Incentives Design in the Presence of Externalities

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th>Citation</th>
<th>Rao, Malvika. 2015. Incentives Design in the Presence of Externalities. Doctoral dissertation, Harvard University, Graduate School of Arts &amp; Sciences.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:23845457">http://nrs.harvard.edu/urn-3:HUL.InstRepos:23845457</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Incentives Design in the Presence of Externalities

A dissertation presented
by
Malvika Rao
to
The School of Engineering and Applied Sciences
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Computer Science

Harvard University
Cambridge, Massachusetts
September 2015
Incentives Design in the Presence of Externalities

Abstract

The design of incentives becomes challenging when faced with externalities. In this thesis, I resolve this difficulty in two settings: position auctions and software economies. The first part of the thesis studies value externalities in position auctions. I develop a constraint-based model that allows an advertiser to submit, along with its bid, additional constraints to state how its value for clicks depends on the positions of the other ads with which it is allocated. I establish complexity results for winner determination and prove the existence of Nash and envy-free equilibria under certain conditions.

This thesis makes important contributions towards a foundation for software economies. I first study a setting in the private software economy consisting of a single task, a worker, and a manager. This is a combination of a repeated principal-agent problem and a prediction problem. I characterize a scoring system that elicits truthful information, leading to accurate predictions from both agents and best effort from the worker.

In the public software economy, I consider the problem of how to incentivize deep fixes to bugs from both computational as well as theoretical perspectives. In the computational work, I introduce a dynamic model of the software ecosystem and propose subsumption mechanisms as a solution. Next, I adapt an approximation methodology that is well-suited to large market settings, known as mean field equilibrium, to the model. I conduct an experimental study to characterize the system in equilibrium and derive lessons for market design.
Further, I perform theoretical analysis of a simple mean field model of deep fixes and prove the existence of a mean field equilibrium. I define a new type of dynamic market equilibrium, called correctness equilibrium, and prove its existence. Finally I consider the relationship between mean field equilibrium and correctness equilibrium, showing that mean field equilibrium need not satisfy a notion of efficiency whereas correctness equilibrium does.
Contents

Abstract ................................................................. iii
Acknowledgments ....................................................... xii

1 Introduction ......................................................... 1
  1.1 Overview of research area ........................................ 4
     1.1.1 Position Auctions ........................................... 4
     1.1.2 Software Economies ....................................... 6
  1.2 Contributions .................................................. 10
     1.2.1 Expressing Value Externalities In Position Auctions .... 10
     1.2.2 Predicting Your Own Effort ............................... 11
     1.2.3 Incentivizing Deep Fixes ................................. 12
     1.2.4 A Theoretical Model of the Public Software Economy ... 14
  1.3 Citations to Published Work ................................. 15

2 Expressing Value Externalities in Position Auctions ............. 18
  2.1 Introduction ................................................... 18
  2.2 Related Work .................................................. 21
  2.3 Preliminaries .................................................. 22
  2.4 Expressive GSP ................................................ 24
     2.4.1 Equilibrium Concepts .................................... 25
  2.5 Incentives in eGSP .......................................... 26
     2.5.1 Semi-Truthfulness ....................................... 28
  2.6 Equilibria in eGSP for Exclusion Constraints ................. 29
  2.7 Algorithmic Considerations .................................. 34
     2.7.1 Complexity on Bounded-Degree Graphs .................... 36
  2.8 Fixed-Parameter Algorithms .................................. 37
     2.8.1 Category-specific Constraints ........................... 38
     2.8.2 Local-exclusion Constraints .............................. 40
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td>Soft Constraints</td>
<td>43</td>
</tr>
<tr>
<td>2.10</td>
<td>Conclusion</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>Predicting Your Own Effort</td>
<td>47</td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>47</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Related Work</td>
<td>49</td>
</tr>
<tr>
<td>3.2</td>
<td>The Basic Model</td>
<td>52</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Desirable Properties of Scoring Systems</td>
<td>53</td>
</tr>
<tr>
<td>3.3</td>
<td>Characterization of Scoring Systems</td>
<td>54</td>
</tr>
<tr>
<td>3.3.1</td>
<td>A Family of Scoring Rules</td>
<td>56</td>
</tr>
<tr>
<td>3.4</td>
<td>Task Decomposition</td>
<td>58</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Independent Subtasks</td>
<td>60</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Correlated Subtasks</td>
<td>62</td>
</tr>
<tr>
<td>3.5</td>
<td>Simulation</td>
<td>65</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusion</td>
<td>69</td>
</tr>
<tr>
<td>4</td>
<td>Incentivizing Deep Fixes</td>
<td>72</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>72</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Main Results</td>
<td>75</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Related Work</td>
<td>76</td>
</tr>
<tr>
<td>4.2</td>
<td>A System of Bugs and Fixes</td>
<td>78</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Bit String Model</td>
<td>78</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Externally Observable</td>
<td>81</td>
</tr>
<tr>
<td>4.3</td>
<td>The Model of the Software Ecosystem</td>
<td>82</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Bug Generation</td>
<td>83</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Model Dynamics</td>
<td>84</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Subsumption Mechanisms</td>
<td>85</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Other Mechanisms</td>
<td>90</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Preemptive Fixing and Reuse</td>
<td>92</td>
</tr>
<tr>
<td>4.3.6</td>
<td>The Worker</td>
<td>93</td>
</tr>
<tr>
<td>4.3.7</td>
<td>The User</td>
<td>94</td>
</tr>
<tr>
<td>4.4</td>
<td>Mean Field Equilibrium</td>
<td>95</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Adapting MFE to our Setting</td>
<td>96</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Limited Workers</td>
<td>97</td>
</tr>
<tr>
<td>4.5</td>
<td>Experimental Study</td>
<td>100</td>
</tr>
<tr>
<td>4.6</td>
<td>Lessons for Market Design</td>
<td>116</td>
</tr>
</tbody>
</table>
5 A Theoretical Model of the Public Software Economy

5.1 Introduction ........................................ 121
5.2 The Jar Model ........................................ 123
  5.2.1 Subsumption Mechanism ......................... 124
  5.2.2 Mean Field Equilibrium ......................... 126
5.3 Correctness Equilibrium ............................ 133
  5.3.1 Preliminaries .................................. 135
  5.3.2 Worker Utility and Aggregate Supply .......... 136
  5.3.3 User Utility and Aggregate Demand ........... 138
  5.3.4 Definition and Properties ...................... 140
  5.3.5 Existence .................................... 143
  5.3.6 The First Fundamental Theorem of Welfare Economics .... 153
5.4 Relating Equilibrium Concepts .................... 155
5.5 Discussion ........................................ 162

6 Conclusion and Future Work .......................... 164

Appendix A Appendix to Chapter 2 .................... 167
A.1 Proof of Lemma 5 .................................... 167
A.2 Proof of Theorem 3 .................................. 168
A.3 Structural Observations ......................... 169
A.4 Proof of Theorem 5 .................................. 173
A.5 Greedy algorithm for soft constraints .......... 175
A.6 The “One Enemy” Special Case .................... 177
A.7 Scheduling with Precedence Constraints ........ 178
  A.7.1 Tractable Special Cases ....................... 179
  A.7.2 Discussion .................................... 182

Appendix B Appendix to Chapter 4 .................... 184
B.1 Likelihood Ratio Test .............................. 184

References ............................................. 186
List of Tables

2.1 Exclusion-only dynamic programming example with $C = \{(1, 2), (2, 3)\}$ and mutually excluded bidders closer than $L = 2$. 42

4.1 Mechanisms and the set of fixes used to preempt bugs from entering the market 118
4.2 Mechanisms and competition 118
4.3 Mechanisms and transfers 118
List of Figures

1.1 Coke insists on being placed above Pepsi. Pepsi wants to be above Perrier and Orangina. .................................................. 5
1.2 In the public software economy, different software components $R_1, R_2, \ldots$ generate bugs, encoded as bit strings, which may receive fixes, also encoded as bit strings. Bit strings representing bugs are placed above each horizontal line while fixes are placed below. In this model, a fix $f$ fixes a bug $b$ if $f$ AND $b = b$. ................................................................. 13
1.3 The Bull by Pablo Picasso, December 1945 – January 1946. A series of 11 lithographs that depict the creature in various stages of abstraction. ........... 17
2.1 Each sports shoe company’s bid is conditioned on being allocated above the other company. The general retailer’s bid is conditioned on neither shoe company’s ad being in slot 1. ............................................... 19
2.2 Classes of negative value-externality constraints. ........................... 21
3.1 Timeline of the worker-manager game. ....................................... 51
3.2 Average score and average number of prediction targets under the best policy, varying $p \in [0.5, 1]$ for $C = -1.9$ and $q = 0.5$. ....................... 67
3.3 Average score for policies 1 through 5, varying $p \in [0.5, 1]$ for $C = -1.9$ and $q = 0.5$. .......................................................... 67
3.4 Average score and average number of prediction targets under the best policy, varying $C \in [-5, 1]$ for $p = 0.8$ and $q = 0.5$. ......................... 69
3.5 Average score for policies 1, 3, 4, and 6, varying $C \in [-5, 1]$ for $p = 0.8$ and $q = 0.5$. ......................................................... 70
4.1 Overview of the software ecosystem ........................................... 78
4.2 Root causes $R$ generate bugs $b$, which in turn may receive fixes $f$. ...... 79
4.3 Root causes $R$ generate bugs, which may receive fixes. Bugs and fixes are represented as bit strings. ............................................... 82
Acknowledgments

First, I would like to thank my advisor, Professor David C. Parkes. My PhD has been a journey. I started my degree with certain ideas about research. Along the way, I discovered many new things. I opened my eyes. I ended up doing the kind of research I could not have imagined, culminating in this thesis. It is impossible to overstate the role that David has played in my journey. Our many hours of meetings – in person and on Skype, and conversations are at the heart of this thesis. I thank David for giving me this opportunity, for sharing his unique, extraordinary vision of research with me, and for seeing me through to the end. Working with him has been an honour and an inspiration.

I am deeply grateful to the other members of my dissertation committee: Professor Margo Seltzer, Professor Yiling Chen, and Dr. David F. Bacon. They have been so generous with their time and feedback, taken such a keen interest in my research, and actively helped to make my work stronger. I would like to thank Margo for her incredible energy, for stimulating discussions, and for providing detailed feedback that left no stone unturned.

I am grateful to David Bacon for inspiring me to pursue research in software economies. Every time David Bacon visited, my research would turn a corner. For all those creative, sparkling conversations, I thank him.

I would like to thank Yiling for providing thoughtful, insightful discussions at critical times, and for serving on both my dissertation and qualifying committees. I would also like to thank the other members of my qualifying committee, Professor Michael Mitzenmacher and Professor Steven Gortler, for giving me early encouragement along my research path.

Looking further back, I wish to express my sincere gratitude to Professor Sue Whitesides, who advised me at McGill University, for introducing me to research and for believing in my abilities.

I gratefully acknowledge the three years of PhD fellowship I received from NSERC.
Needless to say, the work done during my PhD would not be what it is without the many researchers I was fortunate to collaborate with. A special mention goes to Florin Constantin. Our days spent on auction externalities were some of the most exciting moments of my PhD. For being an ideal collaborator and, above all, for being a friend I thank him. In addition, I wish to thank all my co-authors: David F. Bacon, Eric Bokelberg, Yiling Chen, Florin Constantin, Chien-Chung Huang, Ian A. Kash, David C. Parkes, Margo Seltzer, Shrutivandana Sharma, and Manu Sridharan.

The EconCS and AIRG groups at Harvard have been my home over the years and have provided a lively and stimulating research atmosphere. I would like to thank the members of EconCS and AIRG for their support. I would also like to thank my fellow-students for making it so much fun to come to work everyday. In particular, I wish to thank Hossein Azari Soufiani, Eric Balkanski, Alice Gao, Thibaut Horel, Shaili Jain, John Lai, Hongyao Ma, Andrew Mao, Mike Ruberry, Sven Seuken, Greg Stoddard, Panos Toulis, Bo Waggoner, Ming Yin, and Haoqi Zhang. During my time at Harvard I made some wonderful friends – I thank Moritz Bacher, Uri Braun, Thibaut Horel, Masoumeh Izadi, Avinash Kishore, Kirstin Petersen, Shardul Phadnis, and Ming Yin for their friendship.

I am blessed to have a loving and supportive family. I am deeply grateful to my aunt and uncle, Priyanka Naithani and Vikram Simha, for welcoming me into their home amidst major changes in their lives. During my final year, they hosted me for indefinite periods of time, encouraged me to make their home my own, and watched over me. For that and more, I will be ever grateful. I also wish to extend my sincere appreciation to my cousins in Cape Cod, Uma and Jayant Rajagopal, for their hospitality and for good times. The love and care I have received from my aunt and uncle in Wisconsin, Poornima and Subbanna Jayaprakash, has been immeasurable and I thank them. I wish to thank my brother and sister-in-law, Abhijeet Rao and Neha Kaul, for their brilliant sense of humour that would work its magic on me even from afar.
I thank my husband, Jean-Francois Minardi, for finding me and then finding me again, for holding my hand through every obstacle, for solving all my most important decision problems, for inspiring me, for lighting my life with joy. For all that he is.

Most importantly, I would like to thank my parents, Gowri and Arun Rao. Words cannot express how grateful I am for their invaluable presence in my life. Their unconditional love, perceptive intelligence, and zest for life make everything possible. They are my strength and my essence. All that I am and ever will be is thanks to them. Amma, Appa, thank you.
I dedicate this dissertation to my parents, Gowri and Arun Rao.
Chapter 1

Introduction

In recent years there has been revolutionary growth in research at the intersection of computer science and economics. As computer systems become more interconnected, representing different self-interested entities and competing for scarce resources, the resulting interactions and behaviours resemble economic phenomena. At the same time, as markets and other economic mechanisms are increasingly operated by computers, an economic analysis of these structures must incorporate computational considerations. As these two worlds collide, new mathematical models are required that can capture, understand, and answer the fundamental questions underlying these phenomena.

Examples of modern engineering systems with the characteristics described above are abundant. For instance, Skype is a peer-to-peer telecommunications application that provides users with video chat, voice call, and file exchange services via the Internet, and can be accessed from computers and mobile devices. Whatsapp is a text messaging service available on mobile devices that has been developed using open source software. Wikipedia is a free encyclopedia accessible on the Internet, where the content is provided by volunteers. These typify interconnected, decentralized computing systems that resemble socio-economic phenomena. The online auctions operated by Google and eBay and the digital currency Bit-
coin are examples of economic systems operated by computers. Online labour markets are another example of the intersection of computing and economic systems. An interesting case is TopCoder, which matches companies that require software solutions to a global community of programmers. To address a specific client request, programmers compete in a contest with cash awards and the best solution is chosen. Similarly, BountySource is a funding platform for improving open source software, where users can post rewards or bounties on issues they would like tackled and developers can devise solutions and claim the bounty.

A central question in the creation of such hybrid systems is how to design incentives in the presence of externalities. Externalities express the concept of interdependencies, where the utility of an agent may be influenced by the actions or information of other agents in a system. Externalities arise naturally in markets and have received considerable attention in the economics community [13, 51, 52, 53, 54]. However the incentives aspect of externalities are relatively little studied in computer science, although there is a growing literature in the context of networks and auctions [5, 33, 44, 60, 79]. A general definition of an externality, given by Mas-Colell et al. [74], is as follows:

An externality is present whenever the well-being of a consumer or the production possibilities of a firm are affected by the actions of another agent in the economy.

Although ubiquitous, externalities take on different forms in different settings. Two important types of externalities are allocative and informational externalities [52]. In an allocative externality, the value of an agent depends not only on the bundle of goods he is allocated, but also on the bundle of goods allocated to other agents. In an informational externality, the value of an agent for an outcome is affected by the information that other agents may have. For example, an agent’s value for a painting may be influenced by the private information of another agent regarding the painting’s quality. The presence of externalities complicates the design of incentives in ways that traditional solutions cannot overcome [44]. For instance, an
agent’s value for an outcome no longer has a simple representation and can run to multiple dimensions. Another difficulty is that problems that must contend with externalities may be computationally intractable. Moreover, interdependencies make the design space more complex, thereby making it harder to design mechanisms with good properties and without simplifying assumptions.

I answer the question of how to design incentives in the presence of externalities in two settings: i) the first part of the thesis studies a type of auction, referred to as a position auction, where the items being auctioned are positions – for example, positions on a webpage that bidders might use to place advertisements; ii) the second part introduces software economies, a vision of software development inspired by market economies, where supply and demand drive the evolution of the system. In the context of position auctions allocative externalities must be considered, where a bidder’s value for the position it is allocated depends also on the positions allocated to the other bidders in the auction. In a software economy, the design of incentives must contend with informational and allocative externalities. For instance, the quality of the fix to a software bug submitted by one worker now may influence the type of bug that appears later and affect the payoff of the worker assigned to the later bug. In addition, a fix for a particular bug, reported by a particular user, may fix other bugs as well. This results in a positive externality to a different set of users who may encounter those bugs and experience some disutility. Another case is that of a manager and a worker who each have private information regarding the completion time of a task. The information held by the worker, for example, might influence the prediction provided by the manager about the completion time of the task, and further, the worker’s prediction may influence how much effort he exerts to complete the task. In this dissertation, I resolve the challenge posed by the presence of externalities, and design and analyze the unique incentives at play in each setting.

I provide a more detailed description of the research areas addressed in this thesis as well
1.1 Overview of research area

1.1.1 Position Auctions

Online sponsored search auctions, also known as position auctions or ad auctions, are a major source of income for Internet companies, generating billions of dollars of revenue each year. In a position auction, a bidder states a bid associated with a set of search keywords that are related to the product it wants advertised. A search engine offers positions (slots) for advertisements (ads) adjacent to the organic search results, where the ad with the highest bid receives the best position. The generalized second price auction (GSP) is the industry standard for allocating ad slots. Clearly this is an important area of auction design with considerable economic impact. I extend the research in this area to consider the effect of allocative externalities.

Consider the following scenario: Coke and Pepsi both bid to display their ads on a search results webpage when a user searches using keywords “soft drink” or “cola”. Consumer data suggests that slots for ads on a webpage that are placed higher are considered more seriously by users [25]. Hence Coke submits along with its bid, an additional constraint that it will only accept an ad placement anywhere above Pepsi’s ad but not below (see Figure 1.1). This scenario portrays a simple form of externality effect where advertisers competing in position auctions may have different values for the slot allocated depending on their relative position with respect to other advertisers. This value effect occurs in addition to the effect on the quantity of clicks received due to co-location with other ads. That is, not only does Coke stand to lose user attention, in terms of the number of times users click on the Coke ad, if appearing below Pepsi’s ad, but the attention that it does receive may be only from those users that were not interested in purchasing drinks in any case.
Figure 1.1: Coke insists on being placed above Pepsi. Pepsi wants to be above Perrier and Orangina.

Of interest to my research is the value an advertiser has for a click, conditional on receiving one. Unlike quantity externalities, value externalities cannot be learned by a search engine because they are bidders’ private information and therefore are not observable through data collected by the search engine. Hence value externalities must be captured via a bidding language. It is natural to consider the implications of allowing an advertiser to submit, along with its bid, additional constraints to state how its value for impressions or clicks depends on the (relative or absolute) positions of the other ads with which it is allocated. For example, Coke may bid stating that it has positive value $b$ if it receives a higher slot than Pepsi and zero value otherwise, reflecting a negative value externality that Coke may experience in regard to its competitor Pepsi.

Standard models of position auctions do not consider externalities [27, 96]. While quantity externalities have received attention [4, 13, 35, 39, 62, 87], value externalities are relatively less studied. Aggarwal et al. [5] analyze prefix position auctions where an advertiser can specify interest in only the top $k$ positions. Muthukrishnan [79] considers an auction where advertisers bid on the eventual maximum number of ads shown. Ghosh and Sayedi [34] study position auctions where the user submits two bids: one for solo placement and another for placement alongside other ads. In Chapter 2 I argue that the above value
externality models [5, 34, 79] fall within the framework proposed in this thesis.

The goal of my research is to study the means and effect of incorporating negative value externalities in position auctions. Specifically I investigate the following questions:

- What is a bidding language that allows an advertiser to express the value externalities he may experience in regard to co-located advertisers?
- How can the standard GSP auction be extended to accommodate value externalities, and how is the winner determination problem solved?
- What equilibria exist under these conditions?

1.1.2 Software Economies

Software systems have evolved from a Turing machine “input tape to output tape” model of computation to one of continuous interaction with other large systems and with the physical world. In the meantime traditional software engineering techniques have failed to scale accordingly, leading to increased inefficiencies and errors. A study commissioned by the U.S. National Institute of Standards and Technology (NIST) concluded that software errors alone cost the U.S. economy approximately $59.5 billion annually [82]. What is more, the traditional, ideal notion of *absolute correctness* seems ill-adapted to the size and complexity of today’s software systems and is increasingly unachievable. All of this points to the need for a new paradigm in creating and evolving such systems.

In fact software systems have come to resemble systems where behaviour is decentralized, interdependent, and dynamic – rather like market economies. This suggests that the principles of market design and mechanism design will find an important role in complementing traditional engineering techniques. Market mechanisms provide important advantages: markets enable us to directly target incentives issues that abound in any development
process; they elicit valuations from users and allow users greater power to influence the direction of software development; they allow the aggregation of supply and demand thereby providing the scale needed to solve long neglected engineering problems; and prices act as signals and play a crucial role in the efficient allocation of resources.

In this thesis I present the concept of *software economies*. Software economies refer to a vision for a software development process where supply and demand drive the allocation of work and the evolution of the system. The goal is to use the methods of market design and mechanism design to create a self-regulating system that drives the software ecosystem towards a type of dynamic market equilibrium that is well-suited to a model of computation of continuous interaction and allows for a more achievable concept of correctness.

Software economies consist of a *private software economy* and a *public software economy*. A private software economy deals with the internal incentives of managers and employees. Here, well-designed incentives have the potential to reduce costs, increase predictability, and provide insights into which processes yield the greatest benefits in the development cycle. A public software economy considers how to design a market for software bugs and features. A public software economy must contend with much larger scales than the private economy, such as a large user base, a large number of workers, and a large number of bugs and features in the market. If well-executed, a public software economy could provide the key advantages of a market, leading to greater efficiency in managing the software ecosystem.

Prior literature has considered economic approaches towards related aspects of the software engineering industry, but with a different focus from the research in this thesis. For instance, the use of market-based approaches in improving vulnerability reporting systems has been explored. Schechter [90] describes a vulnerability market where a reward is offered to the first tester that reports a specific security flaw. Ozment [83] likens this type of vulnerability market to an open first price ascending auction. Other papers examine the role of
incentives in information security and privacy and propose policies to improve security and privacy [7, 8, 9, 24, 59, 68, 91]. In the economics literature, several papers [14, 57, 58, 70] have studied open source movements from the perspective of community formation and contribution. For instance, Athey and Ellison [14] look at the dynamics of how open source projects begin, grow, and decline, addressing issues such as the role of altruists, the evolution of quality, and the pricing policy. Other research has examined software from the point of view of technological innovation [19, 65], as well as from the point of view of the design and modularity of software [17, 72].

Although the market-based system that is modelled and analyzed in this thesis does not (yet) exist, there are several real-world examples of emerging market-based or crowdsourced platforms for software development. Topcoder [94] and BountySource [21], mentioned earlier, are two such platforms. Other similar platforms include Bountify [20], BugCrowd [22], and GetACoder [32]. For a comprehensive survey of crowdsourced platforms for software development as well as the associated academic literature, see Mao et al. [73]. However these platforms tend to address software tasks that are small and modular. The solutions tend to be human-verified and payment schemes consist of simple one-shot payments. In this thesis I aim to design models and mechanisms that capture a greater degree of the complexity of software economies.

Next I describe the specific research questions that I tackle in the private and public software economies. The problems I solve in this part of the thesis are motivated by incentives issues in the software economy. However these problems are not unique to the software engineering process and can occur elsewhere. I contribute solutions that are general and adaptable to other domains.

Private software economy. I consider a setting that consists of a single task with two agents, a worker and a manager. Each agent has private information regarding the completion
time of the task. The goal is to elicit truthful information leading to accurate predictions from the worker and the manager, as well as the best possible effort from the worker. The challenge that must be overcome is that this is a combination of a repeated principal-agent problem and a prediction problem. Typically, in a principal-agent problem, a principal wishes to elicit a desired effort level from an agent but the agent is not involved in providing information used to make predictions (see for example, [29, 86]). In a prediction problem, accurate predictions of an event’s outcome are sought from the agent but without considering that the agent doing the prediction may control the outcome (see for example, [38, 45, 66, 78]). In contrast I seek to obtain both accurate prediction as well as best effort.

The research questions I address in this setting are as follows:

- How can the game between the worker and manager be modelled?
- Is there a mechanism that elicits truthful prediction from both agents and best effort from the worker in a collusion-proof way?

**Public software economy.** Designing a market mechanism for software bugs and features is fraught with challenges. An important question is how to design incentives to obtain robust or “deep” fixes rather than “shallow” fixes to bugs, at least where users would like a deep fix. A deep fix attempts to correct the root cause of the problem so that another bug with the same root cause is found only after a long time or not at all. In contrast a shallow fix suppresses the bug at a superficial level so that other bugs with the same root cause may appear soon after. While this is a known challenge, there has been no prior work in the academic literature.

My technical approach adopts the *mean field equilibrium* (MFE) methodology, an approximation methodology that is useful for analyzing the behaviour of large systems populated with self-interested participants. Mean field equilibrium derives its name from mean field models in physics where large systems display macroscopic behaviour that is easier to analyze than their microscopic behaviour. MFE have been studied in various settings in
economics [1, 18, 98, 99] and control theory [47, 48, 67]. More recently, a series of papers have analyzed MFE in market settings [2, 3, 12, 42, 43, 50].

The goal of my research is to propose a foundation for the public software economy. I consider the following research questions:

- How can the complexity of the software ecosystem be distilled into a model?
- What mechanisms incentivize deep fixes?
- How can the mean field methodology be adapted to this problem?
- What are the market design lessons that may be drawn?
- What can be said about correctness in this setting?

1.2 Contributions

In this section I describe the contributions of this dissertation, chapter by chapter.

Part I Position Auctions

1.2.1 Expressing Value Externalities In Position Auctions

- I develop a constraint-based model that allows an advertiser to submit, along with its bid, additional constraints to state how its value for clicks depends on the positions of the other ads with which it is allocated. Moreover, the model incorporates different types of constraint classes.

- I show that the winner determination problem is NP-hard via reduction from the IndependentSet problem.
• I contribute fixed-parameter tractable algorithms for exact winner determination for a subclass of constraints, and establish a connection to scheduling problems. To the best of my knowledge this work is one of the first to apply the theory of fixed-parameter tractability to auction design.

• On the incentives side, I introduce a natural extension of the GSP auction, the expressive GSP (eGSP) auction, that induces truthful revelation of constraints for a rich subclass of unit-bidder types, namely downward-monotonic unit-bidder constraints.

• I prove existence of Nash equilibrium for a class of constraints, called exclusion constraints, for which standard GSP has no Nash equilibrium.

Part II Private Software Economy

1.2.2 Predicting Your Own Effort

• I model the setting as a 3-stage game: 1) the worker shares information with the manager; 2) the manager predicts completion time under worker’s best effort; 3) the worker exerts effort and completes the task in some actual time.

• I characterize a scoring system for worker and manager that provides the following properties: information is truthfully reported by both agents leading to accurate prediction, best effort is exerted by the worker, and collusion-proofness is achieved.

• I study the effect of the scoring system on whether the worker splits a task into multiple prediction targets (subtasks). There is greater propensity to split tasks into subtasks when the completion times are correlated than when the completion times are independent.

• I validate the qualitative observations in the theoretical analysis via a simulation study.
Part II Public Software Economy

In the next two chapters, I approach the public software economy, and in particular, the problem of incentivizing deep fixes, from a multitude of angles. First I present a rich computational model and build a simulation. Following this, I study a simplified abstraction of the computational model from a theoretical angle. From a modelling perspective, I consider how to design a model and mechanisms that capture the essence of the problem. From the point of view of a system designer, I evaluate the strengths and limitations of the model in simulation by measuring several performance metrics. In considering the equilibria in this setting, I take a game-theoretic approach and apply the mean field equilibrium methodology to the problem formulation. This work is the first to adapt the mean field equilibrium concept to a setting in software economies. Moreover, I extract lessons for market design and seek to understand a suitable notion of competitive equilibrium. To this end, I define a correctness equilibrium, which is a type of competitive equilibrium that is adapted to handle dynamic settings.

Technical contributions specific to each chapter are described next.

1.2.3 Incentivizing Deep Fixes

- The first and foremost challenge is to distill the complexity of the software ecosystem into a simple model, abstracting away the details until what is left is the fundamental essence of the problem. The process of distilling is difficult to describe in words but can be vividly illustrated by a picture as shown in Figure 1.3. I contribute a dynamic model of the software ecosystem, comprising an abstract model of bugs and fixes as well as a model of the world. The model is flexible, and therefore its elements can be treated as building blocks that can be decoupled and reconfigured to explore different questions in the software ecosystem. For example, one can plug in various payment
Figure 1.2: In the public software economy, different software components $R_1, R_2, \ldots$ generate bugs, encoded as bit strings, which may receive fixes, also encoded as bit strings. Bit strings representing bugs are placed above each horizontal line while fixes are placed below. In this model, a fix $f$ fixes a bug $b$ if $f \text{ AND } b = b$.

schemes, other user and worker models, different relationships between bugs and fixes, and so forth.

- I introduce the bit string language, an abstract representation that is designed to capture the essence of the phenomenon of deep fixes as they interplay with bugs. The bit string language is versatile and can encode different settings in the software economy. The model of the ecosystem along with the bit string language is depicted in Figure 1.2.

- I design and analyze mechanisms that use externally observable information only, called subsumption mechanisms. In a subsumption mechanism, deeper fixes can subsume or replace shallower fixes and a worker’s payoff increases if his fix subsumes another fix. I further distinguish between different types of subsumption: eager, eager with reuse, and lazy subsumption.

- My technical approach is to frame the problem in the context of the mean field method-
ology. I adapt this technique to this computational setting and obtain convergence under different parameterizations of the environment.

- I develop models of worker and user utility. The worker utility model estimates expected utility via look-ahead sampling to sample future trajectories and by using the mean field distribution to understand how other workers may act in the future. User utility is a metric that unifies several performance measures, such as a user’s cost (amount spent towards a reward for a fix), a user’s wait time for a fix (the number of time periods elapsed from the bug being reported to a fix being submitted), and side-effects of a fix (a deeper fix pre-empts future bugs).

- An experimental study is carried out to evaluate the model using performance metrics. The study reveals that subsumption mechanisms perform robustly across all environment configurations examined, and satisfy important criteria for market design.

- I conclude by describing some lessons learnt for market design, both from the experimental study as well as from qualitative analysis of the mechanisms, and make recommendations regarding the suitability of the different mechanisms for various market design criteria.

1.2.4 A Theoretical Model of the Public Software Economy

- I propose a simple mean field model that is an abstraction of the richer computational model of the preceding chapter.

- I use Brouwer’s fixed point theorem to prove existence of mean field equilibrium.

- I develop a new type of dynamic competitive equilibrium, called correctness equilibrium, that is adapted to handle the inter-temporal nature of the model in this work.
Correctness equilibrium introduces beliefs into the standard competitive equilibrium model. Thus correctness equilibrium requires both market-clearing prices as well as consistency of beliefs.

- I prove existence of correctness equilibrium.
- Correctness equilibrium suggests a new, more tractable definition of correctness than the traditional notion of absolute correctness, which requires that all bugs in a software should be fixed. I propose that a software is market-correct if the market for bug fixes for that software has reached correctness equilibrium.
- I consider the relationship between mean field equilibrium and correctness equilibrium, showing that mean field equilibrium need not satisfy a notion of efficiency whereas correctness equilibrium does.

1.3 Citations to Published Work

Parts of Chapter 1 have appeared in:


Chapter 2 is based on the following two papers:


Chapter 3 has appeared in:


Chapter 4 is a significantly extended version of a preliminary contribution presented at a workshop:

Figure 1.3: The Bull by Pablo Picasso, December 1945 – January 1946. A series of 11 lithographs that depict the creature in various stages of abstraction.
Chapter 2

Expressing Value Externalities in Position Auctions

2.1 Introduction

A search engine offers positions (slots) for ads adjacent to the organic search results, with slots lower on the page tending to generate fewer clicks. The generalized second-price auction (GSP) is the industry standard for allocating ad slots. It is the product of design updates in response to increasingly sophisticated individual bidder behaviour. In GSP, each ad is associated with a per-click bid indicating the advertiser’s willingness-to-pay for a click. This implies an expected (bid) value for a slot, which we can think of simplistically as the bid times the click-through rate (CTR), which is the probability of a user click. A greedy algorithm is used to allocate ads to slots in decreasing order of per-click bid. Whenever an ad in slot $j$ receives a click, the advertiser pays a price equivalent to the bid of the bidder in the “next slot”. This is the smallest per-click bid of the advertiser that would have resulted in it retaining slot $j$.

However, for a given slot, an ad’s effectiveness and the distribution of clicks that it attracts
Figure 2.1: Each sports shoe company’s bid is conditioned on being allocated above the other company. The general retailer’s bid is conditioned on neither shoe company’s ad being in slot 1.

also depend on the other ads shown, e.g., via the number of other ads [87] or their relative position [25, 39, 56]. Preferences taking into account such issues cannot be expressed in the current bidding language in GSP and their effect on bidder behaviour is not well understood. While additional expressiveness (e.g., “maximize the number of clicks” or “try to balance my spending across the week”) is increasingly being provided to bidders, the GSP auction remains at the centre of search-engine advertising systems.

We term this kind of allocative externality a quantity externality. We address an orthogonal kind of allocative externality, referred to here as a value externality, where an advertiser’s value, given a click, depends on which other bidders are simultaneously allocated and what locations they are allocated. Whereas quantity externalities are observable to a search engine, value externalities are private to bidders and need to be expressed via a bidding language.

In this work, we introduce unit-bidder constraints (UBC), enabling a flexible class of languages for expressing negative value externalities in position auctions. Each bidder can condition its own bid for a particular slot on another bidder not being allocated some other slot. Each bidder can submit multiple such constraints. The constraints associated with a bid bind only if the bidder is allocated: the bidding language does not force the auctioneer to select a particular allocation, but instead precludes allocations. See Figure 2.1. For example, shoe company 1 can say “my bid is only valid if I am allocated above shoe company 2.” In a more general language we introduce soft-constraints, where a smaller non-zero bid is adopted when a constraint is violated. We refer to the first case, with bid values equal to zero.
if constraints are violated, as a hard constraint model.

We extend the standard GSP auction to allow bids in the UBC language. This expressive GSP (eGSP) auction greedily allocates bidders to slots. To be eligible for allocation to the next slot, an ad must not be in conflict with the constraints of any bidders already allocated. The bid value for an unallocated ad depends in turn on the allocations already made and the ad’s own constraints. A “next price” payment rule is used, with the payment of an allocated bidder equal to the minimum bid it could have made and still won the same slot given its constraints. The choice of a greedy algorithm is motivated by the need for rapid response time for search engines; moreover, achieving even a reasonable approximation to the optimal allocation in the UBC model is NP-hard.

Although eGSP is not strategy-proof, our first result establishes that reporting truthful constraints is a dominant strategy in eGSP for downward monotonic UBC, whatever the bid value and whatever the bids of others. While fruitful bid manipulations already exist in standard GSP, our result shows that augmenting the preferences of bidders with downward-monotonic UBC does not introduce new types of fruitful manipulations. Downward-monotonicity insists that a bidder dissatisfied with a particular slot given an allocation to other bidders is also dissatisfied with any lower slot. The downward-monotonic UBC languages include natural languages in which a bidder precludes being below other bidders (identity-specific), or cares about the range of slots it or other bidders are in (slot-specific). See Fig. 2.2.

We also consider the special case of exclusion externalities, in which a bidder insists that it is never allocated simultaneously with another bidder. An exclusion externality is both an identity-specific and a slot-specific externality. Our second result establishes the existence of envy-free Nash equilibrium in eGSP when each bidder is involved in at most one exclusion constraint. The result is obtained by a delicate reduction to the equilibrium in a GSP with bidder-specific reserve prices [28]. In contrast, there exists no pure-strategy Nash equilibrium in the standard GSP in the same setting. We also establish existence of
envy-free equilibria for general degree exclusion constraints. In a deviation from the existing literature [27], we observe that envy-free equilibrium and Nash equilibrium are incomparable for eGSP.

Turning to algorithmic results, we provide a tight bound on the approximation ratio of the greedy algorithm. For two parametrized classes of constraints, category and local-exclusion, we identify polynomial time optimal algorithms, assuming fixed parameters.

Omitted technical details are described in Appendix A.

2.2 Related Work

Position auctions (e.g., GSP) are an active research area [27, 96], and some of this work focuses on quantity externalities [4, 13, 35, 39, 62, 87]. Although relatively less studied, there are papers that consider value externalities, the focus of the present work. Our class of downward-monotonic UBC languages for expressing value externalities encompasses existing models. Slot-specific constraints generalize the “bid-to-the-top” model of Aggarwal et al. [5], where an advertiser can restrict its bid to appear above some position. The authors describe an easy to implement mechanism and show existence of equilibrium in their special case. UBC can also encode the model of Muthukrishnan [79], where bidders have bids
that depend on the maximum number, referred to as configuration, of ads shown. The latter
work proposes a social-welfare maximizing algorithm and critical value pricing scheme, but
does not address incentives. Ghosh and Sayedi [34] consider a model in which an advertiser
submits two bids: one for solo placement and another for placement alongside other ads.
Revenue and efficiency tradeoffs are examined, in what is a special case of exclusion UBC
with soft constraints. In an incomparable model to ours, Ghosh and Mahdian [33] consider
a setting where an advertiser’s value depends on its quality relative to other ads shown, but
irrespective of their location. Other algorithmic work related to externalities includes Krysta
et al. [64] for combinatorial auctions and Kash et al. [60] for secondary markets for wireless
spectrum. Externalities are well-studied in economics [51, 53, 54], but without a focus on
computational or representation issues.

Even-Dar et al. [28] show the existence of envy-free equilibria in GSP when there are
bidder-specific reserve prices. We adopt their characterization in establishing envy-free equi-
libria in expressive GSP for exclusion constraints.

2.3 Preliminaries

Let $N = \{1, \ldots, n\}$ denote the bidders in a position auction with $m$ slots. As is standard,
we assume $m = n$ (since there is essentially an unlimited number of slots, on multiple pages
of search results.) Each bidder $i$ is associated with a per-click value $v_i \geq 0$. We assume
that the click-through rate (CTR) falls off from one slot to the next according to discount
factor $\delta \in (0, 1)$ and we normalize the first slot’s CTR to 1. Slots $1, 2, \ldots, m$ have CTRs
$1, \delta, \ldots, \delta^{m-1}$.

In our basic model of value externalities, we associate bidder $i$ with constraints $C_i$, so
that his expected value for slot $j \in \{1, \ldots, m\}$ is $v_i \delta^{j-1}$ as long as constraints $C_i$ on the
allocation are satisfied, and zero otherwise. Later, we also allow “soft constraints” wherein
a bidder’s value is $v_i$ or $0 \leq v_i^- \leq v_i$ if the constraints are violated. Both value and constraints are private to each bidder $i$, that submits a bid $b_i$ and constraints $\hat{C}_i$ to the seller, perhaps untruthfully. Given reported bids and constraints, the seller would like to solve WDP, i.e., compute an optimal allocation.

**Definition 1.** Given bids $b = (b_1, \ldots, b_n)$ and constraints $\hat{C} = (\hat{C}_1, \ldots, \hat{C}_n)$, the winner determination problem WDP is to find a set of winners $W \subseteq N$ and an allocation $A$ (with any $i \in W$ winning slot $A_i$) solving:

$$\max_{(W,A) \in \mathcal{F}} \sum_{i \in W} b_i \delta^{A_i}$$

(2.1)

where $\mathcal{F}$ is the set of feasible solutions $(W,A)$: $A_i \neq A_j$ for all $i \neq j$ (both in $W$), and $A$ satisfies $\hat{C}_i$ for every $i \in W$.

Apart from constraints, we assume a standard quasi-linear utility-maximizing bidder model. Bidder $i$’s expected utility equals $(v_i 1_C_i - p_i) \delta^{A_i - 1}$, where $p_i$ is the per-click payment, given that $i$ is allocated slot $A_i$ and $1_C_i$ equals 1 is all constraints in $C_i$ are satisfied and 0 otherwise.

We introduce unit bidder constraints (UBC), a natural and expressive constraint model, as argued below. In UBC, any bidder $i$ has a set $C_i$ of $L_i \geq 0$ constraints (encoded as triples)

$$C_i = \{(\text{pos}_i, B_\ell, \text{pos}_\ell)\}_{\ell=1,\ldots,L_i}$$

(2.2)

where triple $(\text{pos}_i, B_\ell, \text{pos}_\ell)$ imposes the requirement that if bidder $i$ is allocated to slot $\text{pos}_i$ then bidder $B_\ell$ ($\neq i$) cannot be allocated to position $\text{pos}_\ell$.

Different languages impose restrictions on the specific kinds of UBC constraints enabled. For example, UBC can encode identity-specific constraints, where a bidder specifies a set of bidders above which it must be allocated. One can conceptualize this as a directed graph on bidders, with an edge from $i$ to $j$ indicating such an “enemy” of $i$. With 3 slots, this would
be encoded in UBC as

\[ C_i = \{(2, j, 1), (3, j, 2), (3, j, 1)\} \] (2.3)

UBC can also encode slot-specific constraints, where a bidder imposes a requirement that its bid is only valid if it is no lower than some slot and one or more other bidders are no higher than bidder-specific slots. For example, if \( i \) wants to be in slots 1 or 2 and \( j \) should not be in slots 1 or 2, and there are 4 slots, then this can be encoded in UBC as

\[ C_i = \{(3, *, *), (4, *, *), (*, j, 1), (*, j, 2)\}, \] (2.4)

where ‘*’ indicates that the entry is instantiated for all valid values. An exclusion constraint between \( i \) and \( j \), in which \( i \) excludes \( j \) and \( j \) excludes \( i \), is encoded as \( C_i = \{(*, j, *)\} \). Next, we introduce a natural generalization of GSP to UBC.

### 2.4 Expressive GSP

Expressive GSP (eGSP) with hard constraints takes reported UBC constraints \( \hat{C} \) and bid values \( b \) as input and implements a greedy allocation, collecting next-slot payments. The pseudocode for eGSP is given below. An unallocated bidder \( i \) is eligible for slot \( j \) if this allocation is not precluded by the constraint of some already allocated bidder in slots 1, \ldots, \( j - 1 \), or by a constraint between \( i \) and some already allocated bidder. The allocation rule for eGSP repeatedly allocates the next slot to the unallocated, eligible bidder (if any) with the highest bid price, breaking ties at random. Let \( A_i(b, \hat{C}) \in \{1, \ldots, m\} \) denote the slot allocated to a winner, with \( A_i(b, \hat{C}) = 0 \) otherwise. Let \( i_k \in N \) denote the bidder (if any) allocated slot \( k \).

**Expressive GSP (eGSP)**

**Input:** bids \( b_1, \ldots, b_n \), constraints \( \hat{C}_1, \ldots, \hat{C}_n \)

**For** slot \( k = 1 \) to \( m \)

**Eligible** \( \leftarrow \{i: \text{allocating } k \text{ to } i \text{ satisfies } \hat{C}_{i_1}, \ldots, \hat{C}_{i_{k-1}}, \hat{C}_i\} \)
$i_k \leftarrow \max b_i$ in Eligible (if any)

**End**

The per-click price for bidder $i$ allocated to slot $k$ is

$$p_i(b, \hat{C}) = \min b'_i \text{ s.t. } A_i(b'_i, b_{-i}, \hat{C}) = k,$$

(2.5)
i.e., the smallest bid $b'_i$ (given $\hat{C}$) for which a bidder is allocated the same slot, with $b_{-i} = (b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_n)$.

### 2.4.1 Equilibrium Concepts

As is standard in sponsored search auctions, we study complete information Nash equilibrium:

**Definition 2.** Bid profile $(b, \hat{C})$ is a Nash equilibrium (NE) in eGSP if $\forall i$ and fixing the bids $b_{-i}$ and reported constraints $\hat{C}_{-i}$ of others, there is no report $(b'_i, C'_i)$ with higher utility for the bidder than $(b_i, \hat{C}_i)$ (given its true $v_i$ and $C_i$).

The motivation for studying complete information NE in sponsored search is that advertisers can learn each others’ types over time via bidding dynamics. Also of interest in sponsored search auctions is envy-free equilibrium. In the following, let $p_i$ denote the per-click price for bidder $i$ given the bid profile and $A_i$ denote the slot allocated to bidder $i$.

**Definition 3.** Bid profile $(b, \hat{C})$ is an envy-free (EF) equilibrium in eGSP if

1. for every allocated $i$, $\delta^{A_i-1}(v_i - p_j) \leq \delta^{A_i-1}(v_i - p_i)$ for all bidders $j \neq i$ for which the allocation would satisfy the constraints $C_i$ if $i$ and $j$ switch positions, and
2. for every unallocated bidder $i$, $\delta^{A_i-1}(v_i - p_j) \leq 0$, for all bidders $j$ for which the allocation would satisfy $C_i$ if $i$ was allocated $j$’s slot (with $j$ going unallocated).

The standard definition of envy-free [27] is not specific about the effect of $i$ receiving a different slot on the rest of the allocation. But this is crucial here because of externalities.
The envy-free equilibrium property captures a dynamic stability requirement. Consider an allocated bidder $i$. If (1) is violated then $i$ would like to compete with bidder $j$ for its allocated slot, to drive up $j$’s price, and without fear of $j$ retaliating by making $i$ take $j$’s slot at $j$’s price. Bidder $j$ can always do this by bidding just below $i$’s bid price, making $i$ win $j$’s slot at $i$’s bid price (which was in turn setting $j$’s price.) Similarly, consider an unallocated bidder $i$. If (2) is violated then this bidder would like to compete for $j$’s slot and do so without fear of $j$ bidding just below $i$ to make $i$ win the slot.

2.5 Incentives in eGSP

We will establish that eGSP is “semi-truthful”, namely that bidders cannot benefit from misreporting constraints.

**Greedy is incompatible with truthfulness.** Before continuing, we briefly explain by example why this property would not hold if one was to use a naive application of the standard payment rule used to achieve incentive-compatibility in auctions.

**Definition 4.** An allocation algorithm that takes bids and UBC constraints is monotonic if, for all $\hat{C}$, all $b_{-i}$, all $i$ and all $b_i$, then $A_i(b'_i, b_{-i}, \hat{C}) \leq A_i(b_i, b_{-i}, \hat{C})$, for all $b'_i \geq b_i$.

**Lemma 1.** The greedy algorithm is monotonic (with respect to bids) given unit-bidder constraints.

**Proof.** Fix constraints $\hat{C}$ and bids $b_{-i}$. Suppose bidder $i$ is allocated in slot $k$ for bid $b_i$. Then the bidder is at least allocated in slot $k$ when bidding $b'_i > b_i$ because if it remains unallocated when slot $k$ is allocated, the state of the algorithm is unchanged from when bid $b_i$ is submitted because earlier decisions are oblivious to the bid values of unallocated bidders. 

Definition 4 insists that a higher bid value can only lead to a higher slot (and thus a lower slot index). The greedy algorithm is monotonic in this sense. Fix reported constraints
\( \hat{C} \) and let \( f_i(b, \hat{C}) = \delta^{A_i(b, \hat{C})-1} \) while \( i \) wins, and 0 otherwise. Following Myerson [80], the standard approach to achieve a truthful auction would charge a winner \( i \) an (expected) payment for its allocation to slot \( A_i(b, \hat{C}) \) of,

\[
b_i \cdot f_i(b, \hat{C}) - \int_{w=0}^{b_i} f_i(w, b_{-i}, \hat{C}) \, dw
\]  

(2.6)

But we see from the next example that this would not provide truthfulness with respect to constraints.

**Example 1.** Consider 2 slots and bidders 1 and 2, with values 30, 20, where bidder 1’s true constraint precludes bidder 2 from appearing in the top slot when 1 is allocated (but is happy for 2 to appear below 1.) Discount \( \delta = 0.9 \). Bidder 1’s (expected) payment given this constraint is \( 30 - [(30 - 20)] = 20 \). If bidder 1 did not report this constraint, then it would still win slot 1 but with expected payment of \( 30 - [(30 - 20) + (20 - 0)(0.9)] = 2 \).

The difficulty in achieving truthfulness with this payment rule is that the expected payment (for the same slot) is not independent of the bidder’s report (namely reported constraints), a well-known condition for truthfulness. A bidder can pay less by omitting from its bid any constraints that leave the allocation unchanged but would constrain its allocation for lower bid values. Given the uniqueness of the Myerson payment rule in providing truthfulness in regard to the bid value for fixed constraints, we see that it is impossible to achieve full truthfulness with a greedy allocation method. The classical Vickrey-Clarke-Groves mechanism for achieving truthfulness is undesirable since in our case it requires solving NP-hard optimization problems. We show now however that eGSP achieves truthfulness for constraint reports for a large class within UBC.
2.5.1 Semi-Truthfulness

We turn now to the incentive properties of the next-price payment rule in eGSP. We also note that eGSP has the useful property that the price is invariant to bid price \( b_i \) while the allocated slot remains unchanged.

**Definition 5.** A slot auction with UBC constraints is semi-truthful if for any reported \( b_{\neq i} \) and \( \hat{C}_{\neq i} \), of other agents, any \( v_i \) and \( b_i \leq v_i \) of agent \( i \), it is a dominant strategy for agent \( i \) to report its constraints \( \hat{C}_i = C_i \) truthfully.

**Definition 6.** UBC \( C_i \) are downward-monotonic (DM) if

\[
(pos_i, j, pos_j) \in C_i \implies (pos_i + 1, j, pos_j) \in C_i \tag{2.7}
\]

Fixing the allocation to other bidders, if \( i \) is dissatisfied with slot \( pos_i \), then it is also dissatisfied with any lower slot.

**Theorem 1.** The eGSP auction is semi-truthful for bidders whose value externalities can be expressed with downward-monotonic UBC constraints.

**Proof.** Fix any \( b_i \leq v_i \). Let \( k \) denote the slot allocated to \( i \) when reporting true \( C_i \). Conditioned on report \( \hat{C}_i \) not changing the allocated slot \( k \), the payment does not change because constraints have no effect on other bidders until a bidder is allocated and so the eligible set is unchanged. Moreover, if \( i \) is allocated a slot \( k \) then by reporting \( \hat{C}_i \neq C_i \) it cannot be allocated a higher slot (for the same bid value) because it is already eligible for slot \( k \) and thus all higher slots by DM. Also, \( i \) cannot achieve a lower slot by misreport \( \hat{C}_i \neq C_i \), fixing its bid value, because any change to preclude \( i \) from being eligible for slot \( k \) will preclude \( i \) from being eligible for all subsequent slots \( k' > k \) by DM. Finally, an agent that is unallocated but becomes allocated to slot \( k' \) by reporting \( \hat{C}_i \neq C_i \) must have a true constraint that is violated upon allocation to slot \( k' \), since the allocation for earlier slots, \([1, \ldots, k' - 1]\), does not change. \( \square \)
Lemma 2. Identity-specific constraints and slot-specific constraints satisfy DM.

Proof. For identity-specific constraints, \((k_i, j, k_j) \in C_i \Rightarrow (k_i + 1, j, k_j) \in C_i\) for \(k_i < m\).

Intuitively, if \(i\) in slot \(k_i\) is placed below \(j\) then the same holds for \(i\) in slot \(k_i + 1\). For a bidder \(i\) who wants placement above \(j\), if \(j\) is in slot \(k_j\) thereby requiring that \(i\) not be in slot \(k_i\) then \(i\) will not want to be in a lower slot \(k_i + 1\).

Since slot-specific constraints specify the cutoff that bidder \(i\) should be no lower than some slot and bidders \(j\) should be no higher than some bidder-specific slots, DM immediately follows.

Thus identity-specific and slot-specific constraints comprise natural DM classes of UBC. DM is also necessary for eGSP’s semi-truthfulness as demonstrated by the following example.

Example 2. Consider 3 slots and 3 bidders with values 60, 40 and 10, and discount \(\delta = 0.9\).

If bidder 1 is truthful then he wins slot 1 and pays 40 for payoff \(60 - 40 = 20\). But by reporting constraint “I do not want slot 1”, he wins slot 2 and pays 10 for payoff \((60 - 10)0.9 = 45 > 20\). This constraint is not DM.

Achieving truthfulness for constraints, as we do here, is important since standard GSP is not truthful (for bids only). But the next-price payment rule is nevertheless interesting because similar to the rule used in current practice, it provides local stability so that a bidder can only improve the outcome by deviating enough to change the slot allocated.

2.6 Equilibria in eGSP for Exclusion Constraints

We focus now on a subset of DM UBC constraints and establish a clean separation between the existence of Nash equilibrium in eGSP and its inexistence in standard GSP. For this, we focus on max degree one exclusion constraints, so that bidder \(i\)’s bid is only valid when
is not allocated and so at most one of \(i\) and \(i'\) can be allocated. Max-degree one exclusion constraints insist that each bidder participates in at most one such constraint (with the terminology coming from the natural graph-theoretic interpretation), and the exclusion is reciprocal. We denote an exclusion constraint between \(i, i'\) as \(i \leftrightarrow i'\).

\[\textbf{Theorem 2.} \text{ Standard GSP may have no pure-strategy NE for bidders with max-degree one exclusion constraints.}\]

\[\text{Proof.} \text{ Consider 2 slots and 4 bidders } 1, 2, 3, 4 \text{ with } 1 \leftrightarrow 2, 3 \leftrightarrow 4 \text{ and values } v_1 = v_2 + \varepsilon, v_3 = v_4 + \varepsilon \text{ such that } v_1 > v_3 \text{ and } v_1 - v_4 < \delta(v_1 - 0). 1 \text{ must bid at least } v_2, \ \text{else} \ 2 \ \text{can win by bidding } v_2. \ \text{Similarly } 3 \ \text{must bid at least } v_4. \ \text{In response, bidders } 2 \ \text{and } 4 \ \text{must drop out of the auction. Then } \{1, 3\} \ \text{win slots } \{1, 2\}. \ \text{But by the assumption on } \delta, \ \text{bidder } 1 \ \text{prefers bidding below } 3, \ \text{and so this cannot be an NE.} \]

Our main theoretical results in regard to the equilibrium properties of eGSP are:

\[\textbf{Theorem 3.} \text{ There exists an envy-free, Nash equilibrium of eGSP under max-degree one exclusion constraints.}\]

\[\textbf{Theorem 4.} \text{ There exists an envy-free equilibrium of eGSP under (general degree) exclusion constraints.}\]

We establish existence by reduction to GSP with bidder-specific reserve prices (rGSP) [28]. The proof of Theorem 3 is in Appendix A.2. A sketch of the proof is provided below.

In a significant deviation from the standard literature, envy-free equilibria and Nash equilibria are incomparable for eGSP – an envy-free equilibrium need not be a Nash equilibrium. In the standard GSP model however, any envy-free equilibrium is also a Nash equilibrium. rGSP operates just as standard GSP except that the price for slot \(k\) to bidder \(i\) is \(\max(r_i, b_{k+1})\), where \(b_{k+1} = 0\) if no bidder is allocated in slot \(k + 1\). Even-Dar et al. [28]
provide a tâtonnement algorithm to construct an envy-free (and, in their case, Nash) equilibrium for rGSP. They insist that \( b_i \geq r_i \) for all bidders (which is achieved through our reduction.)

For simplicity, we adopt in what follows the convention that the bidders are indexed according to the slots allocated.

**Definition 7.** A bid profile \( b \) is an envy-free (EF) equilibrium in rGSP given reserve prices \( r \) if 
\[
\delta^{i-1}(v_i - \max(p_j, r_i)) \leq \delta^{i-1}(v_i - \max(b_{i+1}, r_i)), \forall j \neq i.
\]

The utility to every bidder in an envy-free equilibrium of rGSP is at least what it would receive if it could exchange positions with any other bidder. An EF equilibrium of rGSP is also a NE. The reduction identifies a set of *candidates* and reserve prices, such that when an equilibrium is determined for rGSP on the candidates, we can construct a bid profile that is an equilibrium in eGSP. Non candidates will be unallocated in the equilibrium of eGSP. The technical challenge is to establish that the strategic effect of non-candidates on candidates in eGSP is equivalent to the effect of the reserve price on bidders in rGSP. The construction generates an envy-free equilibrium of eGSP that is also a Nash equilibrium for the special case of max-degree one exclusion.

Fleshing this out, assume for simplicity distinct values of bidders. Let \( E_i \) denote the *enemies* of \( i \). Namely, the set of bidders with which \( i \) has an exclusion constraint. We first determine a *pseudo outcome* \((K, X, r, \triangleright)\) of eGSP:

- Run eGSP with bids \( b = v \) and constraints \( \hat{C} = C \). The allocated bidders comprise the set of candidates \( K \).
- Let \( X_i \subseteq E_i \) denote bidders that are excluded when \( i \) is allocated, i.e., for whom no other enemy was allocated before \( i \). Define \( r_i = \max_{j \in X_i} \{v_j\} + \epsilon' \), for a small \( \epsilon' > 0 \) if \( X_i \neq \emptyset \) and \( r_i = 0 \) otherwise. \( \epsilon' \) is smaller than the minimum gap between bidder values.
• Define a priority order $\succ$, where $i \succ j$ if $i$ is allocated before $j$ and they share an enemy (necessarily a non-candidate).

If, e.g., $3 \leftrightarrow 2 \leftrightarrow 1$, where values equal IDs, then 3 and 1 are candidates, $X_3 = \{2\}$, $r_3 = 2 + \epsilon'$, $X_1 = \emptyset$, $r_1 = 0$, and $3 \succ 1$. If $4 \leftrightarrow 1, 3 \leftrightarrow 2$ then 4 and 3 are candidates, with $X_4 = \{1\}, X_3 = \{2\}, r_4 = 1 + \epsilon', r_3 = 2 + \epsilon'$ and no priority order constraints (the case for any max-degree one exclusion as each candidate excludes at most one bidder).

The following observation about the pseudo-outcome is straightforward and stated without proof:

**Lemma 3.** In a pseudo-outcome, $v_i > v_j$ and $r_i > v_j$ for candidates $i, j$ with $i \succ j$, and $v_i > r_i$ for all candidates $i$.

In constructing an envy-free NE for our expressive GSP, we will make use of a constructive method to identify envy-free NE for a reserve price GSP [28]. In keeping with the model in the present work, and to keep things simple, we define the reserve price GSP in which there is no bidder-specific quality factors. Even-Dar et al. [28] stipulate that $b_i \geq r_i$ for all bidders. This is very reasonable for our purposes, given that $v_i > r_i$ by Lemma 3, and will pose no difficulty in our analysis.

**Definition 8** (reserve price GSP). Consider a reserve price GSP in which $k \geq \ell$, for $\ell$ bidders and $k$ slots. Each bidder $i$ bids a per-click value $b_i \geq r_i$, for reserve price $r_i$. Allocate the bidders to slots in order of decreasing $b_i$ (breaking ties at random), where bids are assumed indexed so that bidder $i$ is in slot $i$ (with unallocated bidders given index $i > k$.) The price to bidder $i$ is $p_i = \max(r_i, b_{i+1})$, where $b_{i+1} = 0$ if no bidder is allocated to slot $i + 1$.

Consider now running the rGSP on the candidate bidders $K$ and with reserve prices $r$.

**Lemma 4.** In an EF equilibrium of rGSP on the pseudo-outcome, all higher priority $j \succ i$ for any candidate $i$ are allocated to a higher slot than $i$. 

32
Proof. We have $b_j \geq r_j > v_i$ (where the first inequality must hold if $j$ is allocated and thus active). For $i$ to be allocated to a higher slot than $j$, we must have $b_i \geq b_j$ and so the price $p_i \geq b_j > v_i$ and bidder $i$ would have negative utility. This cannot be an envy-free equilibrium in rGSP.

Denote by $b^r$ the envy-free equilibrium of rGSP (a bid profile for the candidates $K$). Based on this, we construct an equilibrium bid profile $b^*$ for eGSP as follows:

- All constraints $\hat{C} = C$ are reported truthfully.
- Bid $b_i^* = b^r_i$ if $i \in K$ and $b_i^* = v_i$ otherwise.

By the consistent ordering property of Lemma 4, we have

**Lemma 5.** Given bids $b^*$ and $C = (C_1, \ldots, C_n)$ reported to eGSP, the outcome is identical to that of rGSP on the pseudo-outcome, under a particular tie-breaking rule.

The proof is by induction on position allocated, from top to bottom (see Appendix A.1). The essential insight is that while the bid ordering corresponding to $b^r$ need not respect the truthful ordering assumed in determining the pseudo-outcome of eGSP, it is sufficient for the bids to respect the priority constraints across bidders that share mutual enemies for the strategic equivalence between bids of non-candidates and reserve prices to hold.

In completing the proof for Theorem 3, it is necessary to establish that the Nash constraints hold for all bidders, both candidates and non-candidates. For a non-candidate, we can start with the observation that the price of an allocated enemy is at least the reserve price of the enemy, which is at least the value of the non-candidate by Lemma 3. For a candidate, in considering the Nash constraint to preclude the benefit from a deviation to a higher slot, we inherit this result from the NE of the $b^r$ equilibrium in rGSP. For a downwards deviation, one needs to also argue that the (effective) reserve price (as induced by bids of non-candidates) remains as assumed in the rGSP equilibrium analysis. This part of the
reduction needs the assumption of max-degree one. Theorem 4 on EF equilibrium follows
directly from the outcome equivalence between rGSP on \( b' \) and eGSP on \( b^* \) (Lemma 5).

To understand why the max-degree one requirement is important to obtain NE, consider
an instance with 3 bidders, values 8, 4 and 6 and exclusion constraints \( 8 \leftrightarrow 4 \leftrightarrow 6 \). Let bidder
values equal IDs. We get pseudo-outcome \( K = \{8, 6\}, X_8 = \{4\}, X_6 = \emptyset, r_8 = 4, r_6 = 0, \)
and \( 8 \succ 6 \). The corresponding instance of rGSP has values 8, 6 and reserve price 4 and 0. An
EF equilibrium is \( b' = (8, 6) \) for \( \delta < 1/2 \). In this case, we have \( 8 - 6 > \delta(8 - \max(4, 0)) \)
and \( \delta(6 - 0) > (6 - 6) \). The candidate equilibrium in eGSP is \( b^* = (8, 4, 6) \), and indeed,
the outcome is equivalent to that of rGSP under these bids, with 8 and 6 allocated slots 1 and
2, at prices 6 and 0 respectively. However \( b^* \) is an EF but not a Nash equilibrium in eGSP.
Bidder 8 can bid 5 instead, in which case 6 is allocated, 4 is eliminated, leaving 8 to receive
slot 2 for price 0. For \( \delta > 1/4 \) this is better for bidder 8, with \( 8 - 6 < \delta(8 - 0) \). The effect of
non-candidate 4 is no longer strategically equivalent to a reserve price of 4 to bidder 8 when
8 deviates downwards and plays after bidder 6 in eGSP.

2.7 Algorithmic Considerations

The winner determination problem, to select the bids that maximize total value given con-
straints, is NP-hard. For exclusion constraints the problem is equivalent to \textsc{IndependentSet}.
This, together with the need for fast algorithms for slot auctions, motivates the greedy algo-
thesis. We first bound the approximation ratio (for social welfare) of the greedy algorithm.
The result is stated for any UBC constraints, not only exclusion ones.

**Definition 9.** A constraint graph is defined as a directed graph \( G \), with bidders as vertices,
in which there is an edge from \( i \) to \( i' \) if \( C_i \) contains at least one constraint \( (\text{pos}_i, i', \text{pos}_{i'}) \)
with \( \text{pos}_i > \text{pos}_{i'} \): if \( i \) is in \( \text{pos}_i \) then \( i' \) cannot be in some higher slot \( \text{pos}_{i'} \). Denote the
corresponding value maximization problem by \( \text{WDP}^\text{pre}_C \).
Theorem 5. Let \( d \) denote an upper bound on all vertices’ in-degrees in \( G \). The greedy algorithm for the WDP\(_C^{pre} \) problem achieves an \( \frac{1-\delta}{1-\delta^{d+1}} \) approximation\(^1\) when \( \delta < 1 \). Formally, letting \( b(W, A) \) be the social welfare in the greedy algorithm,

\[
b(W, A) \geq \text{OPT}(b, C) \cdot \frac{1-\delta}{1-\delta^{d+1}}
\]

where \( \text{OPT}(b, C) \) is the maximum allocation value given bids \( b = (b_1, \ldots, b_n) \) and constraints \( C = (C_1, \ldots, C_n) \).

Note that the graph in Theorem 5 is well-defined for any set of UBC constraints. In the special case of exclusion constraints, the in-degree bound is given by the number of exclusion constraints in which an agent \( i \) can participate. For identity-specific constraints, it is a bound on the number of other agents for whom agent \( i \) is considered an enemy. The proof of Theorem 5 is in Appendix A.4.

The ratio in the theorem is tight. Consider a setting with \( d + 1 \) bidders where every bidder in \( \{1, \ldots, d\} \) bids 1 and has \( d + 1 \) as common enemy: \( C_i = \{d+1\} \). Bidder \( d + 1 \) bids 1.01 and has \( C_i = \emptyset \). Then the greedy algorithm achieves value 1.01 while the optimal solution achieves value \( 1 + 1\delta + \ldots + 1\delta^{d-1} + 1.01\delta^d \). For a general graph (arbitrary \( d \)) and \( \delta < 1 \) we recover the trivial approximation bound \( (1-\delta) \). For exclusion constraints this dependence is intrinsic: INDEPENDENTSET is NP-hard to approximate within a \( 2^{O(\sqrt{\log d})/d} \) factor [88]. For max-degree one exclusion, we can conclude that the greedy algorithm is optimal.

A drawback of this solution is that the approximation ratio is stated in terms of a bound on the indegree of the problem. By Theorem 6 in Sec. 2.7.1, a certain dependence of approximation ratios on the constraint graph structure is however unavoidable. Whereas a bidding language can easily constrain the outdegree of a graph by limiting the number of constraints

---

\(^1\)An algorithm is said to achieve a \( \rho \)-approximation if the value of the allocation it outputs is within a multiplicative factor of \( \rho \leq 1 \) of the value of the optimal solution, for all possible instances.
a bidder is allowed, it is more difficult to see how to control a priori the maximum indegree.

2.7.1 Complexity on Bounded-Degree Graphs

In this section, we provide computational hardness results for \( \text{WDP}^\text{pre}_C \) with equal bids by a unified reduction from two well-studied graph-theoretical problems. For the reduction, we adopt the following construction of an instance of \( \text{WDP}^\text{pre}_C \) from a given graph \( G = (\{1, \ldots, n\}, E) \): construct an \( \text{WDP}^\text{pre}_C \) instance where each vertex \( i \in N \) with bid \( b_i = 1 \), and for each edge \( a = (i, j) \in E \) we place \( i \in C_j \) (note that if \( G \) is undirected we also place \( j \in C_i \)), that is, we add the corresponding edge(s) to the constraint graph \( G_c \). Clearly, a set of winners \( W \) in \( \text{WDP}^\text{pre}_C \) is feasible if and only if \( W \) is an acyclic set in \( G \) (if \( G \) is directed) and an independent set in \( G \) (if \( G \) is undirected). If so, then there is a feasible allocation of slots to \( i \in W \): by Lemma 25 in Appendix A.3 if \( G \) is directed and by the lack of any constraints within \( W \) if \( G \) is undirected. Since all bids are equal, the value-maximizing \( W \) is a maximal set with the corresponding property in \( G \).

Thus, for any \( \delta \in (0, 1] \), the \( \text{WDP}^\text{pre}_C \) problem is NP-hard for general constraint graphs by the immediate reduction from \( \text{INDEPENDENT SET} \). Moreover, this provides an inapproximability lower bound for \( \delta = 1 \), namely \( \min(n^{1-\epsilon}, \frac{\epsilon^{1/2-\epsilon}}{2}) \) for any fixed \( \epsilon > 0 \), where \( c \) is the number of constraints [46], relying on the \( \text{NP} \neq \text{ZPP} \) complexity assumption. We obtain instead two statements we regard as more informative for practical settings where the number of constraints expressed by a bidder is bounded. Let \( \text{WDP}^\text{pre}_d \) denote the restriction of \( \text{WDP}^\text{pre}_C \) to constraint graphs with degree at most \( d \) and equal bids. Via an analogous hardness of approximation result [15] for \( \text{INDEPENDENT SET} \) on bounded-degree graphs relying on the \textsc{Unique Games} conjecture [63] our construction yields:

**Theorem 6.** Let \( l(d) = \frac{\log^2 d}{d} \) and \( \phi(d) = \frac{1-\delta^1+(\delta^{-1})^l(d)}{1-\delta^n} = \frac{1+\delta+\ldots+\delta^{(n-1)}l(d)}{1+\delta+\ldots+\delta^{(n-1)}} \). Symmetric (\( i \in C_i \iff i' \in C_i \)) \( \text{WDP}^\text{pre}_d \) is \textsc{Unique Games}-hard to approximate to within a \( O(l(d)) \).
factor for $\delta = 1$ and to within a $O(\phi(d))$ factor for $\delta < 1$.

**Proof.** The result for $\delta = 1$ follows from our construction and [15]. We prove that for $\delta < 1$, any algorithm $A$ with a better approximation factor on any instance must have a better approximation factor than $l(d)$ for $\delta = 1$. Consider an instance with maximum independent set of size $s \leq n$. We claim that $A$ must output an independent set of size at least $s' \geq s \cdot l(d)$.

Otherwise the approximation ratio of $A$ is at most $\frac{1+\delta+\ldots+\delta s l(d)}{1+\delta+\ldots+\delta s}$ which in turn is at most

$$\frac{1+\delta+\ldots+\delta (n-1) l(d)}{1+\delta+\ldots+\delta (n-1)}.$$

In fact, the problem is computationally challenging even for a bound of 2 on the in-degree and out-degree of each vertex.

**Theorem 7.** For all $\delta \in (0, 1]$, $WDP_2^{\text{pre}}$ is NP-hard.

**Proof.** We use the NP-hardness [31] of MINIMUMFEEDBACKVERTEXSET (MFVS) under the same in-degree and out-degree bound of 2. Recall that the MFVS problem is to determine a minimum set of vertices $S$ whose removal makes a given graph $(V, E)$ acyclic. Our construction provides a simple reduction from the MFVS problem to the $WDP_2^{\text{pre}}$ problem: $S$ is a MFVS in a directed graph $G$ if and only if $V \setminus S$ is a maximal acyclic set in $G$, i.e. a bidder set whose allocation maximizes value given constraint sets $C_i$ for $i \in \{1, \ldots, n\}$. □

Structural observations about the constraint graph are contained in Appendix A.3.

### 2.8 Fixed-Parameter Algorithms

As positive results for exact winner determination, we identify two parametrized subclasses of constraints with tractable algorithms for $WDP_2^{\text{pre}}$. The complexity of these algorithms is polynomial for any fixed parameter value.
2.8.1 Category-specific Constraints

The category-specific model is a special case of the identity-specific model in which every bidder is associated with a category and value externalities are limited to choosing to require placement above all bidders in the same category as a bidder\(^2\). The category-specific model moves towards a more anonymous setting and provides additional structure to enable an optimal algorithm.

The algorithm presented below computes an optimal allocation in polynomial time in \(m\) and \(n\) if the number of categories \(g\) is a constant. To motivate this model, suppose the user query is “cleats” which is a specific type of sports shoes. For this query, there will be bidders for exact match (e.g., Nike and Adidas bidding precisely on cleats) and bidders for broad match (e.g. Amazon bidding on sports shoes). In this example bidders belong either to the exact match category or the broad match category and would be able to express externalities only in these terms.

Let \(G = \{1, \ldots, g\}\) represent the categories, defining a partition of the bidders \(N\). Let \(c_i \in G\) denote the category of bidder \(i \in N\). Each bidder is offered a binary choice when submitting a bid, of having constraints with respect to all other bidders that belong to the same category or having no constraints at all. Let \(F_c \subseteq N\) denote the set of bidders in category \(c \in G\) who have chosen to target all other bidders in category \(c\) through constraints. In a feasible allocation only one bidder in \(F_c\) can be allocated and clearly it is the maximum value bidder \(f_{max}^c\). Let \(F = \bigcup_{c \in G} f_{max}^c\). Let \(Q_c\) denote the set of bidders in category \(c\) who have no constraints. Let \(Q = \bigcup_{c \in G} Q_c\). Finally let \(Q_{free} = \bigcup_{c \in G} Q_c\) such that \(F_c = \emptyset\). In other words \(Q_{free}\) is the set of free bidders who are nobody’s “enemies”. Let \(S\) be the list containing chosen slot positions. Note that we assume, in the algorithms described next, that

\(^2\)Notice that if each bidder is a category unto himself then we are back in the identity-specific model. However for a number of categories strictly less than the number of bidders, the category-specific model is a restriction.
once bidders are allocated they are removed from the sets to which they belong.

**Algorithm** AllocateCategories($G, F, Q$)

- **For** each permutation $F' \subseteq F$.
  - **For** each slot combination $S$ for $F'$.
    - Run Subroutine($G, F', Q, S$).
    - Store the resulting allocation.
  - **End**
- **End**
- Output the allocation of highest value.

**Algorithm** Subroutine($G, F', Q, S$)

- Initialize $j = 0$, $c = 0$, $Q_0 = \emptyset$.
- Build and sort $Q_{free}$ in decreasing bid order.
- Sort $S$ to list slots in order of increasing index.
- **While** there remain available slots and unallocated bidders
  1. Update $Q_{free} = Q_{free} \cup Q_c$.
  2. Place next bidder $f \in F'$ in slot determined by the next unused slot position in $S$. Let $c$ represent $f$’s category.
  3. Allocate top bidders $i \in Q_{free}$ in decreasing bid order in free slots above $f$.
- **End**

**Proposition 1.** Algorithm AllocateCategories outputs the optimal allocation.

**Proof.** The algorithm enumerates and compares all feasible allocations involving max value bidders $f_{max}^c \forall c \in G$ and bidders in $Q_c \forall c \in G$. The only allocations not considered are those involving lower value bidders in $F_c$ who impose the same constraints as $f_{max}^c$. Other than due to constraints, no low bidder can be placed right above a higher bidder. This reduces the space of candidate optimal allocations to only the ones we consider.

We introduce some notation to analyze the runtime of AllocateCategories. Let $P(z, t)$ denote the number of $t$-permutations of a set of $z$ elements. $P(z, t) = \frac{z!}{(z-t)!}$. In order to enumerate all feasible allocations involving max value bidders $f_{max}^c \forall c \in G$, each permutation of bidders within a particular subset $F'$ must be evaluated.
Proposition 2. If $g$, $n$, and $m$ represent the number of categories, the number of bidders, and the number of slots respectively, then an optimal allocation can be computed in time $O((n \log n + gn)(m^g)(g^g))$.

Proof. Sorting $Q_{\text{free}}$ and $S$ takes $O(n \log n)$. Steps 1-3 in Subroutine take $O(n)$. The while loop is run at most $g$ times. Hence Subroutine takes $O(n \log n + gn)$. The number of slot combinations for each subset $F'$ is $\binom{m}{|F'|}$ where $|F'| \leq g$. $\binom{m}{g} \leq m^g$. Since the permutation of the subset $F'$ is important in determining the optimal allocation, the total number of subsets (denoted $TF$) is $P(g, 1) + P(g, 2) + \ldots + P(g, g-1) + g! + 1$. Hence $TF = g! + \sum_{h=1}^{g-1} P(g, h) \leq g(g!) \leq g^g$. 

2.8.2 Local-exclusion Constraints

Suppose bidders only have exclusion constraints to bidders within some distance $L$ in the bid ranking. $L$ is a locality measure that turns out to be the tree-width of the constraint graph, a standard algorithmic concept. This allows tractable algorithms for constant $L$.

By relabeling bidders, assume that bids are sorted decreasingly: $b_i \geq b_{i+1}, \forall i \in 1..n-1$. Clearly, winners are allocated in this order. Otherwise the allocation could be improved, while still feasible, by swapping slots.

Lemma 6. If $i, i' \in W$, for $i < i'$ then $A_i < A_{i'}$.

We provide a dynamic programming approach for the associated WDP based on a standard tree-width decomposition technique (see for example [81]). This approach has time and space complexity exponential in the tree-width of the graph, a quantity associated in our model with a locality measure of the constraints, to be defined shortly. Our algorithm is thus reasonably efficient for constraint graphs with a local structure.

Let $L$ be a constraint locality measure such that if $(i, i') \in C$ then $|i-i'| \leq L-1$ (note that indices do not wrap around). $L$ measures how far apart can two mutually excluded
bidders be in the sorted order of bids. Lemma 6 is the critical property enabling DP in this context; in particular if the highest bidder 1 is allocated (necessarily in the first slot) then in all slots bidders in 2..L only need to be excluded because of 1. Lemma 6 (and this approach) fails if constraints are not symmetric (i.e. the WDP\(_c^{pre}\) problem).

Let an \((i, L)\)-byte be an \(L\)-digit binary word \(B : \{i, \ldots, i+L-1\} \to \{0, 1\}\), where \(B(i+\ell)\) for \(\ell \in \{0, \ldots, L-1\}\) indicates whether bidder \(i+\ell\) is allocated. Let \(#_B^c(\ell)\) denote the number of 1s in \(B(i..i+\ell-1)\). An \((i, L)\)-byte \(B\) is feasible if no constraints are violated, i.e. if \((i_1, i_2) \notin C\) whenever \(B(i_1) = B(i_2) = 1\).

We use \(m^c_i \in \{1, \ldots, m\}\), \(i \geq 1\), for storing how many bidders are allocated before \(i\) in the corresponding optimum. For each state \(X_i = \{i, \ldots, i+L-1\}\), we compute in a table called \(T_i\) via DP the value of each allocation given the optimum solutions in table \(T_{i-1}\). The value of the optimal allocation is found in \(T_{n-L+1}\) and this allocation can be traced back in lower-indexed states via standard DP techniques.

**Initialization.** For all \((1, L)\)-bytes \(B\), let \(m^c_1 = 0\) and initialize in \(X_1\) the total value of allocating according to \(B\)

\[
v_{X_1}(B) = \begin{cases} 
\sum_{i=1}^{L} B(i) \cdot b_i \cdot \delta^{#_B^c(i)}, & \text{if } B \text{ feasible} \\
-\infty, & \text{otherwise}
\end{cases}
\]  
(2.9)

**Dynamic programming.** We describe how to populate \(T_i\) from \(T_{i-1}\) where \(i \geq 2\). Table 2.1 illustrates \(L = 2\) and \(i = 2\). \(^3\)

For all infeasible \((i, L)\)-bytes \(B\) let \(v_{X_i}(B) = -\infty\). Fix now a feasible \((i, L)\)-byte \(B\). For \(\beta \in \{0, 1\}\), let \(B_{\beta} = \beta B_{i+L}^-\) be the \((i-1, L)\)-byte obtained by prepending the bit \(\beta\) to \(B\) (i.e. \(B_{\beta}(i-1) = \beta\)) and deleting \(B\)’s last bit.

\(^3\)\(L = 1\) amounts to no constraints. The greedy algorithm allocating bidders in decreasing order of their bids and removing excluded bidders is always optimal for \(L = 2\) but not for \(L = 3\): consider \(C_1' = \{(1,2), (1,3)\}\) with \(\delta = 1\) and \(b_i' = 1 - (i-1)\varepsilon\) for \(i = 1, 2, 3\).
Table 2.1: Exclusion-only dynamic programming example with $C = \{(1, 2), (2, 3)\}$ and mutually excluded bidders closer than $L = 2$.

<table>
<thead>
<tr>
<th>11+1 $m_i^&lt; v_{X_1}$</th>
<th>2 2+1 $m_i^&lt; v_{X_2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0/1 $\max{0, b_1}$</td>
</tr>
<tr>
<td>0 1 0 $b_2$</td>
<td>0 1 0/1 $\max{b_1 + \delta^1 + b_3, b_2}$</td>
</tr>
<tr>
<td>1 0 0 $b_1$</td>
<td>1 0 0 $\max{b_2, -\infty}$</td>
</tr>
<tr>
<td>1 1 0 $-\infty$</td>
<td>1 1 0 $-\infty$</td>
</tr>
</tbody>
</table>

Table $T_1$ for state $X_1$ Table $T_2$ for state $X_2$ from table $T_1$

$B$’s value is the best value from either allocating $i - 1$ or not (i.e. $B_1$’s and $B_0$’s values), plus the consequent value $V^+(\beta)$ resulting from whether $B(i + L - 1) = 1$, i.e. whether $i + L - 1$ is allocated. Critically for Eq. (2.10), bidders $i - 1$ and $i + L - 1$ cannot exclude each other. $V^+(\beta)$ depends on $\beta$ only by how many bidders in $1..i-1$ are allocated in $v_{X_{i-1}}(B_\beta)$. $\delta$’s exponent in Eq. (2.11) plus one equals $i + L - 1$’s slot number $A_{i+L-1}$, i.e. one plus the number of allocated higher bidders: $m_i^<(B_\beta)$ in $1..i-1$ and $\#_B(L)$ in $i..i + L - 2$.

\[
v_{X_i}(B) = \max_{\beta \in \{0, 1\}} \{v_{X_{i-1}}(B_\beta) + V^+(\beta)\} \quad \text{where} \quad (2.10)
\]

\[
V^+(\beta) = B(i + L - 1) \cdot b_{i+L-1} \cdot \delta^{m_{i-1}^<(B_\beta)+\#_B^<(L)} \quad (2.11)
\]

$m_i^<(B)$ is also updated according to the max in Eq. (2.10). We get a DP time and space complexity exponential in $L$.

**Proposition 3.** For constraint locality $L$, the dynamic programming technique in Eqs. (2.9) and (2.10) correctly computes optimal allocation $OPT$ in $O(n2^L + m)$ time and $O(2^L + m)$ space. $OPT$’s value can be recovered from table $T_{n-L+1}$.

**Proof.** Only table $T_{i-1}$ is needed for updating $T_i$. Each table has size $2^L$ and one entry in $T_i$ is computed in Eq. (2.10) in constant time\(^4\) from two entries in $T_{i-1}$.

\(^4\) $\delta$’s power can be looked up in a pre-computed vector $\delta^{2^m}$.
2.9 Soft Constraints

A natural extension is a soft constraint model where, in addition to constraints $C_i$, a bidder has a pair $(v_i, v_i^-)$ of per-click values, defining its value when no constraints, or at least one constraint in $C_i$ is violated, respectively. The standard hard constraint model has $v_i^- = 0$.

The eGSP auction is generalized as follows: in allocating the next slot, the eligible bidders are those for whom the allocation would not violate a first constraint for an already allocated bidder. The approximation ratio in Theorem 5 continues to hold (see Appendix A.5). For an eligible $i$, the price adopted is then either (1) $b_i$ or (2) $b_i^-$, depending on whether or not $i$’s constraints are still satisfied given the current allocation. If allocated slot $k$, then the price is the minimal value of $b_i$ or $b_i^-$, for case (1) or (2) respectively, such that bidder $i$ would still retain the same slot.

Semi-truthfulness no longer holds for soft constraints even with DM constraints:

**Example 3.** Consider 4 bidders and 3 slots, with bids $(100, 100c)$, $(70, 70c)$, $(50, 50c)$, and $(30, 30c)$ where $c = 0.5$. Let the discount factor be $1 - \epsilon$, for a small $\epsilon > 0$. If no one has any constraints then eGSP allocates bidders 1, 2, 3 to slots 1, 2, 3. In particular bidder 2 pays 50 for a utility of $70 - 50 = 20$. Now suppose bidder 2 lies and specifies a constraint stating it must be placed above bidder 1. Then eGSP allocates bidders 1, 3, 2 to slots 1, 2, 3. Bidder 2 now pays 30; its utility is $70 - 30 = 40$. Bidder 2 can achieve the same result by misreporting its values instead as $(40, 40c)$.

The eGSP auction is semi-truthful in the case of DM UBC hard constraints, meaning that it is a dominant strategy for bidders to report their constraints truthfully. Thus eGSP for hard constraints introduces no new manipulations beyond what already exists in standard GSP. While this “strong” form of semi-truthfulness no longer holds with soft constraints, we can recover a weaker form of semi-truthfulness. The weaker form of semi-truthfulness stipulates that for any report containing a set of untruthful constraints, there exists another report with
a set of truthful constraints and such that the bidder is no worse off under this latter report. Interestingly, this result does not require downward monotonicity, and extends easily to the earlier hard constraint model.

**Theorem 8.** In eGSP with soft-constraints, a bidder $i$ always has a best-response, for any $C_i$, any $(v_i, v_i^-)$, and any reports $(b_{-i}, \hat{C}_{-i})$, of other bidders, in which $i$ reports $C_i$ truthfully along with some pair $(b_i, b_i^-)$ of bid values.

**Proof.** Fix reports (constraints and bids) of other bidders. Consider bidder $i$. First observe that, conditioned on bidder $i$ being allocated in slot $k$, then the allocations of bidders to earlier slots is invariant to changes in the bid and reported constraints from $i$. For every slot $k' = 1, \ldots, k - 1$, the constraints and bid values from $i$ are irrelevant in deciding which of the other bids are allocated (while $i$ remains unallocated.) Because of this, then it is immediate to verify that the price to bidder $i$, conditioned on being allocated to slot $k$, is invariant to its reported bid and constraints. If its reported constraint holds (c.f. fails to hold), given the allocation to slots $1, \ldots, k - 1$, then the price it faces is $\min b_i$ (c.f. $\min b_{-i}^-$) such that it retains slot $k$. Either way, the price is the same because the calculation is unchanged.

To complete the proof, we just need to show that for any first report $\hat{C}_i$ and bid values $b'_i, b_i^-$, there exists a second report, with bid values $b_i, b_i^-$ and $C_i$, such that: (1) the slot allocated is unchanged (and thus the price is unchanged), and (2) the subsequent allocation decisions are no worse under the second report than the first report. Proceed by case analysis.

(Case 1.) $\hat{C}_i$ is violated but $C_i$ holds when $i$ is allocated to slot $k$ under report one. Set $b_i = b_i^- = b'_i$. Slot $k$ is still allocated because the bid value with the second report is only lower while considering higher slots, and the same for the allocated slot. Second, the subsequent allocation is only improved under $C_i$ than $\hat{C}_i$ because $C_i$ holds when allocated and so no other subsequent allocation will be made that bidder $i$ considers undesirable. (Case 2.) Both $\hat{C}_i$ and $C_i$ are violated when $i$ is allocated to slot $k$ under report one. In this case, we again set $b_i = b_i^- = b'_i$. Slot $k$ is still allocated by the same argument as for case 1. Now,
the subsequent allocation is the same because both constraint sets are already violated when
i is allocated. (Case 3.) Constraints $\hat{C}_i$ hold but $C_i$ do not hold when i is allocated to slot $k$
under report one. In this case, set $b_i = b_i^- = b_i'$. The same slot is allocated because the bid
value of i is unchanged until slot $k$ is allocated. The subsequent allocation dynamics might
change, but $C_i$ is already violated anyway and so $i$ is indifferent. (Case 4.) Constraints $\hat{C}_i$
and $C_i$ both hold. In this case, set $b_i = b_i'$, with $b_i^-$ irrelevant. The same slot is allocated,
and to the extent that the subsequent dynamics differ, $i$ prefers them under $\hat{C}_i$ because this
precludes any future allocation that $i$ considers undesirable. □

The idea of the proof is to establish that for any report with an untruthful constraint
set, there exists a report, with bid values $b_i, b_i^-$ and truthful constraint set $C_i$, such that the
slot allocated is unchanged (and thus the price is unchanged), and the subsequent allocation
decisions are no worse under the second report than the first report.

2.10 Conclusion

The problem of value externalities in auctions is relatively little studied. Unlike quantity
externalities, value externalities cannot be estimated by a search engine. Hence capturing
bidders’ value externalities must necessarily entail designing ways in which they can express
this value. To this end we have developed a constraint-based model of the effect of co-
location on a bidder’s value, conditioned on receiving a click.

We have introduced unit-bidder constraints, an expressive language for negative value
externalities in position auctions, and studied the algorithmic and incentive properties of
expressive GSP with downward-monotonic constraints, a class which encompasses exist-
ing models of externalities. We have designed expressive GSP, a semi-truthful mechanism
adapted to handle value externality constraints. We obtain a “semi-truthfulness” property
of eGSP with respect to misreports of downward-monotonic constraints. In this sense, the
modified eGSP is as truthful as the standard GSP and there are no new manipulations. We exhibit a class of such constraints for which Nash equilibria fail to exist in standard GSP, but exist and can be easily constructed in eGSP. We provide a constructive approach to prove the existence of Nash and envy-free equilibria for expressive GSP with exclusion constraints, a subclass of DM UBC. A weaker but still useful notion of truthfulness in regard to constraints is established for a generalization of UBC where bidders have a smaller but non-zero bid value for violated constraints. On the algorithmic side, we consider maximizing social welfare and give a tight bound on the approximation ratio of our algorithm for winner determination in general unit-bidder constraints. Interestingly we are able to identify parametrized subclasses of UBC with optimal polynomial time algorithms that optimize social welfare.

For future work, it would be interesting to characterize equilibria for more general UBC, thereby enabling revenue and efficiency comparisons to standard GSP. Turning to complexity results, an intriguing problem that remains open is that of the WDP$^\text{pre}_C$ problem where each bidder has a constraint against at most one other bidder (see Appendix A.6). We colloquially refer to this as the “one enemy" special case. We find this case appealing because it could be achieved through a simple restriction to a bidding language.

Another compelling direction is to expand on the possibility of connections with scheduling under precedence constraints (see Appendix A.7), which is equivalent to a version of our problem in which a subset of bidders are selected to be allocated but the order of allocation is to be determined.
Chapter 3

Predicting Your Own Effort

3.1 Introduction

Software engineering is one of many domains with complex and modular tasks. There are often information asymmetries, both between the worker performing a task and the manager supervising and between the two of them and the rest of the organization. In such environments, it is important for the organization to be able to elicit accurate predictions from worker-manager teams in regard to when individual tasks are expected to complete. Eliciting accurate predictions enables good decision-making in regard to scheduling resources to projects (such as bug fixes or new features), and coordinating different projects.

A particular challenge is that a worker with information relevant to the prediction task also controls the completion time through the amount of effort exerted on the task. There are informational externalities in this setting. For example, the information held by the worker might influence the prediction provided by the manager about the completion time of the task. In modelling this, we consider a single worker and a single manager. The worker works on a sequence of tasks and receives a score for each task. It is assumed that the organization is able to couple the total score received by a worker over some period of time
with incentives, be they social-psychological such as praise or visibility, or material rewards through prizes or the payment of bonuses. The particular mechanism for this is not modeled here. Rather, we assume that workers will be content to maximize total expected score within the context of an incentive mechanism designed to promote information sharing.

The role of the worker is to share information relevant to the expected completion time of the task with the manager, in order to enable accurate predictions, and also to decide on whether to work at “best effort” or less than best effort. The role of the manager is to combine information received from the worker with her own information (if any), and make accurate predictions to the organization regarding the completion time of tasks. We tackle the issue of how to elicit truthful information and thus accurate predictions from the worker and manager, as well as how to elicit best effort from the worker.¹

In essence, our problem is a combination of a repeated principal-agent problem and a prediction problem. In a principal-agent setting, a principal wishes to elicit a desired effort level from an agent but does not require the agent to make any predictions. On the other hand in a prediction problem, accurate predictions of the outcome of an event are sought but without considering that the distribution on outcomes might be something that can be controlled by the agent doing the prediction. In contrast we seek to establish both accuracy and the investment of best effort.

Our main technical result is a characterization of a class of scoring rules that are able to align incentives with both accurate prediction and the investment of best effort. In addition, the scoring rules inherently preclude the possibility of collusion between the worker and manager in their participation in the scoring system. For example, it is not useful for a manager and worker to agree that the worker will deliberately slow down in return for a

¹Note that we seek to introduce no new incentives for a change in behavior from best effort to less than best effort. The scoring system we design is not the primary incentive scheme. Rather it is layered on top, and should work with (not against) existing incentive schemes such as pay, promotion, and so forth within an organization.
prediction task with lower variance and thus the potential for higher total score to the worker-manager pair.

In addition, we consider the effect of a scoring system on whether or not a worker will choose to split a task into multiple prediction targets. For this purpose, we model a task as a sequence of subtasks, where a subtask is conceptualized as a unit of work with a well-defined end point, and for which the time to complete the unit of work may be informative as to the time to complete other subtasks that comprise a task. With this in mind, we study the incentives for a worker to “split-out” a subtask for the purpose of a separate prediction target. The qualitative result we obtain is that there is a greater propensity to split subtasks for which the completion times are positively correlated than those for which the completion times are independent.

Our viewpoint is that it is the worker, not the manager, who is privy to information in regard to subtasks. Moreover, we can imagine situations in which predictions in regard to subtasks rather than in regard to the aggregate time for a task is useful; e.g., for sharing information with other workers, for replanning, and in order to collect data to enable the training of predictive models in order to enable better organizational efficiency going forward. A simulation study completes the chapter, providing a quantitative analysis of the trade-off between the frequency of “splitting” prediction into subtasks, the degree to which the distribution on subtask completion time is correlated, and a parameterization of the scoring rule that affects how much payment is made per subtask target versus how much payment must be made in catch-up upon the completion of a task.

3.1.1 Related Work

Scoring rules have been developed to measure the performance of experts who are solicited to reveal their probability assessments regarding uncertain events. They have been used in a variety of scenarios, from weather forecasting to prediction markets [38, 45, 66, 78].
*Proper* scoring rules incentivize truthful reporting of likelihood estimates. An overview of the theory behind proper scoring rules can be found in Gneiting and Raftery [38]. Proper scoring rules typically require that the outcome of the uncertain event will be revealed and the agent whose assessment is elicited can not influence the outcome. In our setting, the prediction of effort required to complete a task and the outcome or realized effort are not independent; both are influenced by the worker. Shi *et al.* [92] consider situations where agents may be able to take actions that influence the outcome. They propose principal-aligned mechanisms that do not incentivize agents to take actions that reduce the utility of the principal. Their setting considers eliciting a probability distribution and the outcome space is discrete. Our setting allows for continuous effort level and we seek to elicit the expectation as well as incentivize best effort. The result of Shi *et al.* [92] can be generalized to the setting of eliciting the expectation for a random variable over a continuous outcome space using the characterization of Savage [89], which is also used to derive our characterization in Section 3.3. With this generalization, it is possible to derive our Theorem 10 by assigning a particular utility function to the principal and applying the result of Shi *et al.* [92]. However, we believe that this approach is unnecessarily complicated in our setting, so we derive our results by directly considering desirable properties of the incentive mechanism.

There is a vast literature on principal-agent models [29, 74]. In a classical principal-agent model with hidden action, an agent chooses an action to take that is costly for him but beneficial for the principal in exchange for a promise of payment. The principal cannot directly observe the agent’s action, but the stochastic correlation between actions and outcomes, i.e., the probability of observing an outcome given that the agent takes an action, is common knowledge. For example, the agent’s action can be a level of effort exerted with the probability of success for a project an increasing function of the level of effort. Knowing the stochastic correlation, the principal can incentivize the agent to take a desirable action using contracts with payments based on the outcome.
Radner [86] considers an infinitely repeated setting for the principal-agent problem. In Radner’s setting, the game is composed of sequences of review and penalty periods. By allowing the players’ actions in one period to depend on the history of previous periods, the principal can observe the results of the agent’s actions and *punish* the agent if the agent’s performance fails some statistical test of efficiency. Radner shows that for high enough discount factors, there exist equilibria, consisting of reward-decision pairs, of the infinitely repeated game that are strictly more efficient than the short-term equilibrium. Our setting is different in that it combines the challenge of eliciting desirable action and that of eliciting information from an agent. Our setting introduces information asymmetry about the stochastic correlation between the action and the outcome. The agent has some private information about this stochastic correlation. The principal would like to elicit the information from the agent so as to obtain a better prediction of the stochastic correlation, which is then used by the principal to set the reward for the agent. Because the reward of the agent now depends on his reported information, he may have incentives to lie about his information and/or act in a suboptimal way. Given this tension, the incentive design in this paper aims to achieve truthfully elicitation of the private information as well as the elicitation of the desirable action.
3.2 The Basic Model

We consider the incentive design problem for a company (the principal), whose goal is to truthfully elicit information from its employees as well as incentivize them to exert best effort. The basic model considers a single task with two agents, a worker and a manager, each with private information in regard to likely completion time of the task. The worker shares information with the manager, who then combines this information with his own private information and makes a prediction. The worker then exerts effort, and at some subsequent time, the task completes, and the worker informs the system (and the manager) of this event. We assume that only a truly completed task can be claimed as complete, but allow a worker to reduce effort below best effort, including to pretend a task is not complete when it has been completed. Eventually, a score is assigned to both the worker and the manager.

Let $X$ denote the random variable for the time the task takes to complete under the best effort by the worker. Assume that the realized value of $X$ is nonnegative and upper bounded, i.e., $x \in (0, x_{\text{max}})$. Neither the manager nor the worker knows the realization of $X$. But they each have some private information, denoted as random variables $I_m$ and $I_w$ respectively, on the completion time under best effort. The joint distribution $\Pi(X, I_m, I_w)$ is common knowledge to them, but not known to the company. Assuming $\Pi$ is not known to the company ensures a broad range of priors are considered possible. In particular, this allows $E[X|I_m]$, $E[X|I_w]$, and $E[X|I_m, I_w]$ all take on all values in $(0, x_{\text{max}})$. This ensures that rules we derive work for a broad range of beliefs, similar to proper scoring rules requiring truthful reporting be optimal for all probability distributions. If the company believes that only a significantly restricted set of priors is possible, there may be additional rules our results do not characterize. Note that these expectations are well defined because $X$ is bounded.

The manager and the worker play a three-stage game as shown in Figure 3.1. In stage 1, the worker can communicate with the manager and share information. In stage 2, the
manager makes a prediction $\hat{x}$ about the completion time of the task under the worker’s best effort. In stage 3, the worker exerts some effort and completes the task in time $x'$. While the worker cannot exert more than his best effort and complete the task in time less than $x$, he can work at a more slack pace and take time $x' > x$ to complete the task. However, we require that $x' \leq x_{\text{max}}$ because otherwise it will be clear to both the manager and organization that he is not working efficiently.

We assume that both the manager and the worker are risk neutral. We further assume that the worker, all things being equal, is indifferent between working at best effort or “slacking.” In other words, if the worker can get a higher expected score through a best-effort strategy rather than slowing down, then this is the approach the worker will take.\(^2\) We consider incentive mechanisms (we refer to them as scoring systems) that reward the manager and the worker based on the manager’s prediction of the completion time and the worker’s actual completion time. At the end of stage 3, a manager is rewarded according to the score $S_m(x', \hat{x})$ and the worker according to the score $S_w(x', \hat{x})$. We require $S_m$ and $S_w$ to be differentiable with respect to $x'$ and $\hat{x}$. The goal of a scoring system is to incentivize the report of an accurate prediction and the exertion of the best effort.

### 3.2.1 Desirable Properties of Scoring Systems

Our model is a simple two-player three-stage game. We hence consider the perfect Bayesian equilibrium of the game and desire good behavior of the manager and the worker at the equilibrium. The following are four properties we would like a scoring system to achieve at the equilibrium:

\(^2\)In practice, one can imagine that the scoring system exists in the context of existing incentives within the organization that provide for an existing preference for best effort over slacking. In this sense, our positive results continue to hold, in that we show that best effort is an equilibrium even if the worker is only indifferent between best effort and slacking.
1. **Information sharing.** For all $\Pi$, the worker shares his private information $I_w$ honestly with the manager in stage 1.

2. **Report-the-mean.** For all $\Pi$, when estimating the time required to complete a task under best effort of the worker, the manager’s optimal report in stage 2 is $\hat{x} = E[X|I]$ where $I$ is all information available to the manager at the time, given equilibrium beliefs.

3. **Best effort.** For all $\hat{x}$, it is optimal for the worker to exert his best effort and choose $x' = x$ for all realizations $x$ in stage 3.

4. **Collusion-proofness.** For all $\Pi$, the total expected score of the manager and the worker is maximized by reporting $\hat{x} = E[X|I_w, I_m]$ and exerting best effort such that $x' = x$ for all realizations $x$.

If the above four properties are satisfied, we will have a perfect Bayesian equilibrium where the worker shares all his information with the manager, the manager truthfully report his expectation of the completion time under best effort given both pieces of information, and the worker completes the task as quickly as possible, and this equilibrium is collusion-free, such that no joint deviation can lead to an increase in the total expected score.

### 3.3 Characterization of Scoring Systems

We proceed to characterize scoring systems that satisfy our desirable properties. The main technical challenge is to simultaneously address the need for accurate prediction and retain incentives for the worker to adopt best effort.

First, we consider the best effort property. It’s easy to see that if choosing $x' = x$ is optimal for the worker given any $x$ and prediction $\hat{x}$, the worker’s score $S_w(x', \hat{x})$ must be a decreasing function of $x'$. 

54
Observation 1. A scoring system satisfies best effort if and only if
\[ \frac{\partial S_w(x', \hat{x})}{\partial x'} \leq 0. \]

For example, a simple scoring rule \( S_w(x', \hat{x}) = 2\hat{x} - x' \) can incentivize the worker to exert his best effort. Given the best effort property, we know that \( x' \) is set to \( x \) at the equilibrium. The report-the-mean property requires a scoring system to incentivize the manager to honestly report her expected completion time given all available information. This is exactly the problem that proper scoring rules for eliciting the mean of a random variable are designed for. Proper scoring rules for eliciting the mean of a random variable satisfies the property that reporting the mean maximizes the expected score of the reporter. Hence, we have an immediate solution based on the definition of proper scoring rules.

Observation 2. If the best effort property is satisfied, the scoring system satisfies the report-the-mean property if and only if
\[ E(X|I) \in \arg\max_{\hat{x}} E(S_m(X, \hat{x})|I). \]

We can use any proper scoring rule as the manager scoring rule, in conjunction with a worker scoring rule that incentivizes best effort, to achieve the report-the-mean property. For example, \( S_m(x', \hat{x}) = b - (\hat{x} - x')^2 \) for an arbitrary parameter \( b \) uses a quadratic scoring rule. While it is easy to achieve both best effort and report-the-mean properties at an equilibrium, satisfying information sharing and collusion-proofness is less straightforward.

Consider the pair of the worker and manager scoring rules mentioned above, \( S_w(x', \hat{x}) = 2\hat{x} - x' \) and \( S_m(x', \hat{x}) = b - (\hat{x} - x')^2 \). The worker may not want to share his information with the manager if his information will lead to a lower prediction \( \hat{x} \) by the manager. In addition, the total score can be increased if the worker and the manager collude. To see this, note that the manager can report a larger prediction and the worker can work slowly to perfectly match the manager’s prediction, which increases the worker’s score while maximizing the manager’s score! Below, we characterize the conditions for achieving all four desired properties simultaneously.
3.3.1 A Family of Scoring Rules

We first consider how to satisfy the information sharing property. This will require that the worker is also being rewarded for a more accurate prediction.

**Lemma 7.** *If the best effort and report-the-mean properties are satisfied, the information sharing property is satisfied if and only if \( E(X|I) \in \arg \max_{\hat{x}} E(S_w(X, \hat{x})|I) \).*

*Proof.* The worker can influence the prediction \( \hat{x} \). In an extreme case, when all relevant information is possessed by the worker, the prediction is effectively made by the worker. In order for the worker to predict the mean, the worker scoring rule needs to be a proper scoring rule for the random variable \( X \). Because \( E(X|I) \) maximizes a worker’s score given any information set \( I \), for any \( I_m \) and \( I_w \), \( E(X|I_w, I_m) \) maximizes the worker’s expected score \( E(S_w(X, \hat{x})|I_w, I_m) \). Hence, the worker is better off sharing the information with the manager to have the manager report \( E(X|I_w, I_m) \). \( \square \)

Next, we consider achieving collusion-proofness. Let \( S_T(x', \hat{x}) \) denote the sum of the worker and manager scores. If the manager and the worker collude to report a prediction \( \hat{x} \) and complete the task in time \( x' \), collusion-proofness requires that the manager-worker pair is incentivized to report the mean and exert best effort. These are analogous to achieving information sharing and best effort when the worker has all information and the manager has no information. Let \( S_T(x', \hat{x}) = S_w(x', \hat{x}) + S_m(x', \hat{x}) \) be the total scoring rule. The following result follows immediately.

**Lemma 8.** *Collusion-proofness is satisfied if and only if \( \frac{\partial S_T(x', \hat{x})}{\partial x'} \leq 0 \) and \( E(X|I) \in \arg \max_{\hat{x}} E(S_T(X, \hat{x})|I) \).*

This means that if a scoring system satisfies best effort, report-the-mean, and information sharing we essentially get collusion-proofness for free with the mild additional condition that the total scoring rule also satisfies best effort (a sufficient condition for which is that
the manager’s scoring rule satisfies best effort). Combining the results characterizes scoring systems that satisfy all four desirable properties.

**Lemma 9.** A manager-worker scoring system satisfies information sharing, report-the-mean, best effort, and collusion-proofness at a perfect Bayesian equilibrium if and only if the following conditions are satisfied:

- \( \frac{\partial S_w(x', \hat{x})}{\partial x'} \leq 0. \)
- \( \frac{\partial S_T(x', \hat{x})}{\partial x'} \leq 0. \)
- \( E(X|I) \in \arg \max_{\hat{x}} E(S_m(X, \hat{x})|I). \)
- \( E(X|I) \in \arg \max_{\hat{x}} E(S_w(X, \hat{x})|I). \)

for all information sets \( I. \)

Intuitively, Lemma 9 requires that the worker score and the manager score are all given by a proper scoring rule for eliciting the mean (it is immediate that the total score must also be given by a proper scoring rule), in addition to the worker and total scores being a decreasing function of the actual completion time. For example, \( S_w(x', \hat{x}) = S_m(x', \hat{x}) = f(x') + 2cx'\hat{x} - c\hat{x}^2, \) where \( f'(x') + 2c\hat{x} < 0 \) and \( c > 0 \) is a family of scoring systems that satisfy all four desirable properties. A theorem due to Savage [89] characterizes all (differentiable) proper scoring rules for eliciting the mean.

**Theorem 9** (Savage [89]). *For \( S \) differentiable in \( \hat{x}, E(X|I) \in \arg \max_{\hat{x}} E(S(X, \hat{x})|I) \) if and only if \( S(x', \hat{x}) = f(x') + G(\hat{x}) + (x' - \hat{x})G'(\hat{x}) \) where \( E[f(X)|I] \) is finite for all \( \Pi \) and \( G \) is a differentiable convex function.*

Note that a sufficient condition for \( E[f(X)|I] \) to be finite for all \( \Pi \) is that \( f \) is bounded on \((0, x_{\text{max}}). \) Combining Theorem 9 with Lemma 9 yields a more precise characterization.
Theorem 10. A manager-worker scoring system satisfies information sharing, report-the-mean, best effort, and collusion-proofness at a perfect Bayesian equilibrium if and only if the following conditions are satisfied:

- \( S_w(x', \hat{x}) = f_w(x') + G_w(\hat{x}) + (x' - \hat{x})G'_w(\hat{x}) \) where \( f_w \) is a differentiable function such that \( E[f_w(X)|I_w] \) is finite for all \( \Pi \) and \( G_w \) is a differentiable convex function.

- \( S_m(x', \hat{x}) = f_m(x') + G_m(\hat{x}) + (x' - \hat{x})G'_m(\hat{x}) \) where \( f_m \) is a differentiable function such that \( E[f_m(X)|I_m, I_w] \) is finite for all \( \Pi \) and \( G_m \) is a differentiable convex function.

- \( f'_w(x') + G'_w(\hat{x}) \leq 0 \) for all \( x', \hat{x} \in (0, x_{\text{max}}) \).

- \( f'_w(x') + f'_m(x') + G'_w(\hat{x}) + G'_m(\hat{x}) \leq 0 \) for all \( x', \hat{x} \in (0, x_{\text{max}}) \).

Finally, note that this means we can derive a scoring system from a differentiable convex pair of \( G_s \) whose derivatives we can upper bound by taking \( f'_w(x') = -|\sup_{\hat{x}} G'_w(\hat{x})| \) and similarly for \( f_m \).

3.4 Task Decomposition

Continuing, we now consider that a task has substructure, with a task represented as a series of subtasks. Based on this, we allow a worker-manager team to elect to split-off individual subtasks (or contiguous subtasks) to become identified prediction tasks in their own right; i.e., essentially partitioning the task into a distinct set of pieces, each of which has an associated prediction problem.

In increasing the realism of the model, we also situate the prediction task for a single task in the context of a repeated version of the problem, in which a worker has a sequence of tasks. In this context, the following property is useful:

5. **Always non-negative.** The score of the worker and the manager is always non-negative for all realizations of \( x \) and all reports \( \hat{x} \).
If the score is always non-negative, then our best effort property immediately guarantees that best effort is also optimal for a worker facing a sequence of tasks, in that this will maximize both the total score for sequence of tasks and the score per unit time. In contrast, suppose the score assigned for the completion of a task is negative. In this case, a worker may prefer to spend 10 hours and earn a score of $-2$ than to spend 1 hour and earn a score of $-1$, because in those additional 9 hours the worker would be completing additional tasks for more negative scores.

This noted, we can focus back on a single task and introduce formalism for the idea of a subtask. Let $X = X_1 + \ldots + X_k$ denote a task $X$ composed of $k$ subtasks $X_1, \ldots, X_k$. The worker decides which sets of subtasks are to become targets of the scoring system. For example, the worker might prefer to make a single prediction, thereby being scored just once after completing the task in its entirety. Another option is that the worker may prefer to make $k$ predictions (hence receiving $k$ scores), one for each subtask. Alternately, instead of these two extremes, the worker may end up requesting to have subtask $X_1$ as a target, then subtask $X_2$, and then subtasks $X_3, \ldots, X_k$ aggregated into one chunk of work for the purpose of prediction.

We assume that the degree to which the prediction problem associated with a task may be split-out into subtasks is knowledge that is private to the worker and a priori not known to the manager. We allow the worker to make online decisions about which subtasks to split-out as separate prediction targets. That is, if the worker initially decides to get scored for $X_1$, after this is done he can then choose whether to next get scored for $X_2$ or instead to combine $X_2$ with some number of subsequent subtasks (we assume subtasks must be completed in order). As we are focusing on decisions made by the worker, we will only discuss $S_w$. The report-the-mean and collusion proofness properties are assumed to be retained through an appropriate choice of $S_m$. To be able to make concrete statements, we focus on the special case $S_w(x, \hat{x}) = f(x') + 2cx'\hat{x} - c\hat{x}^2$. 

59
3.4.1 Independent Subtasks

For a simple model, consider a worker with two subtasks, denoted by random variables $X_1$ and $X_2$, and each with discrete support $\{a, b\}$, with $0 < a < b \leq 1$ and $x_{\text{max}} = b$. For this setting with two subtasks, the choice of the worker in regard to prediction targets is as follows:

- Adopt the complete task as a prediction target, share information in regard to $X = X_1 + X_2$ (with the manager making a prediction), work on them both, and then receive a score.

- Split-out $X_1$ as the first prediction target, share information with the manager (with the manager making a prediction), work on $X_1$ and receive a score, then share information in regard to $X_2$, work and receive a score.

Lemma 10. Let $S_w(x, \hat{x}) = f(x') + 2cx'\hat{x} - c\hat{x}^2$ satisfy best effort and always non-negative. Then for a task with two subtasks, it is always optimal for the worker to split independent subtasks into separate prediction targets.

Proof. For any distribution of effort $X$ the worker’s expected score from truthful reporting (which is optimal) is

$$E[S_w(X, E[X])] = E[f(X)] + cE[X]^2.$$ 

To deal with $E[f(X)]$, we make use of two bounds regarding $f(x)$. First, we know that $f'(x') < -2c\hat{x}$ for all $\hat{x}$, so in particular this is true for $\hat{x} = x_{\text{max}}$. By always non-negative, $f(x_{\text{max}}) \geq 0$. Thus, $f(x) \geq (x_{\text{max}} - x)2cx_{\text{max}}$. Second, for $a < b$, $f(a) - f(b) \geq (b - a)2cx_{\text{max}}$. We now show that $E[S_w(X_1, E[X_1])] + E[S_w(X_2, E[X_2])] > E[S_w(X_1 + X_2, E[X_1 + X_2])]$. Note that we use the unconditional expectation over $X_2$ here because $X_1$
and $X_2$ are independent.

$$E[S_w(X_1, E[X_1])] + E[S_w(X_2, E[X_2])]$$

$$- E[S_w(X_1 + X_2, E[X_1 + X_2])]$$

$$= E[f(X_1)] + cE[X_1]^2 + E[f(X_2)] + cE[X_2]^2$$

$$- E[f(X_1 + X_2)] - cE[X_1 + X_2]^2$$

$$= E[f(X_1) + f(X_2) - f(X_1 + X_2)] - 2cE[X_1]E[X_2]$$

$$\geq E[(x_{\text{max}} - X_1)2c x_{\text{max}} + ((X_1 + X_2) - X_2)2c x_{\text{max}}]$$

$$- 2cE[X_1]E[X_2]$$

$$= 2c(x_{\text{max}}^2 - E[X_1]E[X_2]) > 0.$$
Because this bonus is invariant to any aspect of the prediction or effort, including this does not change the rest of the analysis in any way.

### 3.4.2 Correlated Subtasks

To gain an additional qualitative understanding of the effect of our scoring rules on the propensity to split-out subtasks as separate targets, we adopt a simple model of correlation. The joint distribution on \((X_1, X_2)\) is parameterized with \(q \in (1/2, 1]\) and \(r \in [0, 1]\).

The distribution on time to complete task 1 under best effort is \(a\) with probability \(q\) and \(b\) with probability \(1 - q\). With probability \(r\), the time to complete task 2 is the same as for task 1 (i.e., \(X_2 = X_1\)). Otherwise, with probability \(1 - r\) the time to complete task 2 is independently sampled according to probability \(q\).

We use the scoring rule \(S_w(x, \hat{x}) = f(x') + 2cx'\hat{x} - c\hat{x}^2\) with \(f(x') = C - kx'\) and \(c = 1\), where \(k > 2x_{\text{max}}\) and \(C\) is a constant. We show that, for appropriate choice of \(C\), the incentive to split-out subtasks increases as \(r\) increases, and thus as there is more positive correlation between the time to complete the subtasks under best effort.

In particular, the choice of \(C\) sets a threshold for \(r\). If \(r\) is below this threshold then the subtasks are independent enough that the worker does not want to split them. If \(r\) is above this threshold then the subtasks are correlated enough that splitting them to learn is worthwhile. Increasing \(C\) decreases this threshold, but increases the cost to the scoring rule. Thus the choice of \(C\) allows a trade-off between encouraging the accurate sharing of predictions on subtasks and cost. However, past a certain point, the worker will want to split-out all subtasks regardless, and increasing \(C\) will simply increase the cost.

**Lemma 11.** Consider a task with two sub-tasks. Let \(S_w\) be as above with \(C < 2E[X_1]E[X_2]\). Let \(r^* = \sqrt{\frac{2E[X_1]E[X_2] - C}{q(1-q)(a-b)^2}}\). If \(r \geq r^*\) then it is optimal for the worker to split-out subtasks. If \(r \leq r^*\) then it is optimal for the worker to not do so.
Proof. Unlike in Lemma 10, $X_1$ and $X_2$ are no longer independent. In particular, this means that the expected score for task two if they are split is no longer simply $E[S_w(X_2, E[X_2])]$. Instead, the worker learns something after completing the first task so, a priori, the expected score is $E_{X_1}[E[S_w(X_2, E[X_2|X_1 = x])|X_1 = x]]$. Hence we can write the expected gain from splitting as follows:

$$E[S_w(X_1, E[X_1])] + E_{X_1}[E[S_w(X_2, E[X_2|X_1 = x])|X_1 = x]] - E[S_w(X_1 + X_2, E[X_1 + X_2])]$$

$$= E[C - kX_1 + E[X_1]^2] + E_{X_1}[E[C - kX_1 + E[X_1][(E[X_2|X_1 = x])^2]|X_1 = x]] - E[C - k(X_1 + X_2) + E[X_1 + X_2]^2]$$

$$= C + E[X_1]^2 + E_{X_1}[(E[X_2|X_1 = x])^2] - E[X_1 + X_2]^2$$

$$= C + E[X_1]^2 + E[X_2]^2 - E[X_1 + X_2]^2 + E_{X_1}[(E[X_2|X_1 = x])^2] - E[X_2]^2$$

$$= C - 2E[X_1]E[X_2] + E_{X_1}[(E[X_2|X_1 = x])^2] - E[X_2]^2.$$
For the particularly simple distribution we have chosen, we can expand the last two terms as

\[
E_{X_1}[(E[X_2|X_1 = x])^2] - E[X_2]^2
= q(ra + (1 - r)E[X_2])^2 + (1 - q)(rb + (1 - r)E[X_2])^2 - E[X_2]^2
= (1 - r)^2E[X_2]^2 + qr^2a^2 + (1 - q)r^2b^2 + 2qar(1 - r)E[X_2]
+ 2(1 - q)br(1 - r)E[X_2] - E[X_2]^2
= (1 - r)^2E[X_2]^2 + qr^2a^2 + (1 - q)r^2b^2 + 2r(1 - r)E[X_2]^2
- E[X_2]^2
= qr^2a^2 + (1 - q)r^2b^2 - r^2E[X_2]^2
= r^2(qa^2 + (1 - q)b^2 - (qa + (1 - q)b)^2)
= r^2(qa^2 + (1 - q)b^2 - q^2a^2 - (1 - q)^2b^2 - 2q(1 - q)ab)
= r^2((1 - q)qa^2 + q(1 - q)b^2 - 2q(1 - q)ab)
= r^2q(1 - q)(a - b)^2
\]

Thus, splitting is optimal if and only if \( C - 2E[X_1]E[X_2] + r^2q(1 - q)(a - b)^2 \geq 0 \).

Solving for \( r \) yields the desired inequalities. \( \square \)

In a situation where it is important that the cumulative score for each task is non-negative, then a mitigating aspect of this trade-off is that for smaller values of \( C \) the scoring system must assign a larger bonus \( B \) upon task completion to correct for the possibility of an accumulation of negative scores on subtasks.

One might also wonder whether it is possible to modify our scoring rules to allow a worker-manager team to push new predictions in regard to a particular prediction target over time, and without leading to new strategic considerations. For example, suppose the worker elects not to split-off any subtasks and have as the target the entire task. But now as work is completed on each subtask, perhaps the worker has updated information in regard to when
the task will likely be completed. Perhaps surprisingly, this issue turns out to be quite subtle.

For example, associating the score with the average of the score from \( m \) predictions fails, because the worker-manager team could maximize its realized score by simply pushing a lot of predictions just before completing a task when there is high confidence about how long the task will take. Insisting that predictions are made at fixed intervals of time could lead to a preference for slacking in order to be able to make an additional prediction. Adopting a time-averaged score, integrated over the different predictions made over time in regard to a prediction target, could lead to a preference to work more slowly on subtasks about which the prediction is higher quality. We leave a full reconciliation of this problem to future work.

### 3.5 Simulation

Our simulation study is designed to validate the three qualitative observations in our theoretical analysis: (a) for subtasks with more correlation the worker will tend to split out more subtasks as targets, (b) for a higher value of \( C \) the worker will tend to split out more subtasks into targets, and (c) for a higher value of \( C \) the average score received by the worker will tend to increase.

For this purpose, we consider a task \( X \) with 3 subtasks \( X_1, X_2, \) and \( X_3 \). With probability \( q \) the task is low difficulty, and with probability \( 1 - q \) the task is high difficulty. Given that the task is low difficulty, then a subtask takes time \( a = 0.5 \) under best effort with probability \( p \in [0.5, 1] \), and \( b = 1 \) under best effort otherwise. For a high difficulty task, a subtask takes time \( b = 1 \) with probability \( p \), and \( a = 0.5 \) otherwise (both under best effort.) In this way, \( p \) controls the correlation between effort on subtasks. High \( p \) yields high correlation.

We simulate each possible policy a worker might adopt in deciding which subtasks to split-off into separate prediction targets. Altogether, there are six possible policies:

1. Policy 1: Work on each subtask separately. First target is subtask \( X_1 \), then \( X_2 \), fol-
2. Policy 2: First target is $X_1$. If completion time of $X_1$ is observed to be $a$ then the second target is $X_2$, followed by $X_3$. If completion time of $X_1$ is $b$ then the second target is $X_2 + X_3$ as a chunk.

3. Policy 3: First target is $X_1$. If completion time of $X_1$ is observed to be $b$ then the second target is $X_2$, followed by $X_3$. If completion time of $X_1$ is $a$ then the second target is $X_2 + X_3$ as a chunk.

4. Policy 4: First target is $X_1$. The second target is $X_2 + X_3$ as a chunk.

5. Policy 5: First target is $X_1 + X_2$ as a chunk. The second target is $X_3$.

6. Policy 6: The first and only target is the entire task $X = X_1 + X_2 + X_3$ as a single chunk.

For concreteness, the scoring rule that we adopt is

$$S_w(x, \hat{x}) = C - 2x'x_{\text{max}} + 2x'\hat{x} - \hat{x}^2$$

In considering the score, we allocate a bonus $B$ upon completion of the entire task, set to the minimal value such that the score is guaranteed to be positive for all contingencies. To determine this value, we first computed all the possible different scores that could be obtained for each policy and the lowest score was chosen as that policy’s worst score. The lowest score amongst the 6 worst scores of the 6 policies was then chosen to determine the bonus.

Given this setup, we compare the average score and the average number of prediction targets as the amount of positive correlation (reflected by $p$) and the parameter in the scoring rule $C$ varies. For each policy, and for different values of $C$, $p$ and $q$, we run at least 10,000
Figure 3.2: Average score and average number of prediction targets under the best policy, varying $p \in [0.5, 1]$ for $C = -1.9$ and $q = 0.5$.

Figure 3.3: Average score for policies 1 through 5, varying $p \in [0.5, 1]$ for $C = -1.9$ and $q = 0.5$. 
trials and determine the average score. The policy that we assume the worker adopts for a triple \((C, p, q)\) is that which maximizes the average score.

Figure 3.2 is obtained by varying \(p \in [0.5, 1]\) for \(C = -1.9\) and \(q = 0.5\), and shows for each value of \(p\) the average score and the average number of targets for the optimal policy for that value of \(p\). As \(p\) increases there is greater correlation which results in more splitting and a higher score. Figure 3.3 corroborates this by showing that as \(p\) approaches a value of 0.7, the optimal policy changes from Policy 4 to Policy 3. Since Policy 3 varies between 2 and 3 splits, we get an average number of targets equal to 2.5. We have omitted Policy 6 in Figure 3.3; its average score remained in the range \([2.8, 2.9]\) throughout.

Figure 3.4 is obtained by varying \(C \in [-5, 1]\) for \(p = 0.8\) and \(q = 0.5\), and shows for each value of \(C\) the average score and the average number of targets for the optimal policy for that value of \(C\). It shows that as \(C\) increases there is more splitting, while the average score also increases. Figure 3.5 shows the average score of the different policies. The optimal policy is initially Policy 6 (no splitting), and hence there is only 1 prediction target. With increasing \(C\) the optimal policy changes to those with greater splitting, finally ending up at Policy 1 (full splitting). We have omitted policies 2 and 5 in Figure 3.5, as their scores were very close to the scores of policies 3 and 4 respectively (policies 2 and 5 scored slightly less than policies 3 and 4 respectively for all values of \(C\)). For \(C < -3.75\), the value of \(B\) was determined by policy 1, which is why the curve for policy 1 is initially flat and the others are decreasing. For larger values of \(C\), \(B\) is determined by policy 6 so it is flat while the others increase.

The basic trends we see in these plots are consistent with the theory, which allows for a tradeoff between the degree to which tasks are split and the cost to the mechanism.
Figure 3.4: Average score and average number of prediction targets under the best policy, varying $C \in [-5, 1]$ for $p = 0.8$ and $q = 0.5$.

3.6 Conclusion

We have introduced the problem of incentivizing a worker-manager team to both commit best effort to a task while also making accurate predictions. In studying this question, we have characterized a family of scoring rules with natural properties, and considered the effect of the rules on decisions in regard to which subtasks to split-out into explicit prediction targets.

The problem was motivated by an extant outcomes-based incentive system that was applied to IBM’s internal IT initiatives. In this system, software professionals (developers, software designers, testers, etc.) execute tasks assigned by their project managers to produce project deliverables. Each task is associated with a “Blue Sheet” that records the manager’s prediction of required effort for the task, along with its actual completion time. Blue Sheet data is used to compute scores for both ‘workers’ and ‘managers,’ and top scorers are recognized for their achievement.

The Blue Sheet system had been in place since 2009 and has provided some useful initial
insights on process differences across internal groups. However, the Blue Sheet scoring system did not satisfy any of the four properties outlined in Section 3.2.1. It is difficult to derive any strong conclusions about the impact of these missing properties from existing Blue Sheet data (much of the information is self-reported), but the data suggests some evidence of collusion between ‘workers’ and ‘managers.’

In future work, we would like to pilot a new scoring system based on the current work. It would be interesting, as a next step, to consider additional factors that might be important in a practical deployment. These factors include the impact of a scoring system on the kinds of tasks that worker-manager teams choose to take on, for instance in regard to their inherent predictiveness.

The Blue Sheet system includes some additional aspects that are outside of our model. These include a self-assessment of the deliverable quality against specified standards, and also in regard to the extent that reuse of pre-existing assets was leveraged to complete the deliverable. From this perspective, we are interested to understand the impact of a scoring
system on how to decompose work into subtasks in the first place, that is, on the modularization of tasks. A key goal of the Blue Sheet system is to incentivize the creation and application of reusable software components, thereby making the development process more efficient. Devising incentive schemes that directly encourage creating reusable components, e.g., by rewarding the component author when others reuse the component, remains as future work.
Chapter 4

Incentivizing Deep Fixes

4.1 Introduction

The size and complexity of software systems have increased to such an extent that it is beyond our ability to effectively manage them. A study commissioned by the U.S. National Institute of Standards and Technology (NIST) concluded that software errors alone cost the U.S. economy approximately $59.5 billion annually [82]. Software often ships with both discovered as well as undiscovered bugs, because there are simply not enough resources to address all issues [71, 10]. Further, through an empirical study of 277 coding projects in 15 companies, Wright and Zia [100] determine that software maintenance actually introduces more bugs: each subsequent iteration of fixes has a 20 – 50% chance of creating new bugs.

In fact software systems have come to resemble systems where behaviour is decentralized, interdependent, and adapts over time – rather like market economies. This suggests that the principles of market design and mechanism design, coupled with traditional engineering techniques, might be more effective in managing software. Software economies refers to a vision for a software development process, inspired by economies, where supply and demand drive the allocation of work and the evolution of the system.
Software economies consist of a private and a public component. A private software economy, studied in Chapter 3, deals with the internal incentives of managers and their employees, such as allocating resources and predicting completion times. A public software economy refers to a market mechanism where users bid for coveted bug fixes and features. Over time the software reaches an equilibrium in which all fixes and features for which there is enough market value have been implemented. However designing such a market mechanism is fraught with challenges. This work addresses the design of a public software economy, and is a response to a challenge raised in Bacon et al. [16].

Specifically the problem we are interested in is how to design incentives to obtain “deep” rather than “shallow” fixes to bugs, at least where the disutility of users warrants a deep fix. A deep fix attempts to correct the root cause of the problem so that another bug with the same root cause is found only after a long time or not at all. In contrast a shallow fix suppresses the bug at a superficial level so that other bugs with the same root cause may appear soon after. We solve this problem by proposing subsumption mechanisms. In a subsumption mechanism, deeper fixes can replace (or subsume) shallower fixes, and a worker’s payoff increases if his fix subsumes other fixes. A subsumption mechanism employs an installment-based payment rule that stops paying the worker when his fix is subsumed, transferring the remaining reward to the competitor responsible for the deeper fix.

We study the effect of different mechanisms for a dynamic model of the software engineering ecosystem comprising workers, users, root causes of bugs, bugs, fixes, user values, worker costs, and payments. The user base reports bugs and offers payments for fixes. Workers are tentatively matched to bugs, and decide which fix to submit given considerations about cost, payment, and market dynamics. Note that this setting gives rise to many instances of externalities. For example, a fix for a particular bug, reported by a particular user, may fix other bugs as well, resulting in a positive externality to a different set of users who may encounter those bugs and experience some disutility.
Importantly, we insist that the payments and other comparisons performed by the market process involve only those features of the software development process that are externally observable. In other words we use only information that can be observed by the market from the interaction between submitted fixes and reported bugs, without allowing the possibility of such steps as code inspection. Some examples of externally observable features include, the time taken for the next bug to appear, the number of bugs fixed by a submitted fix, and the amount of payment available for a fix. The model presented in this work is flexible and its elements can be viewed as building blocks that can be decoupled and reconfigured to encode different issues in the software economy.

A real-world example is BountySource [21], which is a funding platform for open-source software, where users post rewards on issues they want solved, while developers devise solutions and claim rewards. GitHub [36] is a code repository that allows programmers to work on portions or versions of the code by providing operations such as forking, merging, and syncing. In addition, current practice in software testing uses technology that is consistent with our paradigm. For example, black box testing tests the functionality of the software without examining its internal workings – this resembles the externally observable paradigm that we insist on in our approach. Regression testing tests whether a new fix has reintroduced old bugs. In a similar spirit, a subsumption test would require that past versions of the software are merged with the new fix and tested to see which bugs are fixed. The aforementioned platforms and practices seem like natural precursors to the externally observable market-based system proposed in this chapter.

Our technical approach adopts the mean field equilibrium (MFE) approach, which is an approximation methodology that is useful for analyzing the behaviour of large systems populated with self-interested participants. In large markets, it is intractable and implausible for individuals to best-respond to competitors’ exact strategies [3]. Instead it is assumed in the MFE approach that in the long run, temporal patterns in agents’ actions average out.
Hence agents optimize with respect to long run estimates of the marginal distribution of other agents’ actions. Using this methodology, we compute an approximate MFE of the software engineering ecosystem under different parameterizations of the environment. We study variants on subsumption mechanisms, namely eager subsumption, lazy subsumption, and eager with reuse. In an experimental study we explore different environment settings and compare the performance of the above variants with respect to other mechanisms that do not involve subsumption, using metrics such as user utility and percentage of deep fixes received. We conclude by drawing lessons for market design. We find that subsumption mechanisms satisfy important criteria for market design and perform robustly across all environment configurations examined in the experimental study. Our results demonstrate that the basic premise in this work has merit: by using externally observable comparisons we can drive competition and get deeper fixes and better utility. Through this research we have learnt how to design incentives when there are inter-relationships between tasks in a stochastic setting.

4.1.1 Main Results

- Our main result is that subsumption mechanisms satisfy important criteria for market design and perform robustly across all environment configurations examined in the experimental study. In contrast, this is not the case for mechanisms without subsumption.

- In general we find that installment-based payment schemes (including subsumption mechanisms) create incentives for deep fixes. Installment-based payment schemes that allow transfers, that is, unpaid installments from a fix that are added to the reward for a later fix, perform even better. Transfers augment a low payment for a fix, making deeper fixes more affordable for a worker.

- Subsumption mechanisms pay in installments and allow transfers conditional on sub-
The use of subsumption, including externally observable fix-to-fix comparisons, introduces competition amongst workers and produces deeper fixes and higher user utility.

- There are many variations on subsumption mechanisms. One refinement involves subsumption mechanisms that use fixes. That is, subsumed fixes are not discarded permanently, rather they are stored and reused to address new bugs.

- Surprisingly, a mechanism that pays the entire reward for a fix in a single shot is able to produce deep fixes, but at low rewards only and with a high wait time.

4.1.2 Related Work

Research into vulnerability reporting systems has explored a market-based approach. Schechter [90] describes a vulnerability market where a reward is offered to the first tester that reports a specific security flaw. The reward grows if no one claims it. At any point in time the product can be considered secure enough to protect information worth the total value of all such rewards being offered. Ozment [83] likens this type of vulnerability market to an open first price ascending auction. While vulnerability markets as well as existing bug bounty programs (e.g., Mozilla security bug bounty) motivate testers to report flaws, such systems do not capture users’ valuations for fixes.

Le Goues et al. [41] share our view of software as an evolving, dynamic process. However where we approach software engineering from a market design perspective, they are influenced by biological systems. They apply genetic programming for automated code repair [40, 97].

Our work is most closely related, from a methodological perspective, to a series of recent papers that analyze MFE in various market settings. Motivated by sponsored search, Iyer et al. [50] consider a sequence of single-item second price auctions where bidders learn their
private valuations as they win auctions. The authors prove existence of MFE by appealing to Schauder fixed point theorem. In addition, they show that the agent’s optimal strategy in an MFE is to bid truthfully according to a function of his expected valuation. In a related paper, Gummadi et al. [43] examine both repeated second price and generalized second price (GSP) auctions when bidders are budget constrained. The authors show that the effect of long-term budget constraints is an optimal bidding strategy where agents bid below their true valuation by a shading factor. The existence of MFE is established via Brouwer’s fixed point theorem. Other settings have also been analyzed in the mean field context [2, 3, 12, 42]. These papers present a theoretical analysis whereas we take a computational approach. The analytical approaches used to characterize optimal strategies do not seem to extend to our setting, which is much more complex.

There are a number of organizations that crowdsource work for the purpose of software development. A comprehensive survey of crowdsourced software platforms for software development as well as the associated academic literature is presented by Mao et al. [73]. A notable example of a crowdsourcing platform that employs a contest architecture is TopCoder [94], where programmers compete in a contest with cash awards to submit the best solution to a particular stage of a software project. The structure and performance of different contest architectures have been studied in the economics literature [75, 76, 95]. More recently, Archak and Sundararajan [11], DiPalantino and Vojnovic [26] and Chawla et al. [23] study crowdsourcing contests such as TopCoder and model them as all-pay auctions. However crowdsourcing contests are an altogether different scenario to ours. In our model submissions do not happen in a simultaneous way. Moreover we cannot directly observe the quality and judge which submission is best. Instead the system evolves over time to retain deeper fixes.
4.2 A System of Bugs and Fixes

The model of the software ecosystem consists of two parts that interact with each other: a model of software bugs and fixes, and a model of the world comprising time periods, workers, users, and payments (see Figure 4.1). We first describe the system of bugs and fixes, before presenting the entire ecosystem in a later section.

We present an abstract model of software as a set of independent root causes,\(^1\) where each root cause generates a series of related bugs. To draw an analogy with a real world scenario, a root cause may be thought of as a specific component or functionality of a software; for example, one root cause might be a synchronization error in the user interface component, while another root cause might be a buffer overflow in the graphics component.

4.2.1 Bit String Model

We devise a bit string model that captures how a particular root cause can generate several bugs and encodes the relationships amongst bugs and fixes. Each bug permits a set of fixes. Each root cause is associated with a particular bit string length \(l > 0\). The set of bugs

\(^1\)Root causes may be referred to as simply roots.
belonging to this root cause comprises the $2^l - 1$ non-zero bit strings of length $l$. The set of permissible fixes for this set of bugs consists of the set of $2^l$ bit strings, including the $2^l - 1$ non-zero bit strings that address the bug, as well as the zero bit string that models a worker who chooses not to submit a fix (referred to as the null fix). A larger $l$ signifies a root cause that generates more bugs.

Armed with the bit string model we proceed to define concrete properties relating bugs and fixes. The bit string representation of bugs and fixes combined with the rules defining their relationships gives us a compact mathematical language that we can use to capture a complex, interdependent phenomenon such as the one studied in this work. In our model, fixes pertaining to a particular root cause cannot be used to fix bugs generated by other root causes. Thus all relationships and properties are relevant for only those bugs and fixes that belong to the same root cause. In what follows, we refer to a bit whose value is 1 as an ON-bit.

**Definition 10.** [Valid fix] A fix $f$ is valid for bug $b$ if it includes all the ON-bits in $b$, i.e. $f \geq b$ bit-wise. Thus an AND operation between $f$ and $b$ must result in $b$.

We refer to the set of fixes for a bug plus the null fix (the zero bit string) as the set of
feasible fixes for the bug. Different bugs can have different numbers of feasible fixes. Bug 1110 has only 3 feasible fixes. In contrast bug 0001 has $2^3 + 1$ feasible fixes.

**Example 4.** A root cause with $l = 4$ can generate the set of bugs \{0001, 0010, \ldots, 1111\}, where each bit string represents a single bug. Consider bug $b = 1110$. Bug $b$ is fixed by two fixes: $f_1 = 1110$ and $f_2 = 1111$. Fix 0111 cannot fix $b$. The set of feasible fixes given bug $b$ is \{1110, 1111, 0000\}.

**Definition 11.** [Fix depth] The fix depth of fix $f$ refers to the number of ON-bits in the bit string of $f$, and is denoted $|f|$.

Continuing with the above example, $f_1$ and $f_2$ have fix depths equal to 3 and 4 respectively. We can now define, in the context of the bit string representation, what constitutes a shallow or deep fix with respect to a given bug.

**Definition 12.** [Shallow fix] Given a bug $b$, a shallow fix $f$ is a valid fix for which $|f| = |b|$.

**Definition 13.** [Deep fix] Given a bug $b$, a deep fix $f$ is a valid fix with $|f| > |b|$.

In other words, a shallow fix is the fix that meets the essential requirement of having the same ON-bits as the bug being fixed and no more. A deep fix is a valid fix that is not a shallow fix. The deepest fix not only fixes the bug in question but all bugs of that root cause.

**Example 5.** Consider bug $b = 1100$ generated by a root cause with bit string length $l = 4$. A shallow fix for $b$ is $f_1 = 1100$. A deep fix for $b$ is $f_2 = 1101$ or $f_3 = 1110$. The deepest possible fix is $f_4 = 1111$.

Next we consider the relationship between fixes.

**Definition 14.** [Ordering relation] A fix $f_k$ is deeper than a fix $f_i$ ($f_k \succ f_i$) if $f_k \geq f_i$ interpreted bitwise, with $f_k[j] > f_i[j]$ for at least some $j$. 80
The ordering relation provides a partial order, as shown in Example 5 where we have
\[ f_4 \succ \{f_2, f_3\} \succ f_1. \] Here \( f_2 \) and \( f_3 \) are incomparable.

The bit string language is an abstraction of the ways in which bugs and fixes may relate
and interact with one another in our setting. Our language is designed to capture the essence
of the phenomenon of deep fixes as they interplay with bugs, and it is this interplay that is
the focus of this work. While the model of bugs and fixes is richer than could be studied
theoretically, it is nonetheless not so complicated that the results cannot be interpreted. The
bit string language is flexible and can encode different settings in the software economy, and
possibly other settings that deal with complex interdependencies.

In practice there are no general rules that define the relationships between different types
of bugs and fixes. Instead there are several plausible interpretations of the bit string model.
For example, one interpretation may consider each ON-bit as a location in code and therefore
a fix with more ON-bits would fix more locations than one with fewer ON-bits. Furthermore
a bug with a particular ON-bit could imply that a problem has occurred in that specific
location. In this context the stipulation that a valid fix must at least include the same ON-bits
as the bug it addresses makes sense – a fix must at least fix the location where the problem
has occurred.

4.2.2 Externally Observable

So far we have described how bugs and fixes relate to one another in the context of the bit
string model. Moving on, we consider those properties that are externally observable, and to
that end, introduce a key concept, that of subsumption.

**Definition 15.** [Subsumption relation] A fix \( f_k \) subsumes another fix \( f_i \) \( (f_k \succ f_i) \) with respect
to a set of bugs \( B \) if the set of bugs fixed in \( B \) by \( f_k \) strictly contains the set of bugs fixed in \( B \)
by \( f_i \).
Figure 4.3: Root causes $R$ generate bugs, which may receive fixes. Bugs and fixes are represented as bit strings.

Continuing with Example 5, suppose $b$ receives fix $f_2$. Now suppose the root cause generates another bug $b' = 0110$ which receives the fix $f' = 1110$. $f'$ subsumes $f_2$ since $f'$ fixes all the bugs fixed by $f_2$ as well as $b'$. However $f' \not\succ f_2$ does not imply $f' \succ f_2$. Clearly, we have $f_2 \not\succ f'$. But it may be that $f' \succ f_2$ or that $f'$ and $f_2$ are incomparable, as is the case here.

4.3 The Model of the Software Ecosystem

We are now ready to present the model of the entire ecosystem. We study a setting with discrete time periods and a large population of workers who submit fixes. Users discover and report bugs, and may offer payment for fixes. At any point in time, we assume that the software consists of a set $R$ of root causes of bugs (of some fixed cardinality) and that all root causes have bit string length $l$. 
4.3.1 Bug Generation

In each time period we randomly sample from all $2^l - 1$ bugs associated with a root cause, regardless of whether some of those bugs might be already fixed (i.e., sampling with replacement). A new bug enters the system only if we choose an unfixed and as yet unreported bug. Note that an unreported bug may be preemptively fixed by a fix already submitted for a different reported bug. This reflects a natural situation where a root cause may generate several bugs initially, but fewer and fewer as fixes accumulate because an increasing number of bugs are preemptively fixed.

Upon generation, a bug is associated with a payment available to a worker who submits a fix. This is explained in detail in Section 4.3.7 below.

A root cause $x \in \mathcal{R}$ that has not generated a new bug in some period of time (this is a parameter of the model) is called *inactive* and is regenerated with probability $\beta_x \in (0, 1]$. This models the root cause as having received deep enough fixes that it is now unlikely to generate more bugs, and that the user base shifts its attention to a new set of bugs. When a root cause is regenerated, it is removed and replaced with a new root cause (one with all bugs yet to be generated) of the same bit string length. The presence of multiple regenerating root causes ensures an infinite stream of bugs, which is necessary for the stationarity in long-run statistics that motivate the MFE approach.
4.3.2 Model Dynamics

The actions in the market are simulated through a sequence of fix-verify cycles that occur over time (see Figure 4.4). Each fix-verify cycle takes place in a single time period $t$ at a given root cause. Thus at time period $t$ we “round-robin” around all root causes in the system, executing fix-verify cycles at each root cause. This process is repeated at time period $t + 1$. A fix-verify cycle for a particular root cause proceeds as follows:

1. The root cause is queried once to see if it generates a new bug, which is associated with a payment from the associated user.

2. A worker is selected at random to be tentatively assigned to a bug, perhaps an unfixed bug or a newly generated bug.

3. The worker submits a feasible fix (perhaps null) that maximizes his expected utility, given a model of the future.

4. The fix is verified by the market infrastructure in regard to the assigned bug.

5. The total amount to be paid to the worker over some time horizon is calculated. This depends on the specifics of the payment rule.

6. The worker is paid in a single step, or in installments, depending on the specifics of the payment rule.

We make four assumptions. First, a user payment is associated with a bug at the time it is generated. No further payments are solicited from users for the same bug and thus its total user payment does not continue to accumulate over time. Second, a worker submits a fix in the same time period that the worker is assigned the bug. Third, a worker is selected at random, and assigned to work on a bug. Fourth, the likelihood that the same worker is
repeatedly assigned bugs belonging to the same root cause is low, hence the worker considers only those scenarios where future work on the same root cause is performed by other workers.

Each worker works on one bug at a time and submits a fix before starting work on the next bug. Hence an individual worker works sequentially. Moreover, in any given time period, each root cause generates at most one new bug, at most one worker is assigned to an unfixed bug, and at most one bug is fixed. This process ensures that at most one fix is submitted per time period, per root cause, and avoids situations where two or more overlapping or equivalent fixes might be submitted simultaneously. While work within a particular root proceeds in sequential order, work across different roots may happen in parallel.

4.3.3 Subsumption Mechanisms

Consider an instantiation of the model where Step 4 of the fix-verify cycle involves a specific check. In a subsumption mechanism, this step involves a check for whether the current fix subsumes any previous fixes. If so, the subsumed fixes are replaced by the current fix. In addition, the worker’s payment may increase if his fix subsumes previous fixes. The basic idea is that a worker now competes with other workers who fix subsequent bugs on the same root cause. Thus the model captures indirect competition in the following way: a later worker might produce a deeper fix for a subsequent bug $k$, that not only fixes $k$ but also subsumes $w_j$’s fix for bug $i$. The payment scheme in a subsumption mechanism pays out in a fixed number of installments over time.

There are variations on subsumption mechanisms. Generally, a subsumption mechanism is defined as a mechanism that uses subsumption relations in at least one of two ways: i) to make externally observable comparisons between fixes, where one fix may subsume another; ii) to determine payments according to the subsumption ordering of fixes. This work focuses on three variations referred to as eager, lazy, and eager with reuse.
Eager Subsumption

Eager subsumption performs externally observable checks for subsumption using only the bugs and fixes seen so far. Consider a fix $f_k$ for bug $k$ at time $t$. Fix $f_k$ subsumes a prior fix $f_i$ if it fixes all bugs fixed by $f_i$, from the set of bugs seen until time $t$. If fix $f_i$ is subsumed then it is discarded. Since subsumption is concluded without waiting to see the entire range of bugs and fixes, eager subsumption only approximates the actual ordering relationship between fixes.

For eager subsumption, we use a simple payment rule that pays out equal installments of the total payment over a fixed number of time periods, $h^*$. Let $i < k$ be the index into all fixes $f_i$ that occurred at a time prior to $f_k$ on this root cause. Let $Q(t)$ represent the set of all bugs at time $t$ on this root cause that were generated, remained unfixed, and are now fixed by $f_k$. Let $\hat{r}_i(t)$ denote the total payment still to be made to the worker associated with fix $f_i$ in periods $t$ and forward.

The total payment remaining to be made for a fix $f_k$ for bug $k$ submitted at time $t$ is,

$$\hat{r}_k(t) = \sum_{i=1}^{k-1} \hat{r}_i(t) I_{\{f_k \succ f_i\}} + \sum_{q \in Q(t)} r_q + r_k \quad (4.1)$$

where $\hat{r}_i(t)$ is the remaining unpaid payment at time $t$ of a previous fix $f_i$ subsumed by $f_k$, $r_q$ is the payment associated with an unfixed bug $q$ that $f_k$ fixes, and $r_k$ is the payment associated with bug $k$.

This total payment is made in installments. Let $h^*$ denote the time horizon over which payments are made. The payment to the worker in time period $t'$, such that $t \leq t' \leq t + h^*$, is,

$$p_k(t') = \hat{r}_k(t') / h^* \quad (4.2)$$

In particular, the worker is paid an installment every time period until all $h^*$ installments
are exhausted or until the worker’s fix is subsumed by a new fix, whichever occurs sooner. In the latter case, the remainder of the worker’s payment is transferred to the subsuming fix. Hence if the worker’s fix is only subsumed after \( h^* \) time periods have passed, or not subsumed at all during the lifetime of the root cause, the worker is paid \( \hat{r}_k \) in its entirety.

If a root cause is regenerated while a worker still has outstanding payments, he is paid the remainder in full.

**Lazy Subsumption**

Eager subsumption can give false positives in regard to the ordering of fixes, as the next example shows.

**Example 6.** Suppose \( b_1 = 0001 \) and its fix \( f_1 = 0011 \). Next suppose \( b_2 = 1000 \) and has fix \( f_2 = 1001 \). The eager subsumption mechanism concludes that \( f_2 \) subsumes \( f_1 \) as it fixes all bugs seen so far. Thus \( f_1 \) is replaced by \( f_2 \). Now consider \( b_3 = 0010 \). \( f_1 \) can fix \( b_3 \) but not \( f_2 \). Had \( f_1 \) been retained it would have precluded the appearance of \( b_3 \). Instead \( b_3 \) enters the market and awaits a new fix.

Lazy subsumption attempts to decrease the likelihood of false positives by delaying judgement. Although a check for subsumption is performed when a new fix is submitted by a worker, subsumption relations are finalized only when a root cause regenerates. Thus, given the same instance, the final partial ordering of fixes obtained with lazy subsumption may be different than the one obtained with eager subsumption.

To illustrate let us consider a single fix \( f_k \) for bug \( k \) submitted at time \( t \). Suppose \( f_k \) fixes all bugs fixed by a prior fix \( f_i \) that have appeared until time \( t \). At this point eager subsumption would decide that \( f_k \) has subsumed \( f_i \), whereas with lazy subsumption this decision is deferred. Continuing, assume that \( f_i \) has not subsumed any fixes. Payments towards \( f_i \) are stopped and \( f_i \) is replaced by \( f_k \), whose payments begin. But in contrast to
eager subsumption, \( f_i \) is not discarded permanently, but is instead removed from the system and placed on a waiting list. Moreover \( f_k \)'s payments do not yet include the transfer of the remaining unpaid amount \( \hat{r}_i(t) \) from \( f_i \). The total payment with lazy subsumption is also given by Equation 4.1. At time \( t \) the worker responsible for \( f_k \) begins to receive installments of the portion of the total payment represented by the second and third terms in Equation 4.1, that is, \( \sum_{q \in Q(t)} r_q + r_k \). However while installments for the rest of the payment begins, the payments associated with the first term in Equation 4.1 are frozen until the root cause regenerates at time \( t^* \). Only then does the mechanism determine, based on all bugs generated and fixes submitted up until time \( t^* \), whether fix \( f_k \) subsumes \( f_i \). At that point only, the lazy subsumption scheme redistributes the payment \( \hat{r}_i(t) \) to the right party: if \( f_k \) is found to have subsumed \( f_i \) at the time the root is regenerated, then \( \hat{r}_i(t) \) is transferred to the worker who submitted \( f_k \), otherwise it is retained by the worker who submitted \( f_i \).

An interesting feature of lazy subsumption is that it reuses fixes that are on the waiting list. In Example 6, lazy subsumption would attempt to reuse \( f_1 \) when \( b_3 \) appears. If \( f_1 \) can be successfully reused, then it is simply reinstated into the current, working set of fixes and the bug does not enter into the economy and no new payment amount is brought into the market. Hence only if none of the fixes on the waiting list can fix \( b_3 \) does the mechanism revert to the standard scenario where the bug is listed and a new payment is associated with it. It is via reuse that subsumption relations, and consequently the partial ordering of fixes, are updated during a root’s lifetime.

Thus the process of determining subsumption in the lazy mechanisms proceeds in two stages, consisting of first an eager check followed by a lazy check, as follows:

1. When a new fix is submitted a check is done, as in eager subsumption, to ascertain if the new fix subsumed any prior fixes based on the bugs and fixes seen so far. The purpose of this check is to freeze payments that may be subsumption transfers – if the new fix has subsumed prior fixes then their payments are frozen. Although any
subsumption transfers (the first term in Equation 4.1) are frozen, the new fix starts to receive installments from payments associated with any previously unfixed bugs (the second and third terms in Equation 4.1).

2. When a root regenerates, subsumption relations are concluded by examining the final partial ordering of fixes. The purpose of this check is to determine how to redistribute frozen payments. This step is carried out by retrospectively stepping through the arrival times of fixes and computing the credit or payment that should be accorded to each fix by considering the subsumption relations implicit in the final partial ordering.

The difference with the eager mechanism, given the same system state, is twofold: i) the stricter criteria of judging subsumption means that the first term in Equation 4.1 might amount to less, and ii) there is a delay in paying out subsumption transfers, thus the per time period payment does not include the first term in Equation 4.1 until the root regenerates. Note that subsumption checks are performed only in the case where a new fix is submitted by a worker. No such checks are performed when a fix from the waiting list is reused to fix a bug and reinstated into the current, working set of fixes. The goal here is simply to alleviate the “mistake” made by eager subsumption in determining subsumption without having seen more of the set of bugs that a root cause may generate. However more than one fix in the waiting list may be able to fix a new bug – in this instance we use a tie-breaking rule such as giving priority to the most recent waiting fix. Because lazy subsumption concludes subsumption relations when a root cause regenerates, which may happen before all bugs are generated, the final subsumption relations concluded by the lazy approach remain an approximation of the true ordering relation amongst fixes.
Eager with reuse

This hybrid mechanism functions like the eager subsumption mechanism and uses the eager payment rule, with total payments given by Equation 4.1. However it deviates from the eager mechanism in that subsumed fixes are not discarded. Instead they are kept on a waiting list and reused as is done in lazy subsumption. What this means is that there are time periods when a new bug is fixed via a reused fix and the bug does not enter the market or get associated with a new payment amount. The effect is as if a new bug was not generated at all and instead was preemptively fixed by an existing fix in the system (which would have been the case had the reused fix not been removed from the set of current fixes in the first place). Accordingly any eager installments that are being paid out proceed as if a bug was not generated in this time period.

Continuing with Example 6 to illustrate the eager with reuse mechanism, $f_2$’s payments include the remaining unpaid payment of $f_1$ as well as the payment for $b_2$. Payments are made as per the eager subsumption rule. However $f_1$ is not discarded, instead it is reused when $b_3$ appears and therefore a new payment is not associated with bug $b_3$.

4.3.4 Other Mechanisms

In this section we present instantiations of the fix-verify cycle that do not involve subsumption. In the following mechanisms, no subsumption comparisons are made between fixes. These provide baselines against which to compare the subsumption mechanisms in our experimental study.

- **Myopic mechanism**: The worker is paid in full as soon as he submits a valid fix. Let $Q(t)$ represent the set of all generated bugs at time $t$ on this root cause that are now
fixed by $f_k$. The total payment for a fix $f_k$ for bug $k$ submitted at time $t$ is,

$$\hat{r}_k^n(t) = \sum_{q \in Q(t)} r_q + r_k$$  \hspace{1cm} (4.3)

where $r_q$ is the payment of a formerly unfixed bug $q$ that $f_k$ has now fixed, and $r_k$ is $k$’s reward.

- **Installment mechanism:** The total payment available to be paid for a fix is described by Equation 4.3. However the worker is paid in a fixed number, $h^*$, of equal installments that stop as soon as a new bug of the same root cause appears. Any remaining payment goes unused and can be considered to be “wasted”.\(^2\) The payment in time period $t'$, where $t \leq t' \leq t + h^*$, is equal to $\hat{r}_k^n(t) / h^*$.

- **Installment with transfer:** This mechanism functions like the installment mechanism, but takes one step closer to eager subsumption by allowing transfers of payments from one fix to another. Installments that remain unpaid when a new bug appears are not thrown away. Rather, they are added to the payment for the new bug. The total payment available to be paid for a fix $f_k$ for bug $k$ submitted at time $t$ is,

$$\hat{r}_k^n(t) = \hat{r}_i(t) + \sum_{q \in Q(t)} r_q + r_k,$$  \hspace{1cm} (4.4)

where $\hat{r}_i(t)$ is the remaining unpaid payment at time $t$ of the previous fix $f_i$ whose installments were interrupted by $k$, and the other terms are as defined in Equation 4.3. Because bugs appear in sequence and payments are interrupted as soon as the next bug appears, there can only be one such earlier fix that is currently receiving payments. The payment in time period $t'$, where $t \leq t' \leq t + h^*$, is equal to $\hat{r}_k^n(t) / h^*$.

\(^2\) As explained in Section 4.3.7, the user in our model commits to a payment even if the work is not done. Hence unused payments are not returned to the user.
4.3.5 Preemptive Fixing and Reuse

We make explicit a subtle variation present amongst the six mechanisms described in the preceding sections. Recall that a new, unreported bug may be preemptively fixed by a fix already existing in the system for a reported bug. We refer to the set of submitted fixes that are currently present in the system as the set of *active* fixes. The composition of the set of active fixes differs amongst the different mechanisms.

Because the myopic and installment mechanisms do not perform fix-to-fix comparisons and thus do not eliminate any submitted fixes from the system, the set of active fixes comprises all fixes submitted so far on a root cause. Hence at time $t$ all fixes submitted at a root cause up until time $t$ are used to preemptively fix new bugs in the myopic and installment mechanisms. In the case of eager subsumption, subsumed fixes are permanently deleted from the system. Therefore the set of active fixes at time $t$ is equal to the set of all fixes submitted so far minus those fixes that have been subsumed by time $t$. As a result the preemptive power of eager subsumption may be lower than that of the myopic and installment mechanisms, specifically when there are false positives as shown in Example 6.

Turning to eager with reuse and lazy subsumption, although subsumed fixes are removed from the system, they are not permanently deleted as they are in eager subsumption. Instead subsumed fixes are stored in a waiting list. The set of active fixes at time $t$ is the same as in eager subsumption: it comprises the set of all fixes submitted so far minus those fixes that have been subsumed by time $t$. However when a new bug is generated that could not be preemptively fixed by any active fix, the eager with reuse and lazy mechanisms check whether any fixes stored in the waiting list may be reused. The effect is that new bugs are introduced into the market at time $t$ only if no fix in the set of all fixes submitted at a root up until time $t$ (i.e., the set of active fixes and the set of subsumed fixes) has already fixed them.
4.3.6 The Worker

A worker is chosen at random to be given the option of working on a bug, in what is referred to as a tentative assignment. The worker’s decision problem is to determine which fix to submit, if any. The worker’s decision to work now does not impact his decision in the future, since the current model assumes that fixes are produced instantaneously and the set of competitors faced by the worker does not include himself.

The optimal fix to submit depends both on the behavior of other workers and on the payment mechanism. To produce a fix, \( w_j \) incurs cost \( c_j(|f_k|) \), that is a non-decreasing function of the number of ON-bits in \( f_k \). A simple example is a linear cost function, with \( c_j = |f_k| \). We assume the worker discounts the future payments according to discount factor, \( \delta < 1 \). The presence of a discount factor represents a preference for earlier payments over later ones. The utility a worker \( w_j \) derives from submitting fix \( f_k \) for bug \( k \) is denoted \( y_{jk} \), and is equal to the discounted sum of the payments he will receive, starting from the time period when he submits the fix, minus his cost for the fix \( f_k \). For concreteness, let’s consider eager subsumption, and normalize time, so that the fix is considered to be submitted in period \( t' = 0 \).

The worker’s expected utility is:

\[
E[y_{jk}] = \mathbb{E}_h \left[ \sum_{t'=0}^{h} \delta^{t'} p_k(t') - c_j(|f_k|) \right],
\]

(4.5)

where \( p_k(t') \) is the per-period payment, and \( h = \min(h^*, H) \) where \( H \) is a random variable representing subsumption time (number of time periods before \( f_k \) is subsumed.) Subsumption times vary amongst the fixes submitted, because each fix precludes a different set of bugs, which affects the time till the next bug (and its fix) appears. Moreover each fix permits a different set of future fixes that can subsume the fix.

Given a model of the behaviour of other workers, and the environment by which bugs
and payments are generated, the worker $w_j$ chooses a fix (perhaps null) that maximizes his expected, discounted utility. The precise way in which this is determined uses the MFE methodology, to provide an equilibrium model of the subsumption time of the feasible fixes under consideration.

4.3.7 The User

The users in our model are not strategic. Rather, the user model is simple and users have a random value for an immediate (and possibly, deep) fix of a new bug, which provides both the payment to a worker and defines a user’s utility given the sequence of fixes that follow a bug report. The user utility is a metric that unifies several performance measures, such as a user’s cost (amount spent towards a reward for a fix), a user’s wait time for a fix (the number of time periods elapsed from the bug being reported to a fix being submitted), and side-effects of a fix (a deeper fix pre-empts future bugs).

When a user encounters a bug, she experiences some disutility or inconvenience, and this has an associated cost. Given a bug $k$ and a fix $f_k$, a user’s utility is equal to the value that the user derives from the fix $f_k$ minus the payment incurred by the user in discovering, reporting, and making a payment. In particular, a user’s utility is modeled as her value minus her cost. In our model the user commits to a payment even if the work is not done.

In what follows, the user’s instantaneous disutility associated with a bug is some number, $r \in [r_{\text{min}}, r_{\text{max}}]$, chosen according to a distribution over the interval. Given this, a user’s realized value for the sequence of fixes that occur following the report of a bug is a function $v(r, \hat{f})$ of the reward $r$ and the wait time $\hat{f}$, where $v$ is decreasing in $\hat{f}$, and $v(r, 0) = r$. The longer the user waits, the lower her realized value for a fix since waiting itself is costly. A user’s realized value is at most the amount $r$.

In particular, if disutility $r = r_{\text{min}}$ then the user is called a short-term user, whereas if $r > r_{\text{min}}$ then the user is a long-term user. In the time period that a fix is first submitted,
both short-term and long-term users get an initial installment of realized value, defined as $v(r_{\min}, \hat{t})$. This represents the value of any fix, even a shallow fix. This is also the final, realized value for a short-term user, modeling the idea that she was only interested in a shallow fix. A long-term user receives the remaining value $v(r, \hat{t}) - v(r_{\min}, \hat{t})$ in $h^*$ equal installments, or fewer if a bug appears subsequent to $\hat{t}$.

In this way, the maximum utility a user can get is zero (representing realized value $r$, net payment $r$), and the utility is generally negative. This user utility model has the following simple, sensible properties:

i If the fix never occurs, user utility $= -r$.

ii If the fix occurs right away and all installments are received, user utility $= -r + r = 0$.

iii Fixes that occur later leave the user with some disutility, and thus user utility $< 0$.

### 4.4 Mean Field Equilibrium

In order to understand the long-term behaviour of workers, we need to consider the equilibrium of the software economy. Once equilibrium is established, performance metrics can be meaningfully measured, and a market designer may be better informed on conditions for desirable market behaviour.

In a subsumption mechanism, workers must contend with competing workers who may subsume their fixes, thereby curtailing their payment installments. Thus the worker faces a naturally strategic situation. The deeper the fix submitted the less likely it is to be subsumed and payments curtailed. Yet deeper fixes cost the worker more. A worker in this market must decide how to best-respond to competitors’ play in order to maximize his expected utility.

We can simplify equilibrium analysis by adopting the approach of mean field equilibrium (MFE). MFE is an approximation methodology suited to large market settings, where
keeping track of the strategies of individual agents becomes prohibitive (both for a modeler, and, presumably for a participant). As the number of agents grows large, it is reasonable to assume that agents optimize with respect to long run estimates of the marginal distribution of other agents’ actions. In particular, each worker best-responds as if in a stationary environment. In this way, MFE approximates the beliefs adopted by each participant about their environment. MFE requires a consistency check: the latter distribution must itself arise as a result of agents’ optimal strategies.

4.4.1 Adapting MFE to our Setting

Recall that a worker assigned a bug must decide which fix to submit, if any. This amounts to estimating the subsumption time of each valid fix.

In particular, we assume that each worker models the future as though there is an i.i.d. distribution $D$ of fix depths submitted by others, where the set of possible fix depths is $\{0, \ldots, l\}$ (including null fixes). The worker assumes that all fixes associated with a particular fix depth occur with equal probability. This induces a probability distribution over the set of all $2^l$ possible fixes. Given this marginal distribution, a worker infers the conditional distribution on feasible fixes for a specific bug, and chooses the fix that maximizes his expected utility by estimating the distribution on subsumption time and thus estimating equation (4.5).

To estimate the utility of each possible, feasible fix, a worker samples possible future trajectories given that workers behave according to belief $D$, given the environment model that dictates how bugs and new root causes are generated, and arrives at a sample over possible subsumption times and thus an estimated, discounted utility.

In particular, given a fix $f$ submitted in the current time period, bugs not (preemptively) fixed by $f$ might appear in future time periods. A future fix to one of these future bugs might subsume $f$, thereby curtailing the number of payment installments. In order to estimate the expected utility of a fix $f$, the worker simulates the software economy environment a number
of times. Several trajectories are sampled in order to realize subsumption time and arrive at an estimate of the utility the worker can expect if he submits $f$ (see Figure 4.5). We stop sampling trajectories once the worker’s estimated utility converges to within a confidence interval. This look-ahead sampling technique is inspired by the sparse sampling algorithm of [61].

Let $\Phi(D)$ be the long-term, empirical distribution that results when workers assume model $D$ and apply their optimal strategy, given the dynamic model of the root cause and bug environment.

**Definition 16.** An MFE is a distribution $D$ of fix depths such that $\Phi(D) = D$.

Continuing, for a given environment and payment mechanism, we estimate the MFE following the approach described in Algorithm 1. To determine convergence we compare distributions using a likelihood ratio test (details in Appendix B.1).

### 4.4.2 Limited Workers

The mean field methodology becomes more behaviourally reasonable with larger numbers of workers. In particular, it assumes that as the number of agents grows large any individ-
**ALGORITHM 1**: Algorithm to estimate MFE

Initialize time period $t = 0$.
Initial distribution on fix depths (worker beliefs), $D_0$.

while convergence not reached do

    $t = t + 1$.

    for each root cause do

        Regenerate root cause according to its regeneration probability.
        Try to generate a new bug.
        if bug generated then

            Generate payment by sampling from reward distribution.
            Assign a worker at random to bug.
            Verify fix submitted by worker (if any).
            Update the empirical distribution on fix depths given this worker’s
            selection and compute $D_{t+1}$.
            Make payment installments on this root cause.
        end

    end

    Update user’s realized utility.

end

Check convergence.

An agent has negligible effect on overall outcomes. In the context of our setting, this does not imply that the number of workers must be greater than the number of bugs in the market. Rather, it simply means that the larger the number of workers the more accurate are equilibrium predictions.

In this section we describe a special case where a worker might experience only local interactions that do not grow with the size of the global market. This can arise in markets where different tasks require different specialized skills, and the set of workers with similar qualifications is limited. To this end we vary the ratio of workers to the amount of work. The change in ratio of workers to work impacts the sampling of future trajectories carried out by the worker to decide what fix to submit. In the standard case a worker never faces himself on the same root where he is currently being paid. In each trajectory sampled, bugs are generated as a consequence of the fix submitted, and future anonymous workers are simulated to submit fixes to these bugs. To determine what fix a future competitor will
choose, the mean field distribution $D$ is used. Thus each decision encountered in a trajectory is made according to $D$.

Whereas in the limited workers case, the same worker $w$ with the current paying fix may be reassigned to the same root with some probability. Consider a trajectory sampled to determine what fix to pick for bug $b$: as a consequence of a fix $f$ being considered, bugs are generated and workers are simulated to submit fixes. As long as these future workers are opponents, $D$ is used to decide what fix is submitted. However what if one of the future workers assigned is $w$ himself? At this point in the decision tree $w$’s future decision is determined recursively – i.e., $w$ computes the best future fix by recursively sampling trajectories and computing the maximum utility at that node. This value is then propagated up the tree and used to calculate the utility of submitting the original fix under consideration, $f$.

Figure 4.6 illustrates the recursion when $w$ computes the average utility for submitting $f$. The black nodes represent instances when a future worker is an opponent and thus a fix is sampled from $D$. The white nodes represent instances when a future worker is $w$ himself, and if so, $w$ must once again sample trajectories to determine what fix to submit. In contrast Figure 4.7 depicts the standard case where all nodes in a look-ahead trajectory are black nodes, and corresponds to one of the fix $f$ nodes in Figure 4.5

![Recursion tree for limited workers case. Value of subtree is propagated up.](image-url)
Figure 4.7: In the standard case workers sample fixes in a look-ahead trajectory via the mean field distribution.

4.5 Experimental Study

We carry out an experimental study to understand the strengths and limitations of the different mechanisms. The following questions motivate our study.

1. How do the simple mechanisms, myopic and installment, fare relative to eager subsumption?

2. Do installment with transfer and eager with reuse bring about an improvement in the performance of installment and eager respectively?

3. What is the relative performance of the three subsumption mechanisms, eager, eager with reuse, and lazy?

4. Which mechanism produces the best results at very low reward levels?

5. What happens when we limit the number of workers?

Next we list the simulation environments considered in the experimental study.

- Root causes are associated with bit string length 4.
- Bugs are generated from a uniform or geometric distribution.

---

3The MFE code is implemented in Matlab and run on the Odyssey cluster supported by the Research Computing Group at Harvard University.
• The worker uses either a linear or exponential cost function, and the worker’s cost $c$ for the first ON-bit is sampled uniformly at random from different ranges.

• The number of installments $h^*$ is either 5 or 10.

• Unless otherwise stated, the discount factor is 0.8.

• Users are either all short-term or a mixture of short-term and long-term with user disutility sampled uniformly at random from different ranges.

Because we focus on the workers’ perspective, and noting that users simply make a payment equal to their disutility for a bug, we refer to payments rather than user disutility throughout this section. Next we describe metrics used to evaluate performance. The metrics are averaged over all observations for each root cause for a fixed length of time after the algorithm has converged. These are:

1. User utility: the average user utility per bug generated, averaged over all bugs seen during a length of time;

2. User cost: this refers to the total payment paid out towards fixes by all users of a particular root cause;

3. Percentage of immediate fixes: this calculates the percentage of all bugs generated at a particular root cause that receive any non-null fix in the same time period that they appear;

4. Percentage of immediate deep fixes: this is the percentage of all generated bugs that receive a deep fix in the same time period that they appear.

5. Percentage of submitted fixes that are deep: this looks at the set of submitted fixes and calculates the percentage that are deep.
Figure 4.8: Percentage of immediate fixes with a range of payments

Figure 4.9: Percentage of immediate deep fixes with a range of payments

Figure 4.10: User utility for myopic, eager, and installment mechanisms
A good mechanism provides high user utility, high percentage of deep fixes, and requires less payment. The above metrics address these concerns. Using multiple metrics allows us to focus on different characteristics of the model.

**Simple mechanisms**  We first compare the performance of eager subsumption with two simple mechanisms, namely the myopic and the installment mechanisms. Eager subsumption can be considered to be the simplest variant amongst the subsumption mechanisms proposed in this work. Bugs are generated uniformly at random, the number of installments is 10, and the worker cost function is linear, with each worker randomly sampling cost for initial bit, $c \sim \text{uniform}(1, 2)$. To understand the ratio of reward to cost in this environment, suppose the reward is 10. Then, for an average initial bit cost of 1.5, we have a ratio of $\frac{10}{1.5} = 6.67$. Consider Figures 4.8, 4.9 and 4.10 (Figure 4.11 is the same data as Figure 4.8 but with y-axis starting at 0).

At relatively low payments, the eager and installment mechanisms, which make payments for a valid fix over time, do not cover the worker’s cost. This is both because the worker discounts installments over time and because payments may be interrupted early due to a new bug appearing or a fix getting subsumed. This leads bugs to accumulate until the combined payment for fixing a number at the same time is sufficient. That bugs linger unfixed in the system is seen in Figure 4.8 that shows that less than 100% of bugs receive immediate fixes at lower payment values. In comparison, myopic pays the entire payment when a valid fix is submitted. This leads to more immediate fixes.

On the other hand, the equilibrium strategy of a worker in the myopic mechanism is to produce the shallowest possible fix. Since the worker is paid in full as soon as he submits any valid fix, he maximizes his payoff by simply producing the least cost (hence shallowest) fix needed to claim the payment. Because of this, its performance levels off once all fixes are affordable and remains unresponsive to a further increase in payment. Figures 4.9 and 4.10
show that increase in payment has no effect on percentage of deep fixes or user utility with the myopic mechanism. The installment mechanism fares better than the myopic mechanism when the payment is high but, because there are no transfers, it lags behind eager subsumption at lower payments. Without transfers, the installment mechanism cannot augment the low payment for a current fix with any remaining unpaid payment from a past fix. These results show the need for something more than the basic myopic and installment mechanisms. In addition, it turns out that we can refine the performance of eager subsumption by adding reuse.
Installment with transfer and Eager with reuse  We now consider more sophisticated variations of the above mechanisms, such as the installment with transfer and eager with reuse mechanisms. In installment with transfer, the remaining amount from a previously halted payment is not wasted but transferred towards a fix for the next bug. Because this transfer happens every time a new bug is generated, and involves no fix-to-fix comparisons, the amount transferred can accumulate for bugs appearing later on a root cause. Hence we expect installment with transfer to perform better than the installment mechanism.

Recall that eager subsumption deletes subsumed fixes. However some of these might be false positives, as explained in Example 6. On the other hand the myopic, installment,
and installment with transfer mechanisms do not delete any submitted fixes. By deleting fixes that turn out to be false positives, eager reduces its ability to preemptively fix new bugs instead of generating them (see bug generation model in Section 4.3.1). This places eager subsumption at a disadvantage with respect to metrics such as user utility. Eager with reuse enhances the eager mechanism’s performance with the ability to reuse subsumed fixes instead of discarding them. Note that we do not count reused fixes when computing the metrics in this study. Instead an instance of reuse is treated as if no bug happened in that time period.

Figure 4.12 shows results from an experiment with the same parameters as used in the case of simple mechanisms, but now including installment with transfer. As evident from Figure 4.12, installment with transfer provides higher user utility than the installment mechanism. Moreover, for the parameter settings considered here, it bridges the gap between installment and eager subsumption.

In the next experiment we consider an environment involving the mechanisms, installment with transfer, eager subsumption, and eager with reuse. In order to create opportunities for reuse of subsumed fixes to occur, we must change some parameter settings in this next environment. To this end, bugs are now generated according to a geometric distribution, and
Figure 4.16: Percentage of immediate deep fixes with costly workers

Figure 4.17: User utility with costly workers

Figure 4.18: User utility with costly workers
worker cost is an exponential function. The worker’s cost for the first ON-bit is \( c = 1 \) and the number of installments is \( h^* = 10 \). To get a sense of the ratio of reward to cost in this environment, suppose the reward is 10. Then, for an average initial bit cost of 1, we have a ratio of \( \frac{10}{1} = 10 \). Figures 4.13 and 4.14 show the results from a run where the rate of bug generation is sped up. Figure 4.15 is the same data as Figure 4.13 but with y-axis starting at 0. This is done by querying a root cause five times to see if it generates a new bug in a time period instead of just once.

Like the installment mechanism, installment with transfer stops paying a worker as soon as the next bug appears. It follows that the portion of the payment that the worker can expect to keep is partly influenced by the rate of arrival of bugs. Moreover submitting the deepest possible fix on a root cause may not be profitable given the worker’s cost and available payment. Hence the worker’s expected payoff for submitting deeper fixes is reduced. We therefore expect that installment with transfer will produce a lower percentage of deep fixes in this setting than the other mechanisms, and accordingly that is what we see in Figure 4.13.

Figures 4.16, 4.17, and 4.18 show results from a setting where workers have an ex-

---

For reuse to occur, we require fixes that are deep enough to be reused but not so deep that they are hard to subsume. When bugs are generated uniformly at random and worker cost is linear, reuse is often precluded because we get bugs of a high bit string value, receiving deeper fixes, arising earlier in a root’s lifetime.
ponential cost function with each worker randomly sampling the initial bit cost \( c \sim \text{uniform}(2.5, 3) \), bugs are generated from a geometric distribution, and the number of installments is 10. To get a sense of the ratio of reward to cost in this environment, suppose the reward is 10. Then, for an average initial bit cost of 2.75, we have a ratio of \( \frac{10}{2.75} = 3.64 \). Figure 4.19 is the same data as Figure 4.16 but with y-axis starting at 0. In this setting eager with reuse seems to do best with regard to user utility whereas the eager mechanism performs better in terms of percentage of deep fixes.

**Lazy subsumption** Moving on, we consider the lazy subsumption mechanism. Lazy subsumption reuses subsumed fixes and finalizes subsumption relations only when a root cause regenerates. In doing so it attempts to correct the false positives produced by the eager mechanism and retain the best, minimal set of fixes in the system. In this experiment bugs are generated from a geometric distribution, the number of installments is 10, and the worker cost model is an exponential function, with each worker randomly sampling \( c \sim \text{uniform}(3, 4) \). Suppose the reward is 10. Then, for an average initial bit cost of 3.5, we have a ratio of reward to cost of \( \frac{10}{3.5} = 2.86 \). See Figures 4.20, 4.22, and 4.21.

As a consequence of reusing subsumed fixes, the eager with reuse and lazy subsumption mechanisms achieve lower user cost, resulting in higher user utility. So far our results provide no clear evidence that the lazy mechanism is significantly better than the eager with reuse mechanism for any of the metrics in this study. A possible reason for this may be that the amount of the worker’s payment that is redistributed at the end of a root’s lifetime is not large enough or exceed a certain threshold to make a difference in the worker’s incentives \(^5\).

One line of future work would be to alter the model so that the amounts redistributed

---

\(^5\)Even adjusting for the effect of discounting of later payments (by increasing redistributed amounts by a small quantity), to level the playing field, we are not able to detect a significant improvement with lazy over eager with reuse. However, this did result in lazy progressing from doing worse than eager with reuse to doing equally well or slightly better.
Figure 4.20: Percentage of immediate deep fixes with lazy, eager, and eager with reuse mechanisms

Figure 4.21: User utility with lazy, eager, and eager with reuse mechanisms

Figure 4.22: User cost with lazy, eager, and eager with reuse mechanisms
by lazy are of greater weight. In general, it may be that different environments and suitable metrics are required in order to capture the advantages of lazy subsumption. For the environments examined in this study, it appears that eager subsumption is sufficient to get the competitive benefits.

**Robustness across environments.** We consider 10 different environments, plotting them on the x-axis, versus the metrics user utility and percentage of deep fixes on the y-axis. Details of each environment are as follows.

1. Short-term users, fixed payment chosen in the range $[2, 5]$, same cost workers with $c = 1, h^* = 5$.

2. Short-term users, fixed payment chosen in the range $[11, 15]$, same cost workers with $c = 1, h^* = 5$.

3. Short-term users, fixed payment chosen in the range $[5, 7]$, worker randomly samples $c \sim \text{uniform}(1, 2), h^* = 5$.

4. Long-term as well as short-term users with payment sampled according to $r \sim \text{uniform}(4, 8)$, worker randomly samples $c \sim \text{uniform}(1, 2), h^* = 5$. 

Figure 4.23: *Percentage of immediate deep fixes with lazy, eager, and eager with reuse mechanisms, with y-axis starting at 0.*
5. Long-term as well as short-term users with payment sampled according to $r \sim \text{uniform}(25, 29)$, worker randomly samples $c \sim \text{uniform}(1, 2)$, $h^* = 5$.

6. Faster rate of bug generation.

7. Low cost workers, where each worker randomly samples $c \sim \text{uniform}(1, 1.5)$, long-term and short-term users with payment sampled according to $r \sim \text{uniform}(5, 11)$, $h^* = 5$.

8. High cost workers, where each worker randomly samples $c \sim \text{uniform}(2, 2.5)$, long-term and short-term users with payment sampled according to $r \sim \text{uniform}(8, 14)$, $h^* = 5$.

9. High cost workers, where each worker randomly samples $c \sim \text{uniform}(2.5, 3)$, long-term and short-term users with payment sampled according to $r \sim \text{uniform}(11, 17)$, $h^* = 5$.

10. Environment with lazy subsumption.

The results are shown in Figures 4.24 and 4.25. The environments are ordered from simple settings with short-term users and all workers having the same cost, to more complicated settings with long-term as well as short-term users, workers with different costs, faster rate of bug generation, and opportunities for reuse. Accordingly we see a shift in the kinds of mechanisms that dominate – the myopic mechanism performs well in the first few environments, but later the installment with transfer and the subsumption mechanisms take over. The utility of long-term users depends on receiving immediate and deep fixes. As we have seen, simple mechanisms like myopic and installment do not offer sufficient incentives to address this requirement. Hence mechanisms that permit transfers and reuse subsumed fixes perform best in environments 5 to 10 in Figures 4.24 and 4.25, in particular, the eager with reuse mechanism. Our results demonstrate that the basic premise in this work has merit: by
Figure 4.24: Percentage of deep fixes in different environments

Figure 4.25: User utility in different environments
using externally observable comparisons we can drive competition and get deeper fixes and better utility.

**Short term users** In a counterintuitive result, we find that the myopic mechanism gives deep fixes at low rewards only as shown by Figures 4.26, 4.27 and 4.28. In this treatment, bugs are generated uniformly at random, the worker cost function is linear with $c \sim \text{uniform}(1, 2)$ for all workers, the number of installments is 10, and the payment is a fixed constant (the constant is varied on the x-axis). In these figures, we “zoom in” to observe what happens at low rewards with short-term users.

As explained previously, the myopic mechanism pays the entire payment in one shot, thereby leading to more immediate fixes than the other mechanisms (see Figure 4.26). Since we are in a regime with short-term users only, the myopic mechanism achieves the maximum user utility once all bugs receive immediate fixes of any kind (see Figure 4.28). Further, accumulated payments (from cases where the fix remains too costly) provide an incentive for the worker to produce a deep enough fix in order to collect the entire accumulated amount. Thus we see deep fixes even with the myopic mechanism. However as the payment increases and completely covers the worker’s cost for a fix, the incentives for deep fixes go away. The worker simply gives the shallowest possible fix to claim the payment, and there is no further accumulation. As the reward increases beyond the amount shown in these figures, we revert to the case examined earlier in Figures 4.8, 4.9 and 4.10, where the eager mechanism was dominant.

**Limited workers case** In this experiment we explore the limited workers scenario in conjunction with the eager mechanism. We bound the number of recursions that are possible in any single lookahead to be no more than two. The parameter settings are: 5 root causes, bit string length 3, bugs generated uniformly at random, linear worker cost function with $c = 1$ for all workers, 5 installments, and payment sampled according to $r \sim \text{uniform}(5, 9)$. We
Figure 4.26: Percentage of immediate fixes with fixed payments and short-term users

Figure 4.27: Percentage of immediate deep fixes with fixed payments and short-term users

Figure 4.28: User utility with fixed payments and short-term users
vary the number of workers from 3 to 10.

In each time period the maximum number of bugs generated is the number of root causes in the system. Thus, by having fewer workers than bugs per time period, we increase the likelihood that a worker is reassigned to work on the same root cause. A worker can speculate via look-ahead sampling that his current choice of fix may be subsumed by a future fix also submitted by himself. From the worker’s point of view the payments lost due to subsumption are transferred to himself via a future deep fix. Hence we expect incentives to be shifted towards submitting shallow fixes, at least initially, with workers swooping in to submit deep fixes later in the lifetime of a root cause. Figure 4.29 plots the number of workers versus the fraction of submitted fixes that are deep (expressed as a percentage). As expected the percentage of submitted fixes that are deep increases with the number of workers.

### 4.6 Lessons for Market Design

We begin by summarizing the qualitative lessons learned from the experiments conducted. In designing a market, if we care more about the percentage of bugs that receive immediate deep fixes then the mechanism of choice should be eager subsumption. On the other hand,
if user utility is more important then the eager with reuse and lazy mechanisms are better choices. At low reward levels below a certain reward threshold (see Figure 4.27), and with short-term users, the myopic mechanism performs best. However, if the market evolves to higher rewards above the aforementioned threshold, and with long-term users, the performance of eager subsumption surpasses that of the myopic mechanism. In considering the installment mechanisms, installment with transfer bridges the performance gap between installment and eager subsumption. Finally, in a market with a limited number of workers, and implementing eager subsumption, we see that increasing the number of workers produces a higher percentage of fixes that are deep.

Next we classify salient features of the mechanisms examined in this work. Table 4.1 considers the question: what is the set of submitted fixes that are used to prevent new bugs from entering the economy? This can be done either by preemptively fixing new bugs so that they are not generated at all or by reusing a subsumed fix in the waiting list so that no new payments are introduced and no worker is assigned. Table 4.2 orders the mechanisms from simple mechanisms that involve no competition to increasingly more complex mechanisms that involve competition. Another classification of the mechanisms, depicted in Table 4.3, considers whether there is transfer of payments and whether fixes are compared against one another.

An automated market mechanism for bug fixes must take into consideration some important design criteria. These are:

1. Robustness with respect to the environment.
3. Low knowledge requirement. A portable and flexible mechanism makes fewer assumptions about the underlying system and can even be applied to different settings.
Table 4.1: Mechanisms and the set of fixes used to preempt bugs from entering the market

<table>
<thead>
<tr>
<th>All fixes submitted so far (equal the set of active fixes)</th>
<th>Myopic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Installment</td>
</tr>
<tr>
<td></td>
<td>Installment with transfer</td>
</tr>
<tr>
<td>All fixes submitted so far (subsumed fixes are reused)</td>
<td>Eager with reuse</td>
</tr>
<tr>
<td></td>
<td>Lazy subsumption</td>
</tr>
<tr>
<td>Set of active fixes (subsumed fixes are deleted)</td>
<td>Eager subsumption</td>
</tr>
</tbody>
</table>

Table 4.2: Mechanisms and competition

<table>
<thead>
<tr>
<th>No competition</th>
<th>Myopic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment as adversary</td>
<td>Installment</td>
</tr>
<tr>
<td></td>
<td>Installment with transfer</td>
</tr>
<tr>
<td>Competition and MFE</td>
<td>Eager subsumption</td>
</tr>
<tr>
<td></td>
<td>Eager with reuse</td>
</tr>
<tr>
<td></td>
<td>Lazy subsumption</td>
</tr>
</tbody>
</table>

Table 4.3: Mechanisms and transfers

<table>
<thead>
<tr>
<th>Transfers</th>
<th>Fix to fix comparisons</th>
<th>No fix comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eager subsumption</td>
<td>Installment with transfer</td>
</tr>
<tr>
<td></td>
<td>Eager with reuse</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lazy subsumption</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No transfers</th>
<th>Fix to fix comparisons</th>
<th>No fix comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N/A</td>
<td>Installment with transfer</td>
</tr>
<tr>
<td></td>
<td>Myopic</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Installment</td>
<td></td>
</tr>
</tbody>
</table>

The subsumption mechanisms satisfy all three design criteria. On the other hand, the myopic, installment, and installment with transfer mechanisms fail one or more of the design criteria. The subsumption mechanisms perform robustly across all configurations of the environment examined in our experimental study. The use of active fixes to preemptively fix new bugs is part of our model and common to all mechanisms. While this helps to reduce the number of bugs generated and therefore the number of fixes submitted on a root cause, subsumption mechanisms go one step further. Subsumption mechanisms delete fixes that have been judged to be subsumed, thereby striving proactively to eliminate redundant submissions. As a result the system evolves over time to retain the best and minimal set
of fixes. This evolution in the quality of the fixes that comprise a system is important for long-term maintenance. The check for whether one fix subsumes another does not strictly require the assumption of different underlying root causes – rather, it relies on making externally observable comparisons. In general, the problem of incentivizing deep, or equivalently high quality and lasting, fixes is not unique to software engineering. Subsumption mechanisms may be applied to settings other than software engineering with a modular and open design structure, where the quality of the system may evolve over time, and that share the phenomenon of incentivizing deep fixes.

Of the different subsumption variations, the lazy mechanism is more complicated to implement and has to track more information in order to delay subsumption judgements and redistribute payments. Moreover our experimental study was unable to demonstrate situations where lazy might perform significantly better than eager with reuse, thereby distinguishing itself as the mechanism of choice for certain environments. Therefore we believe that the eager subsumption and eager with reuse mechanisms hold the most promise for practical market design.

The installment and installment with transfer mechanisms employ a payment rule that stops paying when the next bug appears. Hence these mechanisms are vulnerable to aspects of the environment such as bug generation rate. In cases where a technology has been newly adopted, the initial quantity as well as rate of generation of bugs may be very high and therefore unresponsive to the depth of the initial fixes submitted. Over time and extensive use, a more stable version of the newly deployed technology should emerge. To this end, it would be desirable to implement a mechanism that produces deep fixes. In such cases, the installment mechanisms would not be the best choice. Furthermore, these mechanisms do not perform fix-to-fix comparisons and consequently do not delete redundant fixes. A bloated system that keeps around all fixes ever submitted is not desirable in practice. Finally the installment mechanisms have a high knowledge requirement. Curtailing workers’ payments
without a priori knowledge that a set of bugs belongs to the same root cause does not seem reasonable in practice.

The myopic mechanism does not meet the first two design criteria. However, in an interesting twist, the myopic mechanism gives deeper fixes and higher user utility than all other mechanisms at very low payment values. A restricted domain with short-term users who cannot afford high payments, but who do not mind waiting for a fix, and where the technology is not expected to last (thus design criterion 2 does not apply) might do well to implement the myopic mechanism. However a more flexible option would be to implement a subsumption mechanism where the number of installments is small (e.g., $h^* = 2$). This option meets all three design criteria while sharing the advantages of the myopic mechanism.

In conclusion, subsumption mechanisms satisfy all three market design criteria and perform robustly across all environment configurations examined in the experimental study. We have studied three subsumption variations in this work, but other variations exist. This class of mechanisms may be adapted to different settings and exhibit good properties for practical market design.
Chapter 5

A Theoretical Model of the Public Software Economy

5.1 Introduction

Our vision of a public software economy centres around the design of a market mechanism to more efficiently manage the development of software and to increase the achievability of correctness in software functionality. The goal is to use the methods of mechanism design and market design to create a self-regulating system that drives the process of developing software and managing the software ecosystem towards a type of market equilibrium in a dynamic setting. The use of a market confers important advantages: a market enables us to directly target incentives issues that abound in any development process via payments, it elicits valuations from users and allows them greater power to influence the direction of software development, it allows the aggregation of supply and demand thereby providing the scale needed to solve long neglected engineering problems, and what is more, prices act as signals and play a crucial role in the efficient allocation of resources in the market.

In the preceding chapter we presented a rich computational model to address a significant
challenge in such a market – that of how to incentivize deep fixes. We continue this line of research here. We consider a model that is a simplified abstraction of the computational model of the previous chapter in order to perform theoretical analysis and obtain further insights into the public software economy. First we prove the existence of a mean field equilibrium in this simple model. We then introduce a type of market equilibrium, that we call correctness equilibrium, that is adapted to handle the dynamic nature of the model presented in this chapter. Correctness equilibrium suggests a more achievable definition of software correctness than the traditional notion of absolute correctness, in that software may be considered correct if the market for bug fixes has cleared. We complete the chapter by examining the relationship between mean field equilibrium and correctness equilibrium in this context.

To begin we describe a simple model of deep fixes, which we refer to as the Jar model. Informally, in this model, we think of a root cause as a jar of water, where the deepest fix depth submitted so far is equivalent to the water level in the jar rising (i.e., increasing depth of water). As the water level rises it drowns out some of the bugs contained in the jar. In this model, a fix is characterized by a depth in \([0, 1]\), which can alternately be thought of as the quality of the fix. A fix depth of 0 indicates no fix submitted while a fix depth of 1 indicates that a fix of the deepest possible depth has been submitted, where all bugs in the jar are fixed (water level completely covers the jar). Thus the decision of a worker in this model is about which depth \(d\) of fix to submit. In the rest of this chapter we will use the phrases “submit a fix of depth \(d\)” and “submit a fix depth \(d\)” interchangeably.

We consider a simplified version of subsumption mechanisms introduced in the last chapter. Strictly deeper fixes belonging to a particular jar subsume less deep prior fixes of that jar. Let the current deepest fix depth be \(\bar{d}\). A single new bug \(k\) is available to fix every time period. For subsumption to occur, the next fix must be of depth \(d\) that is strictly greater than \(\bar{d}\). If so an update \(\bar{d} = d\) is performed. On any given root cause, only the worker that
submitted $\bar{d}$ is paid. Hence when a new fix subsumes the fix associated with $\bar{d}$, current payments stop and payments to the worker that submitted the new fix take over. The simplified subsumption mechanism is closest to the eager subsumption mechanism of Chapter 4 and is described in more detail in Section 5.2.1. Because we consider a dynamic setting where a sequence of subsumption mechanisms are executed over time, we refer to this setting as the repeated subsumption mechanisms game.

5.2 The Jar Model

The Jar model consists of the following parameters.

- Discrete time periods $T = \{1, 2, \ldots\}$.
- Set of workers $W$.
- Set of $n$ discrete, increasing, fix depths $D = \{d_1, d_2, \ldots, d_n\}$, where $d_1 = 0$, $d_n = 1$, and $0 < d_l < 1$ for $l = 2, \ldots, n - 1$.
- Set $RC$ of a fixed number of root causes of bugs.
- Associated with each root cause in $RC$ is the current deepest fix depth, denoted $\bar{d}$.
- Every time period, with probability $(1 - \beta)$, each root cause is regenerated ($\bar{d}$ is set to $0$ ∀ $q \in RC$), where $\beta \in (0, 1)$.
- Each time period a bug $k$ is generated on each root cause in $RC$.
- The user side of the model is minimal – the user is simply represented by a reward. Reward money is collected for a fix for bug $k$, denoted $r_k$ and such that $r_k \in [0, r_{max}]$ sampled according to a continuous distribution $f$, for some $r_{max} \in \mathbb{R}_+$. Note that reward is sampled once for any given bug and does not accumulate.
A worker $j \in W$ has linear cost $c_j d$ to provide fix of depth $d$. Cost parameter $c_j \in [c_{min}, c_{max}]$ is randomly sampled from a distribution over $[c_{min}, c_{max}]$, for some $c_{min}, c_{max} \in \mathbb{R}_+$. 

A worker $j \in W$ derives utility $w_j(d) \in \mathbb{R}$ from submitting a fix of depth $d$, where the utility comes from reward or payments received.

### 5.2.1 Subsumption Mechanism

We consider a dynamic setting that consists of a sequence of instantiations of a subsumption mechanism, such that the instantiations occur over time at each root cause. Let $\bar{d}$ denote the deepest fix depth submitted so far on a given root cause. Continuing we will assume that we always refer to the same root cause and thus omit using indices into the set $RC$. Each subsumption mechanism instance proceeds as follows.

1. Every time period, if a root cause does not regenerate, a single bug $k$, with reward $r_k$, is available to fix at the root cause. If $k$ is unfixed at time period $t$ then it remains available to fix at time period $t + 1$; that is, no new bug is generated and no new reward is sampled in this case.

2. A worker $j$ chosen at random is assigned to bug $k$.

3. Worker $j$ submits a fix depth $d \geq 0$, where a null fix of depth $d = 0$ corresponds to the worker choosing not to work.

4. If $d > \bar{d}$ then it is considered to have fixed the bug and subsumed past fixes. Payments to the worker that submitted $\bar{d}$ are stopped while payments to $j$ are begun. These payments consist of both the reward $r_k$ as well as any remaining reward of the previous fix associated with $\bar{d}$. In addition $d$ now replaces $\bar{d}$. Otherwise if $d \leq \bar{d}$ then existing payments towards $\bar{d}$ continue and $j$ is not paid.
5. The worker associated with $\bar{d}$ is paid an installment of the reward associated with his fix (defined below). Subsequent installments are paid every time period until the fix is subsumed or the root cause regenerates, whichever happens first.

At time period $t$, let the fix depth submitted by worker $j$ towards bug $k$ be $d$ and let the user reward sampled for $k$’s fix be $r_k$. Suppose $d > \bar{d}$, that is $d$ is deeper than the current deepest fix in the system. Then the fraction of the reward $r_k$ paid at time $t$ is denoted $r_k(t) = \frac{r_k}{2}$, and the worker gets half of $r_k$ immediately in the time period when he submits his fix. When $d > \bar{d}$, the total amount of the first installment paid to $j$ at time period $t$ is $\frac{r_k}{2} + \hat{s}(t)$, where $\hat{s}(t)$ is the remaining reward of the previous fix corresponding to $\bar{d}$ and is transferred conditional on subsumption (i.e., $d > \bar{d}$). In a subsequent time period $t + 1$, and provided the fix associated with $d$ is not subsumed, worker $j$ is paid $\frac{r_k(t)}{2}$; in words payments follow the geometric series. Thus the reward available to $j$ in the periods following the time period he submit his fix evolves as follows:

$$r_k(t + 1) = \begin{cases} 0 & \text{if fix associated with } d \text{ is subsumed} \\ \frac{r_k(t)}{2} & \text{otherwise} \end{cases} \quad (5.1)$$

If $d \leq \bar{d}$ then the fix leads to no reward, now or in the future. Worker $j$’s state at any time $t$ is given by $s_j(t) = (r_k(t), \bar{d}(t), \hat{s}(t), q_j(t))$, where $\bar{d}(t)$ is the deepest fix depth submitted until time period $t$, and $q_j(t) \in \{0, 1\}$ refers to $j$’s status at time $t$. $q_j(t) = 1$ if $j$ is submitting a fix at time $t$ whereas $q_j(t) = 0$ if $j$ has already submitted a fix at a previous time or is not assigned a bug. Let $d(t)$ denote the fix depth submitted at time period $t$. Worker $j$’s objective, when he submits a fix depth $d$, is to maximize his long-term expected utility $w_j(d)$. For convenience we normalize time so that time $t$ when worker $j$ submits a fix corresponds to time $t' = 1$. The worker’s utility for submitting depth $d$ is,
\[ w_j(d) = E[-c_j d + [\$\{t\} + \sum_{t'=1}^{\infty} \beta^{t'-1} - \frac{r_k}{2\beta} I_{\{d \geq d(t')\}}]I_{\{d \geq \bar{d}(1)\}}] \] (5.2)

Because the worker submits his fix depth \( d \) at time \( t' = 1 \), \( d(t') \) for \( t' = 1 \) is the worker’s own fix, i.e., \( d(1) = d \). However \( d(t') \) for \( t' > 1 \) is a random variable, referring to future fixes by other workers. Hence the expectation is taken with respect to this random variable in the above equation.

### 5.2.2 Mean Field Equilibrium

Let \( g \) denote the cumulative distribution function (CDF) of workers’ submitted fix depths. In the MFE model, each fix depth is assumed to be sampled according to \( g \). \( g \) is a non-decreasing function with support on \( \{0, \ldots, n\} \), representing depths, and such that \( g(0) \geq 0 \) and \( g(1) = 1 \). Because \( g \) is the CDF of a discrete probability distribution, it is entirely characterized by the set of values \( g(d) \forall d \in D \). Therefore all notions of continuity can be defined with respect to the Euclidean norm. Define \( G = \{g \in \mathbb{R}^n : \|g\|_2 \leq \sqrt{n}\} \) where \( \|\cdot\|_2 \) is the Euclidean norm. Note that when the current deepest fix depth submitted so far, denoted \( \bar{d} \), is high, the probability mass in \( g(d) \) for values of \( d \leq \bar{d} \) is considered to belong to cases where no new depth is submitted.

The probability that depth \( d \) is not subsumed at time period \( t' \) is given by \( g(d)^{t'-1} \). Let \( r, c, \$ \), and \( s \) represent reward, cost, subsumption transfer, and state respectively at time period \( t' \). Let \( V^*(s) \) denote the value function of the single period decision problem described above. Note that if at the time of submission \( d \leq \bar{d} \), where \( \bar{d} \) is the deepest fix depth submitted so far, then \( V^*(s) = -cd \). Otherwise \( V^*(s) \) is as follows.
Equation 5.3 returns the maximum possible value, over all depths, of the worker’s expected utility. Equation 5.8 is obtained by taking the sum of the geometric series. The worker’s best response, when assigned a bug and faced with a reward \( r \) associated with that bug, is to submit either depth 0 or some depth \( d \), where \( \bar{d} < d \leq 1 \). Let \( d^* \) denote the best response depth. Next we consider the relationship between \( d^* \) and \( r \).

**Lemma 12.** \( d^* \) is monotonically increasing with respect to \( r \), for any \( g \in G \) and for any state \( s \).

**Proof.** Consider depths \( d, d' \) such that \( d > d' > 0 \). Suppose that for a given \( g \) and \( r, d^* = d \).

Since \( g \) is non-decreasing in \( d \), \( g(d) \geq g(d') \) and therefore \( \frac{1}{1 - \frac{r g(d)}{2}} \geq \frac{1}{1 - \frac{r g(d')}{2}} \).

Now suppose that the reward is increased from \( r \) to \( r + \epsilon \) for some \( \epsilon > 0 \). Then, all else being the same, we have \( \frac{r + \epsilon}{1 - \frac{r g(d)}{2}} \geq \frac{r + \epsilon}{1 - \frac{r g(d')}{2}} \) and this holds for any \( \epsilon > 0 \). Hence given a state \( s \) and holding all other state variables constant, if you preferred \( d \) to \( d' \) before, you still will at any higher value of \( r \).

Finally let us consider the corner case where \( d > d' \) and \( d' = 0 \). Suppose that for a given \( g \) and \( r, d^* = d \). The worker’s expected utility for submitting \( d' = 0 \) is zero since this
corresponds to the case where the worker chooses not to work, and accordingly is not paid and incurs no cost. Since expected utility is strictly increasing in \( r \) and \( g \) is non-decreasing in \( d \), it follows that \(( -cd + \hat{s} + \frac{r + \epsilon}{1 - \beta g(d)} ) > 0 \) for any \( \epsilon > 0 \). That is, given a state \( s \), if the worker’s expected utility for submitting \( d \) was greater than zero for \( r \) then it will remain greater than zero for any higher value of \( r \).

\[ \text{Lemma 13.} \quad d^* \text{ is characterized by reward thresholds, for any } g \in G \text{ and for any state } s. \]

\[ \text{Proof.} \quad \text{Suppose there is only one non-zero best response depth, } d^* = 1, \text{ possible. Then the worker chooses } d^* \text{ if his utility for depth } 1 \text{ is greater than zero. This implies the following condition on reward } r \in \mathbb{R}_+, \]

\[ r \geq 2(cd^* - \hat{s})(1 - \frac{\hat{g}(d^*)}{2}) \]  \[ (5.9) \]

Substituting \( d^* = 1 \) and \( g(d^*) = g(1) = 1, \)

\[ r \geq 2(c - \hat{s})(1 - \frac{\hat{g}}{2}) \] \[ (5.10) \]

Equation (5.10) defines the threshold for \( d^* = 1 \) in this scenario. For reward range \( 0 \leq r < 2(c - \hat{s})(1 - \frac{\hat{g}}{2}) \) the worker chooses \( d^* = 0 \), and for \( r \geq 2(c - \hat{s})(1 - \frac{\hat{g}}{2}) \) he chooses \( d^* = 1 \).

Now consider the general case where there are \( m \) best response depths possible. For a given \( g \), let the set of best response depths be \( D^* = \{d_1, \ldots, d_m\} \), where \( D^* \subseteq D \) and \( d_{l-1} < d_l \forall d_{l-1}, d_l \in D^* \). Because the best response depth for any particular reward value depends on the distribution \( g \), the set of best response depths \( D^* \) might be a strict subset of the set of depths \( D \), for any given \( g \). In order to choose \( d_l \) over \( d_{l-1} \) we require that,
\[-cd_l + s + \frac{\frac{s}{2}}{1 - \frac{\beta g (d_l)}{2}} \geq -cd_{l-1} + s + \frac{\frac{s}{2}}{1 - \frac{\beta g (d_{l-1})}{2}} \]  \hspace{1cm} (5.11)

Cancelling terms common to both sides of the equation and rearranging we get,

\[\frac{\frac{s}{2}}{1 - \frac{\beta g (d_l)}{2}} \geq \frac{\frac{s}{2}}{1 - \frac{\beta g (d_{l-1})}{2}} + c(d_l - d_{l-1}) \]  \hspace{1cm} (5.12)

\[\frac{r}{2} \geq \frac{c(d_l - d_{l-1})}{1 - \frac{\beta g (d_l)}{2} - \frac{1}{1 - \frac{\beta g (d_{l-1})}{2}}} \]  \hspace{1cm} (5.13)

\[r \geq \frac{2c(d_l - d_{l-1})}{1 - \frac{\beta g (d_l)}{2} - \frac{1}{1 - \frac{\beta g (d_{l-1})}{2}}} \]  \hspace{1cm} (5.14)

Thus the reward threshold \( \tau_l (g) \) where the best response depth switches from \( d_{l-1} \) to \( d_l \) is given by:

\[ \tau_l (g) = \frac{2c(d_l - d_{l-1})}{1 - \frac{\beta g (d_l)}{2} - \frac{1}{1 - \frac{\beta g (d_{l-1})}{2}}} \] \hspace{1cm} \( l \geq 2 \)  \hspace{1cm} (5.15)

Note that \( \tau_1 (g) \) refers to the case where the worker decides between submitting the minimal depth in \( D^* \) that provides positive utility versus submitting \( d = 0 \) and getting zero utility. Recall that nonzero depths that are not in \( D^* \) provide negative utility because non-null fixes of depth less than \( \bar{d} \) receive no reward but do incur a cost. The reward threshold \( \tau_1 (g) \) is therefore the same as in Equation 5.9 and is given by,

\[ \tau_1 (g) = 2(c d_1 - s)(1 - \frac{\beta g (d_1)}{2}) \]  \hspace{1cm} (5.16)

We get the following behaviour. For reward range \( \tau_1 (g) \leq r < \tau_2 (g) \), best response depth \( d^* = d_1 \). For \( \tau_2 (g) \leq r < \tau_3 (g) \), \( d^* = d_2 \). Continuing in a similar fashion, we end
Figure 5.1: Best response depths versus reward thresholds.

with $d^* = d_m$ for $r \geq \tau_m(g)$.

Lemma 14. Each reward threshold is continuous in $g$, for any $g \in G$.

Proof. By Lemma 13, for any $g$ and corresponding set of best response depths $D^*$, the threshold $\tau_l(g)$ where the best response depth switches from $d_{l-1}$ to the next higher depth $d_l$, $\forall d_{l-1}, d_l \in D^*$ and for $l \geq 2$, is described by Equation 5.15.

Consider the first term in the denominator of $\tau_l(g)$, $\frac{1}{1 - \frac{\beta g(d_l)}{2}}$. This term is the quotient of two terms that are each continuous in $g$ and is therefore itself continuous in $g$. The same applies to the second term in the denominator. Since the denominator is the difference of these two terms, it is also continuous in $g$. The numerator in $\tau_l(g)$ does not depend on $g$. Hence $\tau_l(g)$ is continuous in $g$.

Threshold $\tau_1(g)$ where the worker decides to submit the minimal depth in $D^*$ is given by Equation 5.16. Since $\tau_1(g)$ consists of a linear combination of terms, it is continuous in $g$.  

\[ \Box \]
Next let us turn to the mean field game where a worker models the future as facing an i.i.d. distribution $g$ of fix depths submitted by other workers. For each bug, reward $r$ is drawn from a distribution $f$ where $f$ is continuous with support on $[0, r_{\text{max}}]$, for some $r_{\text{max}} \in \mathbb{R}_+$. A worker selected at random then best responds with fix depth $d^*$, resulting in a distribution on fix depths played. We assume that worker’s cost $c$ is sampled from a distribution (see Section 5.2). We further assume that there exist distributions over the values of current deepest depth $\bar{d}$ as well as subsumption transfer $\hat{s}$. In particular let $\chi$ denote the marginal distribution on cost $c$ (i.e., integrated over time), let $\phi$ denote the marginal distribution on current deepest depth $\bar{d}$, and let $\gamma$ denote the marginal distribution on subsumption transfer $\hat{s}$. Further, let $\bar{\psi}(g)$ be the distribution on fix depths played for a fixed $c$, $\bar{d}$, and $\hat{s}$. Let us turn our attention to the worker’s best responses under $g$ versus $g' \in G$, where $\|g' - g\| < \delta$ for some $\delta > 0$. Let $\tau_l \leq \tau_{l+1}$ and $\tau_l' \leq \tau_{l+1}'$ represent successive reward thresholds under $g$ and $g'$ respectively. For thresholds $\tau_l$, the probability that $d_k$ is picked is given by $Pr(d^* = d_k) = \int_{r=\tau_l}^{\tau_{l+1}} f(r)dr$. There are four cases.

**Case 1:** A best response depth $d_k$ is present under $g$ as well as under $g'$. The difference...
in the probability that \( d_k \) is a best response for some reward values, when using \( g' \) versus \( g \) is given by,

\[
\Delta Pr(d^* = d_k) = \int_{r = \tau_i}^{r = \tau_{i+1}} f(r) dr - \int_{r = \tau_i}^{r = \tau_{i+1}} f(r) dr
\]

(5.18)

By Lemma 12, the integral limits are well-defined. Each integral is continuous over the interval defined by the limits of integration. By Lemma 14, the reward thresholds are continuous in \( g \). Therefore the limits of the integrals are continuous in \( g \) and the difference in area of integration is also continuous in \( g \). It follows that the change in probability \( \Delta Pr(d^* = d_k) \) is continuous in \( g \). Similar arguments apply to the other three cases listed below.

**Case 2:** A best response depth \( d_k \) is present under \( g \) but not under \( g' \). Here \( d_k \) is never a best response under \( g' \) and thus the corresponding integral evaluates to 0.

\[
\Delta Pr(d^* = d_k) = 0 - \int_{r = \tau_i}^{r = \tau_{i+1}} f(r) dr
\]

(5.19)

By the same arguments as used for **Case 1**, the change in probability \( \Delta Pr(d^* = d_k) \) is continuous in \( g \).

**Case 3:** A best response depth \( d_k \) is not present under \( g \) but appears under \( g' \). Here \( d_k \) is never a best response under \( g \) and thus the corresponding integral evaluates to 0.

\[
\Delta Pr(d^* = d_k) = \int_{r = \tau_i}^{\tau_{i+1}} f(r) dr - 0
\]

(5.20)

By the same arguments as used for **Case 1**, the change in probability \( \Delta Pr(d^* = d_k) \) is continuous in \( g' \).

**Case 4:** A best response depth \( d_k \) is present under neither \( g \) nor \( g' \). Hence \( \Delta Pr(d^* = d_k) = 0 \).
We have shown that, given \( c, \bar{d}, \) and \( \hat{s} \), the distribution over fix depths played \( \bar{\psi}(g) \) is continuous in \( g \) for any \( g \). By Equation 5.17 it follows that the long run marginal distribution over fix depths played, \( \psi(g) \), is continuous in \( g \). This completes the proof.

\[ \square \]

**Definition 17.** A mean field equilibrium is defined as a distribution \( g \) such that \( \psi(g) = g \).

**Theorem 11.** There exists a mean field equilibrium in the repeated subsumption mechanisms game.

**Proof.** By Lemma 15, \( \psi(g) \) is continuous in \( g \), for any \( g \in G \). \( G \) is a convex, compact subset of Euclidean space. Applying Brouwer fixed point theorem it follows that \( \psi(g) = g \). \( \square \)

### 5.3 Correctness Equilibrium

We introduce a type of market or price-taking equilibrium tailored to our setting called a correctness equilibrium (CE). In the context of a private ownership economy, a market equilibrium is also known as a Walrasian or competitive equilibrium. This section follows the treatment of competitive equilibrium in microeconomics (e.g., [55], [74]), adapting standard techniques to the current scenario.

We model a private ownership production economy in a setting where there is a set of root causes that generate bugs, in any time period root causes regenerate with probability \( (1 - \beta) \), there are users and workers, with demand for and supply of fix depths, and where payments take the form specified by the subsumption mechanism in Section 5.2.1. Because our setting is inherently dynamic, we tailor the notion of a competitive equilibrium to incorporate properties of a mean field equilibrium. Thus correctness equilibrium requires market-clearing conditions to hold, but over multiple time periods that are connected through the subsumption mechanism. What this means is that a worker may not get the entire price
for supplying a particular fix depth, although prices now exist. In order for the worker to estimate how much of the price he may lose due to being subsumed by a later fix, he must factor in beliefs regarding the distribution of depths submitted by other workers. Accordingly a correctness equilibrium insists on both market-clearing prices as well as consistency of beliefs. Hence correctness equilibrium is a type of competitive equilibrium but adapted to include inter-temporal considerations.

While we retain several features of the MFE model in Section 5.2, the model in this section differs from that model in ways that are necessary in order to design a market and analyze the equilibrium in this market. For instance, workers and users are associated with a continuum of types which allows for continuous demand and supply functions. The user side is more developed here and users have a utility function that describes their utility for fix depths received and gives rise to aggregate demand. As is standard in a market setting, workers best respond to prices for the different goods or fix depths in the market instead of facing randomly sampled rewards as in Section 5.2. Further, in the MFE model a worker is selected at random and assigned to a bug. As a result worker-to-bug matchings may be suboptimal where an assigned worker may be unable to produce a fix depth while there exists an unassigned worker that could produce the required fix depth. In contrast workers in a price-taking equilibrium model respond to prices and thus choose to supply the fix depth that optimizes their utility. We also introduce an additional good, “time” as the input to the production of goods. This can be interpreted as the worker purchasing labour hours, representing the cost incurred by the worker to engage in work for a given length of time, in order to be able to produce goods or fix depths and earn wages by selling them in the market. A standard assumption is that consumers are shareholders in the units of production in a private ownership economy. We assume likewise and users in our model earn a share of the total profit of all workers in the market.
5.3.1 Preliminaries

The set of goods in our market consists of \( n \) nonzero fix depths, and an additional good that we call \textit{time}. Time is the input to the worker’s production function, where the amount required is specified by the worker’s cost function \( c(\cdot) \). A worker may submit a fix of depth 0 to indicate that he chooses not to work. Each worker \( j \) is associated with a type \( \bar{t} > 0 \) with support in \( \mathbb{R}_{++} \). Denote the space of worker types by \( \bar{T} \), where \( (\bar{T}, \chi_{\bar{T}}) \) is a measurable space with \( \sigma \)-algebra of subsets \( \chi_{\bar{T}} \) and measure \( \mu_{\bar{T}} \). Worker \( j \) has production vector \( y_j = (y_{j0}, y_{j1}, \ldots, y_{jn}, y_{jn+1}) \), \( y_j \in Y_j \subset \mathbb{R}^{n+2} \), where the first \( n + 1 \) coordinates \( y_{jm} \geq 0 \) represent fix depths supplied by the worker and \( y_{jn+1} \leq 0 \) represents the quantity of time demanded by the worker. Production sets \( Y_j \) are closed and bounded above for all \( j \). A worker produces a single unit of the depth that maximizes his utility and thus at most one of the first \( n + 1 \) coordinates is equal to 1 while the remaining fix depth coordinates are set to 0.

Workers have beliefs regarding the distribution of depths submitted by other workers, denoted \( g \) and defined as in Section 5.2.2. This need not be exactly the same belief \( g \) as in the MFE model although the definition of the CDF is the same. Regardless we will use the same notation here. The distribution of fix depths \( g \) has support on \( \{0, \ldots, n\} \) because submitting depth 0 is part of the set of actions that a worker may take. We include depth 0 in the production and consumption vectors described below in order to be consistent with \( g \). However we do not include depth 0 in the market clearing condition since a worker refusing to work, thereby submitting fix depth 0, is not equivalent to a user choosing to buy nothing and thus demanding none of the nonzero depths.

Each user \( i \) is associated with a type \( \hat{t} > 0 \) with support in \( \mathbb{R}_{++} \). Denote the space of user types by \( \hat{T} \), where \( (\hat{T}, \chi_{\hat{T}}) \) is a measurable space with \( \sigma \)-algebra of subsets \( \chi_{\hat{T}} \) and measure \( \mu_{\hat{T}} \). User \( i \) has consumption bundle \( x_i = (x_{i0}, x_{i1}, \ldots, x_{in}, x_{in+1}) \), \( x_i \in X_i \subset \mathbb{R}^{n+2} \), where the first \( n + 1 \) coordinates \( x_{im} \geq 0 \) represent fix depths demanded by the user and \( x_{in+1} \leq 0 \).
represents the quantity of time supplied by the user. The bundle demanded by the user can be composed of fractional depths as well as multiple different depths. A user of type \( \hat{t} \) has endowment \( \epsilon(\hat{t}) > 0 \) of the good time, and sampled from a continuous distribution with support in \( \mathbb{R}_+ \). In addition users are shareholders in the units of production in this economy. Thus a user of type \( \hat{t} \) has a share \( 0 \leq \theta(\hat{t}) \leq 1 \) of the total profit generated by the set of all workers, and such that \( \int_{\hat{T}} \theta(\hat{t}) \mu_{\hat{T}} = 1 \).

The vector of prices for the set of goods in the market is denoted \( p = (p_1, \ldots, p_n, p_{n+1}) \), \( p \in \mathbb{R}_{n+1}^+ \). Note that while price vector \( p \) can be nonnegative out of equilibrium we require \( p \) to be strictly positive in equilibrium. This is explained in a later section.

## 5.3.2 Worker Utility and Aggregate Supply

Consider a worker \( j \) of type \( \bar{t} > 0 \) assigned to work on a root cause where the current deepest fix depth submitted is \( \bar{d} \). Worker \( j \)'s utility \( u_j \) to produce \( d \), given prices \( p \) and belief \( g \), is given by,

\[
u_j(\bar{t}, d, \bar{d}, p, g) = r(d, \bar{d}, p, g) - p_{n+1}c(\bar{t}, d)\tag{5.21}
\]

The amount of the input time needed to produce a single unit of the output \( d \) by a worker of type \( \bar{t} \) is specified by a continuous production function, \( c(\bar{t}, d) \geq 0 \), that is strictly increasing in \( d \). We choose a function \( c(t, d) \) such that higher types need less of the input (or alternately incur lower cost) to produce a fix of depth \( d \). An example of such a function is \( c(\bar{t}, d) = \bar{d}/\bar{t} \), which we will use henceforth. The worker's expected reward is denoted \( r(d, p, g) \) and defined below. This is different from the randomly sampled reward of Section 5.2. Here \( r(d, p, g) \) tells us how much of the price \( p_d \) for fix depth \( d \) the worker can expect to retain, in addition to any subsumption transfers, given that he is paid according to the subsumption mechanism payment rule described in Section 5.2.1.
Recall from Section 5.2.1 that the worker does not get paid if he submits \( d \leq \bar{d} \), that is in this case \( r(d, \bar{d}, p, g) = 0 \). In the context of a price-taking equilibrium this does not mean that the price of depth \( d \) is zero, but rather that there is no demand for a depth that is less than the current deepest depth at a root cause. Because the worker wants to maximize his expected utility, his best response is to either submit \( d > \bar{d} \) or \( d = 0 \), thereby getting utility \( u_j(\cdot) \geq 0 \). In what follows we will focus on the case \( d > \bar{d} \).

\[
r(d, \bar{d}, p, g) = \begin{cases} 
\hat{r}(d, p, g) + \hat{s}(\bar{d}, p, g) & \text{if } d > \bar{d} \\
0 & \text{otherwise}
\end{cases}
\]

(5.22)

The first term \( \hat{r}(d, p, g) \) is borrowed from Equation (5.8) and represents the expected amount of the price for a fix of depth \( d \) that worker \( j \) retains, while losing the remainder to a competing worker due to subsumption, given that \( j \) is paid according to the subsumption mechanism payment rule described in Section 5.2.1. Clearly \( \hat{r}(\cdot) \) is continuous in \( p \) and \( g \).

The second term \( \hat{s}(\bar{d}, p, g) \) is the amount transferred to \( j \) because his fix (depth) subsumes a previous fix of depth \( \bar{d} \). Thus \( \hat{s}(\cdot) \) is the remainder of payments made towards \( \bar{d} \) leftover unpaid. Note that for simplicity \( \hat{s}(\cdot) \) is paid as one lump-sum amount and not in installments as done with \( p_d \). \( \hat{s}(\cdot) \) is defined below.

\[
\hat{s}(\bar{d}, p, g) = p_d - \hat{r}(\bar{d}, p, g) 
\]

(5.25)

Since \( \hat{r}(\cdot) \) is continuous in \( p \) and \( g \), it follows that \( \hat{s}(\cdot) \) is also continuous in \( p \) and \( g \).
Thus $r(\cdot)$ is continuous in $p$ and $g$. A worker $j$ of type $\bar{t}$ will supply a fix of depth $d$ such that,

$$\max_{d \in \mathbb{D}} u_j(\bar{t}, d, \bar{d}, p, g) \quad (5.26)$$

### 5.3.3 User Utility and Aggregate Demand

Before defining the user’s utility maximization problem, we define the terms that characterize user preferences.\(^1\) Consider the consumption set of user $i$, $X_i \subset \mathbb{R}^{n+2}$. Let $x_i, x_i' \in X$ denote consumption bundles and let $\succeq$ denote the preference relation of users, such that $x_i \succeq x_i'$ means that $x_i$ is weakly preferred to $x_i'$. User preferences are complete if for all $x_i, x_i' \in X$, it is the case that either $x_i \succeq x_i'$ or $x_i' \succeq x_i$ or both hold. Preferences are transitive if for all $x_i, x_i', x_i'' \in X$, if $x_i \succeq x_i'$ and $x_i' \succeq x_i''$ then it must be that $x_i \succeq x_i''$. The assumption that preferences are locally non-satiated stipulates that for any bundle $x_i'$ and within any distance $\epsilon > 0$, there exists a bundle $x_i$ that is strictly preferred to $x_i'$. A stronger assumption than local non-satiation is monotonicity. Preferences are monotone if commodities are considered to be “goods” and not “bads”. This means that a consumption bundle $x_i$, that contains more of some commodities and equal amounts of the remaining commodities than a bundle $x_i'$, is weakly preferred. However $x_i$ is strictly preferred to $x_i'$ if preferences make the stronger assumption of strong monotonicity. Note that monotone preferences are also locally non-satiated. Under convexity of preferences, if for all $x_i, x_i', x_i'' \in X$, if $x_i' \succeq x_i$ and $x_i'' \succeq x_i$ then $\alpha x_i' + (1 - \alpha) x_i'' \succeq x_i$ for any $\alpha \in [0, 1]$. Strict convexity requires that $\alpha x_i' + (1 - \alpha) x_i'' \succ x_i$ for any $\alpha \in [0, 1]$. Informally convexity can be interpreted as users having a taste for diversification. Finally, preferences are continuous if a user who weakly prefers each element in a sequence of bundles $\{x_i^n\}$ to each element of another sequence $\{x_i''^m\}$ retains

\(^1\)For further details see [74].
this preference at the limit points of these sequences.

We make the following standard assumption regarding a user’s preference relation over all commodities in our setting. In particular note that we assume that users in our setting are strongly monotone. This can be interpreted to mean that given two consumption bundles, where one bundle has more of some things, i.e., fix depths or the good time, and at least equal amounts of everything, the user is always strictly better off with the latter bundle. In our context, this suggests that the user could always do with more fix depths supplied as he will eventually use other functionalities in the software and encounter bugs therein.

**Assumption 1.** Each user’s preferences are (i) complete, (ii) transitive, (iii) strongly monotone, (iv) strictly convex, and (v) continuous.

Consider a user $i$ of type $\hat{t} > 0$. User $i$ maximizes his utility $v_i(\hat{t})$ for acquiring a consumption bundle $x_i \geq 0$ given prices and endowment. The user’s utility maximization problem is given by,

$$\max_{x_i \in X_i} v_i(\hat{t}, x_i)$$

subject to

$$p \cdot x_i \leq p \cdot e(\hat{t}) + \theta(\hat{t}) \int_{\bar{T}} u(\bar{t}, p, g) d\mu_{\bar{T}}$$

where $\int_{\bar{T}} u(\bar{t}, p, g) d\mu_{\bar{T}}$ is the total profit generated by the set of all workers. Given complete, transitive, and continuous preferences, there exists a continuous utility function $v_i(\hat{t}, x_i)$ that represents this preference relation (see [74], p. 47). Moreover the demand function $x_i(\hat{t}, p)$ associated with $v_i(\hat{t}, x_i)$ is continuous at all $p \gg 0$ and $e(\hat{t}) > 0$ (see [74], p. 93). A suitable utility function might be one where higher types get higher value for acquiring a fix of depth $d$, for example suppose the bundle $x_i$ consists of a request for just a single fix of depth $d$, then $v_i(\hat{t}, d) = d^{\hat{t}}$.

Note that users consume a bundle that may include fractional goods. Whereas each
worker produces a single fix of a particular depth that maximizes his expected utility. However, because our model consists of a continuum of both users and workers, this difference between the demand side and the supply side does not prevent existence of an equilibrium.

5.3.4 Definition and Properties

We are now ready to present the formal definition of correctness equilibrium. Note that $g^*$ is the distribution obtained by extracting the first $n + 1$ coordinates of $y^*$ and normalizing.

**Definition 18.** The allocation $(x^*, y^*)$, price vector $p^*$, and distribution of depths $g^*$ constitute a correctness equilibrium if the following conditions are satisfied:

(i) Each worker $j$ maximizes his utility, given current fix depth, $\tilde{d}$, at a root cause. $y^*_j$ solves

$$
\max_{y_j \in Y_j} u_j(\tilde{t}, y_j, \tilde{d}, p^*, g) \tag{5.29}
$$

(ii) Each user $i$ maximizes his utility. $x^*_i$ solves

$$
\max_{x_i \in X_i} v_i(\hat{t}, x_i, p^*) \tag{5.30}
$$

(iii) The market clears, $\forall p^* \gg 0$.

$$
\int_{\hat{T}} x^*(\hat{t}, p^*)d\mu_{\hat{T}} = \int_{\tilde{T}} e(\hat{t})d\mu_{\tilde{T}} + \int_{\tilde{T}} y^*(\tilde{t}, p^*, g)d\mu_{\tilde{T}} \tag{5.31}
$$

(iv) The distribution of depths $g^*$ induced by the aggregate supply $y^*$ is consistent with the workers’ belief $g$, i.e., $g = g^*$.

The standard model of market equilibrium does not consider workers’ belief $g$ and does not require condition (iv) above. CE departs from standard market equilibrium by incorporating conditions for a MFE into a market economy. CE requires market-clearing conditions
to hold, but over multiple time periods that are connected through the subsumption mechanism. Because a worker is paid in installments in a subsumption mechanism, he does not receive the full amount of the price for a fix depth immediately. Instead the worker must calculate his expected payment given that his fix may be subsumed by a deeper fix arriving later in time. This requires the worker to take into account the distribution of fix depths submitted by all workers, $g$. Therefore workers in a CE best respond to prices $p$ as well as belief $g$ and the current state of the system.

Although we do not model the system state changing explicitly, the current state is taken into account in the worker’s utility function that takes the current depth as argument and includes subsumption transfers in computing the expected utility. In a CE both prices $p$ and belief $g$ must be stationary in the long run. Because aggregate supply and demand are defined in terms of the utility functions of workers and users, and since these utility functions are defined per time period, the market clears every time period in a CE. We can think of the CE model as an approximation of a more complicated system evolving behind the scenes, where agents must reason about the dynamics of the future.

Next we establish some properties of the aggregate supply and demand.

**Lemma 16.** The depth $d$ chosen by a worker is monotonically increasing in type $t$, given $p$ and $g$.

**Proof.** Consider depths $d$, $d'$ such that $d > d'$. Suppose that for a given $p$ and $g$ a worker of type $\bar{t}$ chooses to supply $d$. Now suppose that the worker’s type is increased from $\bar{t}$ to $\bar{t} + \epsilon$ for some $\epsilon > 0$. Then, all else being constant, the cost to produce $d$ decreases by a factor $d(\frac{1}{\bar{t}} - \frac{1}{\bar{t}+\epsilon})$. Likewise the cost to produce $d'$ would decrease by a factor $d'(\frac{1}{\bar{t}} - \frac{1}{\bar{t}+\epsilon})$. But since $d > d'$, we have that $d(\frac{1}{\bar{t}} - \frac{1}{\bar{t}+\epsilon}) > d'(\frac{1}{\bar{t}} - \frac{1}{\bar{t}+\epsilon})$. Therefore if $d$ was preferred by type $\bar{t}$ then it remains preferred by type $\bar{t} + \epsilon$ for any $\epsilon \geq 0$.

**Lemma 17.** Aggregate supply is continuous in $p$ and $g$. 
Proof. Consider depths \( d_l, d_k \) such that \( \bar{d} \leq d_l < d_k \). If worker \( j \) prefers to supply \( d_k \) to \( d_l \) then it must be that,

\[
\begin{align*}
  u_j(\bar{t}, d_k, \bar{d}, p, g) & \geq u_j(\bar{t}, d_l, \bar{d}, p, g) \tag{5.32} \\
  r(d_k, \bar{d}, p, g) - p_{n+1}c(\bar{t}, d_k) & \geq r(d_l, \bar{d}, p, g) - p_{n+1}c(\bar{t}, d_l) \tag{5.33} \\
  r(d_k, \bar{d}, p, g) - r(d_l, \bar{d}, p, g) & \geq p_{n+1}c(\bar{t}, d_k) - p_{n+1}c(\bar{t}, d_l) \tag{5.34} \\
  \frac{r(d_k, \bar{d}, p, g) - r(d_l, \bar{d}, p, g)}{p_{n+1}} & \geq \frac{d_k - d_l}{\bar{t}} \tag{5.35} \\
  \bar{t} & \geq \frac{p_{n+1}(d_k - d_l)}{r(d_k, \bar{d}, p, g) - r(d_l, \bar{d}, p, g)} \tag{5.37}
\end{align*}
\]

Equation (5.37) describes a type threshold such that for type values \( \bar{t}^* > \bar{t} \) the worker prefers to supply \( d_k \) over \( d_l \). Now consider any two successively deeper depths, \( d_l, d_{l+1} \), where \( d_l \geq \bar{d} \). Then the threshold value where the utility maximizing depth switches from \( d_l \) to \( d_{l+1} \) is given by,

\[
\tau_{l+1} = \frac{p_{n+1}(d_{l+1} - d_l)}{r(d_{l+1}, \bar{d}, p, g) - r(d_l, \bar{d}, p, g)} \tag{5.38}
\]

Thus for type range \( \bar{t} \leq \tau_{l+1} \), the worker prefers to supply depth \( d_l \), for \( \tau_{l+1} < \bar{t} \leq \tau_{l+2} \) the worker prefers to supply depth \( d_{l+1} \), for \( \tau_{l+2} < \bar{t} \leq \tau_{l+3} \) the worker prefers to supply depth \( d_{l+2} \), and so on. Note that in the case where \( d_l = \bar{d} \), a worker of type \( \bar{t} \leq \tau_{l+1} \) will supply \( d = 0 \).

Since \( r(\cdot) \) is continuous in \( p \) and \( g \), the type thresholds described by Equation (5.38) are also continuous in \( p \) and \( g \). Therefore the size of the set of types that supply a depth \( d \) is continuous in \( p \) and \( g \), for any depth \( d \). To see this, note that aggregate supply for \( d \) is given by \( \int_{\bar{t}} y(\bar{t}, d, p, g) d\mu_{\bar{t}} \). The integral is continuous over the interval defined by the limits of
integration. It follows that aggregate supply for $d$ is continuous in $p$ and $g$. This holds for all depths and thus the overall aggregate supply, $\int_T y(\hat{t}, p, g) d\mu_{\hat{T}}$, is continuous in $p$ and $g$. 

**Lemma 18.** Aggregate demand is continuous in $p$, for $p \gg 0$.

*Proof.* As mentioned above, there exists a continuous utility function $v(\hat{t})$ that represents a user’s preference relation (see [74], p. 47), and further, the demand function $x(\hat{t}, p)$ associated with $v(\hat{t})$ is continuous at all $p \gg 0$, given $e(\hat{t}) \gg 0$ (see [74], p. 93). Aggregate demand is given by $\int_{\hat{T}} x(\hat{t}, p) d\mu_{\hat{T}}$. The latter integral is continuous over the interval defined by the limits of integration. Hence continuity of $x(\hat{t}, p)$ in $p$ implies continuity of aggregate demand in $p$, for $p \gg 0$. 

5.3.5 **Existence**

Given workers’ belief $g$, the aggregate excess demand function is given by,

$$z(p, g) = \int_T x(\hat{t}, p) d\mu_{\hat{T}} - \int_T e(\hat{t}) d\mu_{\hat{T}} - \int_T y(\bar{t}, p, g) d\mu_{\bar{T}}$$ (5.39)

Aggregate excess demand is equal to aggregate demand in the economy minus the sum of aggregate supply and aggregate endowment of the goods demanded. In other words it is the total quantity of all goods demanded over and above what consumers in the economy already possess as endowment and the total quantity of all goods supplied.

We first formalize the budget balance property of our system before proceeding to show that Walras’ law holds. Specifically we observe that no payments are “lost” in our system. When a fix subsumes a previous fix, any remaining unpaid installments associated with the previous fix are transferred to the worker who submitted the subsuming fix. Otherwise when a root cause regenerates all remaining installments are paid to the worker that is receiving the current payment. In the case where there is no subsumption and no regeneration of the
root cause yet, the worker simply receives all the payment installments associated with his fix. Fact 1 states that the sum of all transfers is equal to the sum of all payments leftover unpaid due to subsumption interruptions.

**Fact 1.** \( \sum_{d=1}^{n} \int_{\bar{T}} y_d(\bar{t}, p, g)(\hat{s}(\bar{d}, p, g) - (p_d - \hat{r}(d, p, g)))d\mu_{\bar{T}} = 0 \)

Walras’ law states that aggregate excess demand \( z(\cdot) \) is zero for any price vector \( p \in \mathbb{R}^{n+1}_+ \). This result is obtained because when consumer or user preferences are locally non-satiated, each user’s budget constraint binds with equality. In other words each user expends his entire wealth during his lifetime.

**Lemma 19.** Walras’ law holds for any belief \( g \) and for any prices \( p \), i.e. \( p \cdot z(p, g) = 0 \).

**Proof.** Consider \( \int_{\bar{T}} u(\bar{t}, d, p, g)d\mu_{\bar{T}} \), the total profit over all worker types to supply a particular depth \( d \).

\[
\int_{\bar{T}} u(\bar{t}, d, p, g)d\mu_{\bar{T}} = \int_{\bar{T}} y_d(\bar{t}, p, g)(\hat{r}(d, p, g) + \hat{s}(\bar{d}, p, g))d\mu_{\bar{T}} - \int_{\bar{T}} c(\bar{t}, d)p_{n+1}d\mu_{\bar{T}}
\]

(5.40)

Summing over all depths,

\[
\sum_{d=1}^{n} \int_{\bar{T}} u(\bar{t}, d, p, g)d\mu_{\bar{T}} = \sum_{d=1}^{n} \int_{\bar{T}} y_d(\bar{t}, p, g)(\hat{r}(d, p, g) + \hat{s}(\bar{d}, p, g))d\mu_{\bar{T}} - \sum_{d=1}^{n} \int_{\bar{T}} c(\bar{t}, d)p_{n+1}d\mu_{\bar{T}}
\]

(5.41)

\[
= \int_{\bar{T}} u(\bar{t}, p, g)d\mu_{\bar{T}}
\]

(5.42)

Substituting \( p_d - (p_d - \hat{r}(d, p, g)) \) for \( \hat{r}(d, p, g) \),

144
\[
\int_{\bar{T}} u(\bar{t}, p, g) d\mu_{\bar{T}} = \sum_{d=1}^{n} \int_{\bar{T}} y_d(\bar{t}, p, g) p_d d\mu_{\bar{T}} + \\
\sum_{d=1}^{n} \int_{\bar{T}} y_d(\bar{t}, p, g) (\hat{s}(d, p, g) - (p_d - \hat{r}(d, p, g))) d\mu_{\bar{T}} - \\
\sum_{d=1}^{n} \int_{\bar{T}} c(\bar{t}, d) p_{n+1} d\mu_{\bar{T}} 
\]  

(5.43)

By Fact 1 the second term on the RHS in Equation (5.43) is equal to 0. Thus we have,

\[
\int_{\bar{T}} u(\bar{t}, p, g) d\mu_{\bar{T}} = \sum_{d=1}^{n} \int_{\bar{T}} y_d(\bar{t}, p, g) p_d d\mu_{\bar{T}} - \sum_{d=1}^{n} \int_{\bar{T}} c(\bar{t}, d) p_{n+1} d\mu_{\bar{T}} 
\]  

(5.44)

\[
= \sum_{d=1}^{n} \int_{\bar{T}} y_d(\bar{t}, p, g) p_d d\mu_{\bar{T}} - p_{n+1} \int_{\bar{T}} c(\bar{t}) d\mu_{\bar{T}} 
\]  

(5.45)

Letting \(-y_{l}(\bar{t}) = c(\bar{t})\) and combining terms,

\[
\int_{\bar{T}} u(\bar{t}, p, g) d\mu_{\bar{T}} = \sum_{d=1}^{n} \int_{\bar{T}} y_d(\bar{t}, p, g) p_d d\mu_{\bar{T}} + p_{n+1} \int_{\bar{T}} y_{l}(\bar{t}) d\mu_{\bar{T}} 
\]  

(5.46)

\[
= p \cdot \int_{\bar{T}} y(\bar{t}, p, g) d\mu_{\bar{T}} 
\]  

(5.47)

By local non-satiation the budget constraints of all users will bind at any price \(p\). Integrating over all user types and substituting Equation (5.47), we have,

\[
p \cdot \int_{\bar{T}} x(\bar{t}, p) d\mu_{\bar{T}} = p \cdot \int_{\bar{T}} e(\bar{t}) d\mu_{\bar{T}} + \int_{\bar{T}} \theta(\bar{t}) d\mu_{\bar{T}} \int_{\bar{T}} u(\bar{t}, p, g) d\mu_{\bar{T}} 
\]  

(5.48)

\[
= p \cdot \int_{\bar{T}} e(\bar{t}) d\mu_{\bar{T}} + \int_{\bar{T}} u(\bar{t}, p, g) d\mu_{\bar{T}} 
\]  

(5.49)

\[
= p \cdot \int_{\bar{T}} e(\bar{t}) d\mu_{\bar{T}} + p \cdot \int_{\bar{T}} y(\bar{t}, p, g) d\mu_{\bar{T}} 
\]  

(5.50)

By the definition of aggregate excess demand in Equation (5.39),
\[
\begin{align*}
 p \cdot \int_T x(\hat{t}, p) d\mu_T - p \cdot \int_T e(\hat{t}) d\mu_T - p \cdot \int_T y(\hat{t}, p, g) d\mu_T &= 0 \\
 p \cdot (\int_T x(\hat{t}, p) d\mu_T - \int_T e(\hat{t}) d\mu_T - \int_T y(\hat{t}, p, g) d\mu_T) &= 0 \\
p \cdot z(p, g) &= 0
\end{align*}
\] (5.51) (5.52) (5.53)

Next we establish desirable properties of the aggregate excess demand function \(z(\cdot)\) that will be used subsequently to show the existence of CE. In what follows, homogeneity of degree zero implies that if both prices of goods and consumer wealth change in the same proportion then the consumer’s set of feasible consumption bundles does not change. The proof of Property 4 is very similar to the standard treatment in the textbook [55], nonetheless we include it here for completeness.

**Lemma 20.** For any belief \(g\) and price vector \(p\), aggregate excess demand \(z(p, g)\) satisfies the following properties:

1. \(z(\cdot)\) is continuous in \(p\) and \(g\), where \(p \gg 0\) and \(g \in G\).

2. \(z(\cdot)\) is homogenous of degree 0.

3. \(p \cdot z(p, g) = 0, \forall p\).

4. Consider a sequence of price vectors \(\{p_m\}\), where \(p_m \gg 0\), that converges to \(p'\) such that \(p' \neq 0\) and \(p'_k = 0\) for some good \(k\). Let the corresponding sequence of aggregate excess demands be \(\{z(p_m)\}\). Then the aggregate excess demand for at least one such good must be unbounded above, i.e., \(\max\{z_1(p_m), \ldots, z_{n+1}(p_m)\} \to 0\).

**Proof.** Property 1 follows from the continuity of the aggregate demand and aggregate supply in the inputs \(p\) and \(g\). Consider the utility functions of the worker and the user. The set of
feasible production and consumption vectors does not change when all prices and budgets are multiplied by a constant $\alpha > 0$. Therefore demand and supply are homogenous of degree 0 and this implies that $z(\cdot)$ is homogenous of degree 0 (property 2). Property 3 follows from Lemma 19.

Next we prove Property 4. Consider a user $i$ of type $\hat{t}$ with budget constraint $p' \cdot e(\hat{t}) + \theta(\hat{t}) \int_{\hat{T}} u(\hat{t}, p', g) d\mu_{\hat{T}} > 0$. Denote $i$’s demand vector associated with the sequence of prices $\{p_m\}$ by $x_i(\hat{t}, p_m)$. Assume that the sequence of demand vectors $\{x_i(\hat{t}, p_m)\}$ is bounded. Then this bounded sequence must contain a convergent subsequence – let $x^*$ be the demand vector to which the subsequence converges. User $i$ maximizes his utility at prices $p_m$ by demanding bundle $x_i(\hat{t}, p_m)$ subject to his budget constraint. Since user preferences are strongly monotone and the user utility function is strongly increasing, the budget constraint binds for each $m$ as follows,

$$p_m \cdot x_i(\hat{t}, p_m) = p_m \cdot e(\hat{t}) + \theta(\hat{t}) \int_{\hat{T}} u(\hat{t}, p_m, g) d\mu_{\hat{T}}$$

(5.54)

Because in the limit as $m \rightarrow \infty$, $p_m \rightarrow p'$, we can take the limit in the above equation and write,

$$p' \cdot x^* = p' \cdot e(\hat{t}) + \theta(\hat{t}) \int_{\hat{T}} u(\hat{t}, p', g) d\mu_{\hat{T}}$$

(5.55)

Suppose $x^{**} = x^* + (0, \ldots, 0, 1, 0, \ldots, 0)$, where $x^{**}$ contains an increase of 1 in the $k$th position. Then we have user utility as follows,

$$v_i(\hat{t}, x^{**}) > v_i(\hat{t}, x^*)$$

(5.56)

where the strict inequality results from strong monotonicity of preferences and strongly
increasing $v_i(\cdot)$. However because $p_{k'} = 0$ we have,

$$p' \cdot x^{**} = p' \cdot x^* = p' \cdot e(\hat{\imath}) + \theta(\hat{\imath}) \int_{\bar{T}} u(\bar{i}, p', g) d\bar{\mu}_{\bar{T}}$$ (5.57)

Since $v_i(\cdot)$ is a continuous function, this implies that it is possible to effect a small perturbation with $q \in (0, 1)$ such that,

$$v_i(\hat{\imath}, qx^{**}) > v_i(\hat{\imath}, x^*)$$ (5.58)

$$p' \cdot (qx^{**}) < p' \cdot x^*$$ (5.59)

$$< p' \cdot e(\hat{\imath}) + \theta(\hat{\imath}) \int_{\bar{T}} u(\bar{i}, p', g) d\bar{\mu}_{\bar{T}}$$ (5.60)

Further since the sequences $\{p_m\}$ and $\{x_i(\hat{\imath}, p_m)\}$ converge to $p'$ and $x^*$ respectively, this means that there is an $m$ large enough such that,

$$v_i(\hat{\imath}, qx^{**}) > v_i(\hat{\imath}, x_i(\hat{\imath}, p_m))$$ (5.61)

$$p_m \cdot (qx^{**}) < p_m \cdot e(\hat{\imath}) + \theta(\hat{\imath}) \int_{\bar{T}} u(\bar{i}, p_m, g) d\bar{\mu}_{\bar{T}}$$ (5.62)

But the user is supposed to maximize his utility at prices $p_m$ by demanding $x_i(\hat{\imath}, p_m)$ and therefore Equations 5.61 and 5.62 provide a contradiction. It follows that our assumption is false and the sequence of demand vectors $\{x_i(\hat{\imath}, p_m)\}$ must be unbounded. This implies that there must be some good $k'$ whose demand is unbounded in the sequence. However the price sequence $\{p_m\}$ converges to $p'$ and meets the budget constraint (see Equation 5.55) implying that the price of good $k'$ must go to zero in the limit. Finally unbounded demand for good $k'$ implies that aggregate excess demand for good $k'$ is also unbounded.

Because user preferences are strongly monotone in our model, demand can go to infinity.
as prices go to zero. Consequently we restrict the equilibrium price vector to be strictly positive, i.e., \( p^* \gg 0 \). In this case the price-taking equilibrium condition is \( z(p^*, g') = 0 \), which satisfies conditions (i)–(iii) of Definition 18. By Walras’ law, this implies that aggregate excess demand for all goods is zero in equilibrium for \( p^* \gg 0 \).

We are now ready to prove the existence of CE. Because \( z(\cdot) \) is homogenous of degree 0, only relative prices matter and so we can restrict the space of prices searched for an equilibrium to the unit simplex. In addition we restrict the space of prices to lie in the interior of the simplex to ensure that the equilibrium price vector is strictly positive. Suppose \( \epsilon \in (0, 1) \). We normalize prices in the following way:

\[
\triangle_\epsilon = \left\{ p \in \mathbb{R}^{n+1}_{++} : \sum_m p_m = 1 \text{ and } p_m \geq \frac{\epsilon}{1 + 2n} \forall m \right\}
\] (5.63)

In order to prove the existence of CE we will construct a function \( \tilde{f}(p, g) \), such that \( \tilde{f} : \triangle_\epsilon \times G \rightarrow \triangle_\epsilon \times G \), and show that it is continuous with respect to \( p \gg 0 \) and \( g \). In particular \( \tilde{f} \) is composed of two functions, \( \tilde{f} = f_1 \circ f_2 \), such that \( f_1(p, g) \) maps to an updated supply distribution \( g' \) while leaving \( p \) untouched, and \( f_2(p, g') \) adjusts prices and maps to an output \((p', g')\).

**Lemma 21.** For any price vector \( p \) there exists a fixed point \((p, g^*) = f_1(p, g^*)\).

**Proof.** For any \( p \) and \( g \), \( f_1(p, g) \) computes the aggregate supply, \( \int_T y(t, p, g) d\mu_T \), and outputs \( p \) and the distribution \( g' \) obtained by extracting the first \( n + 1 \) coordinates of \( y \) and normalizing. By Lemma 17, the aggregate supply is continuous in \( p \) and \( g \). It follows that \( f_1(p, g) \) is continuous in \( p \) and \( g \). Moreover \( f_1(p, g) \) maps from a convex, compact set to itself. By Brouwer’s theorem, there exists a fixed point \((p, g^*) = f_1(p, g^*)\). \( \square \)

Next, we turn our attention to \( f_2(p, g') \) defined as follows for each good \( k \), for \( p \in \triangle_\epsilon \) and given a fixed \( \epsilon \in (0, 1) \).
\[ f^k_2(p, g') = \left( \frac{h_k(p, g')}{\sum_{m=1}^{n+1} h_m(p, g')} \right) \text{ for } k = 1, \ldots, n + 1 \]  

(5.64)

where \( k = 1, \ldots, n + 1 \) are the goods in the market comprising the nonzero fix depths and the commodity time. The numerator and denominator of Equation 5.64 are comprised of the terms below.

\[ h_k(p, g') = p_k + \epsilon + \max(\hat{z}_k(p, g'), 0) \]  

(5.65)

\[ \sum_{m=1}^{n+1} h_m(p, g') = 1 + n\epsilon + \sum_{m=1}^{n+1} \max(\hat{z}_m(p, g'), 0) \]  

(5.66)

where \( \hat{z}_k(p, g') = \min(z_k(p, g'), 1) \) for a good \( k \). Hence \( \hat{z}_k(p, g') \) is bounded above by 1. We define \( \hat{z}(\cdot) = (\hat{z}_1(\cdot), \ldots, \hat{z}_{n+1}(\cdot)) \) and \( f_2(\cdot) = (f_2^1(\cdot), \ldots, f_2^{n+1}(\cdot)) \). Then we have the property that \( p \cdot \hat{z}(p, g') \leq p \cdot z(p, g') = 0 \ \forall \ p \gg 0 \). As well, \( \sum_{k=1}^{n+1} \frac{h_k(p, g')}{\sum_{m=1}^{n+1} h_m(p, g')} = 1 \). Because \( \epsilon < 1 \) we also have that \( \frac{h_k(p, g')}{\sum_{m=1}^{n+1} h_m(p, g')} \geq \frac{\epsilon}{1+2n} \). Hence for any input \( p \in \Delta_\epsilon \) and \( g' \in G \), \( f_2 : \Delta_\epsilon \times G \rightarrow \Delta_\epsilon \times G \). In what follows, we show that \( f_2(\cdot) \) has a fixed point and this fixed point is a price-taking equilibrium. The proof mimics the standard case described in the textbook [55].

**Lemma 22.** Given belief \( g' \) there exists a fixed point \( (p', g') = f_2(p', g') \) where \( p' \in \Delta_\epsilon \).

**Proof.** Because \( z(\cdot) \) is continuous in \( p \gg 0 \) and \( g' \in G \), \( f_2(\cdot) \) is continuous in \( p \in \Delta_\epsilon \) for any \( \epsilon \), and \( g' \in G \). Further, \( f_2(\cdot) \) maps from a convex, compact set to itself. By Brouwer’s theorem, there exists a fixed point \( (p', g') = f_2(p', g') \). \( \square \)

**Lemma 23.** A fixed point \( (p^*, g') = f_2(p^*, g') \) is a price-taking equilibrium for any given belief \( g' \) and where \( p^* \gg 0 \). Thus \( z(p^*, g') = 0 \).

**Proof.** By Lemma 22 there exists a fixed point \( (p', g') = f_2(p', g') \). This implies that, for \( k = 1, \ldots, n + 1 \),
Substituting the definitions for both sides of the above expression from Equations 5.65 and 5.66, we get, for \( \epsilon \in (0, 1) \) and \( k = 1, \ldots, n + 1 \),

\[
p_k'(1 + n\epsilon + \sum_{m=1}^{n+1} h_m(p', g')) = p_k' + \epsilon + \max(\hat{z}_k(p', g'), 0) 
\]

\[
p_k'(n\epsilon + \sum_{m=1}^{n+1} h_m(p', g')) = \epsilon + \max(\hat{z}_k(p', g'), 0)
\]

Next, consider a sequence of price vectors \( \{p'\} \), where \( p' \in \Delta_\epsilon \), that solve Equation 5.71 as \( \epsilon \to 0 \). Because this sequence is bounded, it must contain a convergent subsequence. Suppose \( p'' \) is a convergent subsequence and suppose it converges to \( p^* \). Let us assume that it is not the case that \( p^* \gg 0 \). Then there must be some \( k \) such that \( p_k^* = 0 \). By Property 4 of Lemma 20 this must mean that there is a good \( k \) such that \( p_k'' \to 0 \) and \( z_k(p'', g') \) is unbounded as \( \epsilon \to 0 \). As a result, since \( p_k'' \to 0 \), any solution to Equation 5.71 must require the LHS term to approach zero as \( \epsilon \to 0 \). However the RHS term of Equation 5.71 is greater than zero because \( \hat{z}_k(p'', g') = 1 \) for unbounded \( z_k(p'', g') \) as \( \epsilon \to 0 \). This is a contradiction since the terms must equal \( \forall \epsilon \in (0, 1) \). Therefore our assumption cannot hold and it must be that \( p^* \gg 0 \).

Because in the limit as \( \epsilon \to 0 \), \( p'' \to p^* \) where \( p^* \gg 0 \), we can take the limit in Equation 5.71 and write, for \( k = 1, \ldots, n + 1 \),
\[ p_k^* \left( \sum_{m=1}^{n+1} h_m(p^*, g') \right) = \max(\hat{z}_k(p^*, g'), 0) \quad (5.72) \]

Multiplying both sides of the above equation by \( \hat{z}_k(p^*, g') \), we have, for \( k = 1, \ldots, n+1 \),

\[ \hat{z}_k(p^*, g') p_k^* \left( \sum_{m=1}^{n+1} h_m(p^*, g') \right) = \hat{z}_k(p^*, g') \max(\hat{z}_k(p^*, g'), 0) \quad (5.73) \]

Summing all \( n+1 \) equations gives,

\[ \sum_{k=1}^{n+1} \hat{z}_k(p^*, g') p_k^* \left( \sum_{m=1}^{n+1} h_m(p^*, g') \right) = \sum_{k=1}^{n+1} \hat{z}_k(p^*, g') \max(\hat{z}_k(p^*, g'), 0) \quad (5.74) \]

By Walras’ law \( \sum_{k=1}^{n+1} p_k^* z_k(p^*, g') = 0 \). As noted earlier, this means that \( \sum_{k=1}^{n+1} \hat{z}_k(p^*, g') p_k^* \leq 0 \). Therefore the LHS term of Equation 5.74 must be less than or equal to zero. By equality so must the RHS term and this requires that \( \hat{z}_k(p^*, g') \leq 0 \). By construction, it must be that \( z_k(p^*, g') \leq 0 \) for \( k = 1, \ldots, n+1 \). But \( p^* \gg 0 \) and Walras’ law requires that \( p^* \cdot z(p^*, g') = 0 \). This implies that \( z(p^*, g') = 0 \) which is the equilibrium condition. Hence \((p^*, g')\) is a price-taking equilibrium.

**Theorem 12.** There exists a fixed point \((p^*, g^*) = \bar{f}(p^*, g^*)\), where \( p^* \gg 0 \) and \( g^* \in G \), and this fixed point is a correctness equilibrium.

**Proof.** \( \bar{f} \) is a composition of \( f_1 \) and \( f_2 \). Because \( f_1 \) and \( f_2 \) are continuous and map a convex, compact set to itself, \( \bar{f} \) likewise has the same properties. By Brouwer’s theorem, there exists a fixed point \((p^*, g^*) = \bar{f}(p^*, g^*)\). Any fixed point of \( \bar{f} \) must by definition also be a fixed point of \( f_1 \) and \( f_2 \) because of the form taken by \( f_1 \) and \( f_2 \) in the construction of \( \bar{f} \). By Lemma 23 this implies a price-taking equilibrium which satisfies conditions (i)–(iii) of
Definition 18. By Lemma 21 condition (iv) of Definition 18 is satisfied too. Therefore the fixed point is a correctness equilibrium. □

5.3.6 The First Fundamental Theorem of Welfare Economics

In this section we establish the first fundamental theorem of welfare economics in the context of CE and show that any allocation at a CE is Pareto efficient. The proof adapts the standard treatment found in texts [55, 74]. To begin we define Pareto efficiency in the public software economy.

**Definition 19.** Given price vector $p$, belief $g$, and current fix depth, $\tilde{d}$, at a root cause, an allocation $(x, y)$ is Pareto efficient if there is no other allocation $(x', y')$ such that $u_j(\tilde{t}, y'_j, \tilde{d}, p, g) \geq u_j(\tilde{t}, y_j, \tilde{d}, p, g)$ and $v_i(\hat{t}, x'_i) \geq v_i(\hat{t}, x_i)$ for all workers $j$ and all users $i$ of all types, and $u_j(\tilde{t}, y'_j, \tilde{d}, p, g) > u_j(\tilde{t}, y_j, \tilde{d}, p, g)$ or $v_i(\hat{t}, x'_i) > v_i(\hat{t}, x_i)$ for some $j$ or $i$.

What Pareto efficiency says is that there is no other way to allocate production and consumption of fix depths such that some agent is better off while no agent is worse off. Note that this does not mean that a Pareto efficient allocation is the socially optimal or social welfare maximizing allocation.

**Theorem 13.** If preferences are locally non-satiated, and if the allocation $(x^*, y^*)$, price vector $p^*$, and distribution of depths $g^*$ constitute a correctness equilibrium, then $(x^*, y^*)$ is Pareto efficient.

**Proof.** Suppose $(x^*, y^*)$ is a CE at prices $p^* \gg 0$, but such that it is not Pareto efficient. By condition (iii) of Definition 18, we have,

$$\int_T x^*(\tilde{t}, p^*)d\mu_T = \int_T e(\tilde{t})d\mu_T + \int_T y^*(\tilde{t}, p^*, g^*)d\mu_T$$  (5.75)
This implies that there must be some other allocation \((x', y')\) such that users are better off, i.e.,

\[
\int_{\hat{T}} v(\hat{t}, p^*, x') d\mu_{\hat{T}} \geq \int_{\hat{T}} v(\hat{t}, p^*, x^*) d\mu_{\hat{T}} \quad (5.76)
\]

and further the allocation \((x', y')\) must provide strictly higher utility to at least one user type. Because \((x^*, y^*)\) was chosen by users even though \((x', y')\) Pareto dominates it, this must mean that at prices \(p^*\), \((x', y')\) is more expensive and thus not affordable to at least one user type, while remaining at least as affordable as \((x^*, y^*)\) to other user types. Accordingly, taking the inner product we can write,

\[
p^* \cdot \int_{\hat{T}} x'(\hat{t}, p^*) d\mu_{\hat{T}} > p^* \cdot \int_{\hat{T}} x^*(\hat{t}, p^*) d\mu_{\hat{T}} \quad (5.77)
\]

Substituting condition (iii) of Definition 18 as written above, and subtracting the endowment from both sides of the equation gives,

\[
p^* \cdot \left( \int_{\hat{T}} e(\hat{t}) d\mu_{\hat{T}} + \int_{\hat{T}} y'(\hat{t}, p^*, g^*) d\mu_{\hat{T}} \right) > p^* \cdot \left( \int_{\hat{T}} e(\hat{t}) d\mu_{\hat{T}} + \int_{\hat{T}} y^*(\hat{t}, p^*, g^*) d\mu_{\hat{T}} \right)
\]

\[
p^* \cdot \int_{\hat{T}} y'(\hat{t}, p^*, g^*) d\mu_{\hat{T}} > p^* \cdot \int_{\hat{T}} y^*(\hat{t}, p^*, g^*) d\mu_{\hat{T}} \quad (5.78)
\]

Consequently it must be that there is a set of non-zero measure of workers of some type \(\bar{t}\), such that for every worker of type \(\bar{t}\) we have,

\[
p^* \cdot y'(\bar{t}, p^*, g^*) > p^* \cdot y^*(\bar{t}, p^*, g^*) \quad (5.80)
\]

But this contradicts the definition of CE which states that at a CE all workers maximize
their utility at prices $p^*$ by choosing production level $y^*$. □

5.4 Relating Equilibrium Concepts

The definition of CE (see Definition 18) incorporates the crucial properties of both an MFE as well as a price-taking or market equilibrium (ME). Theorem 12 demonstrates that a CE exists and simultaneously attains the conditions for an MFE, via Lemma 21, and an ME, via Lemma 23. The relationship among the three equilibrium concepts is depicted in Figure 5.2 and formalized in the following theorem.

**Theorem 14.** A correctness equilibrium satisfies market-clearing conditions as well as consistency of beliefs.

Next we illustrate via an example that an allocation obtained at an MFE need not be Pareto efficient, in contrast to a CE allocation. To this end we map an instance of a CE setting to an instance of an MFE setting, incorporating parameters such as user types and values, that are necessary to reason about social welfare.

**Example 7.** Suppose there are only two types of workers, with type $\bar{\tau} \in [0, 1]$. A worker of high type has $\bar{\tau} = 1$ and a worker of low type has $\bar{\tau} = 0.02$. The likelihood of selecting each type of worker to work in a time period is $\frac{1}{2}$. Users are of type $\hat{\tau}$ drawn at random from
[\[p_{\text{min}}, 1\], where \( p_{\text{min}} = 0.1 \). User types are equivalent to the price that a user pays for a fix, and only one user pays for any particular bug. We set \( p_{\text{min}} \) to a value greater than zero to avoid infinitesimal values where no worker can afford any non-null fix. We define the user’s utility function in the MFE setting as the value obtained from a submitted fix minus payment made towards that fix, where a suitable value function is one where deeper depths provide higher value, for example \( v(\hat{t}, d) = 2d^\hat{t} \). There are a fixed number of identical root causes. The set of depths \( D = \{0, 0.5, 1\} \) and thus a root cause can have at most two bugs. The probability that a root cause does not regenerate in a time period \( T \) is given by \( \beta = 0.8 \). In this example we assume that if a root cause regenerates while a bug remains unfixed, then the user’s payment towards that bug’s fix is returned to the user. The worker’s cost function \( c(\bar{t}, d) = \frac{5}{2}d^6 / 125\bar{t} \).

We will construct a MFE where the strategy adopted by workers is as follows: a low type worker submits \( d = 0.5 \) if possible, otherwise if \( d = 1 \) is the only nonzero option then he submits \( d = 0 \); a high type worker always submits \( d = 1 \). Note that once \( d = 1 \) is submitted at a particular root cause no further bugs can be generated there. Thus a low type worker can only submit \( d = 0.5 \) if assigned to the first bug to appear at a root cause.

The sequence of events at a root cause in a time period in the MFE model is described in Section 5.2.1. Continuing, we depict this problem instance as a Markov chain for any given root cause. Each state represents the different sets of events that are possible at a root cause in a single time period. The different states are as follows.

1. Regenerate: in any time period the root cause may regenerate with probability \( 1 - \beta \).

2. First bug, \( d = 1 \): the first bug on the root cause appears, a high type worker is chosen, and a fix of depth \( d = 1 \) is submitted.

3. First bug, \( d = 0.5 \): the first bug on the root cause appears, a low type worker is chosen, and a fix of depth \( d = 0.5 \) is submitted.
4. Second bug, $d = 0$: the second bug on the root cause appears, a low type worker is chosen, and a null fix, $d = 0$, is submitted.

5. Second bug, $d = 1$: the second bug on the root cause appears, a high type worker is chosen, and a fix of depth $d = 1$ is submitted.

6. No bug: Fix of depth $d = 1$ has been submitted and no further bugs are possible on this root cause. The root cause remains in this state until it regenerates.

Figure 5.3 shows the transition graph, and the transition matrix for states 1, . . . , 6 is given by,

\[
\begin{bmatrix}
1 - \beta & \frac{1}{2}\beta & \frac{1}{2}\beta & 0 & 0 & 0 \\
1 - \beta & 0 & 0 & 0 & 0 & \beta \\
1 - \beta & 0 & 0 & \frac{1}{2}\beta & \frac{1}{2}\beta & 0 \\
1 - \beta & 0 & 0 & \frac{1}{2}\beta & \frac{1}{2}\beta & 0 \\
1 - \beta & 0 & 0 & 0 & 0 & \beta \\
1 - \beta & 0 & 0 & 0 & 0 & \beta
\end{bmatrix}
\]

Given this we can compute the steady-state probabilities for the states as
The conditional probability distribution over the set of states where fixes of depth $d = 0$, $d = 0.5$, and $d = 1$ are submitted is therefore $[0.2, 0.3, 0.5]$. Since this steady-state analysis holds for any root cause and all root causes are alike, the latter distribution holds for the entire system. Thus the best responses described above give rise to the mean field distribution (CDF) on fix depths $g^*$ such that $g^* = [0.2, 0.5, 1]$.

Next we establish consistency by showing that given $g^*$ the workers’ best response is indeed the above strategy. The worker utility function $u_j$, stated below, is adapted from Equation 5.21. Since installments follow the geometric series, note that the maximum amount that can be transferred due to subsumption is $\hat{s}(\bar{d}, p, g) = 0.5$. Moreover transfers can only happen when $d = 1$ is submitted. No transfers can happen when $d = 0.5$ and so in this case $\hat{s}(\bar{d}, p, g) = 0$.

$$u_j(\bar{t}, d, p, g) = r(d, p, g) - c(\bar{t}, d) = \bar{r}(d, p, g) + \hat{s}(d, p, g) = \hat{s}(d, p, g) - \frac{5}{2}d^6$$

Consider the utility of the low type worker when submitting $d = 0.5$, $u_j(0.02, 0.5, p, g^*)$. Substituting input values into Equation 5.83 and rearranging to solve for $p$ such that $u_j(\cdot) \geq 0$, we get that $p \geq 0.025$, implying that $u_j(\cdot) > 0$ for the price range $[p_{\text{min}}, 1]$. Next consider the utility of the low type worker when submitting $d = 1$, $u_j(0.02, 1, p, g^*)$. Substituting input values into Equation 5.83, including $\hat{s}(d, p, g) = 0.5$ which is the maximum value, and rearranging to solve for $p$ such that $u_j(\cdot) \geq 0$, gives the condition that $p \geq 1.2$ which

2Steady-state probabilities are computed via Matlab.

3Note that the worker’s expected payment for submitting $d = 0$ is zero.
is outside the price range \([p_{\min}, 1]\). In other words, for any price \(p \in [p_{\min}, 1]\), the low type worker has \(u_j(0.02, 0.5, p, g^*) > 0\) and \(u_j(0.02, 1, p, g^*) < 0\) even when the maximum transfer amount is used for the case of \(d = 1\). Thus a low type worker has positive utility only when submitting \(d = 0.5\).

On the other hand, for any price \(p \in [p_{\min}, 1]\), the high type worker has \(u_j(1, 1, p, g^*) > u_j(1, 0.5, p, g^*) > 0\) even if the minimum transfer amount \(\hat{s}(\bar{d}, p, g) = 0\) is used for the case of \(d = 1\). Substituting values into Equation 5.83 and rearranging terms, we obtain the condition on \(p\) that is required for the latter inequality to hold. We require \(p > 0.0945\), a threshold that is below the price range \([p_{\min}, 1]\). Thus the high type worker always has greater utility when submitting \(d = 1\). This is consistent with the workers’ best response strategy described above and thus \(g^*\) is an MFE.

Moving on, we define social welfare in the system as the limit of the per-period social welfare, as the number of time periods gets large. Since payments paid by users and earned by workers cancel out, net social welfare is equal to the total value derived by all users minus the total cost incurred by all workers, resulting from the consumption and production of fix depths in the market. Let \(W_l\) and \(W_h\) denote the set of low and high type workers with type given by \(\bar{t}_l\) and \(\bar{t}_h\) respectively. Denote by \(D_t\) the set of fix depths submitted in time period \(t\).

We define the function \(q(\cdot)\) as follows,

\[
q(\bar{t}, d, t, j) = \begin{cases} 
  c(\bar{t}, d) & \text{if } d \text{ is submitted at time } t \text{ by worker } j \text{ of type } \bar{t} \\
  0 & \text{otherwise}
\end{cases} 
\]

Then social welfare in the system is given by,

\[
\lim_{T \to \infty} \sum_{t=1}^{T} \frac{1}{T} \sum_{d \in D_t} \left( \int_{\bar{T}} v(\bar{t}, d) d\mu_{\bar{T}} - \sum_{j \in W_l} q(\bar{t}_l, d, t, j) - \sum_{j' \in W_h} q(\bar{t}_h, d, t, j') \right) 
\]
In the current example social welfare in the system is maximized if only high type workers are assigned every bug at each root cause. However our setting permits several different assignments, each occurring with positive probability. Some cases are: (i) a high type worker is assigned the first bug and no second bug appears, (ii) a low type worker is assigned the first bug followed by a high type worker for the second bug, and (iii) the first two bugs are assigned low type workers. The first two cases result in no further unfixed bugs remaining at the root cause. Consider case (iii). The low type worker will submit $d = 0$ towards the second bug which means that the second bug remains unfixed in the next time period. Unless a high type worker is assigned or until the root cause regenerates, we will get sequences of play consisting of low type workers submitting $d = 0$ every time period. Consider one such sequence where every time period a low type worker is assigned, a high type worker is never assigned, and eventually the root cause regenerates. Clearly there is positive probability that such a sequence can occur. The net social welfare at a root cause where this sequence occurs is,

$$v(\hat{t}, 0.5) - c(0.02, 0.5) + v(\hat{t}', 0) - c(0.02, 0) = v(\hat{t}, 0.5) - c(0.02, 0.5)$$

(5.86)

Suppose instead that a high type worker is assigned the second bug at any point in the sequence. Then the net social welfare at the root cause would be,

$$v(\hat{t}, 0.5) - c(0.02, 0.5) + v(\hat{t}', 1) - c(1, 1) = v(\hat{t}, 0.5) - c(0.02, 0.5) + 2 - 0.032$$

(5.87)

$$> v(\hat{t}, 0.5) - c(0.02, 0.5)$$

(5.88)

Assigning a high type worker for the second bug at any point in the sequence would result in $d = 1$ being submitted, which is a Pareto improvement. Because a low type worker will
submit $d = 0$ towards the second bug, which is equivalent to not doing any work, assigning a high type worker instead would make all workers and users weakly better off.

Next, we consider what may happen in an ME that is not a CE. In other words, conditions (i)–(iii) of Definition 18 are met but condition (iv), the consistency condition, is not enforced. For the purposes of this example we assume user preferences to be locally non-satiated, rather than strongly monotone, and allow equilibrium prices to be zero. As the next example demonstrates this can give rise to a market that clears and yet is Pareto inefficient.

**Example 8.** The set of depths $D = \{0, 0.5, 1\}$ are the only possible depths. User and worker types form a continuum and are sampled at random from the interval $[0, 1]$. The worker cost function $c(\bar{t}, d) = 4d^3/\bar{t}$. The worker and user utility functions are described by Equation 5.21 and Equation 5.27 respectively, where a suitable user utility function $v_1(\hat{t}, d) = d^{\hat{t}}$.

The probability that a root cause does not regenerate in any given time period is given by $\beta = 0.8$. The components of the ME price vector $p^*$ consist of the price of the two nonzero depths, $d = 0.5$ and $d = 1$, as well as the price of the input commodity, time. Moreover each price lies in $[0, 1]$ and the prices in the price vector add up to 1 so that we need consider relative prices only. Let us fix the price of the time commodity to be $p_3 = 0.5$. Consider any set of prices $p_1$ and $p_2$ for depths $d = 0.5$ and $d = 1$ respectively, such that $p_1 + p_2 + p_3 = 1$.

In this instance $d = 1$ is too expensive for any worker type while $d = 0.5$ is affordable for some worker types. Suppose the “un-updated” distribution $g = \{0, 0, 1\}$. Then every worker believes that his fix of depth $d = 0.5$ will be subsumed in the very next time period after submitting it. Hence each worker submitting $d = 0.5$ believes he can only get half the amount of the total reward since payments follow a geometric series. But in that case no worker type can afford either nonzero depth and therefore all workers choose not to work (i.e., all submit $d = 0$). In response prices rise to eliminate excess demand. Because all workers submit $d = 0$ for any price in $[0, 1]$, and given appropriate user budget constraints, the only possible market-clearing equilibrium is one where there is zero demand from users.
and zero supply from workers. Such an outcome provides zero utility to users and workers, and is clearly Pareto dominated by any other outcome where all users and workers have utility \( \geq 0 \) and at least some have utility \( > 0 \).

5.5 Discussion

CE is a type of price-taking equilibrium but embedded in a sequential world. Hence this requires that agents modelled in a CE reason beyond ME and incorporate long-run time-homogenous beliefs about the strategies of agents. Thus a CE also integrates the defining features of an MFE, where workers best respond to a stationary distribution over strategies. The combination of market-clearing prices and consistency of beliefs is apt for a sequential world and enables CE to be Pareto efficient.

An example in the current literature of a market model extended to consider the actions of rival firms is the Cournot oligopoly model [55, 74]. Here firms’ decisions are interdependent and a Cournot-Nash equilibrium is defined as a vector of outputs such that each firm is maximizing profit given the output choices of the other firms. However in contrast to our setting, the equilibrium concept is of simultaneous play taking place in a single time period.

In the public software economy, the concept of CE suggests a new, more tractable definition of correctness than the traditional notion of absolute correctness, which requires that all bugs in a software should be fixed. Instead we propose that a software is *market-correct* if the market for bug fixes for that software has reached correctness equilibrium. In other words all bugs for which users have enough value to pay for and are of low enough cost that workers are willing to work on are fixed. There may still be many latent bugs in the software. However, because there is neither demand for nor supply of fixes to these bugs, such bugs are considered to be irrelevant to the definition of market-correctness.

The public software economy modelled in this chapter is inherently dynamic and is set in
the context of incentivizing deep fixes. Accordingly, CE requires market-clearing conditions to hold, but over multiple time periods that are connected through the subsumption mechanism. In a subsumption mechanism, a worker does not receive the full amount of the price for a fix depth immediately, and must therefore calculate his expected payment contingent on his fix being subsumed by a later, deeper fix. As a result, market-correctness in equilibrium is achieved by workers best responding to both prices as well as to the mean field distribution on fix depths submitted. Thus a market-correct software, in our context, is one where bugs that are fixed have received fixes that are of specific depths with associated prices that satisfy market clearing conditions. Such a definition of correctness, born out of a type of dynamic equilibrium, seems better suited to a model of computation of continuous interaction with other systems and the world.
Chapter 6

Conclusion and Future Work

The central question considered in this thesis is the following:

*How does one design mechanisms with good properties when there are externalities present?*

My dissertation answers this question in two settings, position auctions and software economies, where there are allocative and informational externalities. My work on position auctions extends the GSP auction and shows the existence of Nash equilibrium and envy-free equilibrium for a class of constraints for which standard GSP has no Nash equilibrium. A significant contribution of this thesis is that it proposes a foundation for software economies, tackling problems in the private and public software economy from a multitude of angles.

I conclude by describing directions for future work that stem from my research.

**Position Auctions** On the incentives side, it would be interesting to characterize equilibria for more general classes of unit-bidder constraints, thereby enabling revenue and efficiency comparisons to GSP.

On the algorithmic side, a special case that remains open is that of the $\text{WDP}^\text{pre}_C$ problem with at most one constraint per bidder so that the out-degree of any vertex is at most one. We
colloquially refer to this as the “one enemy” special case. This case is interesting because it could be achieved through a simple restriction to a bidding language.

Another compelling direction is to expand on the possibility of connections with scheduling under precedence constraints, which is equivalent to a version of our problem in which a subset of bidders are selected to be allocated but the order of allocation is to be determined.

**Software Economies** In the private software economy, devising incentive schemes that directly encourage creating reusable components, for example, by rewarding the author of a component when others reuse the component, remains as future work. Further, a mechanism that ranks the value of different components comprising a piece of software would bridge the gap between the private and public software economies.

The scoring system in Chapter 3 was inspired by a preliminary incentives scheme that had been tried at IBM. A revealing direction for future work would be to put theory to practice, by implementing our scoring system in a real-world setting.

Turning to the public software economy, we have provided a treatment of the limited workers scenario. This is an interesting problem that questions the assumptions made by the mean field methodology. A theoretical treatment of this problem that characterizes the limitations of the mean field methodology would be a valuable future direction. Moreover, subsumption mechanisms bear a resemblance to contests – they can be viewed as “sequential contests”. It would be interesting to analyze subsumption mechanisms in the light of contest architectures used by crowdsourcing platforms such as Topcoder.

It would be revealing to explore the possibilities of the computational model by reconfiguring it with different parameter settings. For example, we would like to understand under what environments and metrics lazy subsumption might perform better than eager with reuse. One direction to pursue would be to alter the model so that the amounts redistributed by lazy are of greater weight. In Chapter 4, Figure 4.27, we obtain a crossing point where
the mechanism that produces a higher percentage of deep fixes switches from myopic to eager subsumption. Another line of future work might consider further thresholding behaviour in the computational model. For example, one might ask whether there exists a parameter threshold such that one type of subsumption mechanism performs better when the market is set below the threshold and another type of subsumption mechanism performs for market settings above the threshold.

In this thesis we have focused on the problem of incentivizing deep fixes. However there are other relationships between bugs and fixes that are compelling. For example, what model and mechanism can address the phenomenon of incentivizing good fixes (fixes that do not introduce fresh bugs) instead? The problem of incentivizing good fixes is orthogonal to the problem of incentivizing deep fixes.

To provide a model under which a correctness equilibrium is a mean field equilibrium, and prove this equivalence, would be a valuable result.
Appendix A

Appendix to Chapter 2

A.1 Proof of Lemma 5

*Proof.* Order the bids so that the candidates have decreasing bid value, with the non-
candidates following in some order. We proceed by induction. The inductive hypothesis
is that the outcome (allocation and prices) is the same for the first *i* bidders.

(Base case.) The first bid allocated in eGSP is $b^*_1$ since $b^*_1 \geq b^*_i$ for all other candidates
*i*, and for each non-candidate *j*, there exists at least one candidate $i'$ with $b^*_{i'} \geq r_{i'} > v_j$, and
so $b^*_i \geq b_j = v_j$ for all non-candidates. Given this, then the price in eGSP is $\max(b^*_2, x_1)$
where $x_1 = \max_j \{v_j, 0\}$ over bidders *j* excluded by 1’s allocation. Now, by Lemma 4,
bidder 1 must also be the first across all bidders that share its enemies to be allocated in the
pseudo-run of eGSP, and so $r_1 = x_1$, with the set of excluded agents the same because of
truthful constraints. This establishes the base case, with $p_1 = \max(b^*_2, r_1)$, just as in rGSP.

(Inductive case.) Assume that the i.h. holds for the *i*th candidate, so that the eligible
bidders are candidates from *i* + 1 onwards (which remain eligible because only candidates
were allocated so far, and there are no exclusion constraints between candidates), as well as
remaining, non-excluded non-candidates. The next bid to allocate in eGSP is $b^*_{i+1}$. First, this
bid dominates that of the other candidates, and secondly there continues to exist at least one candidate \( i' \) remaining with \( b_{i'}^* \geq r_{i'} > v_j \) for any non-candidate \( j \), and so \( b_{i+1}^* \) is higher than bid \( b_j = v_j \). Given this, the price in eGSP is \( \max(b_{i+2}^*, x_{i+1}) \) where \( x_{i+1} = \max_j \{ v_j, 0 \} \) over \( j \) excluded by allocating \( i + 1 \). By Lemma 4 and the i.h., the only agents already allocated in eGSP are those that have higher priority, and so the exclusion set is the same as in the pseudo-run of eGSP and we have \( r_{i+1} = x_{i+1} \). This establishes the inductive case, with \( p_{i+1} = \max(b_{i+2}^*, r_{i+1}) \), just as in rGSP.

\[ \square \]

**A.2 Proof of Theorem 3**

*Proof.* For a non-candidate \( i \): bidder \( i \) is not allocated but its enemy, \( j \) is allocated in \( b^* \). To check that \( b^* \) satisfies the requirement of envy-freeness for bidder \( i \), it is sufficient to observe that \( p_j \geq r_j > v_i \), where the second inequality follows from Lemma 3. To check the Nash constraint for bidder \( i \), we need to consider a deviation in which \( i \) is allocated slot \( j \) or above, thus forcing out \( j \). For this, \( i \) needs to bid \( b_i \geq b_j \geq r_j > v_i \), and will want to report its exclusion constraint truthfully (to preclude \( j \) from being allocated). But, we have \( p_i \geq b_j > v_i \), and we see this is not a useful deviation.

For a candidate \( i \): first note that there is no enemy of \( i \) allocated under truthful constraints. So, to check envy-free we consider the slots and prices allocated to all candidates in \( b^* \). That \( i \) has no envy follows immediately from the outcome being envy-free in rGSP, given the definition of EF for rGSP and given the outcome equivalence established in Lemma 5.

Consider now the requirements for NE in regard to a candidate \( i \). A deviation to a higher slot (while still reporting any exclusion constraint truthfully) is not useful because \( i \) would need to pay at least \( b_j \) of a candidate in slot \( j \), and this is not beneficial since \( b_j \geq p_j \) and from envy-freeness. Reporting a false exclusion constraint can only preclude it from an allocation or increase its price and so is not useful. Dropping a true exclusion constraint
does not allow the bidder to avoid payment of at least $b_j$. For a bidder without an enemy and so a zero reserve price it follows from the NE of the bids in rGSP that this deviation is not useful in eGSP. For a bidder with an enemy, then while reporting its constraint truthfully the bidder faces reserve price $r_i$ exactly as in rGSP, and so it follows from the NE of the bids in rGSP that this deviation is not useful in eGSP. Anytime hiding the constraint has an effect on reducing its price then its enemy will be allocated next without stating the constraint, and so this is not a useful strategy. This completes the proof.

\[\square\]

### A.3 Structural Observations

**Lemma 24.** If each vertex in a finite directed graph has indegree at least 1 then there exists a directed cycle.

*Proof.* Suppose each vertex in a directed graph of size $n$ has indegree at least 1. Start at an arbitrary vertex $v_1$. Mark this vertex as “visited.” Pick one of vertex $v_1$’s incoming edges and walk to the adjacent vertex $v_2$. Repeat this procedure until either a marked vertex is encountered (thereby implying a cycle) or the $n$th unmarked vertex $v_n$ is encountered. Since vertex $v_n$ has indegree at least 1, walking along one of $v_n$’s incoming edges will lead us to a previously marked vertex.

**Lemma 25.** Given a constraint graph $G$ and a subset $S$ of bidders, an allocation that allocates every bidder in $S$ is infeasible if and only if there exists a directed cycle in the subgraph induced by $S$ on $G$.

*Proof.* ($\Leftarrow$) Assume that a cycle exists in $G$ and that the vertices $v_{a}, v_{a+b_1}, ..., v_{a+b_h}$ form a directed cycle. Let $Pos[v_a]$ denote the position of $v_a$. To satisfy all constraints in the cycle we require that $Pos[v_a] > Pos[v_{a+b_1}], Pos[v_{a+b_1}] > ... Pos[v_{a+b_n}], and Pos[v_{a+b_n}] > Pos[v_a]$. Clearly it is impossible to satisfy all the constraints in the cycle. Therefore all vertices in $G$ cannot be allocated.
If $G$ is a directed acyclic graph then we output a feasible allocation as follows: pick a vertex $v$ with indegree 0 and place it at the top of the ordering. We know $v$ exists by Lemma 24. Remove $v$ and its incident edges from $G$ and repeat to fill the next $n - 1$ slots in the allocation. The constraints represented by $v$’s outgoing edges are satisfied since $v$ is placed above its adjacent vertices. Note that removing $v$ does not introduce any edges. Therefore if $G$ is acyclic then so is $G \setminus v$. Hence we are always guaranteed to find a vertex with indegree 0 until all vertices have been removed from $G$. \hfill \Box

Consider the constraint graph induced for a fixed set of bidders $S \subseteq N$. For bidders $S$ with an acyclic constraint graph the optimal ordering is defined as the sequence of bidders allocated to slots 1 through $\min \{m, |S|\}$ that maximizes the total discounted bid price. That is, the optimal ordering solves the $\text{WDP}_{\text{pre}}^C$ problem restricted to allocating all bidders in $S$. A contiguous sub-ordering refers to any contiguous subsequence of such an ordering. A sub-ordering is optimal if the ordering of the bidders in the sub-ordering would be maintained if the same restricted $\text{WDP}_{\text{pre}}^C$ problem was solved on just those bidders.

**Lemma 26.** All contiguous sub-orderings of an optimal ordering are optimal. In particular, an optimal ordering cannot have bidder $i$ placed immediately above bidder $j$ where $i$ and $j$ have no constraints between them and $i$’s bid is less than $j$’s bid.

To understand this, notice that any bidders outside of a subsequence are agnostic to a local reordering within a subsequence: if their constraints are satisfied beforehand then they are satisfied for any reordering. For a pair of adjacent bidders in any optimal ordering, a useful swap could be executed because it does not violate any new constraints with other bidders. The only concern is that a constraint between these two bidders themselves is not violated in the rearrangement.

From Lemma 26, it seems that dynamic programming (DP) is a possible technique for the $\text{WDP}_{\text{pre}}^C$ problem. We will highlight the challenge of applying DP on a very simple case,
an increasing path: the constraint graph is a directed path with bids increasing along the path. We have $b_i < b_{i+1}$ and $C_i = \{i+1\}$ for all $i \in \{1, \ldots, n-1\}$, with $C_n = \emptyset$. In light of Lemma 25, we can consider paths as basic building blocks of a potential recursive algorithm for constructing a subset of bidders with an associated directed acyclic constraint graph.

A first approach is to apply DP on the number of slots, by obtaining the optimal solution with $k$ slots from optimal solutions on $k-1$ slots. Such an approach fails, however, as illustrated by the following increasing path instance with 4 bidders and bids 30, 32, 36, 40 respectively. Let $\delta = 0.45$. Then the optimal solution for $k = 2$ slots is to allocate 40 in the top slot followed by 32 (40, 32). This solution has total value of $40 + 32 \cdot 0.45 = 54.4$ whereas allocating 36, 40 yields $36 + 40 \cdot 0.45 = 54$. However the optimal solution for $k = 3$ slots is 36, 40, 30, with a higher value (60.075) than that of any feasible ordering containing bidders 40 and 32 (40, 30, 32; 32, 36, 40; 40, 32). We note that the optimal solution may not allocate all available slots, even if that is feasible: the only feasible allocation of 4 slots has value 55.335, inferior to that using just 3 slots.

An alternate approach is to apply DP on the number of bidders. Recall from our example that 40 is placed in the second slot in the optimal solution for $k = 3$ slots. Two issues that complicate DP appear upon considering the sub-problem (fixing bidder 40 in slot 2) with the other three bidders and slots 1 and 3. First, the 36 bidder cannot be placed in position 3. Second, there is now a gap in discount factors, which are 1 and $\delta^2$ instead of decaying at a constant rate $\delta$. Such issues render us skeptical about the effectiveness of DP on general problem structures.

For an increasing path, however, a successful DP approach exists. It exploits additional structure (via Lemma 26) on the optimal allocation, which must display bidders in decreasing order of value except for contiguous sub-paths (with bidders in increasing order). For example, for $k = 3$, the optimal solution $\{36, 40, 30\}$ has increasing sub-path $\{36, 40\}$, fol-
owed by $30 < 36$. By Lemma 26, sub-ordering $\{40, 30\}$ cannot be improved by a swap.

The DP has one state for each pair of $i \leq n$ and $k' \leq m$, corresponding to the optimal allocation of a subset of bidders $\{1, \ldots, i\}$ on at most $k'$ slots. Sec. 2.8 provides a different DP method for a special case of the slot-specific externalities model, in which bidders can only choose to completely exclude another bidder when allocated.

The next result shows that in sparse constraint graphs, high value bids will win. For example, with no constraints at all, the greedy algorithm is optimal and the top $m$ bids win.

**Lemma 27.** Let $\overline{d} \geq 1$ denote an upper bound\(^1\) on any vertex’s in-degree and out-degree. There is no optimal solution to the WDP\(_C\) problem in which an advertiser whose bid ranks below the $(\overline{d} + 1)m - 1^{th}$ highest ranking bid, for $m$ slots, is assigned a slot.

**Proof.** We prove by contradiction. Let $B$ denote the set of bidders with the $(\overline{d} + 1)m - 1$ highest ranking bids. Assume that in an optimal solution a bid $b(v)$ which is lower than all bid values in $B$ is assigned a slot $A_i$. Then at least $\overline{d}m$ members in $B$ are unallocated. We claim we can use at least one of them to replace $v$ to get a higher-valued allocation without violating the constraints. To see this, note that for each of the unallocated $\overline{d}m$ members $u \in B$, there are two possible reasons why we cannot replace $v$ with $u$: either there are at most $\overline{d}$ vertices $\{v'_1, \ldots, v'_d\}$ having an incoming edge from $u$ such that at least one vertex $v' \in \{v'_1, \ldots, v'_d\}$ is assigned a higher slot $A_{i'}$ for $i' < i$, or there are at most $\overline{d}$ vertices $v'_{1'}, \ldots, v'_{d'}$ having an outgoing edge toward $u$ such that at least one vertex $v' \in \{v'_{1'}, \ldots, v'_{d'}\}$ is assigned a lower slot $A_{i'}$ for $i' > i$.

However, there can be at most $m - 1$ slots occupied by vertices other than $v$. By the Pigeonhole principle, at least one of the $\overline{d}m$ members in $B$ who is unassigned does not have any out-neighbors or in-neighbors in the other $m - 1$ slots (excluding $A_i$). We can safely use this member to replace $v$ without violating any constraints. This contradicts the assumption

\(^1\)The WDP for $\overline{d} = 0$, i.e. no constraints, is straightforward.
that the given allocation is optimal.

Lemma 27 allows us to preprocess input data and discard bidders ranked below the $(\bar{d} + 1)m - 1$ highest bids for any $\bar{d}$ and $m$. If, for a given instance of WDP$_C^{\text{pre}}$, the number of slots $m$ is a constant then, for small $\bar{d}$, an enumerative WDP algorithm examining all feasible allocations becomes practical: its asymptotic runtime is dominated by that of finding the top $(\bar{d} + 1)m - 1$ bids, which is linear in $n$.

A.4 Proof of Theorem 5

We state the greedy algorithm for vertices and edges on a graph, noting that each bidder is associated with a vertex and the edges are such that an edge from $i$ to $j$ indicates that $j \in C_i$. Let $N(v)$ denote the set of in-neighbors of vertex $v$ in the constraint graph $G$. The greedy algorithm proceeds as follows:

**Algorithm**

```
While there is vertex left in $G$
    Choose the remaining vertex $v$ with highest bid
    Assign $v$ to the highest available slot
    Remove $\{v\} \cup N(v)$ from $G$
End
```

**Proof of Theorem 5.** Let $\text{Gre}(v)$ and $\text{Opt}(v)$ be the value collected from vertex $v$ in the greedy algorithm and in the optimal solution respectively. Let $G'$ be the same constraint graph as $G$; we assume that it is “used” by the optimal solution in our discussion. We give an inductive proof showing that in every step of the greedy algorithm, the value of the chosen vertex $v$ is at least $\frac{1-\delta}{1-\delta d+2}$ or $1/(d + 2)$ of the accumulated values of those vertices in $\{v\} \cup N(v)$ in the optimal solution (if they are assigned), and of possibly an additional vertex whose assigned slot position is at most as high as $v$ in the greedy algorithm. In the following, we assume $\delta < 1$; the case $\delta = 1$ follows essentially the same argument.
In the base case, let the chosen vertex in the first step of the greedy algorithm be $v_1^g$. Let $opt$ denote the optimal solution. It is obvious that $b(v_1^g) \geq b(u), \forall u \in V$. In the greedy algorithm, none of the vertices $N(v_1^g)$ can be assigned; however, they along with $v_1^g$, may all be assigned in $opt$. Moreover, it may be the case that none of the vertices $\{v_1^g\} \cup N(v_1^g)$ is assigned the highest slot in $opt$ and that slot is occupied by another vertex $\tilde{v}_1 \not\in \{v_1^g\} \cup N(v_1^g)$. Now remove from $G'$ all the vertices $\{v_1^g\} \cup N(v_1^g)$, and from $G$ all the vertices $\{v_1^g\} \cup N(v_1^g)$ (so after this step, $G \supseteq G'$). We claim that

$$Gre(v_1^g) \geq 1 - \delta \sum_{u \in \{v_1^g, \tilde{v}_1\} \cup N(v_1^g)} Opt(u)$$

(A.1)

For this, assume that in $opt$, $u \in \{v_1^g, \tilde{v}_1\} \cup N(v_1^g)$ wins slot $t$ and among all vertices in $\{v_1^g, \tilde{v}_1\} \cup N(v_1^g)$ that are assigned in $opt$, its position is $i$-th highest (thus $t \geq i$). Then

$$Gre(v_1^g) = b(v_1^g) \geq b(u) = \frac{Opt(u)}{\delta^{t-1}} \geq \frac{Opt(u)}{\delta^{i-1}}$$

(A.2)

Summing the above expression from $i = 1$ to $|\{v_1^g, \tilde{v}_1\} \cup N(v_1^g)|$ gives the RHS expression in equation (A.1). (If not all vertices $|\{v_1^g, \tilde{v}_1\} \cup N(v_1^g)|$ are assigned in $opt$, the RHS expression serves as an upper bound).

For the second step, we observe that the bid of vertex $v_2^g$ in the greedy algorithm must be at least as high as all the remaining vertices in $G'$. This follows from the fact that at this point, $G \supseteq G'$. Moreover, the highest position of a vertex remaining in $G'$ that is assigned in $opt$ can be at most as high as 2. (It can be even lower, since it is possible that the vertex that is assigned the second slot in $opt$ happens to be part of $\{v_1^g\} \cup N(v_1^g)$ and is already removed in our first step). Now we can proceed in the same way as before. We remove $\{v_2^g\} \cup N(v_2^g)$ from $G$ and $\{v_2^g, \tilde{v}_2\} \cup N(v_2^g)$ from $G'$, where $\tilde{v}_2$ is the vertex currently in $G'$ that is assigned the highest position in $opt$. By essentially the same argument, we can show
that

$$\text{Gre}(v^g_2) \geq \frac{1-\delta}{1-\delta d+\varepsilon} \sum_{u \in \{v^g_2, \tilde{v}^g_2\} \cup N(v^g_2)} \text{Opt}(u)$$

(A.3)

Repeating the same argument, since $G \supseteq G'$, $G'$ becomes empty before $G$; moreover opt is fully accounted for during induction.

A.5 Greedy algorithm for soft constraints

We consider a unit-bidder language for externalities in which a bidder can specify

- sets of (soft) constraints of the form

$$C_i = \left\{ (\alpha_i, \beta_j) : \ell \in \{1, \ldots, c_i\} \right\}$$

(A.4)

where $\alpha_i \in \{(i, k_i) : k_i \in 1..m\} \cup \{\text{True}\}$ and $\beta_j \in \{(j, k_j) : j \neq i, k_j \in 1..m\} \cup \{\text{True}\}$, we cannot have both $\alpha_i$ and $\beta_j$ set to True and if neither is True then $i$’s slot is better than $j$’s slot i.e. $k_i > k_j$. Note that different bidders $j_l$ may appear in a bidder $i$’s different constraints.

- two bids $b_i, b'_i$ such that $b_i$ is $i$’s bid given that all its constraints are satisfied and $b'_i$ is $i$’s bid otherwise, i.e. if at least one of its constraints is not satisfied. We call $b_i$ $i$’s base bid. If $i$ has hard constraints only then $b'_i = 0$.

We describe a simple greedy approximation algorithm for the winner determination problem with bids from this language. In this algorithm, a slot $k$ is won by the currently unallocated bidder (if any) $i'$ with highest present bid value for this slot depending on $i'$’s constraints and previous allocations. We denote by $b_i(i_1, \ldots, i_{j-1}) \in \{b_i, b'_i\}$ bidder $i$’s value for slot $j$ given that bidders $i_1, \ldots, i_{j-1}$ are allocated in the better slots $1, \ldots, j-1$. For example, $b_i(\emptyset) = b_i$. 

175
Greedy Algorithm for unit-bidder soft constraints

For slot \( k = 1 \) to \( m \)
\[
i_k = \arg \max_{i \notin \{i_1, \ldots, i_{k-1}\}} b_i'(i_1, \ldots, i_{k-1})
\]
End

Consider a directed graph \( G_c \), with bidders as vertices, in which there is an edge from \( i \) to \( i' \) if \( C_i \) contains at least one constraint of the form \( (i, k_i, i', k_i') \) with \( k_i > k_i' \); thus if \( i' \) is given slot \( k_i' \) then \( i \) cannot be given lower slot \( k_i \).

Let \( d \) denote an upper bound on all vertices’ in-degrees in \( G_c \). The greedy algorithm for the WDP problem achieves an \( \frac{1 - \delta}{1 - \delta d + 1} \) approximation when \( \delta < 1 \). Formally, letting \( W \) and \( A \) be the set of winners and allocated slots in the greedy algorithm, with \( i \in W \) winning slot \( A_i \), and \( b(W, A) \) the social welfare of greedy,

\[
b(W, A) \geq OPT(b, C) \frac{1 - \delta}{1 - \delta d + 1} \tag{A.5}
\]

where \( OPT(b, C) \) is the value of the optimum allocation given bids \( b_1, b_1', \ldots, b_n, b_n' \) and constraint sets \( C_1, \ldots, C_n \).

Proof. We claim that the social welfare \( b(W, A) \) of greedy compares well with \( \bar{V}(b) \), an upper bound on the value of any feasible allocation given bids \( b \). We define \( \bar{V}(b) \) as \( \sum_{j=1}^{m} b_{(j)} \delta^{j-1} \), where \( b_{(1)} \geq \cdots \geq b_{(m)} \) are the \( m \) highest base bids. \( \bar{V}(b) \) is thus the value of the optimum in the absence of any constraints. Clearly,

\[
\bar{V}(b) \geq OPT(b, C)
\]

since \( OPT(b, C) \) has to satisfy all constraints in \( C \). The desired lower bound regarding \( b(W, A) \) follows from

\[
b(W, A) \geq \bar{V}(b) \frac{1 - \delta}{1 - \delta d + 1} \tag{A.6}
\]

that we now establish.
Let $A_h^{-1}$ be the bidder displayed in position $h$ by the greedy algorithm for $h = 1..m$.

We note that the (possibly influenced by bidders displayed above) value of the $j$-th bidder selected by the greedy algorithm is among the $((j - 1)(d + 1) + 1)$ highest base bids, i.e.

$$b_1(A_1^{-1}, \ldots, A_{j-1}^{-1}) \geq b_{((j-1)(d+1)+1)}(\emptyset)$$

Indeed, at least one of the $(j - 1)d + 1$ highest base bids must be eligible for slot $j$ since each buyer appears in the constraints of at most $d$ other buyers. We get that

$$(1 + \delta + \cdots + \delta^d)b(W, A) \geq (1 + \delta + \cdots + \delta^d) \sum_{j \geq 1} b_{((j-1)(d+1)+1)}(\emptyset)\delta^{j-1}$$

$$\geq \sum_{j \geq 1} \sum_{d' = 0}^d b_{((j-1)(d+1)+d'+1)}(\emptyset)\delta^{(j-1)(d+1)+d'} \quad (A.7)$$

$$= \sum_{j' \geq 1} b_{(j')} (\emptyset)\delta^{j'-1} = \mathcal{V}(b)$$

where Eq. (A.7), equivalent to Eq. (A.6), follows from $\delta < 1$ and $j > 1$.

### A.6 The “One Enemy" Special Case

A special case that remains open is that of the $\text{WDP}^\text{pre}_C$ problem with at most one constraint per bidder so that the out-degree of any vertex is at most 1. We colloquially refer to this as the “one enemy" special case. We find this case interesting because it could be achieved through a simple restriction to a bidding language. We have not been able to prove that this is NP-hard or identify a polynomial time algorithm.

In considering this problem, one initial observation is that without the directed externality constraints the problem is just that of the assignment problem. A standard linear programming (LP) formulation adopts $x_{ij}$ to indicate whether bidder $i$ is allocated in position $j$. The feasibility constraints then specify that each bidder (resp. slot) is assigned to at most one slot (resp. bidder). It is well-known that, for general values of each bidder in each
slot but without additional constraints, this problem is totally unimodular and has an integral solution. Using the same encoding, a constraint \( i' \in C_i \) for bidder \( i \) can be specified as a set of linear constraints of the form

\[
x_{ij} + x_{i'j'} \leq 1, \text{ for all } 1 \leq j' < j \leq m
\]

That is, if \( i \) is allocated in slot \( j \), then \( i' \) cannot be allocated in a better slot \( j' \). Let \( LP_2 \) denote the linear program with these constraints. This linear program is no longer totally unimodular and its solutions can be fractional. Consider now an arbitrary LP, \( LP_3 \), with a zero-one constraint matrix \( B \). It is known that such an LP has an integral solution for any linear objective function if \( B \) is the clique-vertex incidence matrix of a perfect graph. Unfortunately, using the hole characterization of perfect graphs, one can show that even a reformulation \( LP_3 \) of \( LP_2 \), strengthened by clique inequalities does not satisfy this condition and can still have fractional solutions.

### A.7 Scheduling with Precedence Constraints

We note here an intriguing connection between the \( WDP_{\text{pre}} \) problem and a classic problem of scheduling non-preemptive jobs on a single machine with precedence constraints to minimize total weighted, discounted completion time.

**Definition 20.** Given a set \( J = \{1, \ldots, n\} \) of \( n \) jobs on a single machine, where each job \( j \) is of length \( p_j \geq 0 \) and has weight \( w_j \geq 0 \), precedence constraints between jobs specified by a directed acyclic graph \( G = (J, R) \) such that \( (i, j) \in R \) implies that job \( i \) must be completed before job \( j \) can be started, and discount factor \( r \in (0, 1) \), the **Discounted Scheduling** problem is to find a schedule that minimizes \( \sum_{j=1}^{n} w_j(1 - e^{-rC_j}) \) where \( C_j \) is the time at which job \( j \) completes. In the scheduling literature this problem is denoted \( 1|\text{Prec}| \sum w_j(1 - e^{-rC_j}) \) [85].
We establish an equivalence between a special case of our WDP$_C^{\text{pre}}$ problem and the D\textsc{ISCOUNTEDSCHEDULING} problem. In particular, we consider a scheduling problem in which jobs have unit processing times, and note that,

\[
\arg\min \sum_j w_j(1 - e^{-rC_j}) = -\arg \max \left(\sum_j w_j e^{-rC_j} - \sum_j w_j\right),
\]

so that the optimal solution to $1|\text{Prec}| \sum w_j(1 - e^{-rC_j})$ also solves the scheduling problem with objective $\arg \max \sum_j w_j(e^{-rC_j})$. Substituting $\delta = e^{-r}$, where $0 < \delta < 1$, this is equivalent to the problem of scheduling jobs to solve $\arg \max \sum_j w_j \delta^{C_j}$, and dividing through by $\delta$ just $\arg \max \sum_j w_j \delta^{C_j-1}$. But with processing time $p_j = 1$, then completion time $C_j$ is equivalently the position of job $j$ in the ordering. For the special problem of WDP$_C^{\text{pre}}$ in which the constraints are acyclic and every bidder must be allocated, we immediately see that this problem is equivalent to D\textsc{ISCOUNTEDSCHEDULING} for $p_j = 1$.

### A.7.1 Tractable Special Cases

Some interesting special cases have been identified for which the D\textsc{ISCOUNTEDSCHEDULING} problem is tractable. One possible motivation to Internet advertising is to a dispatch problem, in which a set of winners has been determined (and all can be feasibly allocated simultaneously) but a dispatcher must determine which slot to which bidder. There is interest, for example, in using offline optimization to guide the allocation by a dispatcher of banner ads to content networks, in meeting campaign targets [84].

The Sidney decomposition algorithm [93] provides a framework by which to solve both the discounted and undiscounted version of the job scheduling problem. The undiscounted version seeks to allocate jobs to minimize the total weighted completion time while respecting precedence constraints. When jobs have unit processing time, this is equivalent to our WDP problem for the case of $\delta = 1$. Not only can the Sidney decomposition be used to iden-
tify polynomial time, optimal algorithms, for special structures on precedence graphs but it also provides the basis for most known 2-approximation algorithms for the undiscounted scheduling problem [6].

Given a problem instance \((J, R)\), a Sidney decomposition partitions \(J\) into subsets \(S_1, S_2, \ldots, S_n\) such that there exists an optimal schedule where jobs from \(S_i\) are processed before jobs from \(S_{i+1}\) for any \(i \in \{1, \ldots, n-1\}\). The Sidney decomposition does not specify any ordering among the jobs within a set \(S_i\). If \(U \subset J\), and \(V \subset J\) then \(U\) has precedence over \(V\) if there exist jobs \(i \in U, j \in V\) such that \((i, j) \in R\). A set \(U\) is said to be initial in \((J, R)\) if there are no jobs in \(J - U\) that must be processed before jobs in \(U\). If \(\alpha\) is any permutation of \(J\) then \(\alpha/U\) is the permutation induced by \(\alpha\) on \(U\). Accordingly \(U\) is initial if and only if there exists a feasible permutation \(\alpha\) of the form \(\alpha = (\alpha/U, \alpha/J - U)\). Let \(\rho\) be a real-valued function whose domain is the set of all subsets of \(J\). \(U \subset J\) is defined to be \(\rho\)-minimal for \((J, R)\) if \(U\) is initial in \((J, R)\), and \(\rho(U) \leq \rho(V)\) for any \(V\) that is initial in \((J, R)\). \(U\) is \(\rho^*\)-minimal for \((J, R)\) if \(U\) is \(\rho\)-minimal for \((J, R)\), and there is no \(V \subset U\) that is \(\rho\)-minimal for \((J, R)\).

For suitable \(\rho\) the following algorithm produces optimal permutations for deterministic problems with either linear or discounted costs.

**Algorithm(\(\rho\))**

\((N, R)\) represents the current network and \(\alpha\) the current permutation in the algorithm. Initialize by setting \((N, R) = (J, R), \alpha = 0\).

1. Find any \(\rho^*\)-minimal subset \(S\) for \((N, R)\).
2. Set \(\beta\) to be any feasible optimal ordering for \((S, R)\).
3. Append \(\beta\) to the end of \(\alpha\).
4. Replace \(N\) by \(N - S\).
5. If \(N\) is not empty then return to Step 1. If \(N\) is empty, stop. \(\alpha\) is optimal.

For the undiscounted version of the problem, the \(\rho\) function is defined as \(\rho(U) = \sum_{i \in U} p_i / \sum_{i \in U} w_i\). The main obstacle to applying this algorithm to arbitrary precedence
structures lies in implementing steps 1 and 2. However Lawler [69] showed that the next subset in a Sidney decomposition (step 1) can be computed in polynomial time. The difficulty in running the algorithm suggested by Sidney is in step 2, which is NP-hard in general [69]. However for certain graph structures (“parallel chain networks, rooted trees, job-modules, and series-parallel networks”) efficient algorithms exist [93]. We illustrate the workings of Algorithm($\rho$) with the example shown in Figure A.1, which is taken from [85]. This is an undiscounted weighted completion time problem. The nodes are labeled as follows: at the top is the number $j$ of the task, on the right is the processing time $p_j$ and on the left is the weight $w_j = 1$. The algorithm produces the optimal permutation, $\alpha$, as follows:

$\begin{align*}
N &= \{1,2,\ldots,7\}, \alpha = \emptyset \\
S &= \{1,3\}, \alpha = \{1,3\}, N = \{2,4,5,6,7\} \\
S &= \{2,4,5\}, \alpha = \{1,3,2,5,4\}, N = \{6,7\} \\
S &= \{6,7\}, \alpha = \{1,3,2,5,4,6,7\}, N = \emptyset
\end{align*}$

For the DISCOUNTEDSCHEDULING problem, a more complicated $\rho$ function is required. Glazebrook and Gittins [37] show how to define the $\rho$ function for the discounted setting but it is not immediately apparent how to compute $\rho$ in general problems. Monma and Sidney [77] show that series-parallel networks \footnote{A series-parallel graph is a graph with a source and a sink that may be constructed by a sequence of series and parallel compositions.} with suitable preprocessing can be solved in $O(n \log n)$ time. Of interest to us are solvable instances where the input graph has the kind of structure that can be produced by a bidding language. Garey [30] proposes an $O(n^2)$ algorithm to solve problems with an acyclic predecessor graph $G$ in which each connected component has the property that either no task in that component has more than one immediate predecessor or no task in that component has more than one immediate successor. In particular a graph may contain some connected components that satisfy the “maximum one immediate predecessor” property while others may satisfy the “maximum one immediate successor” property. An acyclic instance of the WDP$^{pre}_{C}$ problem has this property as long
as a bidder is allowed at most one constraint, and this observation of Garey [30] is relevant for such an instance of our problem, under the additional restriction that all bidders should be allocated.

### A.7.2 Discussion

From the above, we see that there is a rich literature on scheduling under precedence constraints that has strong parallels to the WDP\textsuperscript{pre} problem. In particular, there are special constraint graphs for which there are fast algorithms to allocate every bidder to maximize discounted bid value (and thus, requiring an acyclic constraint graph for these bidders). Based on this, one natural approach for solving the WDP\textsuperscript{pre} problem is to couple this with a search algorithm over sets of winners, for example by establishing a certain local property $P$ of optimal allocations and employing algorithms tailored to $P$. A candidate property $P$ that unfortunately fails is supermodularity of the optimal allocation value, which would couple well with existing efficient algorithms for supermodular maximization (or equivalently submodular minimization[49]). To see this failure of supermodularity, consider the example in Section A.3: the marginal value to the optimal allocation of bidder 32 is less (54.4 − 40)
when added to the 40 bidder than 32, the marginal value when added to the empty set.
Appendix B

Appendix to Chapter 4

B.1 Likelihood Ratio Test

In order to test if the distribution of fix depths has converged after a time, we perform a likelihood-ratio test using the following procedure:

<table>
<thead>
<tr>
<th>Likelihood Ratio Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define a threshold time $\tau$ beyond which the simulation is assumed to be stationary.</td>
</tr>
<tr>
<td>Choose at random two epochs that are not contiguous in time and that occur after time $\tau$.</td>
</tr>
<tr>
<td>Let the two sets of data, $X_1$ and $X_2$, be the set of fix depths submitted in each of the randomly chosen epochs.</td>
</tr>
<tr>
<td>Estimate multinomial parameters for $X_1$, $X_2$ and perform likelihood ratio test.</td>
</tr>
</tbody>
</table>

Note that the choice of $\tau$ can differ depending on the parameters of the simulation, such as the choice of mechanism. We are interested to know whether $X_1$ and $X_2$ are from the same distribution – if so we can confirm that the mean field algorithm has converged. To this end, we would like a null hypothesis that assumes that the distributions of the data $X_1$ and $X_2$ are different and then reject this hypothesis if it is unlikely that the distributions are different according to the test statistic. In what follows, we describe the details of just such a test.
The parameter space corresponding to the null hypothesis is constrained to the multinomial parameter vectors $\theta_1, \theta_2$ such that $\theta_1 \neq \theta_2$, operationalized as $|\theta_1 - \theta_2| > \epsilon$ for some $\epsilon > 0$ (for example $\epsilon = 0.01$). Let $N$ denote the value of the likelihood function under the null hypothesis. In other words, $N$ is the value of the likelihood function evaluated at the maximum likelihood estimator (MLE) on the data $(X_1, X_2)$, given the constraint $\theta_1 \neq \theta_2$.

$$N = \max_{\theta_1, \theta_2} Pr(X_1|\theta_1)Pr(X_2|\theta_2)$$

$$s.t. \ |\theta_1 - \theta_2| > \epsilon$$

Let $D_1$ be the value of the likelihood function evaluated at the MLE on $X_1$ alone. Similarly let $D_2$ be the value of the likelihood function evaluated at the MLE on $X_2$ alone.

$$D_1 = \max_{\theta} Pr(X_1|\theta)$$

$$D_2 = \max_{\theta} Pr(X_2|\theta)$$

where $\theta$ are the parameters of a multinomial distribution. Note that $N < D_1 D_2$ because $N$ is constrained to have $\theta_1 \neq \theta_2$, while the parameters in the denominator are unconstrained.

Continuing, define $\lambda = \frac{N}{D_1 D_2}$. The test statistic $-2 \ln \lambda$ is approximately a chi-squared distribution with 1 degree of freedom. We compute $z = 1 - F(-2 \ln \lambda)$, where $F$ is the cumulative distribution function (CDF) of the above mentioned chi-squared distribution. We reject the null hypothesis if $z < \alpha$ for significance level $\alpha = 0.05$ (the probability of observing a test statistic at least as extreme, given that the null hypothesis is true, is 0.05).

We conclude that the distribution has converged if we reject the null hypothesis.
References


[90] Stuart E. Schechter. How to buy better testing: using competition to get the most security and robustness for your dollar. In In Infrastructure Security Conference, 2002.


