Designing Incentive for Cooperative Problem Solving in Crowdsourcing

Huan Jiang

February 2015

Department of Social Informatics
KYOTO UNIVERSITY

Doctor Thesis of
Ishida and Matsubara Laboratory
Department of Social Informatics
Kyoto University

– –

Abstract

The goal of this thesis is to design incentives for workers in crowdsourcing-based cooperative task solving, with the aim of improving the quality of task solution.

Crowdsourcing is an online, distributed task-solving and web-based business model that has emerged in recent years. It is now admired as one of the most lucrative paradigm of leveraging collective intelligence to carry out a wide variety of tasks with high complexity. This very success is dependent on the potential for replacing a limited number of skilled experts with a multitude of unskilled crowds, by decomposing the complex tasks into smaller pieces of "micro-tasks" (or subtasks), such that each subtask becomes low in complexity, requires little specialized skill, time and cognitive effort to be completed by an individual, and models a step or operation needed in the sequence of producing a final solution to the complex task. However, it is also possible that crowdsourcing-based work will fail to achieve its potential. When facing a geographically distributed workforce who has various knowledge domains and levels of abilities, task requesters always have difficulty to ascertain the quality of the submitted result. Moreover, the strategic behaviors of the rational workers who aim at maximizing their own utilities could have great impact on the quality of the task solution.

Drawing on incentive design perspective, this thesis addresses following issues in crowdsourcing with respect to task-solving model construction, worker's behavior analysis, incentive design and experimental implementation.

1. Designing efficient task decomposition strategy for solving complex

tasks.

In order to facilitate crowdsourcing-based task solving, complex tasks are decomposed into smaller subtasks that can be executed either sequentially or in parallel by workers. These two task decompositions attract a plenty of empirical explorations in crowdsourcing. Our research focuses on the complex tasks that can be decomposed into both a collective of independent and dependent subtasks. Of particular interest are the work that aim at analyzing workers' strategic behaviors, comparing the efficiency of two task decomposition strategies, in terms of final quality, and finally generating explicit instructions on optimal task decomposition. To achieve these goals, we formally present and analyze those two task decompositions as vertical and horizontal task decomposition models, in which the final quality is defined as the sum of the qualities of solutions to all decomposed subtasks. Our focus is on addressing the efficiency (i.e., the quality of the task's solution) of task decomposition when the self-interested workers are paid in two different ways — equally paid (group-based revenue sharing) and paid based on their contributions (contribution-based revenue sharing). By combining the theoretical analyses on worker's strategic behavior and simulation-based exploration on the efficiency of task decomposition, our study 1) shows the superiority of vertical task decomposition over horizontal task decomposition in improving the quality of the task's solution when the final quality is defined as the sum of the qualities of solutions to all decomposed subtasks; and 2) gives the explicit instructions on strategies for optimal vertical task decomposition under both revenue sharing schemes to maximize the quality of the task's solution.

Furthermore, we design and conduct proofreading experiments on Amazon Mechanical Turk. The first series of experiments compare the efficacy of vertical decomposition and horizontal decomposition on proofreading task. The second series of experiments focus on the vertical task decomposition, and compare the situations where the first subtasks with different difficulties. Our experiments ran for two months, and were highly visible on the MTurk crowdsourcing platform. According to the results we collected,

we 1) verify the superiority of the vertical task decomposition strategy over the horizontal one on proofreading task; and 2) apply some of the our proposed instructions on vertically decomposing the proofreading task, which show promise on improving the quality of the final outcome.

2. Limited budget allocation among workers in cooperative task solving.

As discussed in vertical task decomposition, in order to facilitate crowdsourcing-based task solving, complex tasks are recommended to be decomposed into smaller subtasks that are chained sequentially with interdependence. These collective subtasks are required to be executed cooperatively by individual workers. Aiming to maximize the quality of the final solution subject to the self-interested worker's utility maximization, a key challenge is to allocate the limited budget among the sequential subtasks. This is particularly difficult in crowdsourcing marketplaces where the task requester posts subtasks with their payments sequentially and pays the predetermined payment to the worker once the subtask is finished. This study is the first attempt to show the value of Markov Decision Processes (MDPs) for the problem of optimizing the quality of the final solution by dynamically determining the budget allocation on sequentially dependent subtasks under the budget constraints and the uncertainty of the workers' abilities. Our simulation-based approach verifies that compared to some benchmark approaches, the proposed MDP-based payment planning is more efficient at optimizing the final quality under the same limited budget.

3. Quality control on subtask solving.

Even though higher-quality output can be selected or aggregated by harnessing appropriate mechanisms, the quality of the final output can depend heavily on the qualities of the individuals' outputs. Therefore, quality control on subtasks plays an important role in determining the success of the complex task solving. Crowdsourcing competitions have been found to be conducive to acquiring high quality solutions by encouraging redundancy. Such crowdsourcing marketplaces exhibit a similar structure — a microtask is specified, a reward and time period are stated, and during the period

workers compete to provide the best solution. After the competition, workers are then used to verify redundant solutions for quality control, by asking each worker to select and go through some of solutions. Finally, at the conclusion of the period, a subset of selected solutions are accepted as final solutions, and the corresponding workers are granted the reward.

However, the verification procedure presents weakness in terms of the distribution of tasks that have been verified. Because of the various difficulties of the micro-tasks which determine the expected rewards associated with the verification executions, hard micro-tasks on demand endure a risk of long completion time since task selections of workers are congested on easy micro-tasks. Strategic task selection behaviors result in uneven distribution of solved micro-tasks, which can be viewed as low efficiency in crowdsourcing.

The goal of this work is to mitigate the uneven distribution problem to improve the crowdsourcing-based task solving from an efficiency standpoint. We explore crowdsourcing software development process, where the crowdsourcing-based bug detection contest is imported as a solution examination phase. This bug detection contest can be view as a particular mechanism for redundant solution examination, and is readily extended to the cooperative task-solving process we formalized in the first two issues.

To achieve the goal, we first formalize and analyze the bug detection model in which strategic players select code and compete in bug detection contests. Specifically, we model the bug detection contests as all-pay auctions, and rigorously analyze the relationship between strategic code selection behaviors and the offered rewards. Secondly, division strategy is proposed to divide the workers into small divisions, and constrain each worker's code selection behavior exclusively in the division she belongs to. Our study shows that the division strategy can control two features of the bug detection contest, in terms of the expected reward classes and the scales of ability levels, by intentionally assembling workers with particular ability distribution in one division. In this way, division strategy is able to determine the worker' strategic behaviors on code selection, and thus improve the bug detection efficiency. We analyze the division strategy characterized by skill

mixing degree and skill similarity degree and find an explicit correspondence between the division strategy and the bug detection efficiency. Based on our simulation results, we verified that the skill mixing degree, serving as determinant factor of division strategy, controls the trend of the bug detection efficiency, and skill similarity degree plays an important role in indicating the shape of the bug detection efficiency.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, Associate Professor Matsubara Shigeo for the continuous support of my Ph.D study and research. I greatly appreciate his encouragement, his enthusiasm and immense knowledge which always help me out whenever I encounter difficulties in my research. From Associate Professor Matsubara Shigeo, I earned a love and power for scientific research that will last a lifetime. I could not have imagined having a better supervisor and mentor for my Ph.D study.

I own sincere thanks to Professor Toru Ishida for providing me such an excellent opportunity to study to Japan. I am also deeply grateful to my advisory committee members at Kyoto University, Professor Keishi Tajima and Professor Yoshinori Hara, for all their valuable advice and interesting discussions. I would also like to thank the rest of my thesis committee: Professor Hisashi Kashima, for his encouragement, insightful comments, and hard questions. Besides, I would like to express my thanks to Professor Makoto Yokoo and Associate Professor Yuko Sakurai in Kyushu University for providing me the opportunity to visit and giving generous care and attention of my research.

I would also like to show gratitude to the faculty members at Ishida and Matsubara Laboratory: Associate Professor David Kinny, Associate Professor Hiromitsu Hattori, Associate Professor Yohei Murakami, Assistant Professor Yuu Nakashima, Assistant Professor Arisa Ema, Researcher Masayuki Otani, Researcher Takao Nakaguchi. My sincere thanks also goes to Assistant Professor Donghui Lin, who gives me a lot help and support in

my research life in Japan. Besides, I also greatly appreciate our coordinators, Ms. Terumi Kosugi, Ms. Hiroko Yamaguchi, Ms. Yoko Kubota, and Ms. Yoko Iwama, for the help on administrative affairs throughout the academic process.

I thank all the members and alumni in Ishida and Matsubara Laboratory. Special thanks goes to all my fellow PhD students Ari Hautasaari, Julien Bourdon-Miyamoto, Andrew W. Vargo, Chunqi Shi, Mairidan Wushouer, Amit Pariyar, Kemas Muslim Lhaksmana, Xin Zhou, Trang Mai Xuan, Shinsuke Goto, Hiroaki Kingetsu, Xun Cao and Nguyen Cao Hong Ngoc, and all my dearest girlfriends, Linsi Xia, Juan Zhou, Meile Wang, Nan Jin, Ling Xu, Wenya Wu, Meng Zhao, Yating Zhang, Bei Liu and Jie Zhou. Thank you all so much for enriching my life in Japan. We'll be friends forever.

Getting through my dissertation required more than academic support. I would like to thank my parents who have given me the strongest love and support. I would also like to thank my fiancé, Peng Chen, for listening to me, accompanying me, encouraging me and, at times, having to tolerate me, but never force me. I cannot begin to express my gratitude and appreciation for their unconditional love.

Last but not the least, I would like to show gratitude to Professor Zili Zhang and Professor Huiwen Deng in Southwest University, China, for the continuous guidance on my research. Special thanks are also given to Mr. and Mrs. Nomura for treating me as a family member during my stay in Japan.

This research was partially supported by a Grant-in-Aid for Scientific Research (S) (24220002, 2012-2016) from Japan Society for the Promotion of Science (JSPS). My stay in Kyoto University was supported by the Japanese Government (Monbukagakusho) Scholarship.

Contents

Abstract **i**

Acknowledgements **vii**

List of Figures **xiii**

List of Tables **xiv**

1 Introduction **1**
1.1 Objectives 2
1.2 Approaches and Issues 4
1.3 Thesis Outline 6

2 Background **9**
2.1 Characters of Crowdsourcing Applications 10
2.1.1 Crowdsourcing Community 12
2.1.2 Crowdsourcing Platforms 13
2.2 Research Issues in Crowdsourcing 15
2.2.1 Strategic Behavior 15
2.2.2 Reputation Systems 18
2.2.3 Efficiency and Quality Issue 18
2.3 Mechanism Design in Crowdsourcing 19
2.4 Summary 22

3 Efficient Task Decomposition in Crowdsourcing **23**

3.1 Introduction 23
3.2 Related Work 26
3.3 The Model 28
3.3.1 Vertical Task Decomposition 28
3.3.2 Horizontal Task Decomposition 31
3.3.3 Revenue Sharing Schemes 32
3.4 Strategic Behaviors Analysis 33
3.4.1 Independent Behaviors of Individuals 34
3.4.2 Dependent Behaviors in Vertical Task Decomposition 35
3.5 Task Decomposition Strategy Analysis and Comparison . . . 40
3.5.1 Utility Specification 40
3.5.2 Efficiency Comparison 41
3.5.3 Vertical Task Decomposition Strategy 46
3.6 Proofreading Experiments on MTurk 51
3.6.1 Experimental Design: Comparison of Task Decomposition Strategies 53
3.6.2 Experimental Design: First Subtasks with Different Difficulties in Vertical Task Decomposition 55
3.7 Empirical Results 55
3.7.1 Experiment 1 55
3.7.2 Experiment 2 59
3.8 Summary 60
3.8.1 Discussion on Two Revenue Sharing Schemes . . . 61

4 MDP-Based Reward Design for Efficient Cooperative Problem Solving 65
4.1 Introduction 65
4.2 Related Work 68
4.3 The Model and Problem Statement 68
4.3.1 Player-Specific Ability 69
4.3.2 Value of Accomplishing A New Subtask 71
4.4 MDP-based Payment Planning 75
4.4.1 Markov Decision Processes 75

4.4.2 Payment Planning in MDPs 77
4.4.3 Continuous States and Discretization 78
4.5 Planning Algorithm 80
4.5.1 Solving an MDP 80
4.5.2 UCT algorithm 81
4.6 Efficiency Analysis 82
4.6.1 Settings 82
4.6.2 Simulation results 86
4.7 Summary 88

5 A Division Strategy for Efficient Quality Control 91
5.1 Introduction 91
5.2 Related Work 94
5.3 Preliminaries 95
5.3.1 Bug Detection Model 95
5.3.2 Contest Selection 97
5.4 Bug Detection Efficiency Based on Division Strategy 99
5.4.1 Division Strategy 101
5.4.2 Bug Detection Efficiency 103
5.4.3 Example Study: Key Factors in Efficient Division Strategy 106
5.5 Simulation and Analysis 108
5.5.1 Setting 109
5.5.2 Results and Discussion 110
5.6 Summary 116

6 Conclusion 119
6.1 Contributions 119
6.2 Future Directions 121

Bibliography 123

Publications 135

List of Figures

3.1 Positive quality dependence in a two-subtask situation. . . . 31
3.2 Efficiency comparison of vertical and horizontal task decompositions. . . . 41
3.3 Efficiency comparison of two revenue sharing schemes for vertical task decomposition strategy. . . . 45
3.4 Efficiency estimation of vertical task decomposition strategy under group-based revenue sharing scheme. . . . 48
3.5 Efficiency estimation of vertical task decomposition strategy under contribution-based revenue sharing scheme. . . . 50
3.6 Results of proofreading experiment under horizontal task decomposition strategy. . . . 57
3.7 Results of proofreading experiment under vertical task decomposition strategy where errors are identified in phrase-wise. . . . 58
3.8 Results of proofreading experiment under vertical task decomposition strategy where errors are identified in sentence-wise. . . . 59
3.9 Correct rate comparison between phrase-wise and sentence-wise proofreading. . . . 60

4.1 Subtask difficulty and positive quality dependence. . . . 74
4.2 Decision tree representation of MDP-based payment planning with discretized state space for a two-subtask solving process. . . . 79

4.3 Efficiency comparison for average difficulties. 86
4.4 Efficiency comparison for increasing difficulties. 87
4.5 Efficiency comparison for decreasing difficulties. 88

5.1 Group players and contests into different classes 97
5.2 Contest selection behavior in equilibrium. 99
5.3 Relative importance of two key factors of efficient division strategy. 111
5.4 Bug detection efficiency under linear skill distribution. . . . 112
5.5 Bug detection efficiency under concave skill distribution. . . 113
5.6 Bug detection efficiency under convex skill distribution. . . . 114

List of Tables

2.1 Examples of crowdsourcing applications. 10

3.1 Weight combinations for 2-subtask situation. 42
3.2 Weight combinations for 3-subtask situation. 43
3.3 English articles for proofreading task. 52
3.4 Information for experiment 1. 54
3.5 Information for experiment 2. 56

4.1 Summary of parameters in MDP-based budget allocation simulation. 84

5.1 An example of bug detection efficiency increases with the skill mixing degree of division strategy. 105
5.2 An example of bug detection efficiency increases with the skill similarity degree. 107
5.3 Summary of parameters in division strategy simulation. . . . 110

Chapter 1

Introduction

Crowdsourcing is an online, distributed problem-solving and web-based business model that has emerged in recent years. Jeff Howe and Mark Robinson came up with the term "crowdsourcing" in an issue of Wired magazine in 2006 [Howe, 2006]. Generally, tasks or problems that need to be solved are broadcast to an unknown pool of people and an open call is announced for solution collection. The pool of people, known as the workers, select the tasks based on their own preferences, exert their efforts independently or cooperatively and finally submit solutions. Solutions are then verified and the best solutions are selected and possessed by the task requesters who broadcast the task. The winning individuals provided the best solutions usually rewarded, monetarily or non-monetarily. Crowdsourcing now is admired as one of the lucrative paradigms of leveraging the collective intelligence of crowds, including both professionals and amateurs, which outperforms individual experts provided that certain controls on the workers' behavior and the working process [Howe, 2009] [Kittur, 2010]. In addition, crowdsourcing-based task solving has the added advantage of being cheaper to conduct [Kittur et al., 2008].

1.1 Objectives

As a successful business model, crowdsourcing applications are implemented by various organizations. Amazon Mechanical Turk (mturk.com) — one of the most popular crowdsourcing marketplaces, was introduced by Amazon in 2005 [Ipeirotis, 2010]. It allows for easy distribution of various types of tasks to a multitude of unskilled workforce, and shows considerable promise in having the broadcast tasks executed rapidly and successfully. These tasks requiring human intelligence, such as identifying objects in images, finding relevant information, or doing a natural language processing, are typically "micro-tasks", which are characterized by low complexity and require little specialized skill, time and cognitive effort to be completed by independent individuals.

However, these features set the threshold for crowd employers to request complex tasks. In contrast to micro-tasks posted on Amazon Mechanical Turk, much of the work on demand in real-world if often much more complex, consists of interdependent subtasks, and requires various specialized skills, significant time and cognitive effort. Therefore, complex tasks require substantially more cooperative work among co-workers than do the independent and self-contained micro-tasks [Malone and Crowston, 1994].

In order to take the advantage of the micro-task marketplaces for accomplishing complex tasks, there has recently been work on addressing the task-solving workflow design. Kittur et al. [Kittur et al., 2011] made the first contribution to conceptualize a framework for accomplishing complex task and interdependent tasks by using micro-task markets. Kittur et al. defined a general-purpose framework called Crowdforge, which treats *partition* — to break a larger task down into discrete subtasks, *map* — to process a specified task by one or more workers, and *reduce* — to merge the results of multiple workers' tasks into a overall output, as the basic building blocks for distributed workflows, enabling complex tasks to be solved by cooperative workers. Crowdforge delineates the case study on article writing, which consists of three sequential subtasks, as outline construction, information collection and writing. On average, five articles produced by the

above process are of higher qualities than those produced by independent individuals. Related studies expand of this finding. Turkomatic comes up with the innovative idea that ask the crowd to help decomposing the tasks into set of subtasks that can be posted in parallel or must be serially executed [Kulkarni et al., 2011] [Kulkarni et al., 2012]. Soylent contributes the three-stage pattern, Find-Fix-Verify, to text editing services in word processing, with high feasibility and efficiency [Bernstein et al., 2010]. Similarly, PlateMate provides Tag-Identify-Measure for nutritional analysis from photos of meals [Noronha et al., 2011].

Although all these workflows show potential to sum the micro-tasks to significant productive labor, fully leveraging these systems for acquiring high quality task solution remains challenging. First, preparing complex tasks for crowdsourcing marketplaces requires careful attention to the strategy of decomposing tasks into multiple subtasks, since it determines how subtasks dependent on each other and the way multiple workers organized during the task-solving process. However, hindered by the absence of formalized model, which takes into account managing the dependencies among subtasks, explicit analysis on efficient task decomposition is always neglected in previous research.

Secondly, a significant challenge in the above-mentioned crowdsourcing-based complex task solving has been designing an incentive scheme for the task requester to maximize expected output quality subject to a limited budget on the whole complex task, taking into account aspects of the self-interested individual worker's utility maximization. This is particularly difficult when task requesters face workers with unknown and heterogeneous abilities. Given these difficulties, existing approaches typically provide task requesters with simple rules of thumb for budgeting the tasks by looking at the historical data of the crowdsourcing marketplaces. By comparing the nature of the tasks, the historical activity levels of the workers and the qualities of outcomes, task requesters struggle to distribute appropriate rewards among their tasks, to hope for the best possible task solution [Faradani et al., 2011] [Ipeirotis, 2010]. However, without formal models, improper budget allocation commonly leads to task starvation or inefficient

use of capital.

Third, quality assurance on subtask executions can substantially improve the quality of the final task solution. Crowdsourcing competitions have been found to be conducive to acquiring high quality solutions by encouraging redundancy [Archak, 2010] [DiPalantino and Vojnovic, 2009] [Yang et al., 2008a]. Such crowdsourcing marketplaces exhibit a similar structure — a micro-task is specified, a reward and time period are stated, and during the period workers compete to provide the best solution. After the competition, workers are then used to verify redundant solutions for quality control, by asking workers to select and go through some of solutions. Finally, at the conclusion of the period, a subset of selected solutions are accepted as final solutions, and the corresponding workers are granted the reward. However, the verification procedure presents weakness in terms of the distribution of tasks that have been verified. Because of the various difficulties of the micro-tasks which determine the expected rewards associated with the verification executions, hard micro-tasks on demand endure a risk of long completion time since task selections of workers are congested on easy micro-tasks. Strategic task selection behaviors result in uneven distribution of solved micro-tasks, which can be viewed as low efficiency in crowdsourcing.

Therefore, the objective of this thesis is to design workers incentives in crowdsourcing-based cooperative task solving, with the aim of improving the quality of task solution. Of particular interest are the works that aim at task-solving model construction, worker's behavior analysis, incentive design and experimental implementation.

1.2 Approaches and Issues

In order to address issues on incentive design in crowdsourcing-based cooperative task solving, we take the following steps in this thesis.

First we construct the cooperative problem-solving model by defining two task decompositions as horizontal task decomposition for independent

subtasks, and vertical task decomposition for dependent subtasks. We also show that the dependence among subtasks can be formalized as the degree to which a subtask's difficulty depends on the qualities of the other subtasks. Based on the formalized model, we rigorously analyze the strategic behaviors of the workers, with the aim of understanding the relationship between workers' strategic effort exertion and the reward in cooperative task solving. Then simulations are conducted to analyze and compare the efficiency of two task decompositions, aiming at generating explicit instructions on strategies for optimal task decomposition. It is worth noting that we view the workers as self-interested throughout these issues, as it is the fundamental criterion in incentive design. Finally proofreading experiments on Amazon Mechanical Turk are designed and implemented. Empirical results collected in the experiments verify the reasonableness of the model, and show promise of our proposed instructions on vertically decomposing the proofreading task on improving the quality of final output.

Secondly, we address the design of incentive scheme for the task requester to maximize expected output quality, taking into account aspects of sequential quality dependence among subtasks as formalized in the above model. This is particularly difficult in crowdsourcing marketplaces where the task requester posts subtasks with their payments sequentially and pays the predetermined payment to the worker once the subtask is finished. Markov Decision Processes (MDPs) are capable of capturing the task-solving process's uncertainty about the real abilities of a succession of workers, that is, the uncertainty about the qualities of the sequential subtasks. To the best of our knowledge, our work in this thesis is the first attempt to apply MDPs to payment allocation for sequentially dependent subtasks in crowdsourcing. By modeling this problem as MDPs, we show the value of MDPs for optimizing the quality of the final solution by dynamically determining the budget allocation on sequential and dependent subtasks under the budget constrains. Finally, a simulation-based approach verifies that comparing to some benchmark approaches, our MDP-based payment planning is more efficient on optimizing the final quality under the same budget constraints.

The third issue we address in this thesis is to mitigate the uneven distribution problem on solution examination to manage the quality of the subtask's solution. To achieve this goal, we engage in the challenge of balancing the freedom and restriction on solution selection. It is noteworthy that crowdsourcing provides the policy of maximum freedom of task selection, which encourages crowds to solve the tasks based on their own preferences and thus becomes indispensable for the crowdsourcing's prosperity. Because of that, the traditional assignment problem [Kuhn, 1955], which has been well studied to solve the problem of assigning a given set of workers with varying preferences to a given set of jobs in such a way that, maximum efficiency can be achieved, is not appropriate to be used in crowdsourcing situation. By contrast, division strategy is designed and considered to be highly effective. The basic idea of division strategy is to control the workers' skill distribution in each division by strategically dividing the workers with different skills into divisions, and restrict solution examination in the division, i.e., the workers can only select from the solutions produced by the workers in the same division. By doing this, we are able to guide the workers to select from the codes with more appropriate levels of qualities that correspond to their skill levels. Therefore, the division strategy can be reliable to work out a more uniform distribution of examined solutions and thus an improvement in subtasks' qualities.

1.3 Thesis Outline

This thesis consists of six chapters, including this Introduction Chapter.

Chapter 2 aims to provide a brief glimpse of the literature review on the topic of mechanism design in crowdsourcing. By exploring the related literature, we find out both the essential characters of practical crowdsourcing systems and the crucial research issues in crowdsourcing mechanism design. Furthermore, we classify some dominating mechanism design approaches according to those diverse issues. Since there are a lot of crucial research topics in mechanism design in crowdsourcing, we mainly focus on

the issue of incentive design for solution quality assurance and efficiency improvement.

In order to facilitate crowdsourcing-based task solving, complex tasks are decomposed into smaller subtasks that can be executed either sequentially or in parallel by workers. These two task decompositions attract plenty of empirical explorations in crowdsourcing. However the absence of a formal study makes it difficult to provide task requesters with explicit guidelines on task decomposition. In Chapter 3, we formally present and analyze those two task decompositions as vertical and horizontal task decomposition models. Our focus is on addressing the efficiency (i.e., the quality of the task's solution) of task decomposition when the self-interested workers are paid in two different ways — equally paid and paid based on their contributions. By combining the theoretical analyses on worker's behavior and simulation-based exploration on the efficiency of task decomposition, our study 1) shows the superiority of vertical task decomposition over horizontal task decomposition in improving the quality of the task's solution; and 2) gives explicit instructions on strategies for optimal vertical task decomposition under both revenue sharing schemes to maximize the quality of the task's solution. Furthermore proofreading experiments on Amazon Mechanical Turk are designed and implemented. Empirical results collected in the experiments verify the reasonableness of the model, and show promise of our proposed instructions on vertically decomposing the proofreading task on improving the quality of final output.

Aiming to maximize the quality of the final solution subject to the self-interested worker's utility maximization, a key challenge is to allocate the limited budget among the sequential subtasks. This is particularly difficult in crowdsourcing marketplaces where the task requester posts subtasks with their payments sequentially and pays the predetermined payment to the worker once the subtask is finished. Our study in Chapter 4 is the first attempt to show the value of Markov Decision Processes for the problem of optimizing the quality of the final solution by dynamically determining the budget allocation on sequentially dependent subtasks under the budget constraints and the uncertainty of the workers' abilities. Our simulation-

based approach verifies that compared to some benchmark approaches, the proposed MDP-based payment planning is more efficient on optimizing the final quality under the same limited budget.

In Chapter 5 we construct and analyze a crowdsourcing-based bug detection model in which strategic players select code and compete in bug detection contests. We model the contests as all-pay auctions, and our focus is on addressing the low efficiency problem in bug detection by division strategy. Our study shows that the division strategy can control two features of the bug detection contest, in terms of the expected reward classes and the scales of skill levels, by intentionally assembling players with particular skill distribution in one division. In this way, division strategy is able to determine the players' strategic behaviors on code selection, and thus improve the bug detection efficiency. We analyze the division strategy characterized by skill mixing degree and skill similarity degree and find an explicit correspondence between the division strategy and the bug detection efficiency. Based on our simulation results, we verified that the skill mixing degree, serving as determinant factor of division strategy, controls the trend of the bug detection efficiency, and skill similarity degree plays an important role in indicating the shape of the bug detection efficiency.

Finally, Chapter 6 concludes the thesis by summarizing the results of this research. We also discuss possible future work in this section.

Chapter 2

Background

Crowdsourcing is getting thousands of talented people to work for large or small business-related tasks that a company would normally either perform itself or outsource to a professional third-party provider. Crowdsourcing now is admired as one of the best way of leveraging the collective intelligence of crowds, where groups of people, including both professional and amateur, outperform individual experts. Although for task providers, crowdsourcing promises significant cost-savings, quicker task completion times, and formation of expert communities, many aspects of crowdsourcing decentralized management are under vigorous refinement, in order to motivate the customers to exert their best effort independently or cooperatively.

Mechanism design, which emerged from economic game theory in the 1970s, is now shaking hands with information technology, and come to crowdsourcing management's rescue. Indeed, the basic idea of mechanism design and its theoretical conclusions built on a formal mathematical base are enabling advances in crowdsourcing management technology. In this chapter, we review the current research on incentive design for crowdsourcing, trying to 1) find out both the essential characters of practical crowdsourcing systems and the crucial research issues in crowdsourcing incentive design, and 2) classify the dominating mechanism design approaches according to those diverse issues, and provide feasible or potential (relatively new) approaches to mechanism design on crowdsourcing. This survey paper

mainly focus on the issue of incentive design in crowdsourcing.

This Chapter is arranged as follows. Section 2.2 goes through current research literature on real crowdsourcing platforms and applications, aiming to classify these platforms based on their uses and characterize their main features. Research issues in crowdsourcing incentive design are discussed in Section 2.3. In Section 2.4, we classify some dominating incentive design approaches according to those diverse research issues, especially on the incentive design for solution quality assurance and efficiency improvement. Summary is given in Section 2.5.

2.1 Characters of Crowdsourcing Applications

As an emerging and successful business model [Howe, 2009], crowdsourcing applications are implemented by various organizations in the hopes that the participation of an online open call results in the design of products or solving of problems. Such crowdsourcing platforms mushrooms in the past few years (see Table 2.1). The accompanying table shows a set basic CS system types. The set is not meant to be exhaustive; it shows only those platforms that have received most attention.

Table 2.1: Examples of crowdsourcing applications.

Organizer (URL)	**Characteristic**
TopCoder (topcoder.com)	As on InnoCentive, companies can post software development problems on TopCoder.com and crowd compete to provide the best solution. Additionally, TopCoder.com site offers contests itself to help finding the best programmer.
Amazon Mechanical Turk	

Continued on next page

Table 2.1 – continued from previous page

Organizer (URL)	Characteristic
(mturk.com)	Amazon Mechanical Turk is an online labor market where employees are recruited by employers for the execution of tasks in exchange for a reward.
Taskcn (taskcn.com)	Taskcn is one of the biggest Witkey websites in China where users offer monetary awards for solutions to problems. Other users provide solutions in the hopes of winning the awards. Taskcn.com has more than 1.7 registered users.
Stack Overflow (stackoverflow.com)	A language-independent collaboratively edited question and answer site for programmers. It encourages professional and enthusiast programmers to ask practical, answerable questions based on actual problems that they face.
Yahoo! Answers (answers.yahoo.com)	Yahoo! Answers is an online question and answer forum, allows users to ask questions about a variety of topics. Other users compete to provide the best answer to the question posted by the other user. Winner is decided by the questioner.
Threadless (threadless.com)	Threadless.com is a web-based t-shirt company that crowdsources the design process for their shirts through an ongoing online crowd contests. The winner is decided upon the votes given by the same crowd.

Continued on next page

Table 2.1 – continued from previous page

Organizer (URL)	Characteristic
Wikipedia (wikipedia.org)	Wikipedia is an online encyclopedia established in 2001. Mainly because its content is publicly editable, it becomes the most prolific collaborative authoring projects ever sustained in an online environment.

2.1.1 Crowdsourcing Community

Some research tend to clearly differentiate crowdsourcing model from virtual community model, as declared by Haythornthwaite [Haythornthwaite, 2009] that a crowdsourcing model is based on micro-participation from individuals who are unconnected. By contrast, virtual community model is based on strong connections among a committed set of connected members. We respectfully disagree with the above opinion. Although crowdsourcing starts with decentralization, by sourcing tasks traditionally performed by specific individuals to a group of people within a company through an open call, which is different from sites such as Twitter or Facebook, which, at the beginning, do not have open call for contributions, people (crowds) engaged in a crowdsourcing open call task are "connected" in the following ways [Zhang et al., 2007].

– **Similar domain knowledge.** Crowds working on the same task, competitively or cooperatively, possess the similar domain knowledge which provides the foundation for them to communicate. Moreover, some crowdsourcing websites tend to provide support and information to people in certain fields. For instance, Yahoo! Answers and Stack Overflow are both Q&A platforms, as mentioned in Table 1, however, the latter is specialized on programming questions.

– **Communication.** The Web is an instant communications platform, where messages, and thus idea exchange is convenient. Most crowdsourcing websites that display content now also allow user comments and discussions, and attract a huge volume of user posts. For instance, TopCoder display public forum for program developers for every competition category, and it also allow face to face meetings.

– **Public opinion.** Since the crowdsourcing platforms involve significant interactions between users such as forum discussions and face to face meetings, public opinion toward a person or a group of people is formed gradually.

2.1.2 Crowdsourcing Platforms

Collective intelligence [Wolpert and Tumer, 1999] refers to a large distributed collection of interacting computational processes among which there is little to no centralized communication or control, involving groups of individuals collaborating or competing to create synergy, something greater than each individual part [Surowiecki, 2004]. Crowdsourcing is a special case of such Collective Intelligence. It leverages the wisdom of crowds and is already changing the way groups of people produce knowledge, generate ideas and make them actionable [Surowiecki, 2005].

General-purpose.

General-purpose crowdsourcing platform provide a market in which anyone can post tasks to be completed and specify prices paid for completing them. The inspiration of the system was to have human users complete simple tasks that would otherwise be extremely difficult (if not impossible) for computer to perform. Examples of such problems in practice include object recognition, language understanding, text summarization, ranking, and labeling [Von Ahn and Dabbish, 2004] [Fuxman et al., 2008].

– **Amazon's Mechanical Turk.** The Amazon Mechanical Turk system[10] manages task submission, assignment, and completion, matches qualified people with tasks that require particular skills, provides a feedback mechanism to encourage quality work, and stores task details and results, all behind a Web services interface. The first paper investigating Mechanism Turk as a user study platform has amassed over one hundred citations in the past years [Kittur et al., 2008].

– **Witkey Website** is a new type of knowledge market website, in which a user offers a monetary award for a question or task and other users compete for the award. Taskcn.com [Yang et al., 2008b] is one the biggest Witkey community in China.

Knowledge sharing.

Crowdsourcing not only let users capture and validate data, but also share sheer amount of "data" such as products, services, textual knowledge, and structured knowledge [Doan et al., 2011].

– Systems that share products and services include Napster, YouTube, and CPAN.

– Systems that share textual knowledge include mailing list, Twitter, how-to repositories (such as ehow.com, which lets users contribute and search how to articles), Q&A Web sites[Nam et al., 2009] [Harper et al., 2009] (such as Stack Overflow, Yahho!Answers [Adamic et al., 2008], Yelp [Luca, 2011], Google Answers, which is no longer accepting questions), online customer support systems (such as QUIQ, which powered Ask Jeeves' AnswerPoint, a Yahoo! Answers-like site).

– Systems that share structured knowledge (for example, relational, XML, RDF data) include Swivel, Many Eyes, Google Fusion Ta-

bles, Google Base, many e-science Web sites (such as bmrb.wisc.edu, galaxyzoo.org).

Creative co-creation.

Crowdsourcing is using the innovative potential of customers to drive innovation. For example, the demands, wishes, and requirements of customers are often used systematically for new product development. In crowdsourcing, customers are treated not only as the recipients of products, but also as a source of innovation. Crowdsourcing platform examples include:

- Spreadshirt - shirt community
- JuJups - personalized gifts
- Threadless - create and sell your t-shirts
- Naked&Angry - threadless for ties and wall coverings
- Cafepress - shop, create or sell what's on your mind
- Zazzle - create and sell products
- CreateMyTattoo - crowdsourced tattoo design
- Sellaband - crowdfunded bands
- Artistshare - fans funding new artists
- Quirky - community product development
- Jovoto - co-creation and mass collaboration
- Dream Heels - design your dream heels.

2.2 Research Issues in Crowdsourcing

2.2.1 Strategic Behavior

Task selection

It is noteworthy that crowdsourcing provides the policy of maximum freedom of task selection for the problem answerer. According to the empirical study on data analysis, main conclusions are obtained as follows:

– Yang et al. [Yang et al., 2008b] explored the relationship between task properties and user's participation on Taskcn.com. Task property variables that influence participation include task category, skill requirement, workload and reward. Details can be found in the Yang's work.

– Statistics show that the tasks with relatively low reward have high probability to be chosen and be solved. The problem poster, or seekers in InnoCentive parlance, pay solvers anywhere from $10,000 to 100,000 per solution. According to Lakhani et al. [Lakhani et al., 2007], 30% of all problems on InnoCentive have been solved. Dean [Dean and Image, 2008] shows that the rewards for those solved problems are typically in the $10,000 to $25,000 range.

– Empirical study on Topcoder.com [Archak, 2010] shows that in order to deter entry of their opponents in the contest they like, higher rated contestants would register early in the contest, while the lower rated contestants will prefer to wait until the higher rated opponents mad their choice.

– Users are more and more likely to choose tasks with fewer competitors [Yang et al., 2008b]. Intentionally choosing less popular tasks to participate could potentially enhance winning probabilities, even if one's own expertise remains the same.

It implies that users with high abilities tend to compete with users with low abilities in low skill required tasks. It brings some serious problems, for example,

1. users with low abilities have small winning probability. These users with low abilities who are interested in participating might be hindered by the very likely futility of their efforts, since winning plays a crucial role in continuous contribution [Yang et al., 2008b].

2. The posted problems that require high skill, even provide high reward,

are unlikely to be solved.

Timing of submission

The timing of users' submissions is an important participation dynamic, since users can choose to submit early, or wait to see how many other submissions a task receives. Empirical analysis on the solution submission data from Taskcn.com shows [Yang et al., 2008b]:

- For all users, higher task award correlates with users submitting solutions later.
- Users, both who have won at least once (winners) and non-winners alike, are likely to submit slightly later as they participate over time.
- The number of the submissions is negatively correlated with the time when people submit.

Uneven participation and outcome

By investigating the users' structural prestige [Barabási and Albert, 1999], it demonstrates an evident scale-free nature which fact indicates the uneven interactions among the users. The majority of users participates in a few tasks and wins fewer, while there is an extremely small group of users who have had many submissions or have successfully won money. Specifically, by examining the behavior of users of the web-knowledge sharing market Taskcn.com, Yang et al. [Yang et al., 2008b] [Yang et al., 2008a] find significant variation in the expertise and productivity of the participating users: a very small core of successful users contributes nearly 20% of the winning solutions on the site.

2.2.2 Reputation Systems

Crowds are engaged and the wisdom of crowds is being used in a number of way, e.g., by providing their members with access to knowledge of interest in return for their contribution, commonly in the monetary form, or by enabling them to build their own digital reputation. Jøsang, Ismail and Boyd [Jøsang et al., 2007] give an overview of existing and proposed systems that can be used to derive measures of trust and reputation for Internet transactions, to analyze the current trends and developments in this area, and to propose a research agenda for trust and reputation systems.

Typical proposals in reputation system design research are:

- Providing some measure of protection for market participants against dishonest traders, since in the filed of multi-agent systems, the success of an agent may depend on its reliable partners. A particular focus has been on the electronic marketplace, such as eBay [Resnick et al., 2006] [Houser and Wooders, 2006].
- Providing an incentive for good behavior to achieve a positive effect on market quality [Zhang and Van der Schaar, 2012].
- User reputation is used as a heuristic to identify and promote high quality contributions [Tausczik and Pennebaker, 2011] [Sylvan, 2010] [Harper et al., 2008].

2.2.3 Efficiency and Quality Issue

Nielsen [Nielsen, 2006] analyzed the phenomenon of "participation inequality" in online communities. Nielsen advocates a general 90-9-1 rule: In most online communities, 90% of users are lurkers who never contribute, 9% of users contribute a little, and 1% of users account for almost all the action. Based on this, participation inequality is deemed to be a function of human behavior and on that is almost impossible to overcome.

Stewart et al. [Stewart et al., 2010] argue for a SCOUT (Super contribu-

tor, Contributor, OUTlier) framework wherein 33% are the (OUT)liers who do very little, 66% are the (C)ontributors who provide moderate contributions, and 1% are the (S)uper contributors who offer super effort.

2.3 Mechanism Design in Crowdsourcing

The development of mechanism design theory began with the work of Leonid Hurwicz [Hurwicz, 1960]. He defined a mechanism as a communication system in which participants send messages to each other and / or to a "message center", and where a pre-specified rule assigns an outcome (allocation of resources), for every collection of received messages. Until now, lots of the interest focused on two distinguishing strands: (i) the informational and computational costs of this decentralized resource allocation mechanism; and (ii) how can the desired outcome be achievable when individuals act according to their own best interests. That is how to deal with the goal conflicts problem due to the multiplicity of individuals. Hurwicz introduced the key notion of *incentive-compatibility* [Hurwicz, 1972], which enables the analysis to incorporate the incentive of the agents which are self-interested and have private information.

Mechanism design based on the game theory is applied to tackle with the issues we discussed in section 2.2 [Jain and Parkes, 2009]. In this survey, we will mainly focus on the incentive design with the purpose of improving the solution quality and efficiency in crowdsourcing.

In terms of designing incentives to influence an agent's behavior when the agent's preferences are unknown, this work is related to work by Zhang et al. [Zhang et al., 2009] on environment design and policy teaching. Environment design considers the problem of perturbing agent decision problems in order to influence their behavior. Policy teaching considers the particular problem of trying to influence the policy of an agent following a Markov Decision Process by assigning rewards to states.

Traditional economic theory has generally espoused the view that rational workers will choose to improve their performance in response to a

scheme that rewards such improvements with financial gain. Empirical research shows evidence that monetary reward does encourage people's participation [Yang et al., 2008b] [Mason and Watts, 2010] or motivate people to speak their real preference [Jurca and Faltings, 2007]. DiPalantino and Vojnovic [DiPalantino and Vojnovic, 2009] suggest that it is important that monetary award mechanism should be designed to induce the appropriate levels of participation and quality of submissions. They model contests as all-pay auctions and demonstrate the precise relationship between incentives and participation, including task selection in crowdsourcing contest.

Contests where multiple prizes are awarded are ubiquitous and become an effective way to motivate all users to maximize their effort. The research on understanding the incentive effects of multiple prizes on effort investment attracts great attention [Clark and Riis, 1998a] [Clark and Riis, 1998b] [Moldovanu and Sela, 2001] [Cason et al., 2010]. Very general conclusion is that a first prize always results in a positive incentive to invest effort, second and later prizes lead to ambiguous effects.

Crowdsourcing platforms such as Wikipedia harness collaborative effort to accomplish a comparatively complicated task. Cooperative game theory and the Shapley value are used to design revenue sharing schemes to incentivize users [Abbassi and Misra, 2011].

Although some researches reveal the positive relationship between financial incentives and performance in crowdsourcing systems, others gain different opinions [Hars and Ou, 2002], which indicates that the increased financial incentives increase the quantity, but not the quality, of work performed by participants [Mason and Watts, 2010]. The similar conclusions obtained by the research on Google Answers. Chen et al. [Chen et al., 2010] [Jain et al., 2009] find that posting a higher prize leads to a significantly longer answer, but not better.

Instead of relying on monetary reward, some incentive design turned to non-monetary rewards that take the form of reputation points and confer a measure of social status within the communities. Zhang and Schaar [Zhang and Van der Schaar, 2012] provide incentives for workers to exert efforts using a novel game-theoretic model based on reputation system. Yan

and Roy [Yan and Van Roy, 2008] develop a rational expectation equilibrium model to study how incentives created by reputation markets influence community behavior.

For the crowdsourcing contest situation, another effective way for incentive design is to control the contest architecture [Chawla et al., 2012] [Cavallo and Jain, 2012]. Contest architecture specifies how the contestants are split among several sub-contests whose winners compete against each other (while other players are eliminated). Moldovanu and Sela [Moldovanu and Sela, 2006] compare the performance of such dynamic schemes to that of static winner-take-all contests from the point of view of a designer who maximizes either the expected total efforts or the expected highest effort. There are sheer amount of literatures of the optimal (effort-maximizing) structure of multi-stage contest [Moldovanu and Sela, 2006] [Kaplan and Sela, 2010] [Fu and Lu, 2012] [Fu and Lu, 2009].

The traditional job assignment problem has been well studied to solve the task of assigning a given set of workers with varying abilities to a given set of jobs in such a way that the overall working time can be kept to a minimum. The assignment literature, surveyed by Stattinger [Sattinger, 1993], distinguishes between model where assignment is based on worker's preferences over job characteristics, comparative advantage and production complementarities across job. All these three types of assignment model have one fundamental assumption in common, that is workers are endowed with complete information on their abilities, or there are activities (schooling is an example) which generate information on the worker's ability. Based on the ability information, workers of higher ability are optimally assigned to jobs with more capital, unilaterally by firms. However, that is not the case in the crowdsourcing situation. First of all, as mentioned before, the crowds are "undefined". It is hard to distinguish professionals and amateurs since the crowds can register with very basic personal information. Secondly, firms cannot designate one worker, whom they thought competent to do a specific job, because crowdsourcing provides the policy of maximum freedom of task selection for the crowds. Otherwise, the incentives for the crowds to participate could be dramatically hurt.

2.4 Summary

Some researchers point out the pitfalls in mechanism design in crowdsourcing. Robert Kleinberg, a computer science professor at Cornell University, says, mechanisms may be so complicated that users don't understand them and hence won't participate. It's not sufficient to tell them, "don't worry, I have proved a theorem that the best thing for you to do is such-and-such". This indicates that the mechanism designed for crowdsourcing system should be simple and explainable as much as possible, which is a tough problem.

This survey provides a brief glimpse of the literature review on the topic of mechanism design in crowdsourcing. By exploring the related literature, we find out both the essential characters of practical crowdsourcing systems and the crucial research issues in crowdsourcing mechanism design. Furthermore, we classify some dominating mechanism design approaches according to those diverse issues. Since there are a lot of crucial research topics in mechanism design in crowdsourcing, we mainly focus on the issue of incentive design for solution quality assurance and efficiency improvement.

Chapter 3

Efficient Task Decomposition in Crowdsourcing

3.1 Introduction

Crowdsourcing system is an online, distributed problem-solving and web-based business model. It is now admired as one of the most lucrative paradigm of leveraging collective intelligence to carry out a wide variety of tasks with various complexity. This very success is dependent on the potential for replacing a limited number of skilled experts with a multitude of unskilled crowds, by *decomposing* the complex tasks into smaller pieces of subtasks, such that each subtasks becomes low in complexity, requires little specialized skill, time and cognitive effort to be completed by an individual.

After decomposing a complex task into multiple small subtasks, a collective of crowds (or workers) execute the subtasks either *independently* or *dependently*, which depends on whether there are *dependencies* among the subtasks [Malone et al., 2009]. When the subtasks are structured independently, multiple workers are recruited to collaborate *in parallel*, and subtask's quality depends only upon the effort of the worker who performs it. By contrast, when there are dependencies among the subtasks, workers are organized to collaborate *sequentially*, and subtask's quality de-

pends on efforts of multiple workers who jointly produce output. In the sequential process, subtask dependence mainly characterized in the striking feature that one worker's output is used as the starting point for the following worker, which makes the assumption that following worker can do better based on quality solution provided by previous worker hold (see [Kulkarni et al., 2012] [Kulkarni et al., 2011]). Sequential process has been implemented for various types of tasks and exhibits outstanding effectiveness. For instance, CrowdForge [Aniket Kittur and Kraut, 2011] and Soylent [Bernstein et al., 2010] delineate case studies on article writing and word processing respectively with sequential process, and they both come up with high quality final outcomes.

Our research focuses on the complex tasks decomposition in crowdsourcing, wherein the complex tasks can be decomposed and executed in both independent and dependent way. Of particular interest are the work that aim at analyzing workers' strategic behaviors, comparing the *efficiency* of two task decompositions, in terms of *final quality*, and finally generating explicit instructions on strategies for optimal task decomposition. We define two task decompositions as *horizontal task decomposition* for independent subtasks, and *vertical task decomposition* for dependent subtasks. To illustrate the concepts, we refer to the following crowdsourcing-based *proofreading* task as example, wherein an article containing three paragraphs requires spelling, style and grammar error correction. In this context, the article could be horizontally decomposed into three pieces of subtasks that each has one paragraph and be performed by one worker independently, and later combine the revised paragraphs into a coherent article. Meanwhile, the original article could also be vertically decomposed into three sequential subtasks, as "Find-Fix-Verify" proposed by Bernstein et al. [Bernstein et al., 2010]. Specifically, find stage asks a worker to identify the patches that need proofreading throughout the whole article. Fix stage recruits a worker to correct the errors in all patches as many as he can. In the verify stage, all the corrections made in the fix stage are verified by another worker.

Absent explicit task decomposition guidelines, it is difficult for the *task*

requesters to decide how to decompose the task properly. Although some comparative studies of horizontal and vertical task decomposition strategies have been conducted by experimentations [Aniket Kittur and Kraut, 2011], only implicit or even ambiguous conclusions have been drawn. Aiming at providing the task requesters with explicit guidelines on efficient task decomposition, we first construct the formal models for vertical and horizontal task decompositions, which respectively specify the relationship between subtask quality and the worker's effort level in the presence of positive and none dependence among subtasks. Then, the efficiency of task decompositions is explored under two *revenue sharing schemes*, group-based sharing (i.e., equally sharing) and contribution-based sharing, which have received most of the attention so far [Mason and Watts, 2010].

At a broad level, task requester first decomposes the complex task into smaller subtasks, and then presents them to the workers with associated rewards through crowdsourcing websites, seeking to maximize the quality of the task's solution, which is positively related to the efforts devoted by all the workers. The workers perform the subtasks either sequentially or in parallel, and each of them is self-interested and devotes an amount of effort to her own subtask for individual utility maximization. When the complex task is decomposed, it is decomposed into subtasks with varying intrinsic difficulties. We define subtask dependence by characterizing how subtask's difficulty can be altered by the quality of the previous subtask.

We summarize our main contribution in the following. In Section 3.3, we formally construct the models for both vertical and horizontal task decompositions. We show that the dependence among subtasks can be formalized as the degree to which a subtask's difficulty depends on the qualities of the other subtasks. In Section 3.4, we rigorously analyze the strategic behaviors of the workers, and find that contribution-based revenue sharing scheme provides more incentives for workers to exert higher efforts on difficult subtasks. In Section 3.5, we conduct simulations to analyze and compare the efficiency of two task decompositions, aiming at generating explicit instructions on strategies for optimal task decomposition. We conclude that when the final quality is defined as the sum of the qualities of solutions to

all decomposed subtasks, vertical task decomposition strategy outperforms the horizontal one in improving the quality of the final solution, and give explicit instructions on the *optimal strategy* (i.e., arrangement of subtasks with different difficulties for final quality maximization) under vertical task decomposition situation from the task requester's point of view. Specific examples are also demonstrated to show how instructions can be utilized for constructing optimal strategy and selecting the best one from the given strategies.

Furthermore, based on the above studies and results, we design and conduct two series of proofreading experiments on Amazon Mechanical Turk. The first series of experiments compare the efficacy of vertical decomposition and horizontal decomposition on proofreading task. The second series of experiments focus on the vertical task decomposition, and compare the situations where the first subtasks with different difficulties. Experimental design are detailed in Section 3.6. We present the experiment data we collect in Section 3.7. According to the data analysis, we 1) verify the superiority of the vertical task decomposition strategy over the horizontal one on proofreading task; and 2) apply some of the our proposed instructions on vertically decomposing the proofreading task, which show promise on improving the quality of final outcome.

3.2 Related Work

In order to harness crowdsourcing marketplaces, such as MTurk [Ipeirotis, 2010], to accomplish complex tasks, efficient task decomposition in workflow design has been a research problem that received considerable attention from many researchers.

CrowdForge is a rigidly defined framework for task decomposition [Aniket Kittur and Kraut, 2011]. It delineates the case study on article writing, which consists of three sequential subtasks, as outline construction, information collection, and writing. On average, five articles produced by the above process are of higher qualities than those produced by inde-

pendent individuals. Turkomatic comes up with the innovative idea that ask the crowd to help decomposing the tasks into set of subtasks that can be posted in parallel or must be serially executed [Kulkarni et al., 2011]. Afterwards, to guarantee the success of task decomposition, real-time expert intervention is introduced in Turkomatic [Kulkarni et al., 2012]. For specific tasks, particular sequential processes are executed in a previously determined manner. Soylent contributes the three-stage pattern, Find-Fix-Verify, to text editing services in word processing, with high feasibility and efficiency [Bernstein et al., 2010]. Similarly, PlateMate provides Tag-Identify-Measure for nutritional analysis from photos of meals [Noronha et al., 2011]. Although all these workflows show superiority of vertical task decomposition over horizontal task decomposition in complex task solving, and are evaluated to be efficient empirically, further and explicit analysis on efficient task decomposition is still hindered by the absence of formalized model, which takes into account the dependence among subtasks.

Furthermore, iterative paradigm could be introduced into task decomposition as subdecompositions to break subtasks down to small pieces until they can be solved individually. At the same time, iterative paradigm can serve as an workflow alone for complex task solving without task decomposition. TurKit presents the first examples in the literature of iterative workflow for complex tasks [Greg Little and Miller, 2010] [Little et al., 2009]. Such iterative task solving workflows without task decomposition are beyond the scope of this study.

In addition, it is worth noting that, for efficient complex task solving in crowdsourcing, online task assignment has been a research problem that has also attracted attentions from many researchers. Different from the works that provide efficient solutions for applications with independent subtasks [Ho and Vaughan, 2012] [Tran-Thanh et al., 2013], Long et al. [Tran-Thanh et al., 2014a] first investigates the interdependent subtask allocation in crowdsourcing systems, which has the most relevant background to our research.

3.3 The Model

In this section we consider the complex task, e.g., proofreading, that can be both vertically and horizontally decomposed into N $(N > 1)$ subtasks. In both situations, N workers contribute their efforts, such as time and resources, to N subtasks respectively. The amount of effort exerted by worker i to subtask i is characterized by e_i. In the following, we formally introduce the models for vertical and horizontal task decompositions respectively, by specifying the relationship between subtask quality and the worker's effort level in the presence of positive and none dependence among subtasks.

3.3.1 Vertical Task Decomposition

We first take Find-Fix-Verify workflow for proofreading as an specific example to highlight the relationship between subtasks' qualities and the quality of the final solution in the presence of positive dependence among sequential subtasks.

Find-Fix-Verify: This sequential task solving process is mainly characterized in the striking feature that one worker's output is used as the starting point for the following worker. The output of Find stage is the *patches* that may have spelling and grammar errors and need corrections or edits. The quality of patches could be evaluated by how well they cover the true positions (i.e., the positions with actual errors) [Tran-Thanh et al., 2014a]. The output of Fix stage is the corrections of the errors in patches that identified in Find stage. The quality of fix task could be evaluated by the number and the average validity of the proposed corrections. Last, in Verify stage, workers performs quality control. By either accepting or rejecting the corrections and edits, verify task further improves the proofreading result.

It is worth noting that although the output of Verify stage is the final solution of the proofreading task, the quality obtained by Verify stage is not the quality of the final solution — since the quality of Verify stage absolutely owes to the efforts for verification, it merely contributes a additive portion to the final quality. Find and Fix also contribute to the final quality in this

way. In other words, the quality of the final solution can be viewed as the *cumulative qualities* obtained from all subtasks. Formally, we define the quality of the final solution as follows.

Definition 3.1 (Final quality)**.** The quality of the final solution (Q) to the complex task is the cumulative qualities of all the N decomposed subtasks, i.e.,

$$Q(\mathbf{e}) = \sum_{i=1}^{N} q_{vertical}^{i} \tag{3.1}$$

where $\mathbf{e} = (e_1, \cdots, e_N)$, and $q_{vertical}^{i}$ is the quality function of subtask i.

It is reasonable to assume that, the quality of each subtask is positively related to the effort from the worker. Furthermore, since each subtask takes the output from the previous subtask as input, the quality of each subtask is also positively related to the quality of the previous subtask. Formally, we assume the subtask quality function is governed by the following form.

Assumption 3.1. *The quality of subtask i's solution depends not only on the effort exerted by worker i, but also on the quality of previous subtask's solution, i.e.,*

$$q_{vertical}^{i} = f^{i}(q_{vertical}^{i-1}, e_i) \tag{3.2}$$

where f^i increases with e_i at a decreasing rate, i.e.,

$$\frac{\partial f^i}{\partial e_i} > 0 \text{ and } \frac{\partial^2 f^i}{\partial e_i^2} < 0 \tag{3.3}$$

Remark 3.1. The recursive definition of f^i directly implies that the quality of subtask i's solution depends also on the efforts from all the prior workers. Hence, we can rewrite Eq. (3.2) equivalently as $q_{vertical}^{i} = f^{i}(e_1, \cdots, e_i)$, and any increase in the efforts from the previous workers also leads to an improvement on the quality of subtask i, i.e., $\forall k \in \{1, \cdots, i\}$, $\partial f^i / \partial e_k > 0$. Last, note that, for the first subtask ($i = 1$), the quality function is simplified as $q_{vertical}^{1} = f^{1}(e_1)$.

Before we illustrate how does $q^i_{vertical}$ depend on $q^{i-1}_{vertical}$, we first introduce the concept of *subtask difficulty*. Take Find stage for example, articles that consist more frequent in long and compound sentences, indicate more grammatical errors, and thus require considerable effort for locating the true positions. Also, when the patches are required to be identified at phrase-level instead of sentence-level, it demands greater effort. Thus, high difficulty indicates low marginal contribution based on the same level of effort, which is formalized as follows.

Definition 3.2 (Subtask difficulty)**.** We endow subtask i with weight $\omega_i \in (0,1)$ as its difficulty. Subtask i is said to be more difficult than subtask j (i.e., $\omega_i > \omega_j$), iff for any effort level l

$$\frac{\partial f^i(e_i, e_{-i} = \mathbf{e}_{N-1})}{\partial e_i}|_{\,e_i=l} < \frac{\partial f^j(e_j, e_{-j} = \mathbf{e}_{N-1})}{\partial e_j}|_{\,e_j=l} \tag{3.4}$$

where $\mathbf{e}_{N-1}$ is the effort levels of all other $N-1$ workers. Furthermore, $\sum_{i=1}^{N} \omega_i = 1$.

Remark 3.2. The constraints on the weights ($\sum_{i=1}^{N} \omega_i = 1$ and $\omega_i \in (0,1)$) indicate that any subtask is neither replaceable nor dispensable, and only if all the subtasks are accomplished, is the complex task regarded as accomplished.

Now, we continue with the Find-Fix-Verify example to illustrate how does $q^{i-1}_{vertical}$ affect $q^i_{vertical}$ by altering the difficulty of subtask i. As Find stage, Fix stage also has its own intrinsic difficulty, for example, interacting grammatical errors in multiple patches result in high level of difficulty. Nevertheless, its difficulty can be altered by the quality of the Find task. For example, high quality of Find task due to phrase-level error location reduces the difficulty of Fix task, however, low quality due to the noisy patches can make Fix task more difficult. Combined with Eq. (3.3), we formalize the relationship between previous efforts and current subtask's difficulty as follows.

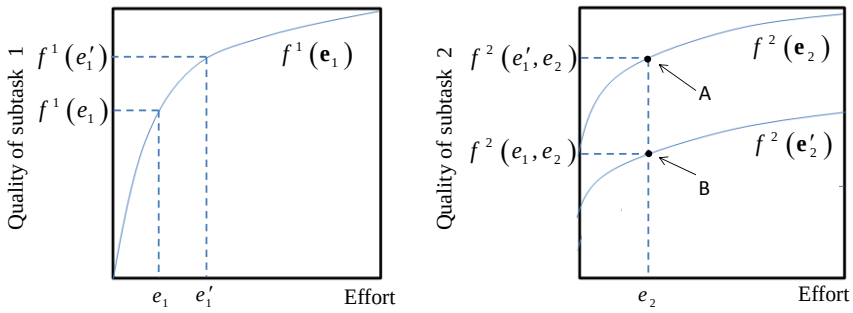


Figure 3.1: For a two-subtask situation, where the quality function for subtask 1 and 2 are both increasing and concave. The quality increase of the first subtask due to the increase of the effort exerted by the first worker (from e_1 to e_1') leads to 1) the quantity increase in subtask 2's quality (from $f^2(e_1, e_2)$ to $f^2(e_1', e_2)$), and 2) the slope increase (slope of point A is greater than that of point B), which indicates the productivity improvement of the second worker.

Assumption 3.2 (Quality dependency)**.** *The difficulty of subtask i decreases as the efforts on previous subtasks increase, i.e.,* $\forall k \leq i-1$*, if* $e_k < e_k'$*, then*

$$\frac{\partial f^i(e_{-k}, e_k)}{\partial e_i} < \frac{\partial f^i(e_{-k}, e_k')}{\partial e_i} \tag{3.5}$$

Remark 3.3. It is worth noting that an increase in the effort previous subtasks not only increases the quality of subtask i in quantity (Eq. (3.3)), but also, according to Eq. (3.5), improves worker i's productivity, which enable worker i to make greater quality improvement by exerting the same level of effort. Fig. 3.1 illustrates the positive quality dependence in a two-subtask situation.

3.3.2 Horizontal Task Decomposition

At a broad level, in horizontal task decomposition situation, the complex task is decomposed into N subtasks with no interdependencies, which im-

plies

$$\partial^2 f^i / \partial e_j \partial e_i = 0, \text{ for } \forall i, j \in 1, \cdots, N.$$

Then, these subtasks are presented to N workers, who devote efforts simultaneously and independently to their own subtasks for individual utility maximization. Finally, the solutions of all subtasks are recomposed into an overall solution, also in a cumulative way. Thus, the quality function of the final solution is defined as in Eq. (3.1), i.e.,

$$Q(\mathbf{e}_N) = \sum_{i=1}^{N} q^i_{horizontal},$$

where $q^i_{horizontal}$ is the quality function of subtask i.

In contrast to vertical task decomposition, where each worker concentrates on a single stage of the workflow (take proofreading for example, Find, Fix or Verify), in horizontal task decomposition, each worker gives considerations to all stages. This makes the worker have to divide her effort among all stages. We assume that the effort e_i, exerted by worker i to subtask i, can be viewed as being distributed among N stages as in the vertical task decomposition situation, proportionally to the difficulties of the stages. This assumption simplifies the results without sacrificing much in terms of generality. Thus, we can define the subtask quality functions by utilizing the form of quality functions defined in vertical task decomposition as follows.

Assumption 3.3 (Horizontal subtask quality function)**.** *The quality of the solution to subtask i only depends on the effort exerted by worker i, which is distributed among N stages proportionally to their difficulties. Hence,*

$$q^i_{horizontal} = \sum_{k=1}^{N} f^k(\omega_1 e_i, \cdots, \omega_k e_i) \tag{3.6}$$

3.3.3 Revenue Sharing Schemes

We formally define two revenue sharing schemes and the corresponding utilities of workers as follows.

Definition 3.3 (Group-based revenue sharing)**.** Under the group-based revenue sharing scheme, each worker receives an equal share of the total revenue given by the task requester, i.e., $R_i = Q(e_i)/N$. Therefore, worker i's utility is

$$\pi(e_i) = \frac{Q(\mathbf{e})}{N} - c(e_i). \tag{3.7}$$

Definition 3.4 (Contribution-based revenue sharing)**.** Under the contribution-based revenue sharing scheme, each worker receives reward determined by her marginal contribution to the final task solution, i.e., $R_i(e_i) = Q_{e_i}(e_i) \cdot e_i$. Thus, worker i's utility is

$$\pi(e_i) = Q_{e_i}(e_i) \cdot e_i - c(e_i) \tag{3.8}$$

Each worker incurs a cost of exerting effort for accomplishing her own subtask. The more effort one devoted, the more costs of time and resources are required. Since the time and resource for individuals are limited, the costs of providing better solutions to subtasks escalate. Based on this argument and previous literature [Holmstrom and Milgrom, 1991], cost is assumed to increase at an increasing rate and of the same functional form for all workers.

Assumption 3.4. *In order to execute subtask i, worker i exerts an effort level e_i with a cost $c(e_i)$. The cost increases with the effort, and it increases at an increasing rate. That is, the cost function is convex, i.e.,*

$$c_{e_i} = \frac{\mathrm{d}c}{\mathrm{d}e_i} > 0 \ \textit{and} \ c_{e_i e_i} = \frac{\mathrm{d}^2 c}{\mathrm{d}e_i^2} > 0 \tag{3.9}$$

3.4 Strategic Behaviors Analysis

In this section we aim to demonstrate the relationship between the incentive and the performance, i.e., the level of effort. This section consists of two parts. Individual behavior analysis focuses on the effort devotion

on the subtasks with different difficulties, and dependent behavior analysis explores the dependent relationship between the effort levels devoted by sequential workers. Our two conclusions of individual behavior analysis are similar to previous theoretical analyses of team-based worker behaviors [Barua et al., 1995] [Wageman and Baker, 1997]. However, the conclusions on workers' dependent behaviors in sequential task solving process are our contributions.

In the following discussion, we assume that the exact efforts exerted by all workers are a common knowledge. This assumption simplifies the results without sacrificing much in term of generality, since similar results readily hold when we extent the exact utilities of workers with commonly known exact efforts to the expected utilities of workers with commonly known effort distribution.

3.4.1 Independent Behaviors of Individuals

Proposition 3.1. *Under both vertical and horizontal task decomposition strategies, when group-based sharing scheme is applied, workers devote higher efforts to easy subtasks than difficult subtasks, for individual utility maximization.*

Proof. Let subtask i be more difficult than subtask j. Suppose each worker chooses an effort level to maximize her utility, and her optimal effort for subtask i and j are denoted by e_i^* and e_j^*, respectively. We want to prove $e_i^* < e_j^*$. For group-based revenue sharing, the first-order conditions for worker on subtask i and j are $Q_{e_i}/N - c_{e_i} = 0$ and $Q_{e_j}/N - c_{e_j} = 0$, respectively. Therefore, under the vertical task decomposition, we have

$$f_{e_i}^i(e_i^*)/N - f_{e_j}^j(e_j^*)/N = c_{e_i}(e_i^*) - c_{e_j}(e_j^*).$$

Suppose $e_i^* > e_j^*$, then $c_{e_i}(e_i^*) > c_{e_j}(e_j^*)$, such that $f_{e_i}^i(e_i^*) > f_{e_j}^j(e_j^*)$. However, since subtask i is more difficult than subtask j, according to Eq. (3.3)

and Eq. (3.4),

$$f^i_{e_i}(e_i = e^*_i, e_j = e^*_j) < f^j_{e_j}(e_i = e^*_j, e_j = e^*_i) < f^j_{e_j}(e_i = e^*_j, e_j = e^*_j)$$

which leads to a contradiction. The supposition $e^*_i = e^*_j$ also leads to contradiction. So $e^*_i < e^*_j$. Similarly, we can prove $e^*_i < e^*_j$ for horizontal task decomposition situation. □

Proposition 3.2. *Under both vertical and horizontal task decomposition strategies, when contribution-based sharing scheme is applied, workers devote efforts to easy subtasks no less than difficult subtasks, for individual utility maximization.*

Proof. The proof is similar to that for Proposition 3.1. □

According to Proposition 3.1 and 3.2, although contribution-based revenue sharing may provide workers with more incentives to perform difficult subtasks than group-based revenue sharing, they both indicate the fact that workers are more inclined to perform easy subtasks. This is consistent with findings in worker behavior studies in crowdsourcing (e.g., [DiPalantino and Vojnovic, 2009] [Yang et al., 2008a] [Zheng et al., 2011]), and highlights the need for task design, which we will explore with respect to task decomposition in Section 3.5.

3.4.2 Dependent Behaviors in Vertical Task Decomposition

Under vertical task decomposition strategy, subtasks are chained together into complementary and sequentially dependent stages in the way, specified in Assumption 3.1, that as the effort exerted to the prior subtask becomes higher, the increase of the quality of the posterior subtask based on the same amount of effort exerted by the following worker becomes larger. In order to explore the dependent relationship between the effort levels of sequential workers, avoiding notational complexity, we use a two-worker situation in

the discussions below. The conclusions are readily extended to N-worker case.

Proposition 3.3. *Consider a two-worker task under group-based revenue sharing scheme, higher effort on the prior subtask from worker 1 incites worker 2 for the posterior subtask to exert higher level of effort, and vice versa. The same holds true when contribution-based revenue sharing scheme is applied.*

Proof. Consider a two-worker situation under group-based revenue sharing scheme. Utility for worker 2 is $\pi(e_2) = (f^1(e_1) + f^2(e_1, e_2))/2 - c(e_2)$. Her optimal efforts e_{21}^* for $e_1 = e_{11}$ and e_{22}^* for $e_1 = e_{12}$ fit the first-order conditions as $f_{e_2}^2(e_{11}, e_{21}^*)/2 - c_{e_2}(e_{21}^*) = 0$ and $f_{e_2}^2(e_{11}, e_{21}^*)/2 - c_{e_2}(e_{21}^*) = 0$. Therefore,

$$\frac{1}{2} f_{e_2}^2(e_{11}, e_{21}^*) - c_{e_2}(e_{21}^*) = \frac{1}{2} f_{e_2}^2(e_{12}, e_{22}^*) - c_{e_2}(e_{22}^*).$$

When $e_{11} < e_{12}$, suppose $e_{21}^* > e_{22}^*$, we have

$$c_{e_2}(e_{21}^*) > c_{e_2}(e_{22}^*) \Rightarrow f_{e_2}^2(e_{11}, e_{21}^*) > f_{e_2}^2(e_{12}, e_{22}^*).$$

However, $f_{e_2}^2(e_{11}, e_{21}^*) < f_{e_2}^2(e_{12}, e_{22}^*)$ for $e_{21}^* > e_{22}^*$ and $e_{12} > e_{11}$, which leads to a contradiction. Therefore, when $e_{11} < e_{12}$, $e_{21}^* < e_{22}^*$. Similarly, when $e_{11} > e_{12}$, we have $e_{21}^* > e_{22}^*$.

In the same way, we can prove when $e_{11} < e_{12}$, $e_{21}^* < e_{22}^*$, and when $e_{11} > e_{12}$, $e_{21}^* > e_{22}^*$ for the contribution-based revenue sharing scheme. □

Since under vertical task decomposition strategy, workers are endowed with the ability to observe the effort levels exerted by the previous workers before him, any extra effort devoted by the first worker can not only benefit the second worker, but also cause an extra effort from the second worker, thus, in an N-worker situation, lead to a "chain effect" that triggers a sequence of extra efforts from all the following workers. This can significantly improve the final quality of the task. By contrast, horizontal task

decomposition strategy inherently cannot have this advantage.

Proposition 3.4. *Consider a two-worker task under group-based revenue sharing, worker 1 is incited to exert higher effort on the prior subtask with the believe that worker 2 will exert high effort for the posterior subtask, and vice versa. The same holds true when contribution-based revenue sharing scheme is applied.*

Proof. Consider a two-worker situation under group-based revenue sharing scheme. Utility for worker 1 is $\pi(e_1) = \frac{1}{2}[f^1(e_1) + f^2(e_1, e_2)] - c(e_1)$. For worker 1, her optimal effort levels, e^*_{11} for $e_2 = e_{21}$, and e^*_{12} for $e_2 = e_{22}$ fit the first-order conditions as follows.

$$\pi_{e_1}(e^*_{11}) = \frac{1}{2}[f^1_{e_1}(e^*_{11}) + f^2_{e_1}(e^*_{11}, e_{21})] - c_{e_1}(e^*_{11}) = 0,$$

$$\text{and } \pi_{e_1}(e^*_{12}) = \frac{1}{2}[f^1_{e_1}(e^*_{12}) + f^2_{e_1}(e^*_{12}, e_{22})] - c_{e_1}(e^*_{12}) = 0.$$

Suppose $e^*_{11} < e^*_{12}$ for $e_{21} > e_{22}$, then $c_{e_1}(e^*_{11}) < c_{e_1}(e^*_{12})$. Therefore,

$$f^1_{e_1}(e^*_{11}) + f^2_{e_1}(e^*_{11}, e_{21}) < f^1_{e_1}(e^*_{12}) + f^2_{e_1}(e^*_{12}, e_{22}).$$

However, when $e^*_{11} < e^*_{12}$ and $e_{21} > e_{22}$, we have

$$f^1_{e_1}(e^*_{11}) > f^1_{e_1}(e^*_{12})$$

$$\text{and } \; f^2_{e_1}(e^*_{11}, e_{21}) > f^2_{e_1}(e^*_{12}, e_{21}) > f^2_{e_1}(e^*_{12}, e_{22}),$$

which lead to a contradiction. Therefore, when $e_{21} > e_{22}$, $e^*_{11} > e^*_{12}$. Similarly, we can prove when $e_{21} < e_{22}$, $e^*_{11} < e^*_{12}$.

In the same way, we can prove when $e_{21} > e_{22}$, $e^*_{11} > e^*_{12}$ and when $e_{21} < e_{22}$, $e^*_{11} < e^*_{12}$ for contribution-based revenue sharing scheme. □

Proposition 3.5. *Consider a two-worker task situation under group-based sharing scheme. When two subtasks have the same difficulty, the same amount of increase in reward incites more efforts from worker 1 than that*

from worker 2.

Proof. Without loss of generality, suppose that for the workers, maximization problem is given by

$$\underset{e_i}{\operatorname{argmax}}\{\frac{\alpha}{2}Q(\mathbf{e})-c(e_i)\},\ \ \alpha\in(0,1],$$

where the reward is able to be modified by adjusting the value of α. Then for worker 1, the first-order condition is $\pi(e_1)=\frac{\alpha}{2}[f^1_{e_1}(e^*_1)+f^2_{e_1}(e^*_1,e_2)]-c_{e_1}(e^*_1)=0$, where e^*_1 represents her optimal effort level. Then we have

$$\frac{\mathrm{d}e_1}{\mathrm{d}\alpha}=-\frac{f^1_{e_1}+f^2_{e_1}}{\alpha(f^1_{e_1e_1}+f^2_{e_1e_1})-2c_{e_1e_1}}>0$$

The positive values of $\frac{\mathrm{d}e_1}{\mathrm{d}\alpha}$ shows that the effort of worker 1 increases with the rewards. Similarly, for worker 2, her the optimal effort level increases as the reward increases, i.e.,

$$\frac{\mathrm{d}e_2}{\mathrm{d}\alpha}=-\frac{f^2_{e_2}}{\alpha(f^2_{e_2e_2})-2c_{e_2e_2}}>0.$$

Notice that we assume that the cost function is the same for all workers, and when two subtasks have the same difficulty, i.e., $f^1_{e_1}=f^2_{e_1}=f^2_{e_2}$ and $f^1_{e_1e_1}=f^2_{e_1e_1}=f^2_{e_2e_2}$, we have

$$\frac{\mathrm{d}e_1}{\mathrm{d}\alpha}>\frac{\mathrm{d}e_2}{\mathrm{d}\alpha}.$$

i.e., the same amount of increase in reward incites more efforts from worker 1 than that from worker 2. □

Proposition 3.6. *Consider a two-worker task under contribution-based revenue sharing scheme. When two subtasks have the same difficulty, the same amount of increase in reward incites less efforts from worker 1 than that from worker 2.*

Proof. Without loss of generality, suppose that for worker i, the utility function is $\pi(e_i) = Q_{e_i} \cdot e_i - c(e_i) + \alpha^2 e_i,\ \alpha \in (0,1]$. For worker 1, the maximization problem is given by

$$\underset{e_1}{\operatorname{argmax}} \{ f^1_{e_1}(e_1) \cdot e_1 + f^2_{e_1}(e_1) \cdot e_1 - c(e_1) + \alpha^2 e_1 \}$$

$$\Rightarrow \ \frac{\mathrm{d}e_1}{\mathrm{d}\alpha} = -\frac{2\alpha}{f^1_{e_1e_1e_1}e_1 + f^2_{e_1e_1e_1}e_1 + 2f^1_{e_1e_1} + 2f^2_{e_1e_1} - c_{e_1e_1}} > 0.$$

That is, for worker 1, her optimal effort level increases as the reward increases.

For worker 2, the maximization problem is given by

$$\underset{e_2}{\operatorname{argmax}} \{ f^2_{e_2}(e_2) \cdot e_2 - c(e_2) + \alpha^2 e_2 \}$$

$$\Rightarrow \ \frac{\mathrm{d}e_2}{\mathrm{d}\alpha} = -\frac{2\alpha}{f^2_{e_2e_2e_2}e_2 + 2f^2_{e_2e_2} - c_{e_2e_2}} > 0.$$

That is, for worker 2, her optimal effort level also increases as the reward increases.

Notice that we assume that the cost function is the same for all referrers, when effort exerted by referrer 1 has the same effect on the influence increase for both of the subtasks, i.e., $f^1_{e_1e_1} = f^2_{e_1e_1} = f^2_{e_2e_2}$ and $f^1_{e_1e_1e_1} = f^2_{e_1e_1e_1} = f^2_{e_2e_2e_2}$, we have

$$\frac{\mathrm{d}e_1}{\mathrm{d}\alpha} < \frac{\mathrm{d}e_2}{\mathrm{d}\alpha}.$$

That is, when the workers are provided with same amount of extra reward for the same extra effort, the first worker exerts lower effort than that exerted by the second worker. □

According to Proposition 3.5, under group-based revenue sharing scheme, increase in reward for the first worker will be more beneficial to the requester. By contrast, according to Proposition 3.6, under contribution-

based revenue sharing scheme, award marginal reward to second worker will be more beneficial to the requester.

3.5 Task Decomposition Strategy Analysis and Comparison

Different from the behavior analysis, which focuses on individual strategic effort devotion, we construct simulation aiming at explicitly evaluating and comparing the efficiencies, in terms of the final quality (Eq. (3.1)), of two task decomposition strategies under two revenue sharing schemes. It is worth noting that, besides the 2-subtask and 3-subtask situation presented in the following, we also explored the situations for $N = 4,\cdots,9$, which not shown here due to limited space but generate the similar results.

3.5.1 Utility Specification

Subtask quality functions

We consider the simulated task solving process under the assumption of subtask quality functions in the Cobb-Douglas form. We assume there are N workers for N subtasks. Specifically, for subtask i, the quality functions under vertical and horizontal task decompositions are respectively defined as

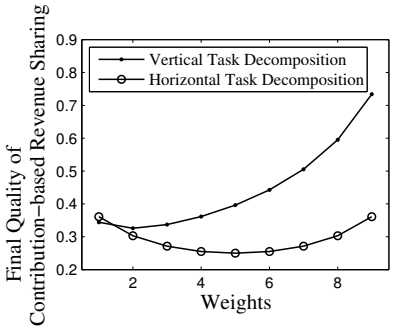
$$q^i_{vertical}(e_1,\cdots,e_i)=\prod_{k=1}^{i} e_k^{\omega_k}$$

and

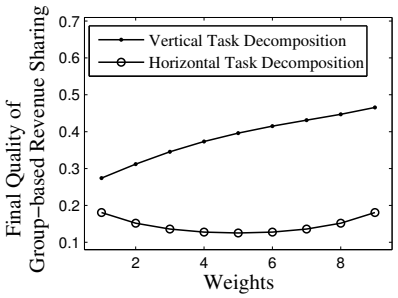
$$q^i_{horizontal}(e_i)=\sum_{k=1}^{N}\prod_{r=1}^{k}(\omega_r e_i)^{\frac{\omega_r}{N}}$$

where $\omega_i\in(0,1)$ is the difficulty of subtask i, and $\sum_{i=1}^{N}\omega_i=1$.

Remark 3.4. According to the specified functions, high value of ω_i corresponds to high difficulty of the subtask, and it can be easily verified that



(a) Final quality comparison under contribution-based revenue sharing.



(b) Final quality comparison under group-based revenue sharing.

Figure 3.2: Efficiency comparison of vertical and horizontal task decompositions (two-subtask situation). Generally, vertical task decomposition strategy outperforms horizontal task decomposition strategy in terms of final quality.

they both satisfy the requirements given in Assumption 3.1.

Cost functions. We simply specify the cost function for worker i as

$$c(e_i) = e_i^2, \;\; i = 1, 2, \cdots, N$$

which satisfies the requirements given in Assumption 3.4.

3.5.2 Efficiency Comparison

Efficiency Comparison of Two Task Decompositions

Two-subtask situation is depicted in Fig. 3.2 as a concise case to illustrate the efficiency difference between vertical and horizontal task decomposition strategies. We endow each of two subtasks with a weight ($\omega_1, \omega_2 \in (0,1)$), which is restricted to one decimal place. Then, we exhaustively examine the final quality under all the combinations of two weights, which are sorted in

increasing order of the first subtask's weight, as given in the Table 3.1.

Table 3.1: Weight combinations for 2-subtask situation.

X-axis	1	2	3	4	5	6	7	8	9
ω_1	1	2	3	4	5	6	7	8	9
ω_2	9	8	7	6	5	4	3	2	1

Note: weight = $\omega_i/10$

Fig. 3.2 (a) and (b) illustrate the final qualities of two task decomposition strategies under group-based and contribution-based revenue sharing schemes, respectively. As can be seen in them, the final quality of vertical situation under both revenue sharing schemes increases along with the difficulty of the first subtask. By contrast, the efficiency of horizontal situation increases with the difficulty of the first subtask only when the first subtask is more difficult than the second one, i.e., ($\omega_2 > \omega_1$). Otherwise, the efficiency decreases with the difficulty of the first subtask.

It is clear from Fig. 3.2 that for final quality estimation, vertical task decomposition strategy is superior to the horizontal one, under both group-based and contribution-based revenue sharing schemes. Furthermore, when the difficulty of the first subtask gets higher, vertical strategy exhibits clearer superiority over the horizontal one. It is worth noting that, for the group-based revenue sharing, in Fig. 3.2 (b), at the first point, corresponding to weight combination (0.1, 0.9), horizontal task decomposition defeats the vertical task decomposition. Intuitively, the reason is that, when the second subtask has high difficulty, the second worker refuses to make much effort because it won't bring much marginal increase to the final quality, let alone half of the total revenue will go to the first worker according to the group-based revenue sharing, which won't compensate the second worker. Thus, the vertical decomposition is less effective.

To sum up, generally, vertical task decomposition outperforms horizontal task decomposition in terms of final quality. Besides the two-subtask situation, we also explored the situations for $N = 3, \cdots, 9$, which not shown here due to limited space but generate similar results.

Table 3.2: Weight combinations for 3-subtask situation.

							(a)						
X-axis	1	2	$\cdots$	8	9	10	$\cdots$	15	16	$\cdots$	21	22	23
ω_1	1	1	$\cdots$	1	2	2	$\cdots$	2	3	$\cdots$	3	4	4
ω_2	1	2	$\cdots$	8	1	2	$\cdots$	7	1	$\cdots$	6	1	5
ω_3	8	7	$\cdots$	1	7	6	$\cdots$	1	6	$\cdots$	1	5	1
							(b)						
X-axis	1	$\cdots$	3	4	5	6	7	8	$\cdots$	10	11	12	13
ω_1	4	$\cdots$	4	5	5	5	5	6	$\cdots$	6	7	7	8
ω_2	2	$\cdots$	4	1	2	3	4	1	$\cdots$	3	1	2	1
ω_3	4	$\cdots$	2	4	3	2	1	3	$\cdots$	1	2	1	1

Note: weight = $\omega_i/10$

Vertical Task Decomposition Efficiency under Two Revenue Sharing Schemes

In Fig. 3.3 we explore 3-subtask situation to compare the efficiencies of two revenue sharing schemes for vertical task decomposition strategy. In the following, we first present the lessons we learn by analysing the data illustrated in Fig. 3.3, then give examples of revenue sharing scheme selection based on what we learn.

As we do for the two-subtask situation, we respectively endow 3 subtasks with weights ω_1, ω_2, ω_3, which are restricted to one decimal place as well, and then exhaustively examine all combinations of the weights, i.e., a permutation of $\{0.1, 0.2, \cdots, 0.9\}$ with a restriction that the sum of three weights equals 1. As shown in Table 3.2, for 3-subtask situation, there are a total of 36 combinations, and they are sorted in a *lexicographic manner*, i.e., $(\omega_1, \omega_2, \omega_3)$ occurs before $(\omega_1', \omega_2', \omega_3')$ iff $\omega_1 < \omega_1'$, or $\omega_1 = \omega_1'$ and $\omega_2 < \omega_2'$, or $\omega_1 = \omega_1'$ and $\omega_2 = \omega_2'$ and $\omega_3 < \omega_3'$.

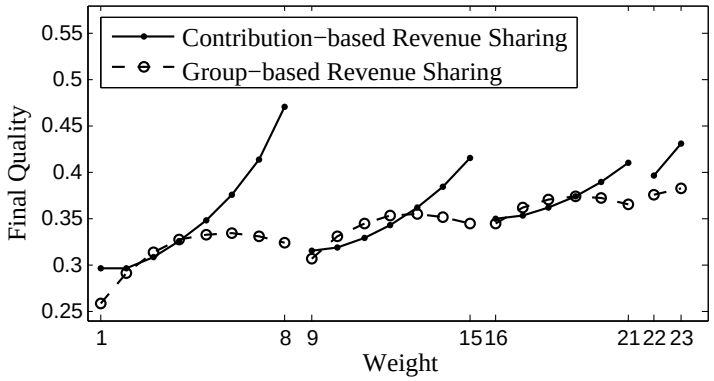
Lesson 1. *When the first subtask is not the most difficult subtask, for the series of subtasks with the strictly concave weights (i.e., $\omega_1 < \omega_2$ and $\omega_2 > \omega_3$), contribution-based revenue sharing scheme leads to higher final qualities than group-based revenue sharing scheme.*

As can be observed in Fig. 3.3 (a) (for the X axis and weight combinations, please refer to Table 3.2 (a)), when the first subtask is not the most difficult subtask, although in the beginning, contribution-based revenue sharing scheme is inferior to group-based scheme, as moving along the weight combinations sorted in lexicographic manner, it becomes superior. The turning point lies at the weight combiantion which begins to be strictly concave. For example, in Fig. 3.3 (a), for the eight series of subtasks begin with weight 0.1 (X axis scale is 1 to 8), the final qualities brought by contribution-based sharing scheme exceed those brought by group-based sharing scheme at the fifth series of weights (5. (0.1, 0.5, 0.4)), and then become completely superior.

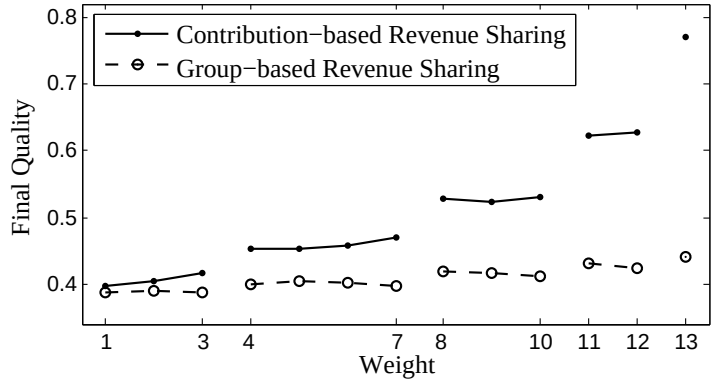
Lesson 2. *When the first subtask is the most difficult subtask, contribution-based revenue sharing scheme leads to higher final qualities than group-based revenue sharing scheme.*

As illustrated in Fig. 3.3 (b) (for the X axis and weight combinations, please refer to Table 3.2 (b)), when the first subtask is one of the most difficult subtasks, contribution-based sharing scheme is superior to group-based sharing scheme. Furthermore, as the weight of the first subtask gets higher, contribution-based sharing scheme exhibits clearer superiority over the group-based one.

Example 3.1. Task requester who aims at optimizing the final quality has to determine the revenue sharing scheme first. For these very similar task decompositions, (0.3, 0.3, 0.4), (0.3, 0.4, 0.3) and (0.4, 0.3, 0.3), the revenue sharing scheme can be decided based on Lesson 1 and 2. For (0.3, 0.3, 0.4), which is neither concave nor starts with a most difficult subtask, group-based scheme is selected. For (0.3, 0.4, 0.3), which is strictly concave, according to Lesson 1, contribution-based scheme is selected. Also, for (0.4, 0.3, 0.3), contribution-based scheme is selected, the reason is different — according to Lesson 2, for the series of subtasks start with the most difficult subtask, contribution-based scheme is superior. All selections can



(a) Final quality comparison when the first subtask is the most difficult subtask.



(b) Final quality comparison when the first subtask is not the most difficult subtask.

Figure 3.3: Efficiency comparison of two revenue sharing schemes for vertical task decomposition strategy. In general, when the series of subtasks begin with a difficult subtask, the contribution-based revenue sharing scheme is superior to the group-based revenue sharing scheme.

be verified in Fig. 3.3.

3.5.3 Vertical Task Decomposition Strategy

In the case of vertical task decomposition, the multiple subtasks are chained into sequentially dependent stages. In this study, we restrict out attention to the case where prior subtasks positively support the subsequent subtasks, as assumed by Eq. (3.5). As the qualities of the prior subtasks improve, the positive support they provide become strong, which makes the subsequent subtasks more dependent on the prior ones. Further, in the extreme situation where all subtasks have the highest qualities with all workers exert their highest efforts ($e = 1$), the dependence among subtasks become strongest, which can be viewed as the intrinsic dependence among the sequential subtasks. Formally, we define the dependence of two sequential subtasks, and then the total dependence among all subtasks as follows.

Definition 3.5. Given a succession of N subtasks and corresponding weights, the sequential dependence between subtasks $i-1$ and i is defined as

$$d(e_{i-1}, e_i) = \frac{\partial}{\partial e_{i-1}} \frac{\partial q^i}{\partial e_i}\bigg|_{e_1=\cdots=e_{i-1}=e_i=1,} = \omega_{i-1}\omega_i, \tag{3.10}$$

and the total dependence among all subtask is

$$\sum_{i=2}^{N} d(e_{i-1}, e_i). \tag{3.11}$$

Lessons Learned on Group-based Revenue Sharing Scheme

In Fig. 3.4, we explore 3-subtask situation and illustrate the final quality of vertical task decomposition strategy under group-based revenue sharing scheme. In the following, we first present the lessons we learn by analysing the data illustrated in Fig. 3.4, then integrate them into explicit guidelines on task decomposition from task requester's point of view, lastly give examples of task decomposition selection based on the lessons we learn.

Lesson 3. *The highest final quality brought by the series of subtasks begin with high difficulty is superior to that brought by the series of subtasks begin with low difficulty under group-based revenue sharing scheme.*

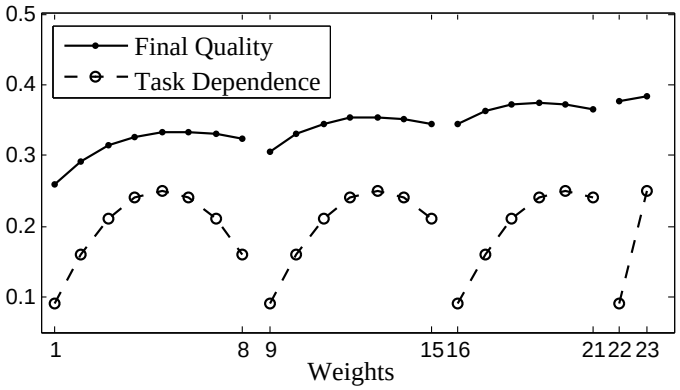
As can be observed in Fig. 3.4 (a), when the first subtask is not the most difficult subtask, the highest quality of the series of subtasks begin with weight equals 0.4 (X axis scale is 22 to 23) is superior to that of the series of subtasks begin with weight 0.3, which then is superior to that of the series of subtasks begin with weight 0.2, and so on. Similar observations can be observed in Fig. 3.4 (b), which illustrates the situation where the first subtask is one of the most difficult subtasks.

Lesson 4. *When the first subtask is not the most difficult subtask, for the series of subtasks begin with the same difficulty, the lowest task dependence leads to the lowest final quality.*

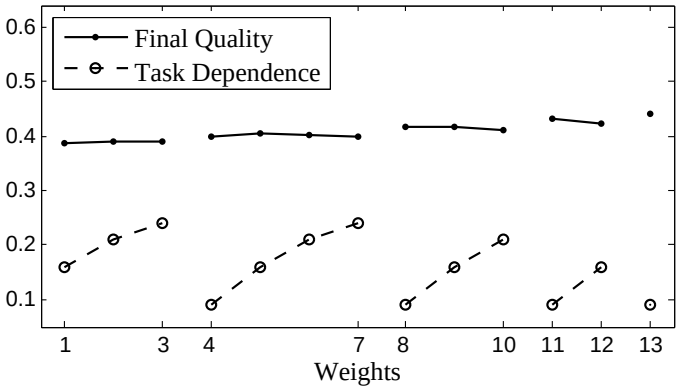
For instance, in Fig. 3.4 (a), for the eight series of subtasks begin with weight 0.1 (X axis scale is 1 to 8), the lowest task dependence, corresponding to x-axis value 1, gives us the lowest final quality among these eight series of subtasks.

Lesson 5. *When the first subtask is the most difficult subtask, for the series of subtasks begin with the same difficulty, i) the highest task dependence leads to the lowest final quality; and ii) the highest task dependence given by the convex weights (i.e.,* $\omega_1 \geq \omega_2$ *and* $\omega_2 \leq \omega_3$*) leads to the highest final quality.*

For instance, as can be observed in Fig. 3.4 (b), for four series of subtasks begin with weight 0.5 (X axis scale is 4 to 7), the highest task dependence, corresponding to x-axis value 7, gives us the lowest final quality among these four series of subtasks. Furthermore, in these four series of subtasks begin with weight 0.5, there are two with convex weights (4. (0.5, 0.1, 0.4) and 5. (0.5, 0.2, 0.3), with task dependence 0.09 and 0.16, respec-



(a) When the first subtask is not the most difficult subtask.



(b) When the first subtask is the most difficult subtask.

Figure 3.4: Efficiency estimation of vertical task decomposition strategy under group-based revenue sharing scheme. In general, series of subtasks begin with high difficulties generate high efficiency with respect to the final quality.

tively), and the higher task dependence, corresponding to x-axis value 5, gives us the highest final quality among these four series of subtasks.

Example 3.2. Suppose the task requester selects the group-based revenue sharing scheme and has to choose among three very similar task decompositions, as (0.3, 0.4, 0.3), (0.4, 0.3, 0.3) and (0.4, 0.4, 0.2). According to Lesson 3, he would prefer the series of subtasks start with a more difficult subtask and eliminate option (0.3, 0.4, 0.3). Furthermore, according to Lesson 5, convex weights (0.4, 0.3, 0.3) is the decomposition, among all series of subtasks starts with difficulty 0.4, that leads to the highest final quality. So the task requester can construct her preference as (0.4, 0.3, 0.3) $\succ$ (0.4, 0.4, 0.2) $\succ$ (0.3, 0.4, 0.3).

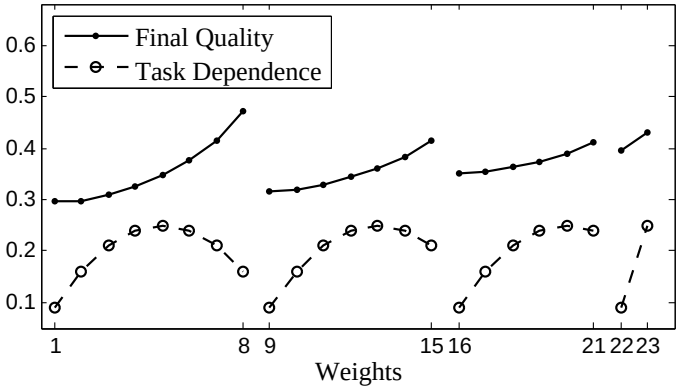
Lessons Learned on Contribution-based Revenue Sharing Scheme

We explore the 3-subtask situation under contribution-based revenue sharing scheme as shown in Fig. 3.5. In the following, we present the lessons we learn by analysing the data illustrated in Fig. 3.5, and integrate them into explicit guidelines on task decomposition from task requester's point of view.

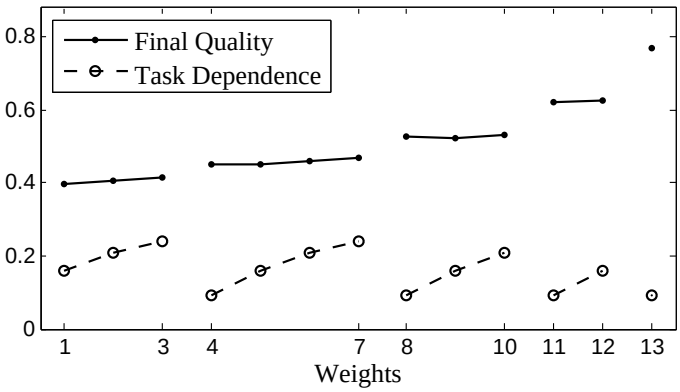
Lesson 6. *When the first subtask is not the most difficult subtask, for an N-subtask situation, given a segment of weights on the first k ($k < N$) subtasks, the highest weight of all the possible weights on the $(k+1)-th$ subtask leads to the highest final quality.*

Take the 3-subtask situation in Fig. 3.5 (a) as an example, given the weight on the first subtask equaling 0.1 (X axis scale is 1 to 8), all possible weights on the second subtask are 0.2, 0.3, $\cdots$, 0.8, then the highest weight on the second subtask which equals 0.8 (X axis value is 8) gives us the highest final quality among all the series of subtasks begin with weight 0.1.

Lesson 7. *When the first subtask is the most difficult subtask, the more difficult the first subtask is, the more efficient is the contribution-based revenue*



(a) When the first subtask is not the most difficult subtask.



(b) When the first subtask is the most difficult subtask.

Figure 3.5: Efficiency estimation of vertical task decomposition strategy under contribution-based revenue sharing scheme.

sharing scheme.

As can be observed in Fig. 3.5 (b), the final quality brought by the series of subtasks begin with weight equals 0.8 (X axis value is 13) is higher than those brought by the series of subtasks begin with weight 0.7, which are higher than those brought by the series of subtasks begin with weight 0.6, and so on.

Lesson 8. *When the first subtask is the most difficult subtask, for the series of subtasks begin with the same difficulty, highest task dependence leads to the highest final quality.*

For instance, as can be observed in Fig. 3.5 (b), for the four series of subtasks begin with weight equals 0.5 (X axis scale is 4 to 7), the highest task dependence, corresponding to x-axis value 7, gives us the highest final quality among these four series of subtasks.

Example 3.3. Suppose the task requester is restricted to start with the subtask of a given difficulty. When the first subtask is not the most difficult subtask, (for example, with difficulty 0.3), according to Lesson 6, the optimal decomposition is the one whose second subtask's difficulty is the highest among all possible difficulties (in this case, 0.1, $\cdots$, 0.6), so the optimal decomposition is (0.3, 0.6, 0.1). When the first subtask is the most difficult subtask, (for example, with difficulty 0.5), according to Lesson 8, the optimal decomposition is (0.5, 0.4, 0.1) with the highest dependence 0.24 among all series of subtasks start with difficulty 0.5.

3.6 Proofreading Experiments on MTurk

MTurk is a crowdsourcing web service that coordinates the supply and the demand of tasks that require human intelligence to complete. The inspiration of the system was to have human users complete simple tasks that would otherwise be extremely difficult (if not impossible) for computers to

Table 3.3: English articles for proofreading task.

Article	Words	Grammar errors	Spelling errors	Logically misused words	Total errors
Tuberculosis	215	7	0	3	10
Change in fashion	167	4	1	5	10
Job interview	263	6	2	2	10
American music	273	5	1	4	10
Weather forecast	190	8	0	2	10
City and county	237	6	0	4	10

perform. Those simple tasks are in various domains, for example, for example to identify objects in images, find relevant information, or to do natural language processing.

Proofreading tasks, i.e., correcting articles to make them error-free, are conducted on MTurk. Six articles that require proofreading are in English. They are all from proofreading and error correction part of the Test for English Majors, Band 8 (TEM-8)[1]. TEM-8 is the highest-level test to measure the English proficiency of Chinese university undergraduates majoring in English Language and Literature and to examine whether these students meet the required levels of English language abilities as specified in the National College English Teaching Syllabus for English Majors [Jin and Fan, 2011]. The vocabulary requirement for TEM$-$8 is 13000 words.

In the experiment, the six articles were chosen to cover various domains, such as medical care, culture, technology, and everyday life. We characterize three different types of error in each article as *grammar errors*, *spelling errors* and *logically misused words*. Table 3.3 gives the statistics on the numbers of errors in each type for all articles.

[1]http://en.wikipedia.org/wiki/College_English_Test

3.6.1 Experimental Design: Comparison of Task Decomposition Strategies

In the first experiment, we attempted to 1) verify the efficacy of vertical task decomposition in improving the quality of the task solution; and 2) show the superiority of vertical task decomposition strategy over the horizontal task decomposition strategy.

To these aims, we conducted two contrast groups of proofreading experiments. One is under the horizontal task decomposition strategy, where the proofreading task for the workers is to "read the English article, and correct the grammar errors, spelling errors and logically misused words as many as possible". In other words, the workers are instructed to use their own judgment on which words to correct and how to correct. One article is supposed to be proofread independently by an individual worker. The article is presented in text format in multiple lines, and each line is endowed with a line number. Workers are required to identify their corrections as

Line No., *incorrect word / phrase* $\rightarrow$ *correct word / phrase*.

The other group is under the vertical task decomposition strategy, where the proofreading task is decomposed into three sequential subtasks, Find-Fix-Verify, as formally modeled in Section 3.3.1. Specifically, in the experiment, the Find task for the workers is to "read the English articles and identify grammar errors, spelling errors and logically misused words as many as possible". Furthermore, the worker are notified of the following working process as "you don't need to provide any correction. Other workers will fix the errors you identified to finally make the articles correct". The article is also presented in text format in multiple lines, and each line is endowed with a line number. Workers are required to identify each error as

Line No., *incorrect word / phrase*.

After the Find task is finished, Fix task will be announced on MTurk, with the potential errors identified in the Find task. It is worth noting that, unlike the exact workflow proposed by Bernstein et al. in their work

Table 3.4: Information for experiment 1.

Experiment	Workflow	Articles	Payment ($) (Bonus*)	Duration (min)	Qualifications Required
Vertical Task Decomposition	**Find-Fix-Verify:** Granularity Levels: Phrase-wise Number of Corrections: 1 Voting Methods: Vote / Veto	Article 1, 2, 3	Find: 0.5 (0.1)	20	None
			Fix: 0.5 (0.1)	20	
			Verify: 0.5 (0.1)	20	
		Article 4, 5, 6	Find: 0.5 (0.1)	20	
			Fix: 0.5 (0.1)	20	
			Verify: 0.5 (0.1)	20	
Horizontal Task Decomposition	**One article for an individual.**	Article 1~ 6	0.5 (0.1)	20/Article	

*When correctness ≥ 90%, bonus is awarded.

[Bernstein et al., 2010], the error correction in our Fix task is not constrained on the identified errors in Find task (see Section 3.3.1). The Fix task for workers is to "read the article and correct the identified grammar errors, spelling errors and logically misused words as many as possible. Some (but not all) possible mistakes are indicated in parentheses, if you think it is not a mistake, please leave as it is". Similarly, the worker are notified of the following working process as "other workers will vote on the corrections you provide to finally make the articles correct". Workers are required to identify their corrections as

Line No., *incorrect word / phrase* $\rightarrow$ *correct word / phrase*.

The last phase is the Vote task, where the corrections given in the Fix task are presented to the workers in the above form, and they are asked to "read the article and examine the corrections". Workers are required to show vote or veto on every correction as

If the correction is correct, tag 'O'; otherwise tag 'X'.

3.6.2 Experimental Design: First Subtasks with Different Difficulties in Vertical Task Decomposition

Explicit instructions on the *optimal strategy* (i.e., arrangement of subtasks with different difficulties for final quality maximization) under vertical task decomposition situation from the task requester's point of view are discussed and proposed in Section 3.5. In this experiment, we applied them to the proofreading task decomposition, with the aim of verifying and revising the instructions such that they are more correct when subjected to real applications.

To this aim, we conducted two contrast groups of proofreading experiments. In both groups, proofreading task is vertically decomposed into three sequential subtasks, as Find-Fix-Verify. As we discussed in Section 3.3, when workers are required to identify the errors in phrase-wise, Find task becomes more difficulty than identifying the errors in sentence-wise. In the experiment, Find task with low difficulty requires workers to identify the errors in sentence-wise. When they find an error, they simply identify it as

Line No.

By contrast, Find task with high difficulty requires workers to identify the errors in phrase-wise, as

Line No., *incorrect word / phrase*.

For both groups, Fix task and Verify task are the same as described in experiment 1. Other information is summarized in Table 3.5.

3.7 Empirical Results

3.7.1 Experiment 1

Under horizontal task decomposition situation, 32 different workers provided 32 proofreading results for six articles. Upon completion of a work

Table 3.5: Information for experiment 2.

Experiment	Workflow	Articles	Payment ($) (Bonus*)	Duration (min)	Qualifications Required
Vertical Task Decomposition	**Find-Fix-Verify:** Granularity Levels: *Phrase-wise* Number of Corrections: 1 Voting Methods: Vote / Veto	Article 1~6	Find: 0.5 (0.1)	20	None
			Fix: 0.5 (0.1)	20	
			Verify: 0.5 (0.1)	20	
	Find-Fix-Verify: Granularity Levels: *Sentence-wise* Number of Corrections: 1 Voting Methods: Vote / Veto	Article 1~6	Find: 0.5 (0.1)	20	
			Fix: 0.5 (0.1)	20	
			Verify: 0.5 (0.1)	20	

*When correctness ≥ 90%, bonus is awarded.

item, the task requester can review the submitted results and approve or reject the payment on the work item. 8 workers out of the above 32 are rejected due to the poor quality (i.e., the correct rate is less than 10%). The remaining 24 proofreading results are accepted, i.e., four proofreading results for each article. Worker response was extremely fast, with 87.5% of the proofreading results received in the one hour after the task was posted, and the remaining ones received in the next one hour.

In Fig. 3.6, we illustrate statistics for proofreading results under horizontal task decomposition strategy, including the average number of the submitted edits, the average number of correct corrections from the submitted edits, and the average number of new errors that introduced during the proofreading. The very high numbers of the submitted edits for each article indicate that all the workers gave fairly good effort to proofreading the articles (with an average number equaling to 8.54). Those submitted edits cover well on the true errors in the articles, with an average correct rate equaling to 43.8% (i.e., averagely, 4.38 corrections for each article). Nevertheless, a substantial number of new errors are also introduced. Specifically, the average number of new errors that introduced into each article is 3.83, which is close to the average number of the corrections. Due to the new introduced errors, although horizontal task decomposition strategy performs well at correcting errors, the *final correct rate*[2] is bad, which equals

[2] $final\ correct\ rate = \frac{number\ of\ corrections}{number\ of\ original\ error + number\ of\ new\ errors} \times 100\%$

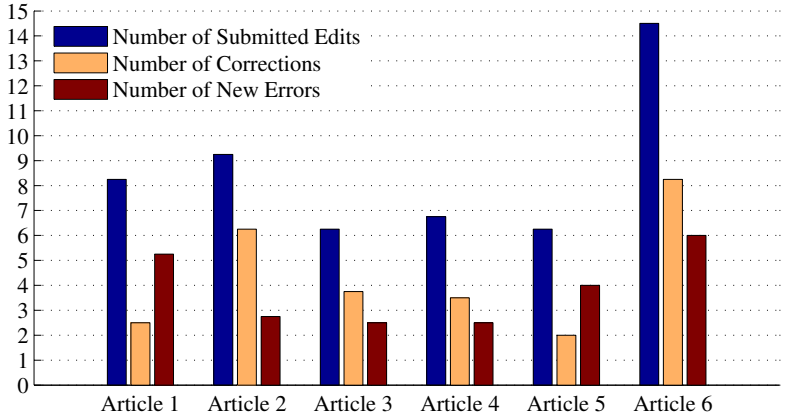


Figure 3.6: Results of proofreading experiment under horizontal task decomposition strategy.

to 31.5% in our experiments.

Under vertical task decomposition strategy, we execute the proofreading task for each article as Find-Fix-Verify procedure described in experimental design in Section 3.6.1. It is worth noting that, for each Verify task, three workers were employed to examine the submitted edits independently. Specifically, one edit wins a vote from a worker only if the worker judges it as correct. Moreover, when one edit wins at least two votes, it is kept as one correction[3]. 68 work items, including all the Find, Fix, and Verify tasks, are executed. Similarly, we rejected 24 of them due to the poor qualities (i.e., the correct rate is less than 10%) and duplicating submissions by the same worker. Finally, same as in horizontal task decomposition experiment, each article received four proofreading results.

In Fig. 3.7, we illustrate statistics for proofreading results under vertical task decomposition strategy, including the average number of the submitted edits which are proposed by the Fix task, the average number of correct corrections from the submitted edits, and the average number of new errors that introduced during the proofreading. In contrast to the horizontal

[3]Majority rule is applied.

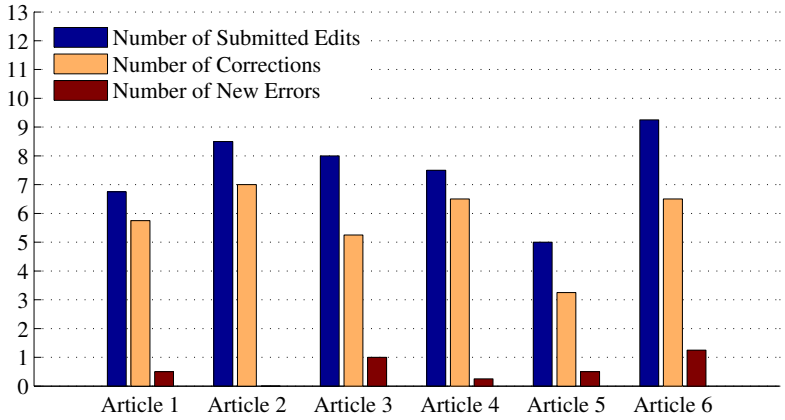


Figure 3.7: Results of proofreading experiment under vertical task decomposition strategy where errors are identified in phrase-wise.

task decomposition situation, the number of submitted edits in vertical situation is smaller (with an average number equaling to 7.50). However, these submitted edits cover better on the true errors, with an average correct rate equaling to 57.1% (i.e., averagely, 5.71 corrections for each article). Moreover, comparing to a substantial number of new errors (3.38), the average number of new errors that introduced into each article by Fix task equals to 0.92. Furthermore, the examination done by workers for Verify tasks leads to a further decrease to the number of new errors. Averagely, only 0.58 new error is introduced into each article. The final correct rate of proofreading task achieved under vertical task decomposition strategy is 54.1%.

Putting the above observations together leads us to the conclusion that vertical task decomposition strategy outperforms horizontal task decomposition strategy in terms of final quality, which is consistent with the conclusion we obtained in Chapter 3.5.2.

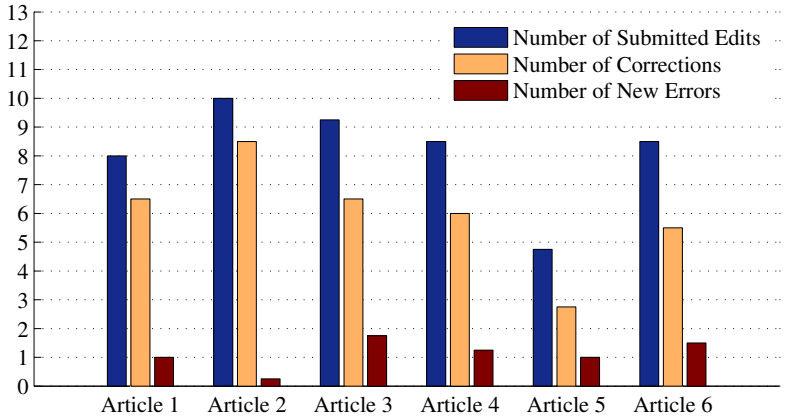


Figure 3.8: Results of proofreading experiment under vertical task decomposition strategy where errors are identified in sentence-wise.

3.7.2 Experiment 2

Aiming at verifying and revising the instructions on vertical task decomposition strategy, in this experiment, we conduct the proofreading task starting with sentence-wise Find task, where the workers are required to identify the lines that have errors. In contrast, sentence-wise Find task has lower difficulty than phrase-wise Find task as we do in experiment 1 where workers are required to identify the incorrect word or phrase. Same as experiment 1, for each Verify task, three workers were employed to examine and vote on the submitted edits independently. Majority rule is applied to determine whether the edit can be kept as a correction. 58 work items, including all the Find, Fix, and Verify tasks, are executed. Similarly, we rejected 16 of them due to the poor qualities (i.e., the correct rate is less than 10%) and duplicating submissions by the same worker. Finally, in this experiment, each article received four proofreading results.

In Fig. 3.8, we illustrate statistics for proofreading results under vertical task decomposition strategy, including the average number of the submitted edits which are proposed by the Fix task, the average number of correct corrections from the submitted edits, and the average number of new errors

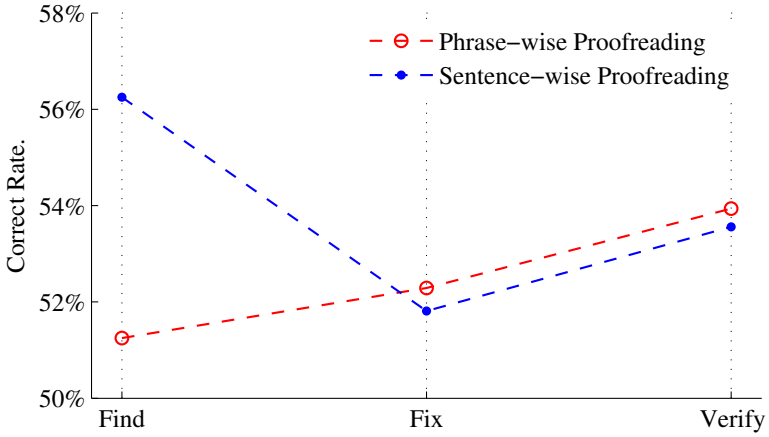


Figure 3.9: Correct rate comparison between phrase-wise and sentence-wise proofreading.

that introduced during the proofreading. In Fix task, the number of submitted edits is between those of phrase-wise proofreading and horizontal proofreading (with an average number equaling to 8.17). Although, these submitted edits cover better on the true errors comparing to those in phrase-wise proofreading, with an average correct rate equaling to 59.6% (i.e., averagely, 5.96 corrections for each article), the average number of new errors that introduced into each article by Fix task is larger than that in phrase-wise proofreading, which equals to 1.13. Finally, after the examination done by workers for Verify tasks, the final correct rate of sentence-wise proofreading task achieved under vertical task decomposition strategy is 53.6%, which is smaller than that of phrase-wise proofreading (54.1%).

3.8 Summary

In this study we have formally presented and analyzed vertical and horizontal task decomposition models which respectively specify the relationship between subtask quality and the worker's effort level in the presence of

positive and none dependence among subtasks. Our focus is on addressing the efficiency of task decomposition when the workers are paid equally and contribution-based. We conducted simulations to analyze and compare the efficiency of two task decompositions, aiming at generating explicit instructions on strategies for optimal task decomposition. We conclude that when the final quality is defined as the sum of the qualities of solutions to all decomposed subtasks, vertical task decomposition strategy outperforms the horizontal one in improving the quality of the final solution, and furthermore give explicit instructions the optimal strategy under vertical task decomposition situation from the task requester's point of view.

Furthermore, based on the previous studies and results, we design and conduct proofreading experiments on Amazon Mechanical Turk. The first series of experiments compare the efficacy of vertical decomposition and horizontal decomposition on proofreading task. The second series of experiments focus on the vertical task decomposition, and compare the situations where the first subtasks with different difficulties. Our experiments ran for two months, and was highly visible on the MTurk crowdsourcing platform. According to the results we collect, we 1) verify the superiority of the vertical task decomposition strategy over the horizontal one on proofreading task; and 2) apply some of the our proposed instructions on vertically decomposing the proofreading task, which show promise on improving the quality of final outcome.

3.8.1 Discussion on Two Revenue Sharing Schemes

Nonetheless, we have noted that although group-based and contribution-based revenue sharing schemes have received most of attention, they are merely used as reward distribution method in traditional companies, where the staff can get reward only after the final solution is worked out. This is not the case in crowdsourcing marketplaces, where rewards for tasks are previously determined and claimed by the task requesters, and workers get reward as soon as they accomplish their tasks and the solutions are qualified. However, task requesters can partially simulate these revenue sharing

schemes by offering bonus rewards or negative rewards [Chen et al., 2011]. Formal studies on optimal task decomposition taking more parameters into consideration, such as the number of subtasks, and the experimental studies that utilize MTurk could be potential topics for future research.

Definition 3.6. *Shirking* is said to occur when the worker's effort level falls short of that required for employer's utility maximization, i.e., $e_i^E < e_i^O$, where e_i^E is the worker's optimal effort level for his utility maximization, and e_i^O represents the optimal effort level from the employer's standpoint.

Proposition 3.7. *The group-based revenue sharing leads to shirking on the part of workers in both horizontal task decomposition and vertical task decomposition.*

Proof. For each referrer, the first-order condition for utility maximization is given by

$$\frac{1}{N} f_{e_i} - c_{e_i} = 0.$$

For the marketer, the first-order condition for utility maximization is given by

$$f_{e_i} - c_{e_i} = 0.$$

Let e_i^E and e_i^O denote the referrer's and the marketer's optimal effort levels respectively for referrer i. Therefore,

$$\frac{1}{N} f_{e_i}(e_i^E) - c_{e_i}(e_i^E) = f_{e_i}(e_i^O) - c_{e_i}(e_i^O) = 0.$$

Suppose $e_i^O \leq e_i^E$. Then $c_{e_i}(e_i^O) \leq c_{e_i}(e_i^E)$. Therefore,

$$f_{e_i}(e_i^O) \leq \frac{1}{N} f_{e_i}(e_i^E) < f_{e_i}(e_i^E),\ for\ N > 1.$$

However,

$$f_{e_i}(e_i^O) \geq f_{e_i}(e_i^E),\ for\ e_i^O < e_i^E.$$

which leads to a contradiction. Therefore, $e_i^O > e_i^E$. □

Proposition 3.8. *The contribution-based revenue sharing leads to shirking on the part of workers in both horizontal task decomposition and vertical task decomposition.*

Proof. For each referrer, the first-order condition for utility maximization is given by

$$f_{e_i e_i} \cdot e_i + f_{e_i} - c_{e_i} = 0.$$

For the marketer, the first-order condition for utility maximization is given by

$$f_{e_i} - c_{e_i} = 0.$$

Let e_i^E and e_i^O denote the referrer's and the marketer's optimal effort levels respectively for referrer i. Therefore,

$$f_{e_i e_i}(e_i^E) \cdot e_i^E + f_{e_i}(e_i^E) - c_{e_i}(e_i^E) = f_{e_i}(e_i^O) - c_{e_i}(e_i^O)$$

Suppose $e_i^O \leq e_i^E$. Then $f_{e_i}(e_i^O) \geq f_{e_i}(e_i^E)$. Therefore,

$$c_{e_i}(e_i^O) \geq c_{e_i}(e_i^E) - f_{e_i e_i}(e_i^E) \cdot e_i^E$$

$$\Rightarrow \quad c_{e_i}(e_i^O) > c_{e_i}(e_i^E).$$

However,

$$c_{e_i}(e_i^O) \leq c_{e_i}(e_i^E), \ for \ e_i^O \leq e_i^E,$$

which leads to a contradiction. Therefore, $e_i^E < e_i^O$. □

According to the analysis, shirking problem exists in both group-based revenue sharing scheme and contribution-based revenue sharing scheme, which means that by dividing rewards equally or according to the marginal contribution among the workers, the individual effort level always falls short of that required for maximum employer payoff. It implies that neither of the above two revenue sharing schemes is effective for encouraging the workers to exert high level of efforts from the employer's point of view.

Chapter 4

MDP-Based Reward Design for Efficient Cooperative Problem Solving

4.1 Introduction

Crowdsourcing is now admired as one of the most lucrative paradigm of leveraging collective intelligence to carry out a wide variety of tasks with various complexity. When a complex task is beyond the capability of unskilled individuals (crowds) on their own to deal with, task requester (or *principal*) *decomposes* tasks into smaller pieces of subtasks, which become low in complexity, and can be completed by individual workers (or *agents*). After the task decomposition, task requesters then present the subtasks to the workers with associated rewards through crowdsourcing websites, such as Amazon Mechanical Turk (MTurk) [Ipeirotis, 2010], seeking to maximize the quality of the task's final solution.

On the other hand, the recruited workers are organized to perform the subtasks cooperatively, which makes the subtask's quality depends on efforts of multiple workers who jointly produce the output. Particularly, in crowdsourcing, those individuals usually cooperate with each other to gen-

erate a portion of a satisfactory solution in *sequential* workflows. A typical example is Soylent [Bernstein et al., 2010], which first contributes a three-phase workflow, Find-Fix-Verify for crowdsourcing-based text editing task. Specifically, take proofreading for example, in Find phase, workers take an article with spelling and grammar errors as input and try to identify the patches that may need corrections. Workers in Fix phase, take the patches as input and provide alternative corrections. Finally, in Verify phase, workers accept or reject the corrections to improve the quality of proofreading results. By this way, these Find, Fix and Verify subtasks are chained together into positively and sequentially interdependent subtask units. The striking characteristic of this solving process is that each subtask takes the output form the previous subtask as input, which makes the *sequential cooperation* assumption that the following workers can do better based on quality solution provided by previous workers hold (see [Kulkarni et al., 2011] [Kulkarni et al., 2012]). It is worth noting that the rewards for each subtask is required to be announced before subtask execution and paid to the worker who performs it once the subtask is finished.

A significant challenge in the above-mentioned crowdsourcing-based complex task solving has been designing an incentive scheme for the task requester to maximize expected output quality subject to the self-interested individual worker's utility maximization, taking into account aspects of sequential quality dependence among subtasks. This is particularly difficult when task requesters face workers with unknown, heterogeneous abilities and have a limited budget to spend on the whole complex task. Given these challenges, existing approaches typically provide task requesters with simple rules of thumb for budgeting the tasks by looking at the historical data of the crowdsourcing marketplaces. By comparing the nature of the tasks, the historical activity levels of the workers and the qualities of outcomes, task requesters struggle to distribute appropriate rewards among their tasks, to hope for the best possible task solution [Faradani et al., 2011] [Ipeirotis, 2010]. However, without formal models, improper budget allocation commonly leads to task starvation or inefficient use of capital.

Against this background, this study addresses the challenge of allocat-

ing limited budget to multiple subtasks which are sequentially interdependent with each other. We first formalize a general model of such crowdsourcing subtasks by specifying the relationship between subtask quality and the worker's effort level, and illustrate sequential cooperation by formalizing how sequential subtasks explicitly depend on each other. Then we pose the problem of maximizing the quality of the final solution with limited budget. The key challenge in this problem is to allocate the limited budget among the sequential subtasks. This is particularly difficult in crowdsourcing marketplaces where the task requester posts subtasks with their payments sequentially and pays the predetermined payment to the worker once the subtask is finished. Markov Decision Processes (MDPs) are capable of capturing the task-solving process's uncertainty about the real abilities of a succession of workers, that is, the uncertainty about the qualities of the sequential subtasks. To the best of our knowledge, our study is the first attempt to apply MDPs to payment allocation for sequentially dependent subtasks in crowdsourcing. By modeling this problem as MDPs, we show the value of MDPs for optimizing the quality of the final solution by dynamically determining the budget allocation on sequential and dependent subtasks under the budget constrains. Our simulation-based approach verifies that comparing to some benchmark approaches, our MDP-based payment planning is more efficient on optimizing the final quality under the same budget constraints.

The rest of this chapter is organized as follows. Section 4.2 summaries the related work, and Section 4.3 describes our model and research problem. We detail in Section 4.4 and Section 4.5 how decision-theoretic techniques can be applied to define and solve our problem. Section 4.6 shows the superiority of our MDP-based approach for payment planning over some benchmark approaches. Finally, we summary our conclusions and propose future work in Section 4.7.

4.2 Related Work

Workers in crowdsourcing marketplaces are motivated by a variety of incentives. In this work, we particularly focus on utilizing the financial strategies as the incentive to improve the quality of crowd work. Work by [Ho and Vaughan, 2012] first formalized the *online task assignment problem*, which addresses the challenge that a single task requester faces when assigning heterogeneous tasks to workers with unknown, heterogeneous abilities, taking into account aspects of task quality and budget constraints. Like other works (see [Azaria et al., 2012] [Karger et al., 2014] [Tran-Thanh et al., 2014b] [Tran-Thanh et al., 2013]), it deal with the independent subtasks, and does not take into account the sequential quality dependence among subtasks. BudgetFix [Tran-Thanh et al., 2014a] is the first work which formalizes the Find-Fix-Verify process, and propose the algorithm that guarantees the quality of the task at a minimal cost by determining the number of interdependent subtasks and the price to pay for each subtask given budget constraints. However, it uses a non-adaptive scheme which makes all payment assignments before any worker arrives, and thus, is not suitable for our settings where the payment on each subtask is determined dynamically during the process of the task solving. Decision-theoretic techniques shows the value on dynamic decision in crowdsourcing. Work by [Dai et al., 2013] designs AI agents that use Partially-Observable Markov Decision Process (POMDPs) for optimizing workflows used in crowdsourcing and obtains excellent cost-quality tradeoffs. However, like TurKit, it explores iterative workflows for complex tasks solving, which is also not suitable for our settings.

4.3 The Model and Problem Statement

At a broad level, the task requester first decomposes the complex task into smaller subtasks. These subtasks are supposed to be positively dependent with each other in a sequential way. Specifically, it requires a sequential

task solving process where one worker's output is used as the starting point for the following worker. Furthermore, any increase in the efforts from the previous workers also leads to an improvement on the quality of the current subtask. These subtasks are then presented to the workers with predetermined payments through crowdsourcing websites, seeking to maximize the quality of the task's solution (under a certain limited budget), which is positively related to the efforts devoted by all the workers. The workers perform the subtasks sequentially, and each of them is self-interested and devotes an amount of effort to her own subtask for individual utility maximization.

4.3.1 Player-Specific Ability

One striking feature of most crowdsourcing marketplaces is that workers who are available to the task can come from a diverse pool, and they are free to choose to complete the tasks which interest them. The workers can come from a diverse pool including genuine experts, novices, and at the extreme, "spammers", who submit arbitrary answers independent of the question in order to collect their fee. The task requesters do not have control over the ability of the workers. As in MTurk, workers remain anonymous to task requesters, and all payment occurs through MTurk platform. However, since task requesters are able to accept submitted solutions or reject those which does not meet their standards, workers on MTurk are only paid if a task requester accepts their work. Based on this regulation, we assume that each worker is diligent and self-interested. That is, the malicious behavior of worker's task execution is outside of the scope of this research, and under a given reward system, each worker selects an effort level (at a cost that depends on her ability) such that her utility is maximized.

Associated with each worker i from the huge pool of potential labor force is a vector of *abilities* $\vec{v_i} = (v_{i1}, v_{i2}, \cdots, v_{iN},)$, where v_{in} ($n \leq N$, and N is the number of the decomposed subtasks), represents worker i's ability to do subtask n. We suppose that the ability vector for each worker is drawn from a continuous joint probability distribution $(0,1]^N$, 1 represents the highest ability. The ability vectors for different workers are drawn

independently from each other, and the distribution is known to the task requester (or *principal*), but the ability vector $\vec{v}_i$ is known only to worker i himself. For simplicity, we shall assume in this research that each worker is endowed with the ability that applies across all subtasks. More concretely, for each worker i, the skill vector $\vec{v}_i$ is simplified as $(v_i, v_i, \cdots, v_i)$, where v_i is drawn from a continuous distribution $F(v_i)$ identically and independently of the abilities of other workers over $(0, 1]$.

In order to solve the subtasks, workers contribute their efforts, such as time and resources, to their subtasks respectively. The amount of effort exerted by worker i to subtask i is characterized by e_i. It is the worker's problem to decide the optimal amount of effort she exerts in order to maximize her utility. After exerting e_i units of effort, and thereby incurring the cost of $C(e_i)$, worker i receives a payment, which is positively related to the amount of her effort, denoted as m_i.

Definition 4.1 (Worker's utility function). For subtask i, the worker's utility, $u(e_i)$, is the individual reward m_i minus her cost, i.e.

$$u(e_i) = m_i - C(e_i), \tag{4.1}$$

where $C(e_i) > 0$ is the total cost of effort e_i.

In our setting, a worker's ability v_i can be interpreted as the amount of effort she is able to exert per unit cost (denoted as $cost_i$) [DiPalantino and Vojnovic, 2009]. Therefore, the utility of worker i can be expressed by

$$\begin{aligned} u(e_i) &= m_i - e_i \cdot cost_i \\ &= m_i - \frac{e_i}{v_i}. \end{aligned} \tag{4.2}$$

In our model, all workers are supposed to be diligent, so they solve their own subtasks to the best of their efforts e^*, where

$$e_i^*(v_i) = m_i \cdot v_i. \tag{4.3}$$

Sequential payment decisions introduce new subtlety into the payment planning problem. That is, each payment decision now have expected value instead of certain value because of the uncertainty about the future workers' abilities.

4.3.2 Value of Accomplishing A New Subtask

We are now focus on the sequential task solving process where the subtasks are carried out sequentially one at a time. Consider the situation where the original task is vertically decomposed into N $(N > 1)$ sequential subtask units. N workers contribute their efforts, such as time and resources, to N subtasks respectively. The amount of effort exerted by worker i is characterized by $e_i \in [0,1]$. As we discussed in Chapter 3, the quality of the final solution can be viewed as the *cumulative qualities* obtained from all subtasks. Furthermore, without loss of generality, we suppose that the benefit of the task requester from the task solution and the quality of the final solution are numerically equal. Formally, the benefit function of task requester is defined as follows.

Definition 4.2 (Benefit function of the task requester)**.** For the the task requester, the benefit (U) from the final task solution is the cumulative qualities obtained from all the N decomposed subtasks.

$$U(\mathbf{e}_N) = \sum_{i=1}^{N} f^i(e_i) \tag{4.4}$$

where $\mathbf{e}_N = (e_1, \cdots, e_N)$ is the effort vector consists of the efforts exerted by all workers on their own subtasks, and $f^i(e_i)$ is the quality function of subtask i.

Remark 4.1. It is worthwhile to note that since all the subtasks are essential subdivisions of the original complex task, none of them is neither replaceable nor dispensable. Hence, it is reasonable to assume that, only if all the subtasks are accomplished, is the original complex task regarded as accom-

plished and the benefit of the task requester becomes positive. Formally,

$$U(\mathbf{e}_{N'}) = \sum_{i=1}^{N'} f^i(e_i) = 0, \ \ \forall \ \ N' < N. \tag{4.5}$$

where N' are integers representing for any subtask except for the last one.

Quality functions of the subtasks are thoroughly discussed in Section 3.3. Attempting to make each chapter self-contained insofar as possible, in the following, we summarize the striking properties of subtask quality functions, and provide necessary explanations in an intuitive way.

- ***Increasing and concave.*** The quality of subtask i is positively related to the amount of effort (e_i) that exerted by the corresponding worker i, and since the time and resources processed by individuals are limited, the marginal contribution ($\partial f^i / \partial e_i$) on the subtask quality of each exerted effort unit is decreasing [Holmstrom and Milgrom, 1991]. These endow the subtask quality functions with two basic properties, as increasing and concave, i.e., the subtask's quality increases with the worker's effort at a decreasing rate (see Eq. (3.3)).

- ***Positive quality dependence.*** The positive dependence of subtask qualities exhibits in two aspects. 1) Since each subtask takes the output from the previous subtask as input, the quality of subtask i's solution depends not only on the effort exerted by worker i, but also on the quality of previous subtask's solution. Specifically, any increase in the efforts from the previous workers also leads to an improvement on the quality of the current subtask (see Eq. (3.2)). 2) For each subtask, it has an intrinsic difficulty. High difficulty indicates low marginal contribution based on the same level of effort. The difficulty of the current subtask decreases as the efforts on previous subtasks increase (see Eq. (3.5)).

In this research, we apply the *Cobb-Douglas production function* as the form of our subtask quality function. Cobb-Douglas production function is

a widely used form for studying important topic for researchers to calculate the contribution rate of various influential factors to the final efficiency growth, such as the economic growth under a Cobb-Douglas production function model [Cheng and Han, 2013]. The general form of Cobb-Douglas production function is expressed as

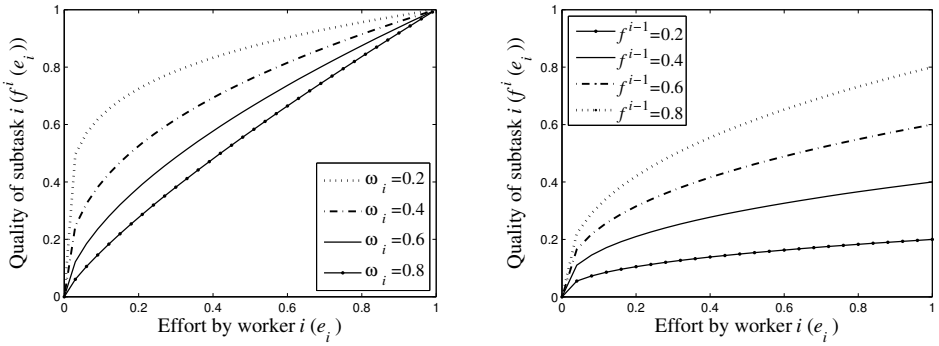
$$Y = AX_1^{\beta_1} X_2^{\beta_2} \cdots X_N^{\beta_N},$$

where X_i ($i = 1, 2, \cdots, N$) denotes the input of the ith factor and Y denotes the output; β_i ($i = 1, 2, \cdots, N$) is the output elasticity of the factor X_i, and A denotes the level of the technical progress.

We now map all the elements in the Cobb-Douglas production function into the sequential task solving process to cast light upon the reasonability of choosing the Cobb-Douglas production function as the form of our subtask quality function. First of all, the input of the ith factor X_i is the effort exerted by worker i on ith subtask. For example, for subtask k ($k = 1, 2, \cdots, N$), the quality function in Cobb-Douglas form is $f^k = Ae_1^{\beta_1} e_2^{\beta_2} \cdots e_k^{\beta_k}$. This can be equivalently rewritten in a recursive way as $f^k = f^{k-1} \cdot e_k^{\beta_k}$, which indicates the positive quality dependence among the sequential subtasks. Secondly, the output elasticity of the factor X_i can be represented by the intrinsic difficulty of subtask i which is decided at the task decomposition stage and denoted as ω_i for subtask i. Once we assume normalization and constraints ($\omega_i \in (0, 1)$ and $\sum_{i=1}^{N} \omega_i = 1$), the implementation of the Cobb-Douglas production function, i.e., for subtask k, $f^k = Ae_1^{\omega_1} e_2^{\omega_2} \cdots e_k^{\omega_k}$, satisfies the increasing and concave property. And finally, since the difficulty of the first subtask cannot be modified at the very beginning, and for analysis simplicity, we set the value of A equals to 1. Formally, we define the subtask quality function in the following form.

Definition 4.3. For subtask i, the quality function (f^i) is

$$f^i(e_1, \cdots, e_i) = \prod_{k=1}^{i} e_k^{\omega_k} \tag{4.6}$$



(a) Subtask difficulties (b) Positive quality dependence

Figure 4.1: (a) illustrates the subtask qualities given the same ability of workers under various subtask difficulties. With a fixed effort, the lower the difficulty value (ω_i), the higher the quality of the subtask ($f^i(e_i)$). (b) illustrates the positive dependence among subtask qualities. With a fixed effort, higher quality of the previous subtask (f^{i-1}) enables the successive subtask to achieve higher quality (f^i).

where $\omega_i \in (0,1)$ is the difficulty of subtask i, and $\sum_{i=1}^{N} \omega_i = 1$.

Remark 4.2. Eq. (4.6) satisfies the requirements in Definition 3.2, where high value of ω_i corresponds to high difficulty and indicates low marginal contribution based on the same amount of effort. Fig. 4.1(a) illustrates the qualities of subtasks with various subtask difficulties. For a fixed level of effort, lower difficulty (i.e., lower value of ω_i) leads to higher quality. Furthermore, Fig. 4.1(b) gives the illustration on the positive quality dependence among two successive subtasks, when subtask $i-1$ comes up with a higher-quality solution, the difficulty of subtask i becomes lower. It is worth noting that the constraints on the difficulty coefficient indicate that any subtask is neither replaceable (i.e., $\omega > 0$) nor dispensable (i.e., $\omega < 1$), and only if all the subtasks are accomplished, is the original task regarded as accomplished.

We now formalize the problem addressed in this study. The task requester, who decomposes a complex task into multiple small dependent subtasks and post them on Crowdsourcing system sequentially, can be viewed as a principal agent. The requester has a budget $M = \sum_{i=1}^{N} m_i > 0$, and a benefit function (Eq. (4.4)). Due to the low complexity, the small subtasks can be performed by individual workers. As we assumed in the above model, each subtask performed by a worker has a value, which equals to the subtask quality, thus the requester's objective is to maximize the final quality of the complex task subject to a fixed budget constraint. More formally, the requester's utility maximization problem can be represented as the following mathematical programming formulation:

$$\begin{cases} \max U(e_1, e_2, \cdots, e_N) \text{ where } e_i = m_i \cdot v_i, \\ \text{s.t.} \\ \quad m_1 + m_2 + \cdots + m_N = M \\ \quad m_i > 0, \;\; i = 1, \cdots, N \end{cases} \tag{4.7}$$

4.4 MDP-based Payment Planning

4.4.1 Markov Decision Processes

We are interested in online crowdsourcing settings, where the task requesters are required to post tasks along with specified payments based on a expected quality of the task solution. Individual workers can then elect to perform any of these tasks, and receive the corresponding payments as soon as they successfully complete the tasks. Due to the uncertainty of workers' abilities, the task requester confronts the problem of deciding the payment for the task / subtask before she observes both the ability of the worker and the quality of the task solution. Moreover, in a budget-constrained sequential subtask solving process, this uncertainty brings even severe difficulty in payment allocation planning.

The sequential payment decision problem induces a Markov Decision

Process (MDP), which is able to represent the uncertainty about the real abilities of a succession of workers through the sequential task solving process, and when solved can be used to implement the efficient payment allocation to guarantee the quality of the final solution to the original task.

Definition 4.4 ([Puterman, 2009])**.** An MDP is defined as a four-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where

- $\mathcal{S}$ is a finite set of discrete *states*.
- $\mathcal{A}$ is a finite set of all *actions*.
- $\mathcal{P}$: $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ (*Transition probabilities*):
  For each state-action pair (s,a), a next-state distribution $P_a(s'|s)$ that specifies the probability of transition to each state s' upon execution of action a from state s.
- $\mathcal{R}$: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ (*Reward distributions*):
  For each state-action pair (s,a), a distribution $R(s,a)$ on real-valued *reward* for taking action a in state s.

Markov Decision Process is commonly used to formulate full-observable decision making under uncertainty problems [Kolobov, 2012]. Putting incentives to one side for now, the MDP execution is described as follows. An agent executes its action in discrete steps starting from some initial state. At each step, the system is at one distinct state $s \in \mathcal{S}$. The agent can execute any action a from a set of action $\mathcal{A}$, and receives a reward $R(s,a)$. The action takes the system to a new state s' stochastically. The transition process exhibits the *Markov property*, i.e., the new state s' is independent of all previous states given the current state s. The transition probability is defined by $P_a(s'|s)$. The model assumes *full observability*, i.e., after executing an action and transitioning stochastically to a next state as governed by $\mathcal{P}$, the agent has full knowledge of the state.

4.4.2 Payment Planning in MDPs

In our setup during the sequential task solving, the agent's (i.e., the task requester's) payment planning problem is defined as follows. As input the agent is given a set of subtasks, and the agent is asked to return a final solution with the highest quality. In pursuit of this goal, the agent is allowed to determines payments for each subtasks within the limited budget. We now model the agent's sequential payment decision problem as a MDP. To do this, we first define a generative model for the subtask execution by crowdsourcing workers of various uncontrolled abilities.

We assume workers are self-interested, so they exert their efforts on their own subtasks to the best of their abilities to maximize their individual utilities. The quality of each subtask increases as the effort gets larger. Moreover, the quality of the current subtask increases as the quality of the previous subtask's solution becomes higher. Specifically, according to Eq. (4.6), given the previous subtask's solution with quality f^{i-1}, the quality of current subtask i with difficulty ω_i is governed by the following probabilistic equation:

$$P_{m_i}(f^i \mid f^{i-1}) = P_{m_i}(f^i = f^{i-1} \cdot {e_i}^{\omega_i}) \tag{4.8}$$

Furthermore, when subtask i is performed by worker i with ability v_i and a predetermined payment m_i is provided, the above transition probability function (Eq. (4.8)) is specified according to the optimal effort of worker i that given by Eq. (4.3) as

$$\begin{aligned} P_{m_i}(f^i \mid f^{i-1}) &= P_{m_i}(f^i = f^{i-1} \cdot (m_i v_i)^{\omega_i}) \\ &= F(v_i) \end{aligned} \tag{4.9}$$

where $F(v_i)$ is the distribution function from which v_i drawn independently of the abilities of other workers.

We now formally define our sequential payment allocation decision problem as planning problem in MDPs.

Definition 4.5. The MDP for our sequential payment allocation problem

with a limited budget M is a four-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where

- $\mathcal{S} = \{(i, U(\mathbf{e}_{i-1}), M_i)\}$. These three elements involving in the state are
  1. $i \in 1, \cdots, N$, is the subtask which is on turn to be executed by worker i;
  2. $U(\mathbf{e}_{i-1}) = \sum_{k=1}^{i-1} f^k(e_i)$, is the quality of the partially completed task, i.e., the accumulative quality of subtask 1 to $i-1$;
  3. $M_i = M - \sum_{k=1}^{i-1} m_k$, represents the remaining budget available for the task requester for the rest of subtasks.
- $\mathcal{A} = \{\text{set payment } m_i \mid m_i \in (0, M_i]\}$, m_i is the accessible payment for performing subtask i.
- $\mathcal{P} = \{P_{m_i}(f^i \mid f^{i-1}) = F(v_i)\}$.
- $\mathcal{R} = \{f^i \mid f^i > 0\}$ f^i is the quality of subtask i.

4.4.3 Continuous States and Discretization

As we see in Definition 4.5, the MDP for online payment planning problem leads unavoidably to a large state space, due to the following two intrinsic reasons. The first is the continuous action space, since the requester can assign an arbitrary amount of payment ($m_i \leq M$) to subtask i. The second is the continuous action-state space. In the payment planning settings, by assigning payment m_i is incapable of guaranteeing the quality of the subtask f^i, since it also depends on the worker's ability v_i according to Eq. (4.6) and Eq. (4.3), which is drawn from a continuous probability distribution $F(v_i)$ over $(0, 1]$. The infinite number of actions and possible abilities of workers makes $\mathcal{S} = \mathbb{R}^{2N}$ an infinite set of states.

In order to solve the continuous-state MDP, we first discretize both action space and state space. By discretizing the budget into $M(> N)$ units, the payment for each subtask is restricted to integer set $\{1, \cdots, M\}$. Furthermore, we maintain the assumption that the values of abilities are independently distributed from $F(v)$, but divide the them into $L(> 1)$ ability levels. We can then approximate the continuous-space MDP via a discrete-state one $\left\langle \bar{\mathcal{S}}, \bar{\mathcal{A}}, \{P_{\bar{s}\bar{a}}\}, \mathcal{R} \right\rangle$, where $\bar{\mathcal{S}} = (M \times L)^N$ is the set of discrete states,

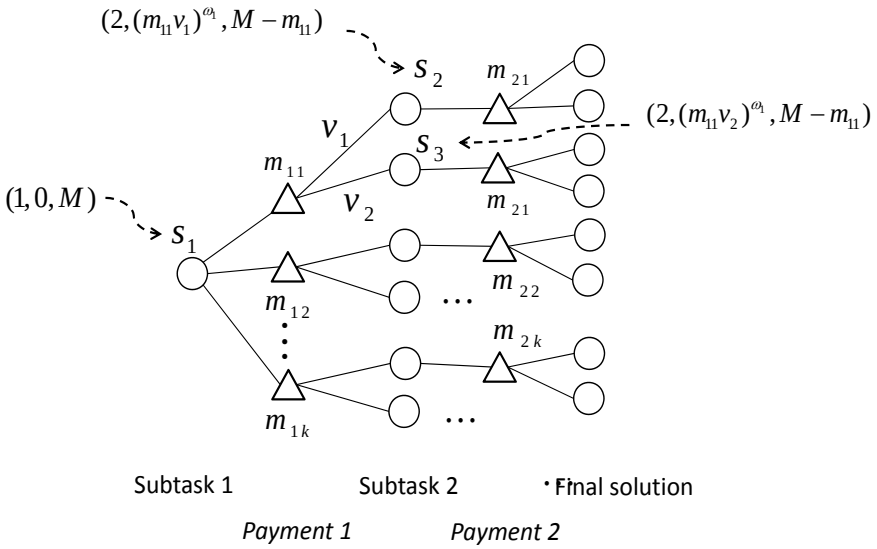


Figure 4.2: Decision tree representation of MDP-based payment planning with discretized state space for a two-subtask solving process.

$\{P_{\bar{s}\bar{a}}\}$ are our state transition probabilities over the discrete states and actions. When our actual payment planning is in some continuous-valued state and we need to decide a payment for the subtask, we compute the corresponding discretized state $\bar{s}$, and execute action $\pi^*(\bar{s})$ according to Eq. (4.14).

In Fig. 4.2 we visualize the MDP-based payment planning model with discretized state space for a two-subtask solving process (i.e., $N = 2$). It is a two-step horizon decision tree. From the starting point s_1 there are $k(< M)$ levels of payments, $m_{11}, \cdots, m_{1k}$. The payment chosen gives probabilities of moving to the next states. The rewards for moving to the next states depend on the abilities of the workers, which are discretized into two levels (i.e., $L = 2$). Notice that, since there are only two subtasks, for subtask 2, the payments access only one choice m_{2k} and each choice is constrained by $m_{1i} + m_{2i} = M$, where $i = 1, \cdots, k$.

4.5 Planning Algorithm

4.5.1 Solving an MDP

Methods for solving an MDP include value iteration [Bellman, 1956], policy iteration, and linear programming [Bertsekas et al., 1995] [Puterman, 2009]. Any solution to an MDP problem is in the form of a *policy*.

Definition 4.6. ([Dai et al., 2013]) A policy, $\pi : \mathcal{S} \to \mathcal{A}$ of an MDP is a mapping from the stage space to the action space.

A policy is *static* if the action taken at each state is the same at every time step. $\pi(s)$ indicates which action to execute when the system is at state s. Solving the planning problem in MDPs consists in computing an *optimal policy* (π^*: $\mathcal{S} \to \mathcal{A}$), which is a probabilistic execution plan that associates with any state $s \in \mathcal{S}$ an *optimal action* $\pi^*(s)$ and achieves the maximum expected sum of rewards. The policy π is evaluated by its *value function*:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} P_{\pi(s)}(s'|s) V^{\pi}(s'). \tag{4.10}$$

where $\gamma \in [0,1)$ called the *discount factor*, which makes the model discounted MDPs. Any optimal policy's value function satisfies the following *Bellman equations*:

$$V^*(s) = \max_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s'|s) V^*(s') \right]. \tag{4.11}$$

The corresponding optimal policy can be extracted from the Bellman equation:

$$\pi^*(s) = \operatorname{argmax}_{a \in \mathcal{A}} \left[R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s'|s) V^*(s') \right]. \tag{4.12}$$

Given an implicit optimal policy π^* in the form of its optimal value

function $V^*(s)$, we measure the superiority of an action by a *Q-function*: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Definition 4.7. ([Dai et al., 2013]) The Q^*-*value* of a state-action pair (s,a) is the value of state s, if an immediate action a is performed, followed by π^* afterwards. More concretely.

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P_a(s'|s) V^*(s'). \tag{4.13}$$

Therefore, the optimal value function can be expressed by:

$$V^*(s) = \text{argmax}_{a \in \mathcal{A}} Q^*(s,a), \text{ for all } s \in \mathcal{S}. \tag{4.14}$$

4.5.2 UCT algorithm

Although we simplied the continuous-state payment planning problem by discretizing both action space and state space, as we can see, in the planning problem settings, the state space is still large, and grows exponentially with the number of subtasks ($\bar{\mathcal{S}} = (M \times L)^N$).

UCT [Kocsis and Szepesvári, 2006] is a planning algorithm that can successfully cope even with an exponentially sized states. It has achieved remarkable success in the challenging game of Go [Gelly et al., 2006]. Exponential states plague other MDP solvers ultimately because these solvers attempt to enumerate the domain of the transition function for various state-action pairs, e.g., when summing over the transition probabilities or when generating action outcomes to build determinations. As a consequence, they need the transition function to be explicitly known and efficiently enumerable. Instead, being a Monte Carlo planning technique, UCT only needs to be able to efficiently sample from the transition function and the cost function.

Based on the multi-armed algorithm UCB [Auer et al., 2002], UCT effectively selects an action in each state based on a combination of two characteristics — an approximation of the action's $Q-$value ($\hat{Q}(s,a)$) and a mea-

Algorithm 1: PaymentPlanning

Data: initial_state s
Result: optimal payment a

```
repeat
    Search(initial_state)
until timeout
return bestPayments
```

sure of how well-explored the action in this state is ($C\sqrt{\frac{ln(n_s)}{n_{s,a}}}$). More specifically, UCT selects the action that maximizes an upper confidence bound on the action value,

$$a^* = \text{argmin}_{a \in \mathcal{A}} \left\{ \hat{Q}(s,a) + C\sqrt{\frac{ln(n_s)}{n_{s,a}}} \right\} \tag{4.15}$$

where $\hat{Q}(s,a)$ is the estimated value of action a in state s, n_s is the number of times state s has been visited, and $n_{s,a}$ is the number of times action a was selected when state s has been visited. Clearly, for each s, $n_s = \sum_{a \in \mathcal{A}} n_{s,a}$. Details are showed in Algorithm 1.

4.6 Efficiency Analysis

4.6.1 Settings

We explore the efficiency of the MDP-based payment planning in the settings giving considerations to the conclusions obtained from numerous empirical and theoretical studies of crowdsourcing task solving and user behavior. Parameters used in simulations are summarized in Table 4.1.

Function Search(*state s*)

Result: optimal reward Q

if *state* == *terminal* **then**
  return 0
end if
if *s has not been visited before* **then**
  $n_s \leftarrow 0$
  for $\forall a \in \mathcal{A}$ **do**
    $n_{s,a} \leftarrow 0$
    $Q(s,a) \leftarrow 0$
  end for
end if
if *random_number* <0.01 **then**
  $a' = random\_action(s)$
  if *there is an untried action in s* **then**
    $a' = random\_action\_of\_untried(s)$
  else
    $a' = \text{argmin}_{a \in \mathcal{A}} \left\{ Q(s,a) + C\sqrt{\frac{ln(n_s)}{n_{s,a}}} \right\}$
  end if
end if
/* State transition from s to s' based on Eq.(4.9) */
$s' \leftarrow$ simulate action a' in state s
$n_s \leftarrow n_s + 1$
$n_{s,a} \leftarrow n_{s,a} + 1$
$Q(s,a) \leftarrow Q(s,a) + \gamma$ Search(*state* s')
return $\hat{Q}(s,a)$

Table 4.1: Summary of parameters

Parameter	Symbol	Value
No. of Ability Levels	L	5
No. of Subtasks	N	5
No. of Budget Units	M	10

Normal ability distribution

We consider the ability distribution function based on the conclusions provided by the empirical user studies in crowdsourcing marketplaces. According to both studies on crowdsourcing market for general-purpose tasks, MTurk [Ipeirotis, 2010] and a crowdsourcing portal with strong market orientation, TopCoder [Archak, 2010], we assume the ability distribution function have a concave cumulative distribution ($\hat{F}(v)$) and in a normal distribution form with mean μ and variance $\sigma^2 > 0$, as follows.

$$\hat{F}(v) = \Phi((v-\mu)/\sigma) \tag{4.16}$$

where $\Phi(x) = \frac{1}{\sqrt{2x}} \int_{-\infty}^{x} e^{-(t^2/2)} dt$. The normal ability distribution indicates that there are more workers with relatively low abilities.

Discretized state space

Online reputation system is a key component of virtually all crowdsourcing website. [Archak, 2010] provides the evidence that worker's reputation is a significant indicator of his ability. Reputation system collects and aggregates the historical records of workers' behavior to form their reputation. For example, users on TopCoder are distinguished into five groups according to their ratings. Without loss of generality, we assume the number of ability level to be 5 ($L = 5$). Furthermore, as discussed in subsection 4.4.3, we discretize the budget into M =10 units.

Subtask types

Crowdsourcing applications which apply sequential task solving process and exhibit outstanding effectiveness, typically have a small set of subtask types. For example, CrowdForge provides a high-level structure for task decomposition which generally consists of three types of subtasks [Kittur et al., 2011]. Soylent also contributes a specific three-stage pattern, Find-Fix-Verify, to text editing services [Bernstein et al., 2010]. Furthermore, this three-stage pattern is attested to be effective for not only word processing, but also photo analysis application [Noronha et al., 2011]. By looking at these empirical studies, we consider a five-subtask ($N = 5$) situation in simulation, considering the task requester may prefer to additional stages for quality control.

Difficulty combinations

By task decomposition, task requester decomposes the complex task into subtasks of various difficulty combinations. Since these subtasks represent elementary units composed in the original task, their difficulties are determined intrinsically by the characteristic of the task. Without loss of generality, we explore MDP-based payments in three difficulty combinations as 1) *average* difficulties, which gives all subtasks with the same difficulties; 2) *increasing* difficulties, which starts with an easy subtask and leaves the most difficult one to the last stage; 3) *decreasing* difficulties, which on the contrary, starts with the most difficult subtask.

Benchmark approaches

Consider an (unrealistic) offline payment planning algorithm with complete access to the true abilities of a succession of workers. The maximal final quality in this setting can be achieved by constrained optimization algorithm. We denote this benchmark by OPT-Quality, i.e., the optimal payment quality. Other ways to predetermine the payments for all subtasks are to divide the total payments proportionally to subtasks' difficulties, or the

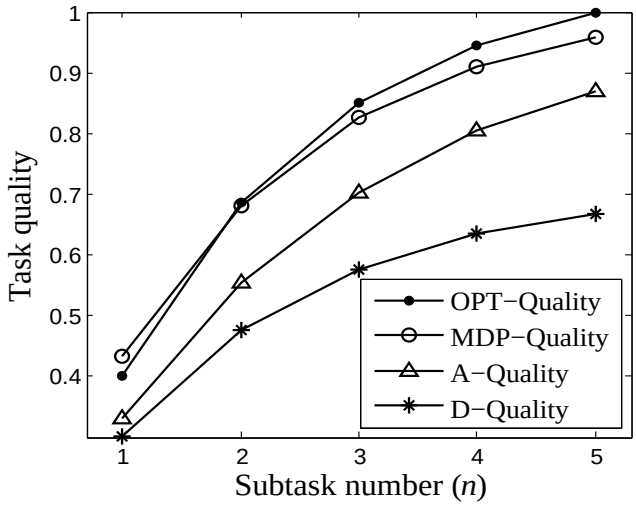


Figure 4.3: Efficiency comparison for average difficulties.

abilities of workers, which we denote with D-Quality, i.e., difficulty-based payment quality, and A-Quality, i.e., ability-based payment quality, respectively.

4.6.2 Simulation results

In Fig. 4.3, 4.4 and 4.5, we compare the efficiency of MDP-based payment planning with three benchmark approaches, for three difficulty combinations. We generate 20 different sets of data for each of the situations, and illustrate the average final qualities respectively. The constant factor C in Eq. (4.15)) equals 0.06, which is determined by trial and error method and generates relatively high convergence speed. Last, we examine three difficulty combinations, and for each combination, we test 20 sets of randomly generated values, and for each test set we run at least to average 3000 trials with UCT algorithm. We finally demonstrate in Fig. 4.3, 4.4 and 4.5, the final quality that averages these 20 data sets. We now present and discuss the findings from our simulation.

As shown in Fig. 4.3, 4.4 and 4.5, generally, when our approach (MDP-

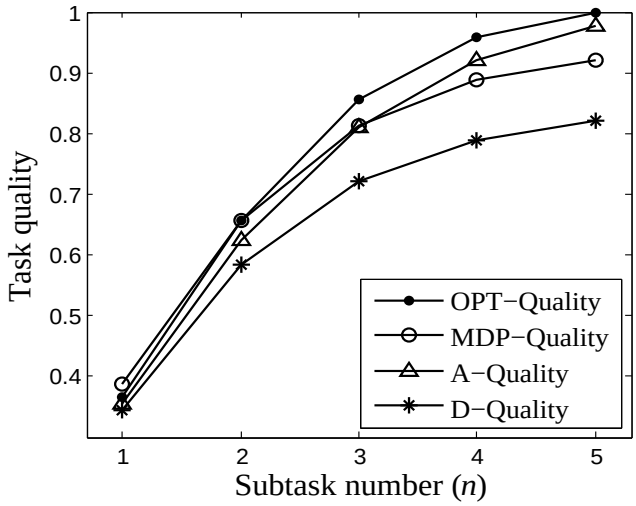


Figure 4.4: Efficiency comparison for increasing difficulties.

Quality) is applied, it allows the task's final quality to approximate the optimal quality brought by the offline optimal payments (OPT-Quality). For all difficulty combinations, Fig. 4.3, 4.4 and 4.5, illustrate clearly that for two of the offline payment planning approaches, ability-based payment planning always outperforms difficulty-based payment planning, and the latter one is an inferior approach to all the others. This indicates that the abilities of workers play more significant role in determining the qualities of the final solutions than the difficulties of subtasks.

We can see in Fig. 4.3, 4.4 and 4.5, that MDP-based payments always generate relatively high qualities of the first two sequential subtasks. Especially, for average and increasing difficulty combination situations (see Fig. 4.3 and Fig. 4.4), the qualities of the first two subtasks exceed the corresponding qualities brought about by the optimal payments. It indicates that since in the sequential subtask solving model the first subtask's quality is crucial to the final quality, the MDP-based payment planning tends to allocate a relatively high payment to the first subtask. This strategy works well under average and decreasing difficulty combinations (see Fig. 4.3 and Fig. 4.5). However, when the difficulties of the subtasks are increasing, this

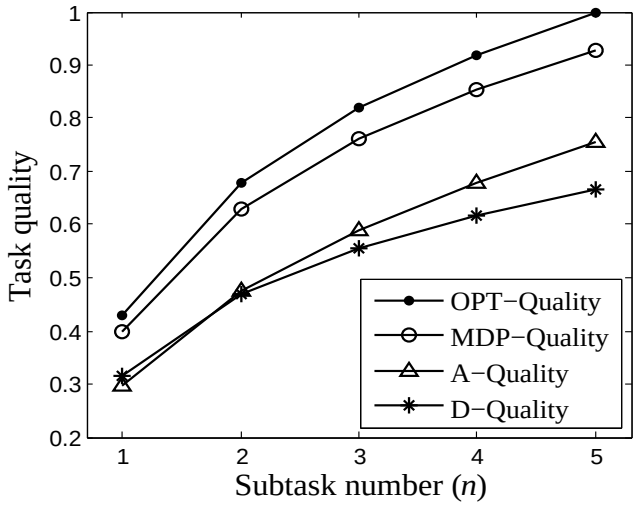


Figure 4.5: Efficiency comparison for decreasing difficulties.

strategy which allocates high payments in the beginning, leaves inadequate budget for maintaining quality growth rate when facing more and more difficult subtasks, as shown in Fig. 4.4.

4.7 Summary

This study addresses the challenge of allocating limited budget to multiple subtasks which are sequentially interdependent with each other. Our first contribution is we formalize a general model of such crowdsourcing subtasks by specifying the relationship between subtask quality and the worker's effort level and how sequential subtasks explicitly depend on each other. Then we pose the problem of maximizing the quality of the final solution with limited budget. A key challenge in this problem is to allocate the limited budget among the sequential subtasks. This is particularly difficult in crowdsourcing marketplaces where the task requester posts subtasks with their payments sequentially and pays the predetermined payment to the worker once the subtask is finished. By modeling this problem as Markov Decision Processes (MDPs) which capture the task-solving pro-

cess's uncertainty about the real abilities of a succession of workers, that is, the uncertainty about the qualities of the sequential subtasks, we show the value of MDPs for the problem of optimizing the quality of the final solution by dynamically determining the budget allocation on sequential dependent subtasks under the budget constrains and the uncertainty of the workers' abilities. Our simulation-based approach verifies that comparing to some benchmark approaches, our MDP-based payment planning is more efficient on optimizing the final quality under the same budget constraints.

Chapter 5

A Division Strategy for Efficient Quality Control

5.1 Introduction

Crowdsourcing is an online, distributed problem-solving and web-based business model that has emerged in recent years [Brabham, 2008], and it is now admired as one of the most lucrative paradigm of leveraging the collective intelligence of crowds. This very success is dependent on the diversity of crowds, not only the diversity of opinion and skill, but also the diversity of *role* played by the crowds in crowdsourcing process.

Common crowdsourcing process proceeds in three phases: (i) a task is announced, usually with its requirements, reward and time period; (ii) crowds work effortfully to compete to provide the best solution; (iii) all of the solutions are examined, a subset of solutions are selected, and the corresponding users are granted the rewards. Although the crowd's primary role is as *task solvers* in the second phase, they become more and more important in the solution examination phase in a wide variety of tasks, work as *solution examiners* to help screening or picking out the best solutions. For example, Stack Overflow, which is a Q&A site for programmers, introduced a *voting system* to help valuable pieces of technical knowledge to become more vis-

ible [Mamykina et al., 2011]. Specifically, users earn rights to examine the contents of posts (questions and answers) of others and vote on the posts if they are considered to be especially useful and appropriate. Upvote on a post helps it to be more visible — appear in the front of the post list. Another example is Threadless, which grants its community members to *score* the designs of t-shirts submitted by other members [Brabham, 2010].

In this study, we explore the model for crowdsourcing software development process, where the crowdsourcing-based *bug detection* is imported as a solution examination phase. Specifically, we model this process as a two-stage process in which the crowds who selected the same programming problem announced by crowdsourcing center (i) in the submission stage, work as *coders* independently, and submit their unique pieces of program codes to the crowdsourcing center and (ii) in the bug detection stage, are presented all of the codes have been submitted to the center, select one piece of them and work as *bug detectors*. Each piece of code is endowed with reward and has a single chance to be proved incorrect. Hence, reward is granted to the bug detector who becomes the first one to provide the test case which can prove his selected code to be incorrect. For instance, the algorithm competition on TopCoder.com includes *coding stage*, where crowds are presented with the same programming problem and are supposed to submit their own codes within limited time, and *challenge stage*, where each of the users has a chance to challenge the functionality of the codes from others. A successful challenge results in a 50-point reward for challenging user [Lakhani et al., 2010].

We are of the opinion that this bug detection process have some strong advantages. Above all, it is a crowdsourcing-based process that can eliminate the false codes and retain the true ones more efficiently and economically than employing a small number of experts to test the large number of codes. Secondly, bug detection by going through others' programs could be beneficial to the programmers since it gives them a chance to learn from others for skill improvement. In contrast, crowds fail to improve in crowdsourcing without such processes [Yang et al., 2008b].

In spite of the above mentioned advantage, the crowdsourcing-based

bug detection paradigm has inherent weakness in term of the distribution of codes that have been selected to be debugged. Because of the various degrees of the codes' qualities, crowds incline to congest on the ones with relatively low qualities that give them high probabilities of successful bug detection, which leaves a substantial part of unchecked solutions and leads to redundant examination on the other part. From the efficiency standpoint, adopting the perspective of a *principal* with the goal to identify maximum number of incorrect codes, we view the uneven distribution of debugged codes as low efficiency.

The goal of this study is to mitigate the uneven distribution problem to improve the crowdsourcing-based bug detection from an efficiency standpoint. To achieve this goal, we encounter the challenge of balancing the freedom and restriction on code selection. It is noteworthy that crowdsourcing provides the policy of maximum freedom of task selection, which encourages crowds to solve the tasks based on their own preferences and thus becomes indispensable for the crowdsourcing's prosperity. Because of that, the traditional assignment problem [Kuhn, 1955], which has been well studied to solve the problem of assigning a given set of workers with varying preferences to a given set of jobs in such a way that, maximum efficiency can be achieved, is not appropriate to be used in crowdsourcing situation.

By contrast, *division strategy* is considered to be highly effective. The basic idea of division strategy is to control the crowds' skill distribution in each division by strategically dividing the crowds with different skills into divisions, and restrict code selection in the division, i.e., the crowds can only select from the codes produced by the crowds in the same division. By doing this, we are able to guide the crowds to select from the codes with more appropriate levels of qualities that corresponding to their skill levels. Therefore, the division strategy can be reliable to work out a more uniform distribution of debugged codes and thus an improvement in bug detection efficiency.

We construct this chapter as follows. In Section 5.3, we model the bug detection contest as *all-pay auction*, and rigorously analyze the relationship between strategic code selection behaviors and the offered rewards. In Sec-

tion 5.4, we present the basic idea of the division strategy and explore two key features in efficient division strategy. In Section 5.5, we conduct the simulation which verifies the effectiveness of division strategy on bug detection efficiency. Finally, Section 5.6 concludes our results, discusses the limitation of our model and suggests potential topics for future research.

5.2 Related Work

There has recently been work on addressing the efficiency problem of crowdsourcing where the crowdsourcing-based contests are modeled as all-pay auctions, a well-studied mechanism that is frequently employed in the contest literature [Baye et al., 1996]. To date, most previous research has focused on the crowdsourcing contests with the format that the principal only benefits from the submission with the highest bid (i.e., the solution with the highest quality), with the aim of optimizing the quality of the best submission [Archak and Sundararajan, 2009] [Chawla et al., 2012] [Ghosh and McAfee, 2012].

Instead of suffering the loss of submissions provided by non-winners, our research takes the full advantage of all-pay auction, with the goal of optimizing the *sum of qualities* for the principal, in connection to bug detection, the total number of debugged codes. Moldovanu and Sela [Moldovanu and Sela, 2001] study the contest model with multiple prizes for the sum-of-qualities objective. Another work [Moldovanu and Sela, 2006] proposed by them studies both the highest quality submission objective and the sum of qualities objective in a multi-stage contest model. Minor [Minor, 2011] studies the incentive design for contest with heterogeneous players to elicit the maximum of the sum of qualities. In contrast, Cavallo and Jain [Cavallo and Jain, 2012] consider the crowdsourcing from a social planner's perspective that seeks to maximize social welfare.

The empirical research shows the evidence that both monetary reward and non-monetary reward, taking the form of reputa-

tion points, provide incentive for crowds to submit high quality solutions [Mason and Watts, 2010] [Yang et al., 2008b]. DiPalantino and Vojnovic [DiPalantino and Vojnovic, 2009] theoretically demonstrate that qualities of submissions are logarithmically increasing as a function of the offered reward. Those results motivate the *reward allocation design* in crowdsourcing contests for improving efficiency. Particularly, contest with *multiple prizes* becomes an effective way to maximize the sum of effort investment [Cason et al., 2010] [Clark and Riis, 1998a] [Moldovanu and Sela, 2001] [Szymanski and Valletti, 2005]. In addition, *contest architecture design* is also used for improving contest efficiency. A contest architecture specifies a *multi-stage contest* in which players are split among several sub-contests whose winners compete against each others. Optimal structure and reward allocation are studied in a sheer amount of literatures [Moldovanu and Sela, 2006] [Fu and Lu, 2012].

Our work differs from both of the above methods mainly in the following two aspects. 1) Instead of considering one contest with multiple rewards, our bug detection model is more akin to a collection of separate contests and each contest has one single reward. 2) Instead of allocating multiple prizes to the winners, we indirectly control the rewards of a set of contests using division strategy. A notably relevant work is [DiPalantino and Vojnovic, 2009].

5.3 Preliminaries

5.3.1 Bug Detection Model

It is natural to view the competitive nature of the bug detection stage as a *contest*, i.e., crowds who selected the same piece of code for bug detection compete among themselves for the reward. We model the contest as an *all-pay auction*, and consider an highest-bid-wins single-item all-pay auction, in which bidders simultaneously submit sealed bids for an item. All bidders forfeit their bids. The auctioneer awards the item to the bidder with the

highest bid, and keeps all the bids.

In the connection to the bug detection contest, the item is the reward, the bids are the actions of bug detection (e.g., constructing test cases), and the sealed value of the bids is the efforts exerted in the bug detection. Specifically, we consider the bug detection stage as a game in which each player, i.e. the bug detector, selects a contest, i.e. a unique piece of code, exerts effort and in each contest the player with the highest effort wins the contest. In the event of a tie, a winner is selected uniformly at random among the highest bidders. Consider there are N players. Associated with each player i is his skill parameter, v_i, where $v_i \in (0,1]$ is drawn from a non-decreasing distribution function $F(v)$ independently of the skills of the other players. Player i's skill is higher than player j's if $v_i > v_j$.

The players can be naturally partitioned into $K(K \geq 2)$ classes, from low skill to high skill, according to the *skill classes* $\vec{\theta} = (\theta_0, \theta_1, ..., \theta_K)$, $\theta_0 = 0 < \theta_1 < ... < \theta_K = 1$. Thus the k-th class contains the players with k-th lowest skills, $v \in (\theta_{k-1}, \theta_k]$. Let N_k be the number of players in class k, then we must have $\sum_{k=1}^{K} N_k = N$.

Let R denote the reward for first successful bug detection for any of the codes, and the value of the reward is common knowledge. However, simply treating the rewards as the same fails to capture the quality difference of codes written by players with different levels of skills, since it is generally acknowledged that there are variations in individual programming performance. One reasonable assumption is that the codes produced by low-skill coders are those with higher probabilities of incorrectness than those produced by high-skill coders. It is implied that the skill parameter v_i stands for both coding and debugging skills of player i. Based on this assumption, we furthermore endow each piece of code with a *weight coefficient* according to its quality, which is positively correlated with the player's skill. Specifically, the codes produced by player from k-th class are associated with weight $\beta_k \in [0,1]$. Intuitively, for each piece of code, the associated weight can be viewed as its probability of having bugs, we have $\beta_1 > \beta_2 > \cdots > \beta_K$. The *expected rewards* for successful bug detection for the codes produced by k-th class of players can be calculated as $ER_k = \beta_k \cdot R$. That is, for

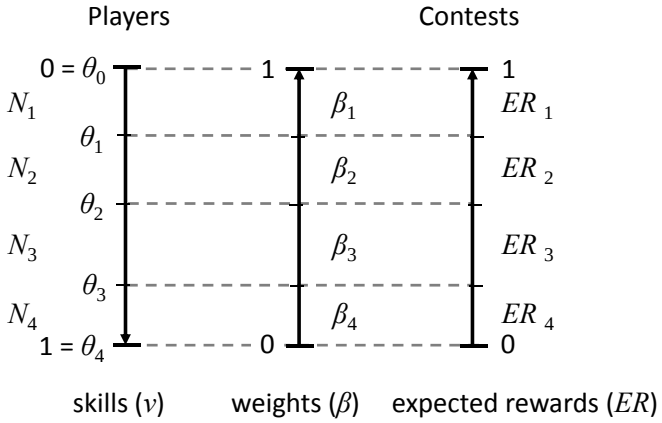


Figure 5.1: Group players and contests into different classes. It shows that the skill distribution of players, in terms of skill classes and the number of players in each class, determines the number of bug detection contests in each corresponding class and the associated rewards.

K classes of codes produced by K classes of players, we have K classes of rewards, $\overrightarrow{ER} = (ER_1, ..., ER_K)$, where $ER_1 > ER_2 > ... > ER_K$, thus, the bug detection contests are naturally partitioned into K classes, which taking on one of these values as its reward.

In Fig. 5.1, we provide an illustration of grouping contests into four classes. It is of particular importance that the skill distribution of players, in terms of skill classes and the number of players in each class, determines the number of bug detection contests in each corresponding class and the associated rewards. For example, since there are N_1 players of the lowest-skill class, we have N_1 lowest-quality pieces of code, which become the N_1 highest-reward contests.

5.3.2 Contest Selection

Proposition 5.1. *(Proposition 3.1 and 4.1 [DiPalantino and Vojnovic, 2009]). There exist a unique symmetric equilibrium to the bug detection contest game.*

Theorem 1. *(Theorem 4.2 [DiPalantino and Vojnovic, 2009]). In the equilibrium, players select unique code as given in the following.*

– **Skill levels.** *Players are partitioned over L skill levels. A skill level l corresponds to the interval of skill values $[v_{l+1}, v_l)$, where*

$$F(v_l) = 1 - N_{[1,l]} \left(1 - \frac{ER_l^{\frac{1}{N-1}}}{H_{[1,l]}(\overrightarrow{ER})} \right), \quad for \;\; l = 1, ..., L \tag{5.1}$$

where

$$H_{[1,l]}(\overrightarrow{ER}) = \left(\sum_{k=1}^{l} \frac{N_k}{N_{[1,l]}} ER_k^{-\frac{1}{N-1}} \right)^{-1}$$

$$and \quad N_{[1,l]} = \sum_{k=1}^{l} N_l.$$

– **Code selection vs. skill level.** *A player of skill v selects a particular contest/code of class j with probability $\pi_j(v)$ given by*

$$\pi_j(v) = \begin{cases} \frac{ER_j^{-\frac{1}{N-1}}}{\sum_{k=1}^{l} N_k ER_k^{-\frac{1}{N-1}}}, & \text{for } j = 1, ..., l \\ 0, & \text{for } j = l+1, ..., K. \end{cases} \tag{5.2}$$

for $v \in [v_{l+1}, v_l)$.

Remark 5.1. In item 1, Eq. (5.1) shows that the intervals of skill levels is determined by the players' skill classes and the number of players of each skill class. It is worthwhile to note that, according to Eq. (5.1), in the equilibrium, two players in the same skill class could be partitioned into different skill levels, and one skill level could be constituted by the players from more than one skill classes. Item 2 shows that, in the equilibrium, a player of skill level l selects a contest that provides one of l highest rewards with probability given by Eq. (5.2). Specifically, the player with skill level l selects contest that provides the corresponding expected reward, that is, the l-th highest reward, with the highest probability, and those that provides

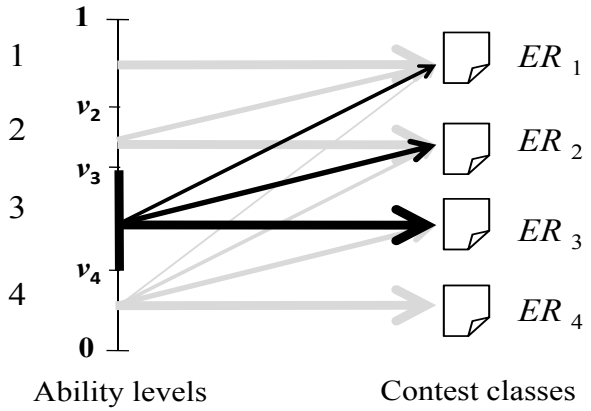


Figure 5.2: Contest selection behavior in equilibrium. Players are partitioned into 4 skill levels. A player of skill level 3 selects the contests that provide the 3*rd* highest expected rewards with the largest probability.

larger expected rewards are selected with lower probabilities.

Fig. 5.2 provides an illustration of the contest selection behavior for K=4 and L=4. The thickness of the arrows indicates the probability that a player selects that particular contest. A player of skill level 3 will select the contests that offer one of 3 highest expected rewards. Moreover, she selects contests that offer the 3*rd* highest expected reward with the largest probability, and that offer the highest expected reward with the smallest probability.

5.4 Bug Detection Efficiency Based on Division Strategy

Aimed at identifying the key features of division strategy that determine the bug detection efficiency, we explore division strategies in two dimensions — horizontal and vertical. For the horizontal comparison, we explore the *skill mixing degree*, which varies from none — *separating*, assembles agents with same skill class in the same division, to very high — *mixing*, assembles

agents with various skill classes in one division. Based on the example study, we make our first appealing conjecture as follows.

– Skill mixing degree is the most significant feature of the efficient division strategy. Specifically, bug detection efficiency is positively related to the skill mixing degree;

Although the skill mixing degree controls the trend of the bug detection efficiency, such a relationship between skill mixing degree and bug detection efficiency is not a one-to-one correspondence. It inspires us to explore the feature that can differentiate the division strategies wherein the same skill mixing degree leads to different bug detection efficiencies. For this vertical analysis, we propose the concept of skill similarity degree which represents the closeness of players' skills in value. Based on the example study, we make our second appealing conjecture as follows.

– Skill similarity degree plays a subordinate role. Specifically, for the situations wherein the same skill mixing degree leads to different bug detection efficiencies, high skill similarity degree indicates high level of bug detection efficiency.

We arrange this section as follows. First we formally construct the definitions for skill mixing degree, skill similarity degree and bug detection efficiency. Then we conduct an example study. At first, we compare the bug detection efficiencies on different skill mixing degrees, which indicates that the bug detection efficiency increases along with the increase in skill mixing degree. Secondly, the comparison among the situations wherein the same skill mixing degree leads to different bug detection efficiencies indicates us that high bug detection efficiency also depends on high skill similarity degree. Finally, weighted average of skill mixing degree and skill similarity degree is constructed for evaluating their relative importance.

5.4.1 Division Strategy

Definition 5.1. The skill mixing degree is defined as the sum of the degrees of the skill mixing of all divisions, as follows

$$M = \sum_{i=1}^{I} M_i \tag{5.3}$$

where I is the number of divisions, and

$$M_i = \frac{\prod_{k\in\Gamma_i} N_{ik}}{\sum_{k\in\Gamma_i} N_{ik}^2} \tag{5.4}$$

where $\Gamma_i = \{k : N_{ik} > 0, k \in \{1,\ldots,K\}\}$, and N_{ik} is the number of the players of k-th skill class in division i.

Definition 5.2. The skill similarity degree is defined as the sum of the degrees of the skill similarity of all divisions, as follows

$$S = \sum_{i=1}^{I} S_i \tag{5.5}$$

where I is the number of divisions, and

$$S_i = 1 - \sqrt{\frac{1}{N_i - 1} \sum_{k=1}^{N_i-1} (v_k - v_{k+1})^2} \tag{5.6}$$

where N_i is the number of the players in division i, and the all players' skills are sorted in non-decreasing order, i.e., $v_1 \leq v_2 \leq \cdots \leq v_{N_i}$.

In order to clarify the efficacy of the division strategy, let's first consider the problem of *uneven distribution* of debugged codes in bug detection contests without division strategy, which is reduced to the situation wherein all players are assembled in the unique division. When players of highest skill class and lowest skill class are assembled in the same division, according

to Eq.(5.2), players with highest skills will exclusively select the codes produced by the lowest-skill player which offer the highest expected rewards. Consequently, in order to avoid competing with the players having higher skills, players with lowest skills would probably select the codes with the highest qualities. Moreover, due to their low skills, they can hardly catch the bugs in those codes. This results in low efficiency, from the principal's point of view, although codes produced by low-skill player would be examined, those produced by high-skill players are left unchecked or checked superficially due to the debuggers' limited skills. In addition, from the player's point of view, it leads to significant inequality and harms the benefits gained by low-skill players.

By applying the division strategy, we can mitigate the uneven distribution of the checked codes and the inequality by restricting the players to select the codes with more appropriate levels of rewards. Specifically, in our bug detection model, we divide N players into I divisions, and restrict the contest selection in the divisions, i.e., the players can only select from the codes produced by the players in the same division. By intentionally assembling players with particular skills classes in one division, the principal can control the essential features of the collection of contests in the division, in terms of expected reward classes and the scales of skill levels, which, according to Eq.(5.1) and Eq.(5.2), determine the players' strategic behaviors on code selection. For instance,when the players of highest skill class and those of lowest skill class are partitioned into different division, the former are restricted to select the codes produced by the players with higher skill classes instead of the lowest skill class.

In the division, the expected reward class is the most significant determinant of the players' strategic behaviors on code selection, and there's one-to-one correspondence between it and the composition of players' skill classes, in terms of the number of the skill classes and the number of players in each skill class. Hence, the concept of skill mixing degree has been constructed for distinguishing the division strategies varying in compositions of players' skill classes, and becomes the most important factor in our bug detection model.

Moreover, it is worthwhile to note that, the exchange of two players belong to two different divisions can bring different effects on the skill mixing degree and skill similarity degree. Specifically, if the two players in two divisions has similar skill values but belong to different skill classes (since the skill values for players are continuous, two players with similar skill values could be in two adjacent but different skill classes.), the exchange of these two player will cause small similarity degree changes in both divisions, but a relatively large change in mixing degree. It is reasonable based on the definitions given by Eq.(5.4) and Eq.(5.6) that skill mixing degree concerns the number of players in different skill classes, but the skill similarity degree concerns the specific value of the players skills. In the same way, we can deduce that if there's a large difference between the skill values of two players in two divisions and belong to different skill classes, the exchange of them will cause relatively small change in mixing degree, but a large change in similarity degree.

5.4.2 Bug Detection Efficiency

Definition 5.3. The bug detection efficiency is defined as the sum of the bug detection efficiency of all divisions, as follows

$$E = \sum_{i=1}^{I} E_i \tag{5.7}$$

where E_i is the efficiency of the i-th division, which is defined as

$$E_i = \sum_{n=L}^{1} \sum_{m=L}^{n} R \cdot \beta_m \cdot N_{im} \cdot \pi_{L-m+1}(v_{L-n+1}) \int_{v_{L-n+2}}^{v_{L-n+1}} v dv. \tag{5.8}$$

where

- R:the reward;
- β_m:the weight coefficient endowed to m-th class of codes, representing its probability of having bugs;
- N_{im}: the number of players of m-th skill class in division i;

– $\pi_{L-m+1}(v_{L-n+1})$: the probability for a player with skill level $L-n+1$ to select a particular code of class $L-m+1$;
– $[v_{L-n+2}, v_{L-n+1})$: the interval of skill level $L-n+1$.

Remark 5.2. The integrals of the skill parameter imply another reasonable assumption that the player's skill of coding is proportional to her skill of bug detection. Therefore, the skill with a player can be viewed as her probability of successful bug detection. Then $\pi \cdot v$ can be understood as the probability of one piece code to be selected and successfully proved to be flawed by a player with skill v, and $\beta \cdot N$ is the number of incorrect codes produced by the player in the same division. Hence, when the reward is normalized to 1, the efficiency of bug detection defined above can be understood as the expected number of codes proved to be flawed in all divisions.

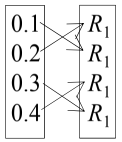
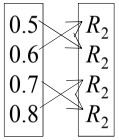
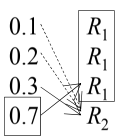
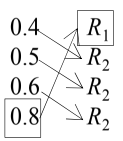
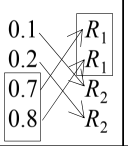
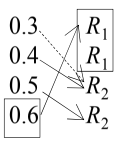
Given reward normalization (R=1), the efficiency of bug detection can be understood as the expected number of codes proved to be flawed. Consider a particularly simple situation that in one division, there are players of two skill levels ($[0, v_{threshold})$, $[v_{threshold}, 1]$), and codes with two classes of expected reward (β_{high}, β_{low}, N_{high}, N_{low}). Players with high skills only contribute to the set of codes with high expected rewards, i.e., select high-expected-reward codes with probability equals to $\frac{1}{N_{high}}$. The expected number of incorrect codes contributed by high-skill players is calculated as

$$\beta_{high} \cdot N_{high} \cdot \frac{1}{N_{high}} \int_{v_{threshold}}^{1} v dv = \beta_{high} \int_{v_{threshold}}^{1} v dv.$$

In contrast, players of low skill level contribute to both classes of codes, with probabilities π_{high} and π_{low} for high-expected-reward codes and low-expected-reward codes respectively. Note that $\pi_{low} > \pi_{high}$, i.e., the low-expected-reward codes are their first choices. The expected number of incorrect codes contributed by low-skill players is calculated as

$$\beta_{low} \cdot N_{low} \cdot \pi_{low} \int_{0}^{v_{threshold}} v dv + \beta_{high} \cdot N_{high} \cdot \pi_{high} \int_{0}^{v_{threshold}} v dv.$$

Table 5.1: An example of bug detection efficiency increases with the skill mixing degree of division strategy.

	Division Strategy I		Division Strategy II		Division Strategy III	
	Division 1_1	Division 1_2	Division 2_1	Division 2_2	Division 3_1	Division 3_2
Division Initialization	0.1 R_1 0.2 R_1 0.3 R_1 0.4 R_1	0.5 R_2 0.6 R_2 0.7 R_2 0.8 R_2	0.1 R_1 0.2 R_1 0.3 R_1 0.7 R_2	0.4 R_1 0.5 R_2 0.6 R_2 0.8 R_2	0.1 R_1 0.2 R_1 0.7 R_2 0.8 R_2	0.3 R_1 0.4 R_1 0.5 R_2 0.6 R_2
Skill Mixing Degree (M)	$M_1 = 0.5$		$M_2 = 0.6$		$M_3 = 1$	
Skill Similarity Degree (S)	$S_1 = 1.8$		$S_2 = 1.62$		$S_3 = 1.6$	
Skill Level (L)	$L_{1\_1} = 1$	$L_{1\_2} = 1$	$L_{2\_1} = 2$	$L_{2\_2} = 2$	$L_{3\_1} = 2$	$L_{3\_2} = 2$
	$<v_2, v_1>=<\underline{0},1>$	$<v_2,v_1>=<\underline{0},1>$	$<v_3, v_2, v_1>$ $=<0, \underline{0.38}, 1>$	$<v_3, v_2, v_1>$ $=<0, \underline{0.79}, 1>$	$<v_3, v_2, v_1>$ $=<0, \underline{0.59}, 1>$	$<v_3, v_2, v_1>$ $=<0, \underline{0.59}, 1>$
Contest Selection						
Bug Detection Efficiency (E)	E_{1_1}=0.8	E_{1_2}=1.04	E_{2_1}=0.68	E_{2_2}=1.24	E_{3_1}=1.32	E_{3_2}=0.84
	E_1=1.84		E_2=1.92		E_1=2.16	

Additionally, the integrals of the skill parameter imply the reasonable assumption that the player's skill of coding is proportional to his/her skill of bug detection.

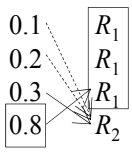
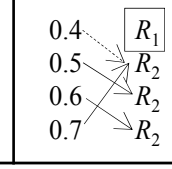
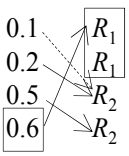
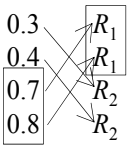
5.4.3 Example Study: Key Factors in Efficient Division Strategy

In Table 5.1, we consider a two-division situation wherein eight players who are endowed with skills v_1,$v_2\cdots$,v_8=0.1, 0.2, $\cdots$, 0.8, belong to two skill classes, $(0, 0.4]$ and $(0.4, 0.8]$. The two classes of codes produced by low-skill players and high-skill players are, without loss of generality, endowed with specific values, R_1=0.8 and R_2=0.4, respectively. We initialize the two divisions with equal number of players. In Division Strategy I, two classes of players are completely separated into two divisions, which leads to the minimum skill mixing degree M_1=0.5. In the equilibrium, players in both divisions are partitioned into only one skill level (L_{1_1}=L_{1_2}=1). Therefore, in Division 1_1, players have equal probability for selecting among the four high-reward contests.

Consider the *common case* in which 1) players in high-skill level select contests with high rewards and players in low-skill level select those provide low rewards; and 2) the "congestion problem" — more than one players compete on the same piece of code — is minimized. Thus, in Division 1_1 no players is supposed to compete on any of the contests, and each piece of code is debugged by a unique player. By contrast, the situation is different in division 2_1 brought by Division Strategy II, where players are partitioned into two skill levels in the equilibrium. There's one player ($v = 0.7$) in the high-skill level, who is also the only one can select from three high-reward contests. The other three players of low-skill level have to compete for the only contest with low reward R_2. Player with the highest skill ($v = 0.3$) wins the reward.

The horizontal comparison among the outcomes under different division strategies demonstrates our first conjecture, that bug detection efficiency is positively related to the skill mixing degree. Specifically,skill mix-

Table 5.2: An example of bug detection efficiency increases with the skill similarity degree.

	Division Strategy II_A		Division Strategy III_A	
M	M_{2_A}= 0.6		M_{3_A} = 1	
L	$<v_3, v_2, v_1>$ =<0, 0.38, 1>	$<v_3, v_2, v_1>$ =<0, 0.79, 1>	$<v_3, v_2, v_1>$ =<0, 0.59, 1>	$<v_3, v_2, v_1>$ =<0, 0.59, 1>
Contest Selection				
S	S_{2_A} = 1.6		S_{3_A} = 1.8	
E	$E_{2\_A\_1}$=0.76	$E_{2\_A\_2}$=0.72	$E_{3\_A\_1}$=0.76	$E_{3\_A\_2}$=1.48
	E_{2_A}=1.48		E_{3_A}=2.24	

ing degrees of Division Strategy I, II, III are $M_1 = 0.5$, $M_2 = 0.6$, $M_3 = 1$, and the corresponding bug detection efficiency are $E_1 = 1.84$, $E_2 = 1.92$, $E_3 = 2.16$.

In Table 5.2, we re-initialize the player allocation in Division Strategy II. By exchanging two players with skills $v = 0.7$ and $v = 0.8$, the bug detection efficiency decreases to $E_{2\_A} = 1.48$ ($< E_2 = 1.92$), without any change to the skill mixing degree ($M_{2\_A} = M_2 = 0.6$). By contrast, the change of player allocation in Division Strategy III leads to contrary outcome. By exchanging the high-skill-class players in two divisions, the bug detection efficiency increases to $E_{3\_A} = 2.24$ ($> E_3 = 2.16$), with the same skill mixing degree ($M_{3\_A} = M_3 = 1$).

The one-to-many correspondence between skill mixing degree and bug detection efficiency inspires us to do the vertical comparison by checking the skill similarity degree of division strategies with the same skill mixing degree. For the Division Strategy II and II_A, with the same skill mixing degree ($M_{2\_A} = M_2 = 0.6$), II_A with lower skill similarity de-

gree ($S_{2\_A} = 1.6 < S_2 = 1.62$) works out lower bug detection efficiency ($E_{2\_A} = 1.48 < E_2 = 1.92$). For the Division Strategy III and III_A, with the same skill mixing degree equals to 1, the increase in skill similarity degree of III_A ($S_{3\_A} = 1.8 > S_3 = 1.6$) leads to a higher bug detection efficiency ($E_{3\_A} = 2.24 > E_3 = 2.16$). This vertical comparison demonstrates our second conjecture that for the situations wherein the same skill mixing degree leads to different bug detection efficiencies, high skill similarity degree indicates high level of bug detection efficiency.

Based on the above analysis, we go one step further by incorporating the relative importance relation of skill mixing degree and skill similarity degree. Using weighted average to combine these two determinant factors allows to consider their relative importance in division strategy for bug detection efficiency. Fig. 5.3 visualizes the shape of the value of the weighted averages over the skill mixing degrees and skill similarity degrees($(M+0.9S)/2$) given in the example. By weighting skill mixing degree and skill similarity degree with coefficients 1 and 0.9 respectively, the average weight of skill mixing degree and skill similarity degree is demonstrated to be consistent with the shape of bug detection efficiency. Although, 0.9 is just an instance for the weight on skill similarity degree, one can easily find that once the coefficient given to skill similarity degree is equal or larger than the one of skill mixing degree, the weighted average becomes different from the bug detection efficiency. The values of weight demostrate that skill mixing degree plays a more important role than skill similarity in the bug detecttion efficiency.

5.5 Simulation and Analysis

In this section, we experimentally evaluate the performance of the division strategies with different skill mixing degrees and skill similarity degrees. First, we randomly generate N players with different skills. Second, we divide them into I divisions in the way that it initials the division strategy with the lowest skill mixing degree. Without loss of generality, we suppose

each division contains the same number of players, i.e., N/I. Based on our model, bug detection efficiency of each division can be computed. Third, by strategically changing the allocation of players, we develop a new division strategy with higher skill mixing degree. By repeatedly conducting the second and the third steps, we can then investigate the conjectures given in the above section.

5.5.1 Setting

We explore the above two features in three environments.

Linear skill distribution function. Suppose the skills of players are uniformly distributed on the unit interval $[0,1]$, we assume the skill distribution function as linear, i.e.,

$$F_{linear}(v) = v, v \in [0,1].$$

Concave skill distribution function. In the situation where the contests attract a lot of players with low skills rather than high skill — for example, a relatively new crowdsourcing platform or the early stages in the multi-stage sequential-elimination contests [Fu and Lu, 2012] — we assume the skill distribution function as concave function which is a normal distribution with mean μ and variance $\sigma^2 > 0$, i.e.,

$$F_{concave}(v) = \Phi((v-\mu)/\sigma),$$

where $\Phi(x)=\frac{1}{\sqrt{2x}}\int_{-\infty}^{x} e^{-(t^2/2)}dt$.

Convex skill distribution function. By contrast, in the situation where the contest attracts more high-skill players than low-skill players — for example, the crowdsourcing contests with minimum skill requirement — we assume the skill distribution function as convex function which, without loss of generality, can be the following power function:

$$F_{convex}(v) = v^{\alpha},\ \alpha > 1.$$

Table 5.3: Summary of parameters

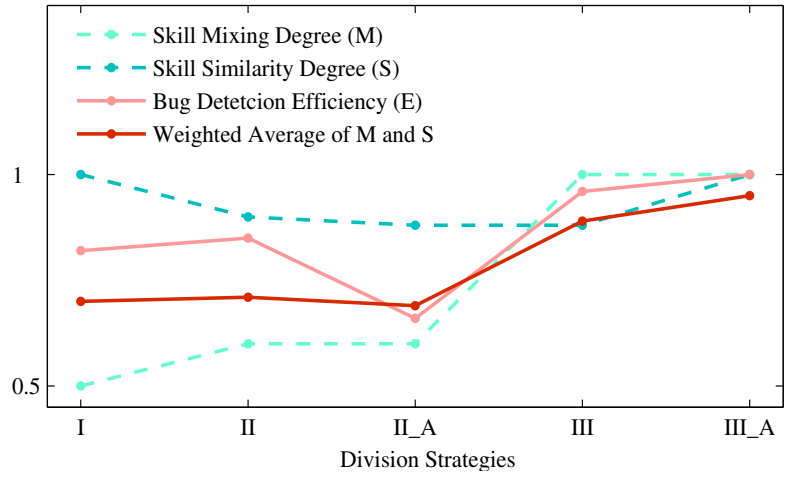
Parameter	Symbol	Value
No. of Players	N	200
Skill of Players	v	$(0,1]$
No. of Skill Classes	K	4
No. of Divisions	I	4
Thresholds of Skill Classes	$\vec{\theta}$	$(0,0.25,0.5,0.75,1)$
Weight Coefficients	$\vec{\beta}$	$(0.8,0.6,0.4,0.2)$

Parameters used in simulation are presented in Table 5.3. 200 players of 4 skill classes are equally divided into 4 divisions. As the weight coefficients show, we don't guarantee the high-quality codes have no bugs, and not all of the low-quality codes are incorrect. We normalize the reward to 1, and we also normalize the simulation results, including bug detection efficiency, skill mixing degree and skill similarity degree in order to compare the effect of different division strategies.

5.5.2 Results and Discussion

Bug detection efficiency of division strategies with different skill mixing degree are investigated under three kinds of skill distributions. Different from the common case we illustrated in the example in section 4, where players in skill level l are only allowed to select the contests providing the l-th class of rewards, we conducted the simulation in less controlled setting. In the simulation, the players are supposed to strategically select contest with probabilities given by Eq.(5). That is, a player of skill level l can randomly select a contest that provides one of l highest rewards. Specifically, the player with skill level l selects contest that provides the corresponding class reward, that is, the l-th highest reward, with the highest probability and those that provides larger expected rewards are selected with lower probabilities.

First, we can easily confirm the conjecture that the bug detection effi-



	Skill Mixing Degree (M)	Skill Similarity Degree (S)	Bug Detection Efficiency (E)	Weighted Average
I	0.50	1.00	0.82	0.70
II	0.60	0.90	0.85	0.71
II_A	0.60	0.88	0.66	0.69
III	1.00	0.88	0.96	0.89
III_A	1.00	1.00	1.00	0.95

Figure 5.3: Relative importance of two key factors of efficient division strategy. Shape of weighted average $((M+0.9S)/2)$ over skill mixing degree and skill similarity degree is consistent with that of bug detection efficiency (E).

ciency increases with the skill mixing degree, for all of the three types of skill distributions. In Fig. 5.4, (a.1) presents the linear skill distribution with mean skill value 0.7, and (a.2) shows the mean efficiency of bug detection versus the skill mixing degree. The linear regression on the bug detection efficiencies indicates an increasing trend for bug detection efficiency through the whole interval of skill mixing degree. Fig. 5.5 (a.2) and Fig. 5.6 (a.2) lead to the same conclusion under both concave skill distribution and convex skill distribution. It is noticeable that, the increase in bug detection efficiency under concave skill distribution situation is particularly

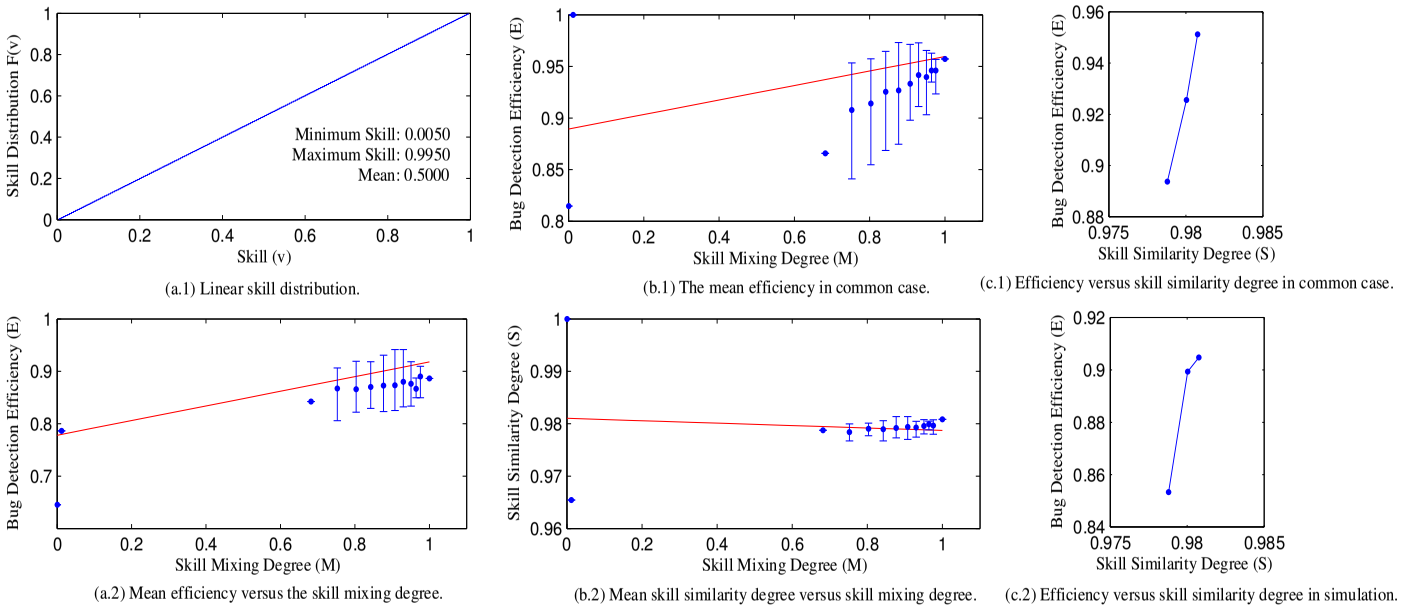
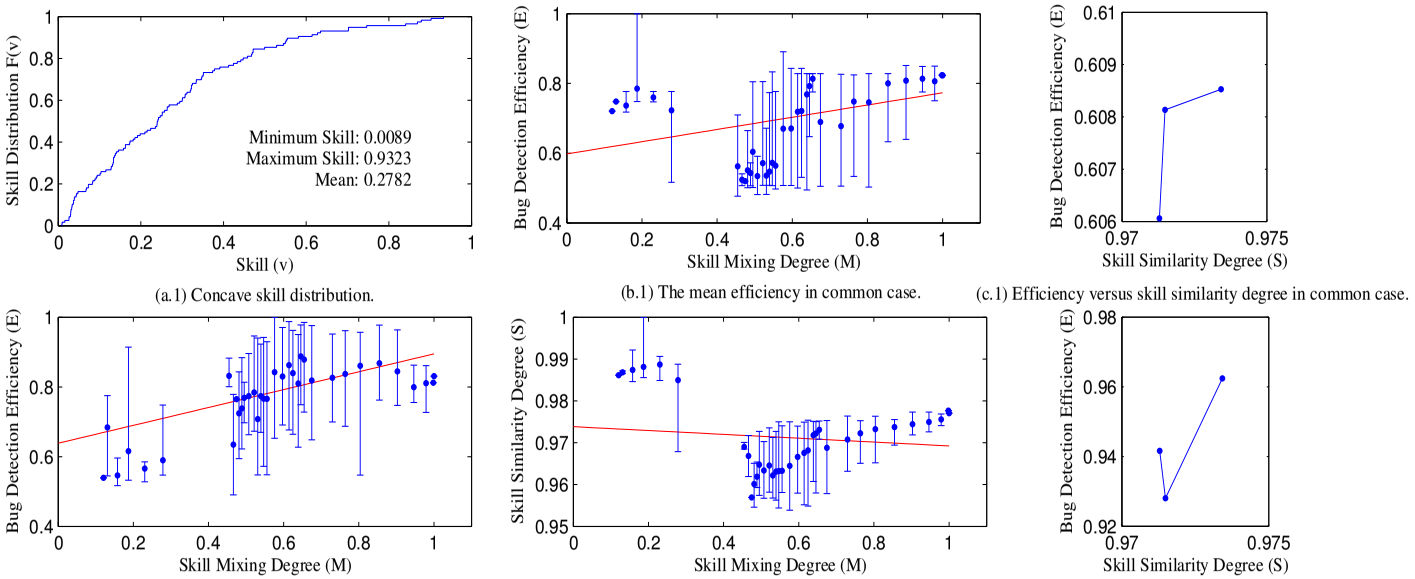


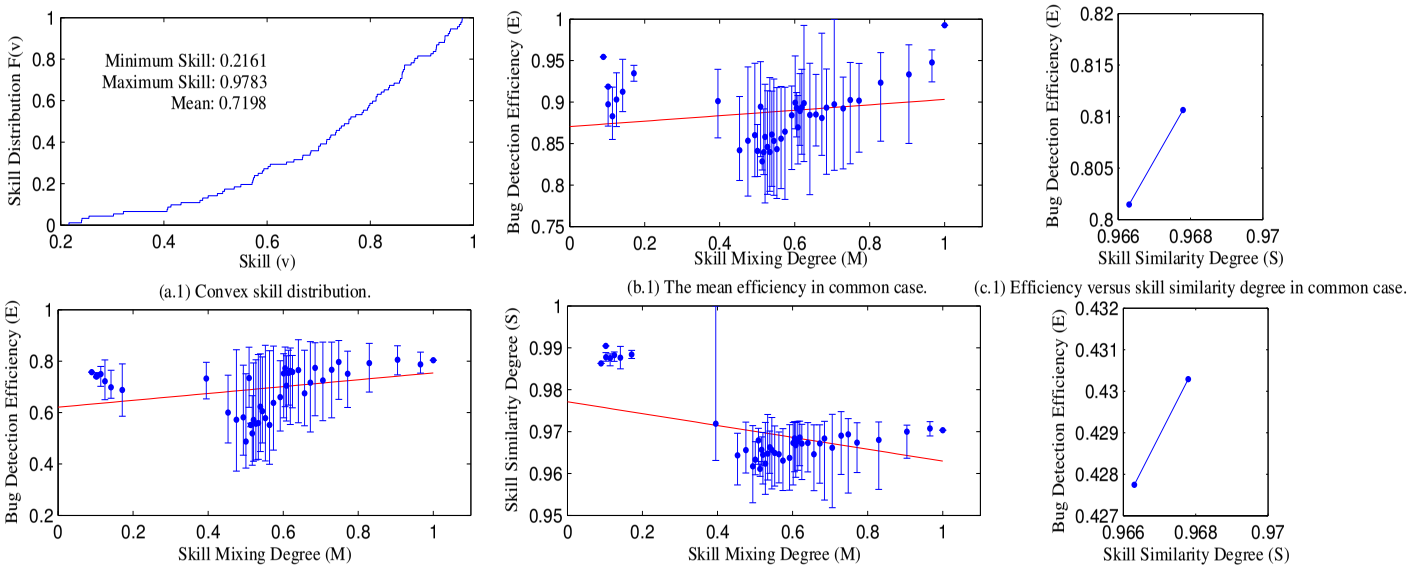
Figure 5.4: Linear skill distribution. Bug detection efficiency increases with the skill mixing degree, and shaped as skill similarity degree.



(a.1) Concave skill distribution. (b.1) The mean efficiency in common case. (c.1) Efficiency versus skill similarity degree in common case.

(a.2) Mean efficiency versus the skill mixing degree. (b.2) Mean skill similarity degree versus skill mixing degree. (c.2) Efficiency versus skill similarity degree in simulation.

Figure 5.5: Concave skill distribution. Bug detection efficiency increases with the skill mixing degree, and shaped as skill similarity degree.



(a.1) Convex skill distribution.

(b.1) The mean efficiency in common case.

(c.1) Efficiency versus skill similarity degree in common case.

(a.2) Mean efficiency versus the skill mixing degree.

(b.2) Mean skill similarity degree versus skill mixing degree.

(c.2) Efficiency versus skill similarity degree in simulation.

Figure 5.6: Convex skill distribution. Bug detection efficiency increases with the skill mixing degree, and shaped as skill similarity degree.

remarkable comparing to the situations of linear and convex skill distributions. Specifically, for the concave skill distribution situation, Fig. 5.5 (a.2) shows the increase in the efficiency is about 0.2 (from 0.6 to 0.8), which is larger than those of the situations for linear skill distributions (Fig. 5.4 (a.2)) and convex skill distributions (Fig. 5.6 (a.2)), wherein the increases in the efficiency is about 0.1 (from 0.8 to 0.9, 0.6 to 0.7, respectively). The reason is easy to understand that for the concave skill distribution, with a low mean skill equaling to 0.2782, the number of potential bugs in all codes is higher than that under both linear and convex skill distribution. In Fig. 5.4 (b.1), Fig. 5.5 (b.1) and Fig. 5.6 (b.1) we investigate the mean efficiency versus the skill mixing degree in common case. The conjecture that bug detection efficiency is positively related to the skill mixing degree is also hold under three types of skill distributions in this case.

Secondly, we illustrate the mean skill similarity degree versus the skill mixing degree in (b.2). As we initialize the divisions with the lowest skill mixing degree at the very beginning by allocating the players of the same skill class in the same division, the skill similarity degree is set to be high at first. Then by exchanging a small number of players in different skill classes among divisions, the skill similarity degree decreases dramatically along with a slight increase in skill mixing degree. As the exchanging process continues, the numbers of the players in different skill classes become closer, which makes the mean of skill similarity degree rebound gradually. Comparing to the skill similarity degree, the corresponding bug detection efficiency in the common case shapes up in a consistent way. This consistence confirms our conjecture that for the situations wherein the same skill mixing degree leads to different bug detection efficiencies, high skill similarity degree indicates high level of bug detection efficiency.

Furthermore, we present the bug detection efficiency versus skill similarity degree under the same skill mixing degree (by simply choosing the median value of efficiency for illustration) in Fig. 5.4 (c.1)(c.2), Fig. 5.5 (c.1)(c.2) and Fig. 5.6 (c.1)(c.2). They show that in common case, as we conjectured, the efficiency increases with the skill similarity degree. However, it is unlikely that this conjecture holds in the simulation which may

due to the random selection on contests. Finally, contrary to the bug detection efficiency, the linear regression lines on the mean skill similarity degree versus skill mixing degree (Fig. 5.4 (b.2), Fig. 5.5 (b.2) and Fig. 5.6 (b.2)) show that the skill similarity degree decreases along with the increase in skill mixing degree. However, this does not reverse the increasing trend of bug detection efficiency, which indicates that skill mixing degree controls the trend, and skill similarity degree indicates the shape.

5.6 Summary

In this study we have presented and analyzed a crowdsourcing-based bug detection model in which strategic players select code and compete in bug detection contests. Our focus is on addressing the low efficiency problem in bug detection. Our study shows that by intentionally assembling players with particular skill distribution in one division and limiting the code selection and bug detection contests in the division, the division strategy, to some extent, can control the essential features of the contests, in terms of expected reward classes and the scales of skill levels, which determine the players' strategic behaviors on code selection. By exploring key factors of division strategy, we conclude that skill mixing degree, serving as determinant factor, controls the trend of the bug detection efficiency, specifically, high degree of skill mixing leads to high level of bug detection efficiency, and skill similarity degree plays an important role in indicating the shape of the bug detection efficiency.

Nonetheless, we have noted that although skill distribution as skill mixing degree and skill similarity degree is the main determinant of player strategic behavior on code selection, it is not the only factor that can influence the bug detection efficiency. Future work could consider the impact of the size and the number of the divisions, and how they might affect the bug detection efficiency. In addition, one limitation on the assignment of codes' rewards in the proposed model also could be considered in future research. Aiming at finding as many incorrect codes as possible, we con-

struct expected rewards from the player's point of view to differentiate the qualities of codes and apply the relationship between reward and player's strategic behavior to encourage more appropriate code selection behavior. However, for other situations wherein the principal requires guarantee of high quality on codes produced by high-skill agents, the rewards assigned to the codes produced by high-skill players should be higher than those provided by low-skill players.

Chapter 6

Conclusion

6.1 Contributions

In this thesis, we have proposed incentive approaches to motivate workers who cooperatively resolve complex tasks in crowdsourcing environment to come up with high-quality solutions. The main contributions of this thesis are summarized as follows:

1. Efficient task decomposition strategy design for complex tasks.

We have formally presented and analyzed vertical and horizontal task decomposition models which respectively specify the relationship between subtask quality and the worker's effort level in the presence of positive and none dependence among subtasks. Our focus is on addressing the efficiency of task decomposition when the workers are paid equally and contribution-based. We conducted simulations and experiments on Amazon Mechanical Turk to analyze and compare the efficiency of two task decompositions, aiming at generating explicit instructions on strategies for optimal task decomposition. We conclude that when the final quality is defined as the sum of the qualities of solutions to all decomposed subtasks, vertical task decomposition strategy outperforms the horizontal one in improving the quality of the final solution. We also give explicit instructions the optimal strategy un-

der vertical task decomposition situation to help the task requester to achieve the highest quality of the final solution.

2. MDP-Based reward design for efficient cooperative problem solving.

We addresses the challenge of allocating limited budget to multiple subtasks which are sequentially interdependent with each other. Our first contribution is that we formalize a general model of such crowdsourcing subtasks by specifying the relationship between subtask quality and the worker's effort level and how sequential subtasks explicitly depend on each other. Then we pose the problem of maximizing the quality of the final solution with limited budget. A key challenge in this problem is to allocate the limited budget among the sequential subtasks. This is particularly difficult in crowdsourcing marketplaces where the task requester posts subtasks with their payments sequentially and pays the predetermined payment to the worker once the subtask is finished. By modeling this problem as Markov Decision Processes (MDPs) which capture the task-solving process's uncertainty about the real abilities of a succession of workers, that is, the uncertainty about the qualities of the sequential subtasks, we show the value of MDPs for the problem of optimizing the quality of the final solution by dynamically determining the budget allocation on sequential dependent subtasks under the budget constrains and the uncertainty of the workers' abilities. Our simulation-based approach verifies that comparing to some benchmark approaches, our MDP-based payment planning is more efficient on optimizing the final quality under the same budget constraints.

3. A division strategy for quality control.

In this issue we have presented and analyzed a crowdsourcing-based bug detection model in which strategic players select code and compete in bug detection contests. Our focus is on addressing the low efficiency problem in bug detection. This bug detection contest can be view as a particular mechanism for redundant solution examination, and is readily extended to the cooperative task-solving process we formalized in the first two issues.

Our study shows that by intentionally assembling players with particular skill distribution in one division and limiting the code selection and bug detection contests in the division, the division strategy, to some extent, can control the essential features of the contests, in terms of expected reward classes and the scales of skill levels, which determine the players' strategic behaviors on code selection. By exploring key factors of the division strategy, we conclude that skill mixing degree, serving as determinant factor, controls the trend of the bug detection efficiency, specifically, high degree of skill mixing leads to high level of bug detection efficiency, and skill similarity degree plays an important role in indicating the shape of the bug detection efficiency.

6.2 Future Directions

Our future research directions of incentive design in cooperative problem solving in crowdsourcing are listed below.

– *Vertical and horizontal task decomposition strategy combination.*

In this thesis we mainly concentrate on modeling and analyzing vertical and horizontal task decomposition independently. Although in terms of quality improvement, vertical task decomposition strategy is superior to the horizontal task decomposition, the latter one is comparatively easier to conduct in crowdsourcing platform, and still powerful in crowdsourcing-based problem solving. It is necessary to combine these two strategy in the workflows design for complex problem solving. Empirical studies are on this issue (e.g., [Kulkarni et al., 2011]), however, absent explicit guidelines, it is difficult for the task requesters to decide how to take fully advantage of these two task decomposition when they work coordinately.

– *Budget-constrained crowdsourcing.*

In this thesis we consider how to allocate limited budget for complex workflows that interleave heterogeneous micro-task (e.g., Find-Fix-Verify).

In our process, all budget is supposed to be spent. In the future work, is it valuable to consider how to balance the final quality and the amount of budget, that is how to make it cheaper and achieve a threshold of quality as well. The MDP-based budget allocation algorithm proposed in this thesis has the potential to deal with the new research problem.

Bibliography

[Abbassi and Misra, 2011] Abbassi, Z. and Misra, V. (2011). Multi-level revenue sharing for viral marketing. In *Proceedings of ACM NetEcon*.

[Adamic et al., 2008] Adamic, L. A., Zhang, J., Bakshy, E., and Ackerman, M. S. (2008). Knowledge sharing and yahoo answers: everyone knows something. In *Proceedings of the 17th International Conference on World Wide Web*, pages 665–674.

[Aniket Kittur and Kraut, 2011] Aniket Kittur, Boris Smus, S. K. and Kraut, R. E. (2011). Crowdforge: crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 16–19.

[Archak, 2010] Archak, N. (2010). Money, glory and cheap talk: analyzing strategic behavior of contestants in simultaneous crowdsourcing contests on topcoder. com. In *Proceedings of the 19th International Conference on World Wide Web*, pages 21–30.

[Archak and Sundararajan, 2009] Archak, N. and Sundararajan, A. (2009). Optimal design of crowdsourcing contests. *ICIS 2009 Proceedings*, page 200.

[Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256.

[Azaria et al., 2012] Azaria, A., Aumann, Y., and Kraus, S. (2012). Automated strategies for determining rewards for human work. In *Proceed-*

ings of the 26th AAAI Conference on Artificial Intelligence.

[Barabási and Albert, 1999] Barabási, A.-L. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509–512.

[Barua et al., 1995] Barua, A., Lee, C.-H. S., and Whinston, A. B. (1995). Incentives and computing systems for team-based organizations. *Organization Science*, 6(4):487–504.

[Baye et al., 1996] Baye, M. R., Kovenock, D., and De Vries, C. G. (1996). The all-pay auction with complete information. *Economic Theory*, 8(2):291–305.

[Bellman, 1956] Bellman, R. (1956). Dynamic programming and lagrange multipliers. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 42, page 767.

[Bernstein et al., 2010] Bernstein, M. S., Little, G., Miller, R. C., Hartmann, B., Ackerman, M. S., Karger, D. R., Crowell, D., and Panovich, K. (2010). Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, pages 313–322.

[Bertsekas et al., 1995] Bertsekas, D. P., Bertsekas, D. P., Bertsekas, D. P., and Bertsekas, D. P. (1995). *Dynamic Programming and Optimal Control*, volume 1. Belmont, MA: Athena Scientific.

[Brabham, 2008] Brabham, D. C. (2008). Crowdsourcing as a model for problem solving an introduction and cases. *Convergence: the International Journal of Research into New Media Technologies*, 14(1):75–90.

[Brabham, 2010] Brabham, D. C. (2010). Moving the crowd at threadless: motivations for participation in a crowdsourcing application. *Information, Communication & Society*, 13(8):1122–1145.

[Cason et al., 2010] Cason, T. N., Masters, W. A., and Sheremeta, R. M. (2010). Entry into winner-take-all and proportional-prize contests: an experimental study. *Journal of Public Economics*, 94(9):604–611.

[Cavallo and Jain, 2012] Cavallo, R. and Jain, S. (2012). Efficient crowd-

sourcing contests. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, pages 677–686.

[Chawla et al., 2012] Chawla, S., Hartline, J. D., and Sivan, B. (2012). Optimal crowdsourcing contests. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 856–868.

[Chen et al., 2011] Chen, J. J., Menezes, N. J., Bradley, A. D., and North, T. (2011). Opportunities for crowdsourcing research on amazon mechanical turk. *Interfaces*, 5(3).

[Chen et al., 2010] Chen, Y., HO, T.-H., and KIM, Y.-M. (2010). Knowledge market design: A field experiment at google answers. *Journal of Public Economic Theory*, 12(4):641–664.

[Cheng and Han, 2013] Cheng, M. L. and Han, Y. (2013). A modified cobb–douglas production function model and its application. *IMA Journal of Management Mathematics*, pages 353–365.

[Clark and Riis, 1998a] Clark, D. J. and Riis, C. (1998a). Competition over more than one prize. *The American Economic Review*, 88(1):276–289.

[Clark and Riis, 1998b] Clark, D. J. and Riis, C. (1998b). Influence and the discretionary allocation of several prizes. *European Journal of Political Economy*, 14(4):605–625.

[Dai et al., 2013] Dai, P., Lin, C. H., Weld, D. S., et al. (2013). Pomdp-based control of workflows for crowdsourcing. *Artificial Intelligence*, 202:52–85.

[Dean and Image, 2008] Dean, C. and Image, E. (2008). If you have a problem, ask everyone. *New York Times*, 22.

[DiPalantino and Vojnovic, 2009] DiPalantino, D. and Vojnovic, M. (2009). Crowdsourcing and all-pay auctions. In *Proceedings of the 10th ACM Conference on Electronic Commerce*, pages 119–128.

[Doan et al., 2011] Doan, A., Ramakrishnan, R., and Halevy, A. Y. (2011). Crowdsourcing systems on the world-wide web. *Communications of the*

ACM, 54(4):86–96.

[Faradani et al., 2011] Faradani, S., Hartmann, B., and Ipeirotis, P. G. (2011). What's the right price? pricing tasks for finishing on time. In *Proceedings of HCOMP11: The 3rd Workshop on Human Computation*.

[Fu and Lu, 2009] Fu, Q. and Lu, J. (2009). The beauty of "bigness": On optimal design of multi-winner contests. *Games and Economic Behavior*, 66(1):146–161.

[Fu and Lu, 2012] Fu, Q. and Lu, J. (2012). The optimal multi-stage contest. *Economic Theory*, 51(2):351–382.

[Fuxman et al., 2008] Fuxman, A., Tsaparas, P., Achan, K., and Agrawal, R. (2008). Using the wisdom of the crowds for keyword generation. In *Proceedings of the 17th international conference on World Wide Web*, pages 61–70.

[Gelly et al., 2006] Gelly, S., Wang, Y., Munos, R., and Teytaud, O. (2006). Modification of uct with patterns in monte-carlo go.

[Ghosh and McAfee, 2012] Ghosh, A. and McAfee, P. (2012). Crowdsourcing with endogenous entry. In *Proceedings of the 21st international conference on World Wide Web (WWW'12)*, pages 999–1008.

[Greg Little and Miller, 2010] Greg Little, Lydia B. Chilton, M. G. and Miller, R. C. (2010). Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 68–76.

[Harper et al., 2009] Harper, F. M., Moy, D., and Konstan, J. A. (2009). Facts or friends? distinguishing informational and conversational questions in social q&a sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 759–768.

[Harper et al., 2008] Harper, F. M., Raban, D., Rafaeli, S., and Konstan, J. A. (2008). Predictors of answer quality in online q&a sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 865–874.

[Hars and Ou, 2002] Hars, A. and Ou, S. (2002). Working for free? motivations for participating in open-source projects. *International Journal of Electronic Commerce*, pages 25–39.

[Haythornthwaite, 2009] Haythornthwaite, C. (2009). Crowds and communities: Light and heavyweight models of peer production. In *The 42nd Hawaii International Conference on System Sciences (HICSS'09)*, pages 1–10.

[Ho and Vaughan, 2012] Ho, C.-J. and Vaughan, J. W. (2012). Online task assignment in crowdsourcing markets. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012)*, pages 45–51.

[Holmstrom and Milgrom, 1991] Holmstrom, B. and Milgrom, P. (1991). Multitask principal-agent analyses: Incentive contracts, asset ownership, and job design. *Journal of Law, Economics, & Organization*, 7:24–52.

[Houser and Wooders, 2006] Houser, D. and Wooders, J. (2006). Reputation in auctions: Theory, and evidence from ebay. *Journal of Economics & Management Strategy*, 15(2):353–369.

[Howe, 2006] Howe, J. (2006). The rise of crowdsourcing. *Wired magazine*, 14(6):1–4.

[Howe, 2009] Howe, J. (2009). *Crowdsourcing: Why the power of the crowd is driving the future of business*. Three Rivers Press.

[Hurwicz, 1960] Hurwicz, L. (1960). *Optimality and informational efficiency in resource allocation processes*. Stanford University Press.

[Hurwicz, 1972] Hurwicz, L. (1972). On informationally decentralized systems. *Decision and Organization*, pages 297–336.

[Ipeirotis, 2010] Ipeirotis, P. G. (2010). Analyzing the amazon mechanical turk marketplace. *XRDS: Crossroads, The ACM Magazine for Students*, 17(2):16–21.

[Jain et al., 2009] Jain, S., Chen, Y., and Parkes, D. C. (2009). Designing incentives for online question and answer forums. In *Proceedings of the*

10th ACM Conference on Electronic Commerce (EC'09), pages 129–138.

[Jain and Parkes, 2009] Jain, S. and Parkes, D. C. (2009). The role of game theory in human computation systems. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 58–61.

[Jin and Fan, 2011] Jin, Y. and Fan, J. (2011). Test for english majors (tem) in china. *Language Testing*, 28(4):589–596.

[Jøsang et al., 2007] Jøsang, A., Ismail, R., and Boyd, C. (2007). A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644.

[Jurca and Faltings, 2007] Jurca, R. and Faltings, B. (2007). Collusion-resistant, incentive-compatible feedback payments. In *Proceedings of the 8th ACM Conference on Electronic Commerce (EC'07)*, pages 200–209.

[Kaplan and Sela, 2010] Kaplan, T. R. and Sela, A. (2010). Effective contests. *Economics Letters*, 106(1):38–41.

[Karger et al., 2014] Karger, D. R., Oh, S., and Shah, D. (2014). Budget-optimal task allocation for reliable crowdsourcing systems. *Operations Research*, 62(1):1–24.

[Kittur, 2010] Kittur, A. (2010). Crowdsourcing, collaboration and creativity. *ACM Crossroads*, 17(2):22–26.

[Kittur et al., 2008] Kittur, A., Chi, E. H., and Suh, B. (2008). Crowdsourcing user studies with mechanical turk. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 453–456.

[Kittur et al., 2011] Kittur, A., Smus, B., Khamkar, S., and Kraut, R. E. (2011). Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 43–52.

[Kocsis and Szepesvári, 2006] Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. In *Machine Learning: ECML 2006*, pages 282–293.

[Kolobov, 2012] Kolobov, A. (2012). Planning with markov decision pro-

cesses: an ai perspective. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 6(1):1–210.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

[Kulkarni et al., 2012] Kulkarni, A., Can, M., and Hartmann, B. (2012). Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 1003–1012.

[Kulkarni et al., 2011] Kulkarni, A. P., Can, M., and Hartmann, B. (2011). Turkomatic: automatic recursive task and workflow design for mechanical turk. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pages 2053–2058.

[Lakhani et al., 2010] Lakhani, K., Garvin, D., and Lonstein, E. (2010). Topcoder (a): Developing software through crowdsourcing. *Harvard Business School General Management Unit Case No. 610-032*.

[Lakhani et al., 2007] Lakhani, K. R., Jeppesen, L. B., Lohse, P. A., and Panetta, J. A. (2007). *The value of openess in scientific problem solving*. HBS Working Paper Number: 07-050, Harvard Business School.

[Little et al., 2009] Little, G., Chilton, L. B., Goldman, M., and Miller, R. C. (2009). Turkit: tools for iterative tasks on mechanical turk. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 29–30.

[Luca, 2011] Luca, M. (2011). Reviews, reputation, and revenue: The case of yelp. com. Technical report, Harvard Business School.

[Malone and Crowston, 1994] Malone, T. W. and Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)*, 26(1):87–119.

[Malone et al., 2009] Malone, T. W., Laubacher, R., and Dellarocas, C. (2009). Harnessing crowds: mapping the genome of collective intelligence. In *MIT Sloan School of Management Working Paper No. 4732-09*.

[Mamykina et al., 2011] Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., and Hartmann, B. (2011). Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'11)*, pages 2857–2866.

[Mason and Watts, 2010] Mason, W. and Watts, D. J. (2010). Financial incentives and the "performance of crowd". *ACM SIGKDD Explorations Newsletter*, 11(2):100–108.

[Minor, 2011] Minor, D. B. (2011). Increasing effort through softening incentives in contests. Technical report, Working Paper, University of California, Berkeley.

[Moldovanu and Sela, 2001] Moldovanu, B. and Sela, A. (2001). The optimal allocation of prizes in contests. *American Economic Review*, pages 542–558.

[Moldovanu and Sela, 2006] Moldovanu, B. and Sela, A. (2006). Contest architecture. *Journal of Economic Theory*, 126(1):70–96.

[Nam et al., 2009] Nam, K. K., Ackerman, M. S., and Adamic, L. A. (2009). Questions in, knowledge in?: a study of naver's question answering community. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 779–788.

[Nielsen, 2006] Nielsen, J. (2006). Participation inequality: Encouraging more users to contribute. *Jakob Nielsen's alertbox, http://www.useit.com/alertbox/participation_inequality*.

[Noronha et al., 2011] Noronha, J., Hysen, E., Zhang, H., and Gajos, K. Z. (2011). Platemate: crowdsourcing nutritional analysis from food photographs. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, pages 1–12.

[Puterman, 2009] Puterman, M. L. (2009). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[Resnick et al., 2006] Resnick, P., Zeckhauser, R., Swanson, J., and Lockwood, K. (2006). The value of reputation on ebay: A controlled experi-

ment. *Experimental Economics*, 9(2):79–101.

[Sattinger, 1993] Sattinger, M. (1993). Assignment models of the distribution of earnings. *Journal of Economic Literature*, pages 831–880.

[Stewart et al., 2010] Stewart, O., Lubensky, D., and Huerta, J. M. (2010). Crowdsourcing participation inequality: a scout model for the enterprise domain. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 30–33.

[Surowiecki, 2004] Surowiecki, J. (2004). The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business. *Economies, Societies and Nations*.

[Surowiecki, 2005] Surowiecki, J. (2005). *The wisdom of crowds*. Random House LLC.

[Sylvan, 2010] Sylvan, E. (2010). Predicting influence in an online community of creators. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1913–1916.

[Szymanski and Valletti, 2005] Szymanski, S. and Valletti, T. M. (2005). Incentive effects of second prizes. *European Journal of Political Economy*, 21(2):467–481.

[Tausczik and Pennebaker, 2011] Tausczik, Y. R. and Pennebaker, J. W. (2011). Predicting the perceived quality of online mathematics contributions from users' reputations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1885–1888.

[Tran-Thanh et al., 2014a] Tran-Thanh, L., Huynh, T. D., Rosenfeld, A., Ramchurn, S. D., and Jennings, N. R. (2014a). Budgetfix: budget limited crowdsourcing for interdependent task allocation with quality guarantees. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2014)*, pages 477–484.

[Tran-Thanh et al., 2014b] Tran-Thanh, L., Stein, S., Rogers, A., and Jennings, N. R. (2014b). Efficient crowdsourcing of unknown experts using

bounded multi-armed bandits. In *Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012)*, pages 768–773.

[Tran-Thanh et al., 2013] Tran-Thanh, L., Venanzi, M., Rogers, A., and Jennings, N. R. (2013). Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, pages 901–908.

[Von Ahn and Dabbish, 2004] Von Ahn, L. and Dabbish, L. (2004). Labeling images with a computer game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 319–326.

[Wageman and Baker, 1997] Wageman, R. and Baker, G. (1997). Incentives and cooperation: the joint effects of task and reward interdependence on group performance. *Journal of Organizational Behavior*, 18:139–158.

[Wolpert and Tumer, 1999] Wolpert, D. H. and Tumer, K. (1999). An introduction to collective intelligence. Technical report, NASA-ARC-IC-99-63, NASA Ames Research Center.

[Yan and Van Roy, 2008] Yan, X. and Van Roy, B. (2008). Reputation markets. In *Proceedings of the 3rd International Workshop on Economics of Networked Systems*, pages 79–84.

[Yang et al., 2008a] Yang, J., Adamic, L. A., and Ackerman, M. S. (2008a). Competing to share expertise: The taskcn knowledge sharing community. In *Proceedings of the International Conference on Weblogs and Social Media*.

[Yang et al., 2008b] Yang, J., Adamic, L. A., and Ackerman, M. S. (2008b). Crowdsourcing and knowledge sharing: strategic user behavior on taskcn. In *Proceedings of the 9th ACM conference on Electronic commerce (EC'08)*, pages 246–255.

[Zhang et al., 2009] Zhang, H., Chen, Y., and Parkes, D. C. (2009). A general approach to environment design with one agent. In *Proceedings of*

the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09), pages 2002–2014.

[Zhang et al., 2007] Zhang, J., Ackerman, M. S., and Adamic, L. (2007). Expertise networks in online communities: structure and algorithms. In *Proceedings of the 16th International Conference on World Wide Web*, pages 221–230.

[Zhang and Van der Schaar, 2012] Zhang, Y. and Van der Schaar, M. (2012). Reputation-based incentive protocols in crowdsourcing applications. In *INFOCOM, 2012 Proceedings IEEE*, pages 2140–2148.

[Zheng et al., 2011] Zheng, H., Li, D., and Hou, W. (2011). Task design, motivation, and participation in crowdsourcing contests. *International Journal of Electronic Commerce*, 15(4):57–88.

Publications

Journals

1. **Huan Jiang** and Shigeo Matsubara, "A Division Strategy for Achieving Efficient Crowdsourcing Contest", *Journal of Information Processing*, pp. 202-209. Vol. 22-2, 2014.

International Conference

1. **Huan Jiang** and Shigeo Matsubara, "Efficient Task Decomposition in Crowdsourcing", In *PRIMA 2014: Principles and Practice of Multi-Agent Systems*, pp. 65-73. Springer International Publishing, 2014.

2. **Huan Jiang** and Shigeo Matsubara, "Improving Crowdsourcing Efficiency Based on Division strategy", In *Proceedings of the 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology*, pp. 425-429. December, 2012.

Workshops and Symposiums

1. **Huan Jiang** and Shigeo Matsubara, "MDP-based Reward Design for Efficient Cooperative Problem Solving", In *Proceedings of the 18th International Workshop on Coordination, Organizations, Institutions and Norm* (COIN 2014), December, 2014, Australia.

2. **Huan Jiang** and Shigeo Matsubara, “Crowdsourcing Software Bug Detection and Division Strategy”, In *Proceedings of the Second Joint Agent Workshop & Symposium* (JAWS2011), October, 2011, Japan.