katsuhide Fujita and Takayuki Ito. "An analysis of computational complexity of the threshold adjusting mechanism in multi-issue negotiations". In: *International Transactions on Systems Science and Applications* 4.4 (2008), pp. 305–311.

[23] Katsuhide Fujita, Takayuki Ito, and Mark Klein. "AN APPROACH TO SCALABLE MULTI-ISSUE NEGOTIATION: DECOMPOSING THE CONTRACT SPACE". In: *Computational Intelligence* 30.1 (2014), pp. 30–47.

[24] Katsuhide Fujita, Takayuki Ito, and Mark Klein. "Preliminary result on secure protocols for multiple issue negotiation problems". In: *Pacific Rim International Conference on Multi-Agents*. Springer. 2008, pp. 161–172.

[25] Katsuhide Fujita, Takayuki Ito, and Mark Klein. "Secure and efficient protocols for multiple interdependent issues negotiation". In: *Journal of Intelligent & Fuzzy Systems* 21.3 (2010), pp. 175–185.

[26] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[27] Stephen J Gould. "Glow big glowworm". In: *Natural History* 95.12 (1986), pp. 10–16.

[28] Rafik Hadfi and Takayuki Ito. "Approximating Constraint-Based Utility Spaces Using Generalized Gaussian Mixture Models". In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer. 2014, pp. 133–140.

[29] Rafik Hadfi and Takayuki Ito. "Low-complexity exploration in utility hypergraphs". In: *Journal of information processing* 23.2 (2015), pp. 176–184.

[30] Rafik Hadfi and Takayuki Ito. "Modeling complex nonlinear utility spaces using utility hyper-graphs". In: *International Conference on Modeling Decisions for Artificial Intelligence*. Springer. 2014, pp. 14–25.

[31] Raiye Hailu and Takayuki Ito. "Efficient Deal Identification by Constraint Relaxation for Collaborative Decision Making Using Negotiation". In: *Journal of Advanced Computational Intelligence and Intelligent Informatics* 18.4 (2014), pp. 608–615.

[32] Shin Harase. "Comparison of Sobol'sequences in financial applications". In: *Monte Carlo Methods and Applications* 25.1 (2019), pp. 61–74.

[33] Hiromitsu Hattori, Mark Klein, and Takayuki Ito. "Using iterative narrowing to enable multi-party negotiations with multiple interdependent issues". In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. 2007, pp. 1–3.

[34] Koen Hindriks, Catholijn Jonker, and Dmytro Tykhonov. "Avoiding approximation errors in multi-issue negotiation with issue dependencies". In: *Proc. of The 1st International Workshop on Agent-based Complex Automated Negotiations (ACAN-2008)*. 2008.

[35] Koen Hindriks, Catholijn M Jonker, and Dmytro Tykhonov. "Eliminating interdependencies between issues for multi-issue negotiation". In: *International Workshop on Cooperative Information Agents*. Springer. 2006, pp. 301–316.

[36] Takayuki Ito. "A multiple-nash-sa mediator for huge design utility spaces in automated negotiations". In: *2024 IEEE International Conference on Agents (ICA)*. IEEE. 2024, pp. 19–24.

[37] Takayuki Ito, Hiromitsu Hattori, and Mark Klein. "Multi-issue Negotiation Protocol for Agents: Exploring Nonlinear Utility Spaces." In: *IJCAI*. Vol. 7. 2007, pp. 1347–1352.

[38] Nicholas R Jennings et al. "Automated negotiation: prospects, methods and challenges". In: *International Journal of Group Decision and Negotiation* 10.2 (2001), pp. 199–215.

[39] Stephen Joe and Frances Y Kuo. "Constructing Sobol sequences with better two-dimensional projections". In: *SIAM Journal on Scientific Computing* 30.5 (2008), pp. 2635–2654.

[40] Catholijn Jonker et al. "Automated negotiating agents competition (ANAC)". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[41] Catholijn M Jonker, Valentin Robu, and Jan Treur. "An agent architecture for multi-attribute negotiation using incomplete preference information". In: *Autonomous Agents and Multi-Agent Systems* 15.2 (2007), pp. 221–252.

[42] Daniel Kahneman. *Thinking, Fast and Slow*. macmillan, 2011.

[43] Ralph L Keeney and Howard Raiffa. *Decisions with multiple objectives: preferences and value trade-offs*. Cambridge university press, 1993.

[44] Mark Klein et al. "Negotiating complex contracts". In: *Group Decision and Negotiation* 12.2 (2003), pp. 111–125.

[45] Thimjo Koça, Dave De Jonge, and Tim Baarslag. "Search algorithms for automated negotiation in large domains". In: *Annals of Mathematics and Artificial Intelligence* 92.4 (2024), pp. 903–924.

[46] Ladislav Kocis and William J Whiten. "Computational investigations of low-discrepancy sequences". In: *ACM Transactions on Mathematical Software (TOMS)* 23.2 (1997), pp. 266–294.

[47] Thomas Kollig and Alexander Keller. "Efficient multidimensional sampling". In: *Computer Graphics Forum*. Vol. 21. 3. Wiley Online Library. 2002, pp. 557–563.

[48] Mario Köppen. "The curse of dimensionality". In: *5th online world conference on soft computing in industrial applications (WSC5)*. Vol. 1. 2000, pp. 4–8.

[49] Quoc V Le et al. "On optimization methods for deep learning". In: *Proceedings of the 28th international conference on international conference on machine learning*. 2011, pp. 265–272.

[50] Zongcan Li, Rafik Hadfi, and Takayuki Ito. "Agenda-Based Automated Negotiation Through Utility Decomposition". In: *International Joint Conference on Artificial Intelligence*. Springer. 2022, pp. 119–135.

[51] Raz Lin et al. “Genius: An integrated environment for supporting the design of generic automated negotiators”. In: *Computational Intelligence* 30.1 (2014), pp. 48–70.

[52] Miguel A Lopez-Carmona et al. “Negoexplorer: A region-based recursive approach to bilateral multi-attribute negotiation”. In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer. 2009, pp. 261–275.

[53] David G Luenberger, Yinyu Ye, et al. *Linear and nonlinear programming*. Vol. 2. Springer, 1984.

[54] Ivan Marsa-Maestre et al. “Effective bidding and deal identification for negotiations in highly nonlinear scenarios”. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. 2009, pp. 1057–1064.

[55] Ivan Marsa-Maestre et al. “From problems to protocols: Towards a negotiation handbook”. In: *Decision Support Systems* 60 (2014), pp. 39–54.

[56] Siddhartha Mishra and T Konstantin Rusch. “Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences”. In: *SIAM Journal on Numerical Analysis* 59.3 (2021), pp. 1811–1834.

[57] Yasser Mohammad et al. “Supply chain management world: a benchmark environment for situated negotiations”. In: *International conference on principles and practice of multi-agent systems*. Springer. 2019, pp. 153–169.

[58] Kevin P Murphy. *Probabilistic machine learning: an introduction*. MIT press, 2022.

[59] Harald Niederreiter. “Point sets and sequences with small discrepancy”. In: *Monatshefte für Mathematik* 104.4 (1987), pp. 273–337.

[60] Harald Niederreiter. *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992.

[61] Makoto Niimi and Takayuki Ito. “Agentm”. In: *Recent advances in agent-based complex automated negotiation*. Springer, 2016, pp. 235–240.

[62] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.

[63] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. MIT Press, 2023.

[64] Rajat Raina, Anand Madhavan, and Andrew Y Ng. “Large-scale deep unsupervised learning using graphics processors”. In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 873–880.

[65] Valentin Robu, D. J. A. Somefun, and Johannes A. La Poutré. “Modeling complex multi-issue negotiations using utility graphs”. In: *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. (2005).

[66] S. Russell and P. Norvig. *Artificial intelligence: a modern approach*. pearson, 2016.

[67] Motoki Sato and Takayuki Ito. “Whaleagent: Hardheaded strategy and conceder strategy based on the heuristics”. In: *Recent advances in agent-based complex automated negotiation*. Springer, 2016, pp. 273–278.

[68] Adi Shamir. “How to share a secret”. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[69] Ilya M Sobol. “Distribution of points in a cube and approximate evaluation of integrals”. In: *USSR Computational mathematics and mathematical physics* 7 (1967), pp. 86–112.

[70] Bálint Szöllősi-Nagy, David Festen, and Marta M Skarżyńska. "A greedy coordinate descent algorithm for high-dimensional nonlinear negotiation". In: *Recent Advances in Agent-based Complex Automated Negotiation*. Springer, 2016, pp. 249–260.

[71] Leigh Thompson. *The Mind and Heart of the*. Prentice-Hall, 2004.

[72] Wietske Visser et al. "A Framework for Qualitative Multi-criteria Preferences." In: *ICAART (1)*. 2012, pp. 243–248.

[73] Wietske Visser et al. "A Framework for Qualitative Multi-criteria Preferences." In: *ICAART (1)*. 2012, pp. 243–248.

[74] John Von Neumann and Oskar Morgenstern. "Theory of games and economic behavior: 60th anniversary commemorative edition". In: *Theory of games and economic behavior*. Princeton university press, 2007.

[75] Chris Voss and Tahl Raz. *Never split the difference: Negotiating as if your life depended on it*. Random House, 2016.

[76] Chenxi Wu et al. "A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 403 (2023), p. 115671.

[77] Farhad Zafari and Faria Nassiri-Mofakham. "Bravecat: iterative deepening distance-based opponent modeling and hybrid bidding in nonlinear ultra large bilateral multi issue negotiation domains". In: *Recent Advances in Agent-based Complex Automated Negotiation*. Springer, 2016, pp. 285–293.

[78] Xiaoqin Shelley Zhang and Mark Klein. "Hierarchical negotiation model for complex problems with large-number of interdependent issues". In: *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. Vol. 2. IEEE. 2012, pp. 126–133.

[79] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Transactions on mathematical software (TOMS)* 23.4 (1997), pp. 550–560.

A

Additional information

A.1. Genius

To simulate and analyze negotiations for this thesis, Genius [51] was used. Genius stands for **G***eneral* **E***nvironment for* **N***egotiation with* **I***ntelligent multipurpose* **U***sage* **S***imulation*, and is the most commonly used automated negotiation simulator available today. The majority of the literature referenced in this thesis regarding automated negotiation was developed and tested in Genius.

Genius is a flexible and easy-to-use environment, built with the aim of facilitating the design of negotiation strategies [51]. To achieve that, negotiations can be simulated using built-in negotiating agents and scenarios, as well as offering an easy integration of new agents and scenarios by means of a graphical user interface.

After a negotiation has been simulated, a variety of tools to analyze the dynamics and outcome of the negotiation, as well as the performance of agents, can be accessed in Genius's analytical toolbox. These include optimal solutions such as the Pareto frontier and the Nash product. Additionally, simulation control and logging components allow users to control and debug the simulations, as well as to obtain the negotiation thread.

Researchers can use the built-in features of Genius to empirically and objectively compare their agent with others in various negotiation scenarios, making Genius an important contribution to the field. Furthermore, the Automated Negotiating Agent Competition (ANAC) has been conducted in Genius since it was first held in 2010 [40]. ANAC is an annual event that brings together researchers from the negotiation community and provides unique benchmarks for evaluating their negotiation strategies, essentially defining the state-of-the-art. For instance, ANAC 14 had nonlinear preferences and large domains [4], ANAC 15 had multi-party negotiations, ANAC 16 had negotiation in smart energy grids, and ANAC 17 had repeated negotiations. Several of the ANAC agents and scenarios are built into Genius, thereof all of the ANAC 14 agents and scenarios, which made it possible to compare the resulting agent from this thesis to those. Results are presented in Appendix C.1.

B

Proofs

B.1. proof of Lemma 2

Proof: Clearly $\mathbb{P}(X = n) = 0$ when $n < 0$ or $n > \min\{\gamma_1, \gamma_2\}$. If $\gamma_1 + \gamma_2 > d$ then the two constraints must be defined by at least $\gamma_1 + \gamma_2 - d$ common issues as $\gamma_1 + \gamma_2 > d \wedge |\Gamma_1 \cap \Gamma_2| = 0$ cannot hold. This establishes the bounds.

Now, fix Γ_1 and view Γ_2 as a uniformly random γ_2-subset of $[d]$, independent of Γ_1. Interpreting the d ground elements as a population with "success" elements precisely those in Γ_1, there are γ_1 successes and $d - \gamma_1$ failures in the population. Drawing γ_2 elements uniformly without replacement (i.e., choosing Γ_2) yields a hypergeometric sampling model. Thus the number of successes observed, which is exactly $X = |\Gamma_1 \cap \Gamma_2|$, has the hypergeometric probability mass function:

$$p_1 = \mathbb{P}(|\Gamma_1 \cap \Gamma_2| = n \,|\, \Gamma_1) = \frac{\binom{\gamma_1}{n}\binom{d-\gamma_1}{\gamma_2-n}}{\binom{d}{\gamma_2}},$$

for all n in the feasible range $\max\{0, \gamma_1 + \gamma_2 - d\} \leq n \leq \min\{\gamma_1, \gamma_2\}$, and zero otherwise.

As any realization of Γ_1 is equally likely, this conditional probability depends only on the sizes (d, γ_1, γ_2), not on the particular realization of Γ_1. Therefore, by the law of total probability, the following holds:

$$\mathbb{P}(|\Gamma_1 \cap \Gamma_2| = n) = \sum_s p_1 \mathbb{P}(\Gamma_1 = s) = p_1 \sum_s \mathbb{P}(\Gamma_1 = s) = p_1$$

By fixing Γ_2, the same steps yield the same results due to the symmetry of the hypergeometric distribution. □

B.2. Proof of Lemma 3.(2)

Proof: As was established in the proof of (1), the total number of posible placements is

$$T = (|I| - v_1 + 1)(|I| - v_2 + 1).$$

Denote the offset between intervals by $\delta = x_2^{\min} - x_1^{\min}$. If $\delta \geq 0$, then $x_2^{\min} = x_1^{\min} + \delta$ and the intersection length J is

$$J = \max\big(0, \min(v_1 - \delta, v_2)\big).$$

Thus $J = v_3$ if and only if one of the following holds:

1. $v_3 < v_2$ and $v_1 - \delta = v_3$, in which case $\delta = v_1 - v_3$;
2. or $v_3 = v_2$ and $d \leq v_1 - v_2$.

Now, for a fixed offset δ, the start point $x_1^{\min}$ must satisfy both $x_1^{\min} \leq |I| - v_1$ and $x_1^{\min} + \delta \leq |I| - v_2$, so the number of valid placements for that offset is

$$N(\delta) = \max\big(0, \min(|I| - v_1, |I| - v_2 - \delta) + 1\big).$$

For $v_3 < \min(v_1, v_2)$, only $\delta = v_1 - v_3$ yields overlap length v_3. There is also a symmetric case $\delta < 0$ with $\delta = -(v_1 - v_3)$, doubling the count, hence the factor 2.

For $v_3 = v_2 \leq v_1$, all offsets $0 \leq d \leq v_1 - v_2$ produce overlap length v_3, counted by summing $N(\delta)$ without the factor 2.

Dividing by the total number of placements T gives (**??**). □

B.3. Proof of Theorem 4

Proof: Since each constraint is placed independently and uniformly at random, the size of their intersection, $|\Gamma_1 \cap \Gamma_2|$, is a hypergeometric random variable, as per Lemma 1. Now, for each possible intersection size $|\Gamma_1 \cap \Gamma_2| \in [0, \min(\gamma_1, \gamma_2)]$, there are $\binom{d}{\gamma_3}$ distinct subsets $S \subset [d]$ of size γ_3 which could represent the actual overlapping dimensions. Assuming uniform random selection of Γ_1, the joint probability of choosing a specific subset S as the intersection is uniformly weighted against all such subsets, yielding

$$\mathbb{P}(\Gamma_1 \cap \Gamma_2 = S) = \frac{1}{\binom{d}{\gamma_3}} \cdot \frac{\binom{\gamma_1}{\gamma_3}\binom{d-\gamma_1}{\gamma_2-\gamma_3}}{\binom{d}{\gamma_2}}.$$

Next, for each dimension $j \in \Gamma_1 \cap \Gamma_2$, the two constraint intervals in that dimension are placed independently and uniformly. Let these intervals be

$$[x_{1j}^{\min}, x_{1j}^{\min} + v_{1j}) \quad \text{and} \quad [x_{2j}^{\min}, x_{2j}^{\min} + v_{2j}).$$

By Lemma 2(2), the probability that two such intervals intersect to share v_{3j} elements is exactly Equation (3.6).

Since intersection across all shared dimensions is required for the constraints to intersect, and by the independence between these dimensions,

$$\begin{aligned}\mathbb{P}\left(\bigcap_{j \in \Gamma_1 \cap \Gamma_2} \left([x_{1j}^{\min}, x_{1j}^{\min} + v_{1j}) \cap [x_{2j}^{\min}, x_{2j}^{\min} + v_{2j}) = v_{3j}\right)\right) \\ = \prod_{j \in \Gamma_1 \cap \Gamma_2} \mathbb{P}\left([x_{1j}^{\min}, x_{1j}^{\min} + v_{1j}) \cap [x_{2j}^{\min}, x_{2j}^{\min} + v_{2j}) = v_{3j}\right)\end{aligned}$$

Finally, the probability of the two constraints intersecting is the probability that $\Gamma_1 \cap \Gamma_2 = \Gamma_3$ multiplied by the probability that all intervals in $\Gamma_1 \cap \Gamma_2$ intersect, which is exactly Expression (3.5). □

B.4. Proof of Lemma 4

Proof: Write $L_1 = [x_1^{min}, x_1^{min}+v_1)$ and $L_2 = [x_2^{min}, x_2^{min}+v_2)$. Define the offset $\delta = x_2^{min} - x_1^{min}$. Since $x_1^{min} \in [0, |I| - v_1]$ and $x_2^{min} \in [0, |I| - v_2]$, the two intervals intersect if and only if

$$x_1^{min} < x_1^{min} + \delta + v_2 \quad \text{and} \quad x_1^{min} + \delta < x_1^{min} + v_1,$$

which is equivalent to $\delta \in (-v_1, v_2)$.

For fixed δ in this range, the number of valid positions x_1^{min} such that both L_1 and L_2 lie entirely within $[0, |I|)$ is

$$n(\delta) = \max\{0,\, \min(|I| - v_2, |I| - v_1 - \delta) - \max(0, -\delta) + 1\}\,.$$

Summing over all admissible δ gives the total number of intersecting configurations:

$$N_{\text{int}} = \sum_{\delta=-v_1+1}^{v_2-1} n(\delta).$$

Now define an "escape" configuration as one where L_1 and L_2 intersect, but shifting L_1 by $+1$ or -1 results in no intersection. For fixed δ, this happens precisely when

$$[x_1^{min} \pm 1, x_1^{min} + v_1 \pm 1) \cap [x_1^{min} + \delta, x_1^{min} + \delta + v_2) = \emptyset.$$

Now, encode this by the indicator $\mathbf{1}_{\text{escape}}(\delta)$ defined in the lemma. The number of such escape configurations is then

$$N_{\text{end}} = \sum_{\delta=-v_1+1}^{v_2-1} \mathbf{1}_{\text{escape}}(\delta)\, n(\delta).$$

Since all valid placements (x_1^{min}, x_2^{min}) are equally likely, the desired conditional probability is the ratio of the number of escape configurations to the number of intersecting configurations:

$$\mathbb{P}(\,(L_1 \pm 1) \cap L_2 = \emptyset \mid L_1 \cap L_2 \neq \emptyset) = \frac{N_{\text{end}}}{N_{\text{int}}}.$$

□

B.5. Generalization of Theorem 4

Theorem 6:

The probability that any two constraints $c_1(\gamma_1, \mathbf{v}_1)$ and $c_2(\gamma_2, \mathbf{v}_2)$ selected from a uniformly generated hypercubic function at uniform random intersect is

$$\sum_{i=0}^{\min(\gamma_1,\gamma_2)} \sum_{\substack{S\subset[d]\\|S|=i}} P(\Gamma_1 \cap \Gamma_2 = S_i) \prod_{j\in S_i}^{i} P([x_{1j}^{min}, x_{1j}^{min} + v_{1j}) \cap [x_{2j}^{min}, x_{2j}^{min} + v_{2j}) \neq \emptyset) \quad \text{(B.1)}$$

where

$$\mathbb{P}(\Gamma_1 \cap \Gamma_2 = S_i) \;=\; \frac{\binom{\gamma_1}{i}\binom{d-\gamma_1}{\gamma_2-i}}{\binom{d}{i}\binom{d}{\gamma_2}} \quad \text{(B.2)}$$

and

$$P([x_{1j}^{min}, x_{1j}^{min}+v_{1j})\cap[x_{2j}^{min}, x_{2j}^{min}+v_{2j}) \neq \emptyset) = 1-\frac{(|I|-v_{1j}-v_{2j}+1)(|I|-v_{1j}-v_{2j}+2)}{(|I|-v_{1j}+1)(|I|-v_{2j}+1)}. \tag{B.3}$$

As this theorem is not used and as its proof is similar to that of Theorem 4, the following is a sketch of the proof

Proof: Go through the steps of the proof of Theorem 4 with the only difference being the use of Lemma 2(1). That is, the probability that the two intervals intersect is:

$$\mathbb{P}([x_{1j}^{\min}, x_{1j}^{\min}+v_{1j})\cap[x_{2j}^{\min}, x_{2j}^{\min}+v_{2j}) \neq \emptyset) = 1-\frac{(|I|-v_{1j}-v_{2j}+1)(|I|-v_{1j}-v_{2j}+2)}{(|I|-v_{1j}+1)(|I|-v_{2j}+1)}.$$

Then summing over all subsets $S \subset [d]$ of size i, and then over all possible intersection sizes $i \in [0, \min(\gamma_1, \gamma_2)]$ then yields the full expression. □

C

Additional results

C.1. Additional results

C.1.1. Linearization vs L-BFGS-B

In [35], the weight ψ in (2.3) would increase when the function was close to an expected utility of the encounter. This cannot be done for this thesis's method as the optimization is over a random subset of the entire domain, and so no claim can be made about an expected utility. They do however not that in these cases, a uniform weight can be used. To illustrate, consider the maximization of the quadratic function

$$-x^2 + 2xy + 6x - y \tag{C.1}$$

over the $[0, 10] \times [0, 10]$ domain. By simply dropping the interaction term $e(x, y) = 2xy$, the optimization reduces to maximizing $-x^2 + 6x$ with respect to x and $-y$ with respect to y. This yields a maximum at $(0, 0)$ which is in fact the global minimum. Alternatively, the interaction term $e(x, y)$ can be linearized to yield $e_x(x)$ and $e_y(y)$ by means of a uniformly weighted average, giving

$$\frac{1}{10}\int_0^{10} e(x, y)dx = 10y = e_y(y)$$
$$\frac{1}{10}\int_0^{10} e(x, y)dy = 10x = e_x(x).$$

This type of weighting effectively redistributes the contribution of the interaction term across the variables that constitute it. Substituting $e(x, y)$ for e_x and e_y in (C.1) results in

$$-x^2 + 26x + 19y.$$

Optimizing this yields the correct global maximum at $(10, 10)$. This example is illustrated in Figure C.1.

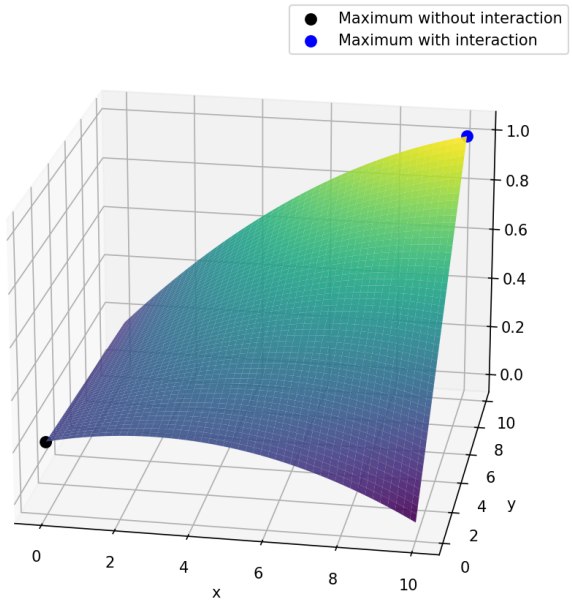


Figure C.1: The function $-x^2 + 2xy + 6x - y$ optimized via linearization including and excluding interaction effects

To test this out, random quadratic functions were generated where the weight matrix A was constructed to not be negative definite. This ensures that the function takes its maximum value at a corner or along an axis of the domain $[0, 2]^d$, corresponding to the cubes that will be used in this thesis. This was done 10 times for each dimension d where $d \in [1, ..., 50]$. The maximum of each function was then searched for via linearization and a single shot of the L-BFGS-B method, and the higher value that was found was used to normalize. Figure C.2 shows the result of this experiment.

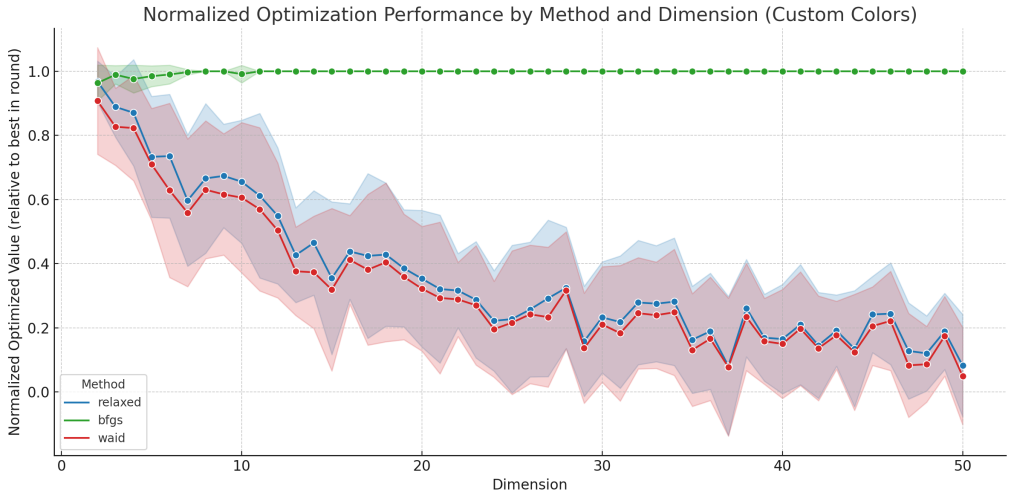


Figure C.2: A single shot of L-BFGS-B versus linearization both by dropping interaction effects (relaxed) and by averaging them out (waid) of quadratic functions in increasing dimension. The maximum of every experiment is used to normalize.

L-BFGS-B is clearly considerably better at finding the maximum with only a single shot

C.1.2. L-BFGS-B initializations

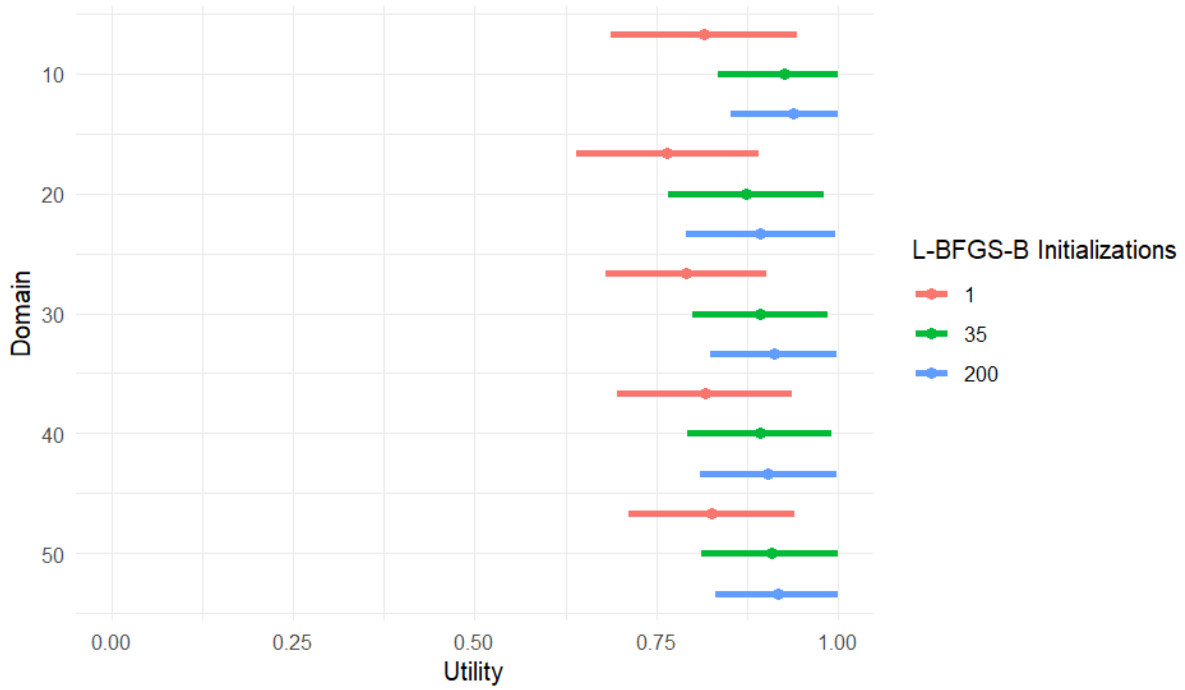


Figure C.3: Utility (mean ± sd) obtained by agent using varying number of L-BFGS-B initializations over varying domain sizes

L-BFGS-B Inits	Mean Utility	Sd Utility	Time(s)
1	0.805	0.127	0.02
35	0.905	0.098	0.04
200	0.915	0.093	0.1

Table C.1: Utility obtained by the agent while using a varying number of L-BFGS-B initializations. Numbers are averaged over the experiments conducted to generate Figure C.3 Time is in terms of one call to Function 8.

C.1.3. Cross effects

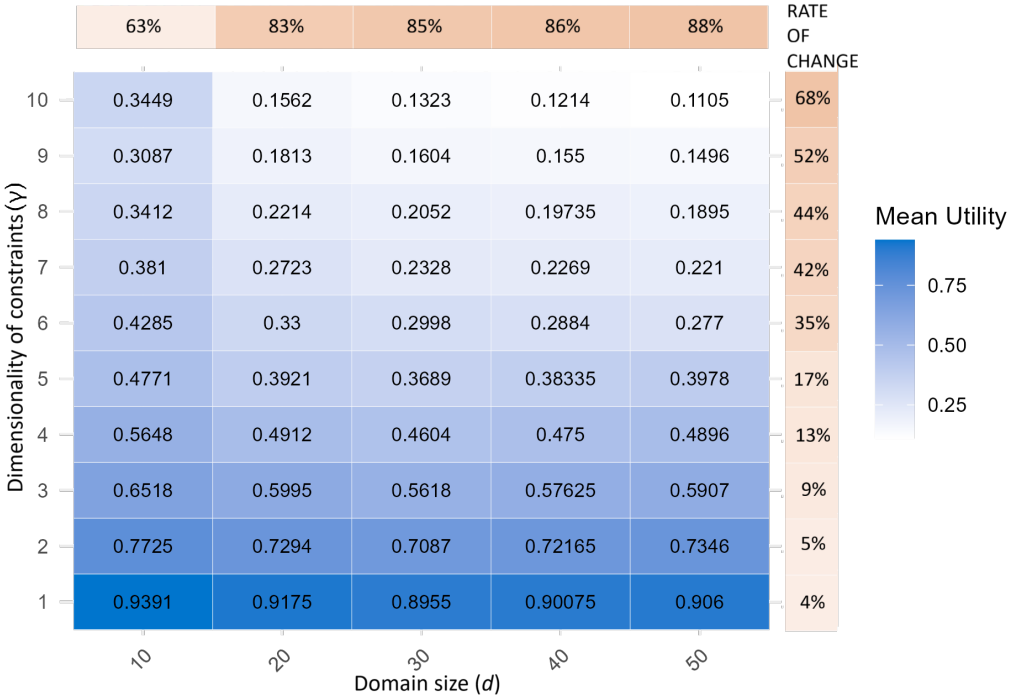


Figure C.4: Average utilities obtained while fixing d and γ and averaging over the other parameters. Rate of change on right (top) side indicates the change in utility over the corresponding row (column)

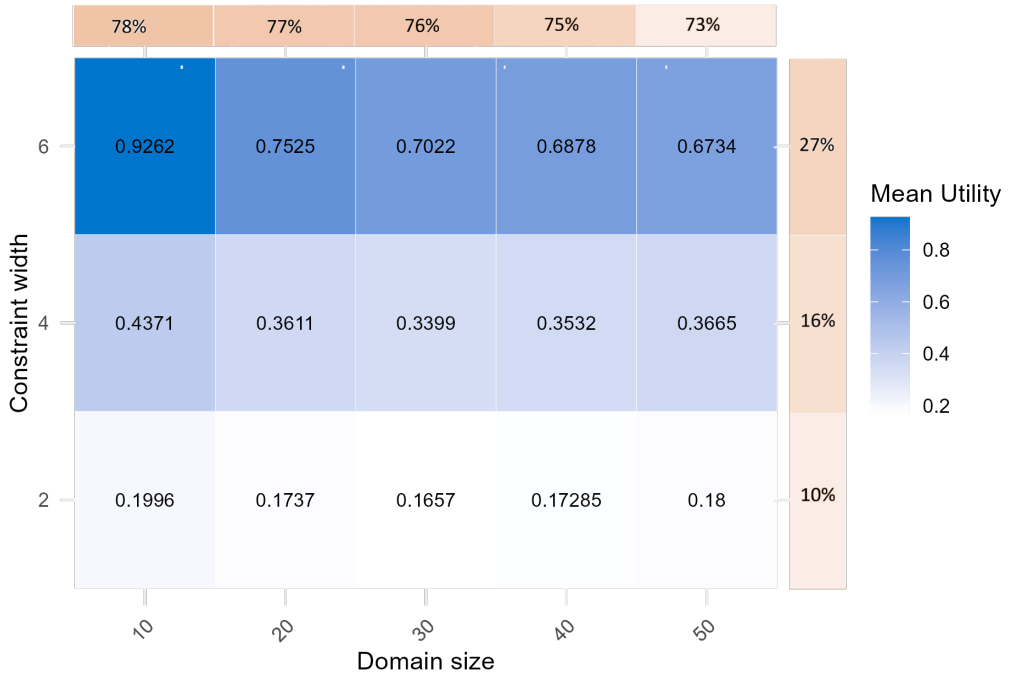


Figure C.5: Average utilities obtained while fixing d and v and averaging over the other parameters. Rate of change on right (top) side indicates the change in utility over the corresponding row (column)

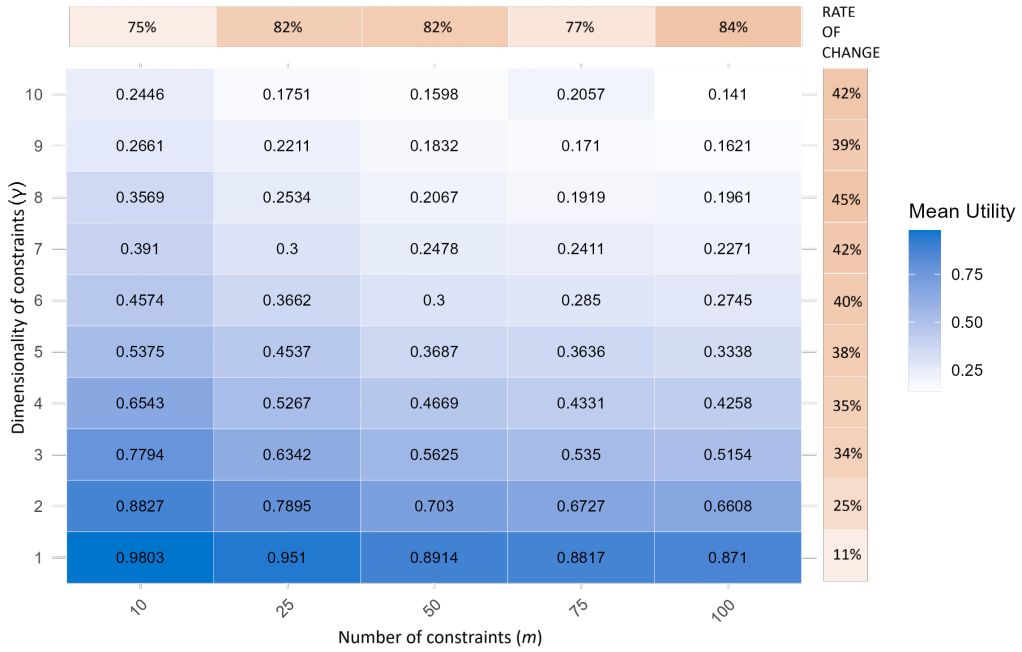


Figure C.6: Average utilities obtained while fixing m and γ and averaging over the other parameters. Rate of change on right (top) side indicates the change in utility over the corresponding row (column)

C.2. Deploying QuadrApprox against ANAC 14 agents

To evaluate the performance of the proposed method in a realistic autonomous negotiation setting, QuadrApprox was tested in 100 bilateral negotiations over 3 domain sizes (1200 negotiations in total) with medium complexity hypercubic utility functions against four prominent agents from the 2014 Automated Negotiating Agent Competition (ANAC). Among these, AgentM and GANGSTER were the official winners of the tournament. The objective of QuadrApprox, as developed in this thesis, is to maximize its own utility. In line with this goal, QuadrApprox consistently outperformed all opponents across this experiment. Notably, it achieved an average win ratio exceeding 80%, demonstrating the effectiveness of searching for bids in surrogate models. A summary of the experimental results is presented in the tables below.

Domain	avg Utility	avg Pareto	avg Rounds	avg Time(s)	Win ratio(%)
10 Issue Domain	0.774	0.430	2077	30	85
30 Issue Domain	0.619	0.603	908	34	80
50 Issue Domain	0.560	0.512	648	32	80

Table C.2: Summary of the performance of QuadrApprox against selected ANAC 14 agents.

Opponent	QA Utility	Opponent Utility	Pareto
AgentM	0.615	0.541	0.517
BraveCat	0.860	0.675	0.324
Gangster	0.412	0.411	0.612
WhaleAgent	0.924	0.704	0.409

Table C.3: The performance of QuadrApprox against selected ANAC 14 agents in a 10 issue Domain ($|\Omega| = 10^{10}$) in terms of individual utility and distance from Pareto frontier.

Opponent	QA Utility	Opponent Utility	Pareto
AgentM	0.859	0.811	0.794
BraveCat	0.732	0.457	0.488
Gangster	0.246	0.240	0.525
WhaleAgent	0.742	0.642	0.616

Table C.4: The performance of QuadrApprox against selected ANAC 14 agents in a 30 issue Domain ($|\Omega| = 10^{30}$) in terms of individual utility and distance from Pareto frontier.

Opponent	QA Utility	Opponent Utility	Pareto
AgentM	0.809	0.803	0.683
BraveCat	0.721	0.446	0.416
Gangster	0.000	0.000	0.458
WhaleAgent	0.713	0.613	0.490

Table C.5: The performance of QuadrApprox against selected ANAC 14 agents in a 50 issue Domain ($|\Omega| = 10^{50}$) in terms of individual utility and distance from Pareto frontier.

D

Declaration on AI use

During the creation of this thesis, I used AI, specifically ChatGPT, for two main purposes. First, I used the tool to review my written text and to correct any grammar mistakes. Occasionally, I asked for suggestions to enhance readability. I carefully reviewed all recommendations made by the tool, and I take full responsibility for the final work.

Second, I used ChatGPT for tasks related to presenting my results. This included refining plots in R and drafting LaTeX table code that matched my preferred formatting.