Mechanisms for Dynamic Setting with Restricted Allocations

by

Yuxin Yu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2011

Abstract

Dynamic mechanism design is an important area of multiagent systems, and commonly used in resource allocation where the resources are time related or the agents exist dynamically. We focus on a multiagent model within which the agents stay, and the resources arrive and depart. The resources are interpreted as work or jobs and are called tasks. The allocation outcome space has a special restriction that every agent can only work on one resource at a time, because every agent has a finite computational capability in reality.

We propose a dynamic mechanism and analyze its incentive properties; we show that the mechanism is incentive compatible. Empirically, our dynamic mechanism performs well and is able to achieve high economic efficiency, even outperforming standard approaches if the agents are concerned about future tasks. We also introduce a static mechanism under the setting of a restricted outcome space; it is proved that the static mechanism is incentive compatible, and its computational complexity is much less than that of the standard VCG mechanism.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Kate Larson. This thesis would not possibly exist without her supervision. Professor Pascal Poupart and Professor Daniel Lizotte are my thesis readers, I would like to give my appreciation to them for reading and helping with the final version. I would like to thank my colleagues and my friends who make this possible as well.

Dedication

This is dedicated to my parents and to my grandma.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In a system of multiple agents, agents compete for resources. One important problem when allocating resources is whether there is a fair competitive environment. For example, one might desire that the agent with the highest priority obtains the resource or that an agent pays a reasonable price for the resource allocated to it. A real example is an auction. Different bidders submit their bids for a single item. There exists a mechanism that ensures the agent with the highest bid wins, and that asks the winner for a payment, so that the winner would have a non-negative utility and the other agents would not submit fake bids to compete.

Dynamic mechanism design is a subarea where agents and resources arrive in and depart from the market. Dynamic mechanisms introduce new concerns, such as the timing of the allocation and how to estimate the utility of the agents. In this thesis we study allocation problems where the arrival and departure of resources are an important feature, and look at computational issues which can arise in such settings.

1.1 Motivation

Suppose there are multiple tasks that arrive dynamically and any of them can be accomplished by any of several working teams. For example, several magazine offices share a single printing factory. The magazine offices have new magazines to be printed at some time; and there are different types of printers in the printing factory. If a printer starts printing one copy of a magazine, it is better for this printer to complete all copies of the magazine; and it is obvious that every printer can only print one magazine at a time.

The tasks for the printing factory are printing different magazines for the offices. Because magazines can be thin or thick, black and white or colored, high or low pixel quality and so on, the printers need different amounts of time to complete the tasks and also have different cost values for the tasks. As the magazines have time constraints on printing since they need to be transported to the readers, the printers should complete the printing by a specific time, which usually can be predicted. Therefore, the factory would like to dispatch the printers to print as many magazines as possible, with a reasonable total cost. Every waiting magazine only exists in the market for a specific time duration, and would be removed if it is not dispatched to a printer by the last possible time period.

This example can be abstracted into the scenario in Figure 1.1. The magazines waiting to be printed are the tasks, the printers are the teams, and the factory is the dispatcher who decides which printer to work on which magazine. There are five teams in total, and seven tasks have arrived; the dispatcher has already decided the printers for the first four tasks. The dispatcher should continually decide which of the {5,6,7} tasks is delivered to the free team 2; or the dispatcher can choose not to dispatch any task now but wait for a future free team, if the expected total cost is lower than dispatching.

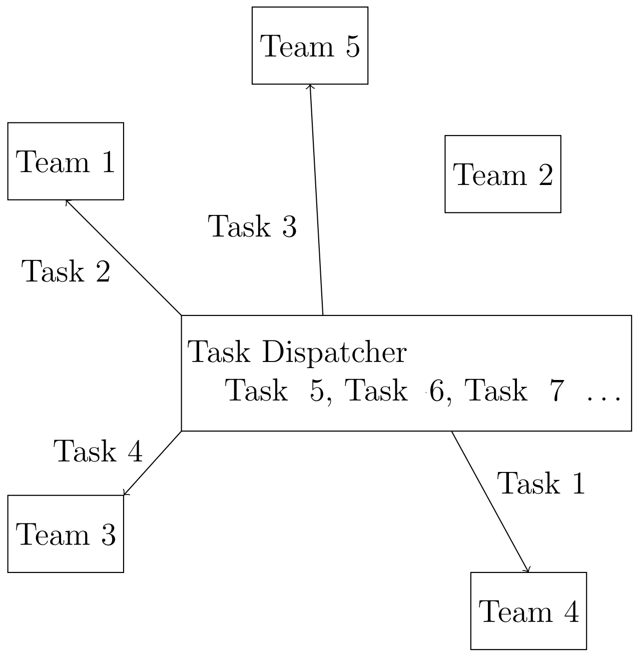

Figure 1.1: Allocating in an example of dynamic scenario

There are other similar situations that can be modeled by the scenario in Figure 1.1. In cloud computing, one service can be called to perform different tasks plenty of times; there are a number of servers to process the tasks. A server could gain some utility by accomplishing a task, so the dispatcher would like to choose a decision that maximizes the teams' values.

In the examples previously discussed, a common assumption was made. It was assumed that a team or agent was capable of processing a single task at a time. Using this assumption, we investigate dynamic models and mechanisms. We argue that this assumption is realistic since it explicitly captures the notion that the agents or teams must accomplish some tasks, and may be restricted in their capabilities. We note that while other researchers have looked at dynamic mechanisms, many have not explicitly included this observation, potentially limiting the applicability of their models.

We start by analyzing this assumption first in a static environment, and a simple consequence is that every agent can only obtain at most one task. In a dynamic environment, an agent could only be allocated a new task after it completes its previous allocations, or if the new task arrives after the current tasks have been completed.

Our mechanisms are extensions of the class of classic Groves mechanisms, known to be the only family of truthful social-welfare maximizing mechanisms for allocation problems. However, we explicitly take into account the computational complexity of the mechanisms we design, which is often overlooked when directly applying Groves mechanisms (in particular the Vickrey-Clarke-Groves mechanism) to problems.

1.2 Contribution and Results

We first study a static setting where agents can only be allocated a single task at a time. We propose a Multi-round mechanism that is truthful and individually rational. We note that the computational complexity of the Multi-round mechanism is $O(n^2)$ in the worst case compared to $O(n^4)$ of the Vickrey-Clarke-Groves(VCG) mechanism in this setting.

The Multi-round mechanism is empirically evaluated. The results show that the worst case of runtime is rare, and the runtime does not change too much as the number of agents and the number of tasks increase. Compared to the VCG mechanism, we do note a loss in social welfare, but argue that this is not a significant issue and additionally that our mechanism appears to be fair in that the tasks it does allocate tend to be balanced in terms of average value.

We then look at a dynamic setting where tasks can arrive and depart and agents can submit bids for the tasks throughout the entire time. We propose a dynamic mechanism for this setting which is truthful. We develop three algorithms for the allocation policy of the mechanism and show that two of our algorithms are polynomial in the number of agents. We also empirically evaluate our dynamic mechanism. The proposed Patience-based VCG mechanism's performance on efficiency is almost the same as the performance of the dynamic VCG mechanism.

This thesis is organized in the following way. First some basic concepts are explained in Chapter 2: *Preliminaries*; then the model on the assumption and the mechanisms are presented in Chapter 3: *Models Where Agents are Allocated At Most One Item at a Time.* We empirically evaluate our proposed mechanisms in Chapter 4: *Implementation and Evaluation.* Some related work is reviewed in Chapter 5: *Related Work*; and finally we conclude with Chapter 6: *Conclusion and Future Work.*

Chapter 2

Preliminaries

This chapter presents the basic definitions and concepts used in this thesis. The Groves mechanism family and the VCG mechanism are the cornerstones of our developed mechanisms. Some essential properties in analyzing mechanisms are also introduced. There is a matching algorithm used in our mechanisms' allocation schemes.

2.1 Mechanism

A mechanism is defined based on the Bayesian game setting.

Definition 2.1.1. *A **Bayesian game** is a tuple (N,O,Θ,p,u), where*

- *N is a finite set of agents;*
- *O is a set of outcomes;*
- *$\Theta = \Theta_1 \times \cdots \times \Theta_n$ is a set of possible joint agents' type vectors;*
- *p is a (common-prior) probability distribution on Θ; and*
- *$u = (u_1 \ldots u_n)$, where u_i: $O \mapsto \Re$ is the utility function for each agent i.*

Assume the agents are self-interested, that every agent is only concerned with the utility that it could obtain from any possible outcome. We assume that agents interact with each other through a mechanism.

Definition 2.1.2. *A **mechanism** on (N,O,Θ,p,u) is a pair (A,M), where*

- $A = A_1 \times \cdots \times A_n$*, where* A_i *is the set of actions available to agent* $i \in \mathbb{N}$*;*
- $M : A \mapsto \prod(O)$ *maps each action profile to a distribution over outcomes.*

An agent i could choose an action from A_i, and the joint action space of other agents is denoted as A_{-i}. The self-interested agent i would like to play an action that maximizes its utility. We assume that agents have quasi-linear utility functions.

Definition 2.1.3. *Agents have **quasi-linear utility functions** (or quasi-linear preferences) in an* n*-player Bayesian game when the set of outcomes is* $O = X \times \Re^n$ *for a finite set* X*, and the utility of an agent i given a joint type* θ *is given by* $u_i(o, \theta) = u_i(x, \theta) - f_i(p_i)$*, where* $o = (x, p)$ *is an element of* O*,* u_i*:*$X \times \Theta \mapsto \Re$ *is an arbitrary function and* $f_i : \Re \mapsto \Re$ *is a strictly monotonically increasing function.*

We will be particularly interested in direct mechanisms in this thesis.

Definition 2.1.4. *A **direct mechanism** allows a single action to every agent, which is revealing the private information, so* $\Theta = A$ *in the mechanism.*

The definition of a mechanism is very general, but one common example of a mechanism is an auction. The actions available to agents in an auction are determined by the bidding rules, and the outcome function describes who should win the item or items given the bids. Bidders also have quasi-linear utilities since their utility depends on their value for winning an item and the payment they must make in exchange.

2.2 Several Properties of Mechanisms

The self-interested agents would like to obtain non-negative utility, and an auctioneer prefers to satisfy this need because agents can negotiate to refuse an allocation bringing negative utility in a real auction. Some auctioneers want to achieve fairness and learn the real private information about the agents' preferences through the agents' reports or bids, so that an agent with real need can obtain its desired resource. Some auctioneers also prefer to improve the social welfare, where the agents and the auctioneer care for the utility of the whole group.

Definition 2.2.1. *The **social welfare** of an outcome o is* $\sum_i u_i(o), \ \forall i \in \mathbb{N}$.

Two important related properties of mechanisms that we try to achieve are truthfulness and incentive compatibility.

Definition 2.2.2. *A mechanism $(\chi, \wp)$ is* ***truthful*** *if and only if, $\forall i \in A$, $\forall \hat{v}_i \in V_i$, and $\forall \hat{v}_{-i} \in V_{-i}$, $v_i - \wp_i(\chi(v_i, \hat{v}_{-i})) \;\geq\; v_i - \wp_i(\chi(\hat{v}_i, \hat{v}_{-i}))$, where A is the set of agents, v_i is the agent i's true type and V_i and V_{-i} are the strategy spaces for the agent i and other agents in A.*

Definition 2.2.3. *A mechanism $(\chi, \wp)$ is* ***incentive compatible*** *if and only if, $\forall i \in A$, $\forall \hat{v}_i \in V_i$, $v_i - \wp_i(\chi(v_i, v_{-i})) \;\geq\; v_i - \wp_i(\chi(\hat{v}_i, v_{-i}))$, where A is the set of agents, v_i is the true type of an agent i, v_{-i} is the real information of the other agents in A and V_i is the strategy space for the agent i.*

If a mechanism is truthful, every agent likes to reveal its true type to the auctioneer, no matter whether other agents submit their true types or not; if a mechanism is incentive compatible, every agent would like to submit its true type when all other agents submit their true types. The truthfulness is also called dominant-strategy incentive compatible; and it is obvious that the truthfulness is stronger than the incentive compatibility.

It is a very essential property in mechanism design that agents' true private information is revealed to the auctioneer. If every agent's utility is maximal when the agents submitting their true information, truth telling is a dominant strategy. In such a scenario, the agents do not have incentives to find another strategy to improve their utilities.

Because an agent receives a payment along with an allocation, the agent's utility can be negative if the payment is larger than the value. Therefore the auctioneer would like to compute the allocations with payments smaller than the agents' actual values for the items being allocated. This constraint is called *individual rationality.*

Definition 2.2.4. *A mechanism satisfies* ***individual rationality(IR)*** *if and only if $u_i = v_i(\chi(\hat{v})) - \wp_i(\chi(\hat{v})) \geq 0$, for any agent i.*

Nevertheless, another property *no-deficit* is about whether the sum of the agents' utilities is negative or not.

Definition 2.2.5. *A mechanism $(\chi, \wp)$ has* ***no-deficit*** *if $\sum\limits_i v_i(\chi(\hat{v})) - \sum\limits_i \wp_i(\chi(\hat{v})) \geq 0$.*

Note that IR is stricter than no-deficit, because the sum of all agents' utilities is non-negative when every agent's utility is non-negative. There is a similar property in the quasi-linear mechanism from the auctioneer's aspect, which is the *budget balanced.*

Definition 2.2.6. *A quasi-linear mechanism is* ***budget balanced*** *when $\sum_i \wp_i(s(v)) = 0$, where s is the equilibrium strategy profile.*

This is a very strict property; the mechanism would return the whole amount of values back to the agents after collecting those values, whatever the agents' types are. This property is suitable for an auctioneer who does not need any utility. There is a relaxed property *weak budget balanced*, which is $\sum_i \wp_i(s(v)) \geq 0$.

Finally, in the previous section we introduced the idea of a direct mechanism. In this thesis we will sometimes restrict ourselves to direct mechanisms. This is without loss of generality since the Revelation Principle[22] states that if you can achieve some outcome through a mechanism then you can also achieve the same outcome through a direct mechanism. The revelation principle provides an important reason for direct mechanisms' convenience to the agents.

Theorem 2.2.7. ***Revelation principle*** *If there exists a mechanism that selects an outcome such that the agents have dominant strategies, then there is a direct mechanism which selects the same outcome and the agents have dominant strategies.*

The revelation principle implies that we only need to consider direct mechanisms when looking at mechanisms. However, it ignores the fact that running the mechanism involves computation and thus the computational complexity of the mechanism is important.

2.3 Groves Mechanism Family

The Groves mechanisms are a family of efficient(i.e. they maximize social welfare) mechanisms for agents with quasi-linear utilities.

Definition 2.3.1. ***Groves mechanisms*** *are direct mechanisms $(\chi, \wp)$, and the agents submit their types $\hat{v}$ to the auctioneer; χ is the decision policy and $\wp_i$ is the payment policy to an agent i;*

$$\begin{aligned} \chi(\hat{v}) &= \arg\max_x \sum_i \hat{v}_i(x), \\ \wp_i(\hat{v}) &= h_i(\hat{v}_{-i}) - \sum_{j \neq i} \hat{v}_j(\chi(\hat{v})). \end{aligned}$$

$h_i(\hat{v}_{-i})$ is an arbitrary function that does not depend on the agent i's type. Under this family of mechanisms, the auctioneer chooses an outcome o that maximizes the sum of every agent's reported value. The payment to an agent i is the difference between the value of the h function and the sum of other agents' reported values. It has been shown that any mechanism that maximizes the social welfare (i.e. is efficient) when agents have quasi-linear utilities must be a Groves mechanism[22].

2.3.2 The VCG Mechanism

While there are many ways to instantiate the h function in the Groves mechanisms, one of the most well known instantiations is the Vickrey-Clarke-Groves (VCG) mechanism.

Definition 2.3.3. ***The VCG mechanism*** *is a mechanism $(\chi, \wp)$ inside the Groves mechanism family, for which*

$$\begin{aligned} \chi(\hat{v}) &= \underset{x}{argmax} \sum_i \hat{v}_i(x), \\ \wp_i(\hat{v}) &= \sum_{j \neq i} \hat{v}_j(\chi(\hat{v}_{-i})) - \sum_{j \neq i} \hat{v}_j(\chi(\hat{v})). \end{aligned}$$

The VCG mechanism uses $\sum_{j \neq i} \hat{v}_j(\chi(\hat{v}_{-i}))$ as the $h_i(\hat{v}_{-i})$ function, thus $h_i(\hat{v}_{-i})$ is the maximal social welfare that can be achieved by other agents if agent i did not exist. In the VCG mechanism, the utility of an agent i is $u_i(\hat{v}) = \sum_j \hat{v}_j(\chi(\hat{v})) - \sum_{j \neq i} \hat{v}_j(\chi(\hat{v}_{-i}))$, so every winning agent's utility is the marginal contribution. The VCG mechanism also satisfies *ex post* individual rationality.

The VCG mechanism chooses an allocation outcome that maximizes the social welfare, which is the optimal allocation. However the computation of the optimal allocation can be NP-hard, which can make the mechanism impractical.

2.4 A Matching Algorithm for Multiple Items

While there are various methods for a mechanism to select an allocation outcome, one method is using a matching. For example, an allocation can be seen as a matching between buyers and sellers in an auction. We provide some background on the matching algorithms used in our mechanisms.

The *Hungarian Algorithm* 2.4.1 by Kuhn and Munkres[13, 16] can compute the maximum weighted matching in a bipartite graph with $O(|V|^3)$ runtime, $|V|$ is the number of the vertices in the graph. The algorithm maintains a labeling function l for the vertices, $l : v \mapsto \mathbb{N}$. The weight $w(a, b)$ of an edge linking two vertices a and b is greater than 0, and $l(a) + l(b) \geq w(a, b)$, $\forall a, b \in V$.

Algorithm 2.4.1 Hungarian Algorithm

```
$\forall x \in X,\ l(x) \leftarrow 0$
$\forall y \in Y,\ l(y) \leftarrow \max_x w(x, y)$
$M \leftarrow \emptyset$

**while** $|M| \neq |V|/2$ **do**
    $S \leftarrow u,\ X \leftarrow X - u,\ T \leftarrow \emptyset,\ NL \leftarrow \emptyset$
    $\forall v \in S \textit{ and } y \textit{ with } l(v) + l(y) = w(v, y),\ NL \leftarrow NL \cup y$

    $a \leftarrow 0$
    **repeat**
        **if** $NL = T$ **then**
            $\alpha \leftarrow min_{x \in S, y \notin T} l(x) + l(y) - w(x, y)$
            $\forall v \in S,\ l(v) \leftarrow l(v) - \alpha$
            $\forall v \in T,\ l(v) \leftarrow l(v) + \alpha$
        **end if**

        $\exists y,\ y \in NL - T$
        **if** $y \notin Y$ **then**
            $\exists z,\ (z, y) \in M$
            $S \leftarrow S \cup z,\ T \leftarrow T \cup y$
        **else**
            $M \leftarrow M \cup (u, y),\ X \leftarrow X - u,\ Y \leftarrow Y - y$
            $a \leftarrow 1$
        **end if**
    **until** a = 1
**end while**
```

A different scenario frequently seen is a buyer can be matched to several items, which is a one-to-multiple matching instead of a one-to-one matching. But it is possible to transform a one-to-multiple matching problem into a one-to-one matching problem. We describe the transformation algorithms in the dynamic model section later in this thesis.

Chapter 3

Models Where Agents are Allocated At Most One Item at a Time

In this chapter we focus on an allocation model with one auctioneer and two types of agents, service providing agents and task agents. A service providing agent has a private value for any interested task agent; the task agents would arrive and depart in the setting. To simplify the following analysis, the service providing agents are called agents and the task agents are called tasks.

In the printing example in section 1.1, the agents are the printers and the tasks are the magazines to be printed. Since each printer can only work on a single magazine at a time, a restriction on the agents is emphasized in the model that an agent can be allocated to at most one existing task, which has arrived and not departed yet. We focus on a static model before analyzing the dynamic model. In this chapter we describe the agents, the tasks, the auctioneer and the mechanisms for the static setting and the dynamic setting respectively.

3.1 The Static Model and Solutions

We start by analyzing a static problem; in particular we assume that the tasks do not depart. The mechanism for this static circumstance is the same as an auction except we enforce a restriction on the allocation outcomes. The assumptions of the allocation problem under the static setting are:

1. An agent has a private cost for any interested task for completing that task by the agent; and an agent has its cost as 0 for a task which the agent is uninterested in.

2. An agent's private information contains the cost for its interested tasks.

3. A self-interested agent prefers a payment not smaller than the cost.

4. The auctioneer prefers to allocate a task to an agent with a small cost value.

5. The auctioneer would allocate at most one task to an agent. For example, since a printer can print one magazine at a time, the printing factory would assign at most a single magazine to a printer in the static setting.

Example 3.1.0.1. *Here is a static allocation problem of two agents and three tasks. Let a bid of an agent is a pair of (task, value) for an interested task. The bids known by the auctioneer are* $\{A1\ (T1, 3), (T2, 5)\}, \{A2\ (T1, 2), (T3, 4)\}$.

An allocation outcome is that the task $T1$ *is allocated to the agent* $A1$*, the task* $T3$ *is allocated to the agent* $A2$ *and the task* $T2$ *is not allocated. Another outcome is that the task* $T2$ *is allocated to the agent* $A1$*, the task* $T3$ *is allocated to the agent* $A2$ *and the task* $T1$ *is not allocated. The first outcome is better than the second one because the total cost is 7 in the first outcome while that of the second one is 9.*

3.1.1 The VCG Mechanism

The auctioneer would like to compute the outcome with the maximal efficiency under the assumptions in the VCG mechanism. Let A be the set of all agents and O be the outcome space. Let c^i be the cost information of an agent i. As the auctioneer prefers small cost, the allocation efficiency should not be defined as $\max_{o \in O} \sum_{i \in A} c^i(o)$; the efficiency cannot be defined as $\min_{o \in O} \sum_{i \in A} c^i(o)$ as well, because none of the tasks allocated is the optimal outcome under that definition.

Furthermore, in most cases the agents would like to obtain the tasks because there is a non-negative benefit for them. For example, the printing factory could receive some benefit from the magazine offices besides the cost, although the printers cannot receive any benefit. To apply the VCG mechanism, we assume an agent also has a positive benefit value besides the cost for every interested task, and the benefit value is the same for every task. Since the standard VCG mechanism uses an optimal algorithm that maximizes the social welfare, we substitute the cost with a virtual value that is the result of the positive benefit minus the cost. The virtual value for a cost c^i_j is $v^i_j = c_M - c^i_j$ where c_M is the positive benefit and is larger than all cost; then the social welfare in the VCG mechanism is computed as $\max_{o \in O} \sum_{i \in A} v^i(o)$.

Definition 3.1.2. *The **VCG** mechanism $(\chi, \wp)$ under the model with a restricted outcome space includes an allocation policy χ and a payment policy $\wp$ for the agents A where v is the virtual value with $v = c_M - c$, c is the agent private cost for tasks.*

The allocation policy is $\chi(\hat{c}) = \arg\max_{\chi} \sum_{i \in A} v^i(\chi(\hat{c}))$, $|\chi^i(\hat{c})| \leq 1$; $|\chi^i(\hat{c})|$ is the number of tasks allocated to the agent i. The payment policy for an agent i satisfies that $\wp^i(\hat{c}) = c_M - (\sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i})) - \sum_{j \in A, j \neq i} v^j(\chi(\hat{c})))$.

Lemma 3.1.3. *As every cost is smaller than c_M, $\sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i}))- \sum_{j \in A, j \neq i} v^j(\chi(\hat{c})) < c_M$ for an optimal allocation policy χ.*

Proof. Let us assume in some cases that, for an agent i, $\sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i})) - \sum_{j \in A, j \neq i} v^j(\chi(\hat{c})) \geq c_M$. Then

$$\begin{aligned} &\sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i})) - (\sum_{j \in A, j \neq i} v^j(\chi(\hat{c})) + v^i(\chi(\hat{c}))) \\ &\qquad = \sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i})) - \sum_{j \in A} v^j(\chi(\hat{c})) \\ &\qquad\qquad \geq c_M - v^i(\chi(\hat{c})) > 0. \end{aligned}$$

Therefore $\sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i})) > \sum_{j \in A} v^j(\chi(\hat{c}))$; this is impossible because χ is an optimal policy. So in any case, $\sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i}))- \sum_{j \in A, j \neq i} v^j(\chi(\hat{c})) < c_M$. $\square$

Theorem 3.1.4. *The VCG mechanism is truthful for agents in the static model.*

Proof. Because every agent can only obtain one task, $c^i(\chi(\hat{c})) < c_M$; and as $\wp^i(\hat{c}) < c_M$ as well, an agent i's utility by the VCG mechanism $(\chi, \wp)$ is $u^i(\hat{c}) = \wp^i(\hat{c}) - c^i(\chi(\hat{c})) = (c_M - c^i(\chi(\hat{c}))) - (c_M - \wp^i(\hat{c})) = v^i(\chi(\hat{c})) - (c_M - \wp^i(\hat{c}))$.

According to the definition of the VCG's payment scheme, $u^i(\hat{c}) = v^i(\chi(\hat{c})) - (\sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i})) - \sum_{j \in A, j \neq i} v^j(\chi(\hat{c}))) = v^i(\chi(\hat{c})) + \sum_{j \in A, j \neq i} v^j(\chi(\hat{c})) - h(-i) = \sum_{j \in A} v^j(\chi(\hat{c})) -$

$h(-i)$, with $h(-i) = \sum_{j \in A, j \neq i} v^j(\chi(\hat{c}_{-i}))$, $h(-i)$ is irrelative to the agent i's value or the agent's announced value.

c^i is the agent i's true cost. Let c_a^i be an alternative report of c^i; the self-interested agent i would submit c_a^i to the auctioneer for a larger utility. There are two outcomes o_1 and o_2, $o_1 = \chi(c^{-i}, c^i)$ and $o_2 = \chi(c^{-i}, c_a^i)$. The utilities based on the value reports c^i and c_a^i are $u^i(o_1) = v^i(o_1) + \sum_{j \in A, j \neq i} v^j(o_1) - h(-i) = \sum_{j \in A} v^j(o_1) - h(-i)$, $u^i(o_2) = v^i(o_2) + \sum_{j \in A, j \neq i} v^j(o_2) - h(-i) = \sum_{j \in A} v^j(o_2) - h(-i)$.

Because χ is an optimal allocation policy and the algorithm's result is o_1 when the agent i submit its true cost, $\sum_{j \in A} v^j(o_1) \geq \sum_{j \in A} v^j(o_2)$; then $u^i(o_1) \geq u^i(o_2)$. Therefore no matter what c^{-i} is, reporting the true cost is the dominant strategy for the agent i. $\square$

The allocation policy is an optimal algorithm, and since every agent obtains no more than one task, the allocation policy can be implemented by a matching algorithm, such as Algorithm 2.4.1. There always exists a bipartite graph, where the nodes on one side are for the agents and the others are for the tasks; suppose v_j^i is the agent i's virtual value for a task j, the corresponding edge's weight equals to v_j^i. Algorithm 2.4.1 can compute an optimal matching with the computational complexity $O((m+n)^3)$, m is the number of agents and n is the number of tasks. Then the computational complexity of the allocation policy is also $O((m+n)^3)$.

An optimal allocation among the agents A/i for every agent i is needed to be computed through the payment policy. So the computational complexity of the payment policy is $O(m \times (m+n)^3)$.

3.1.5 NaiveGreedy Mechanism

We would like to discover a mechanism with lower computational complexity in the static environment. The mechanisms we propose later are directly based on the cost, so the value in following mechanism means the cost. The auctioneer can construct a *value matrix* that contains the values from all agents to compute the social welfare more conveniently. Then a very simple allocation scheme χ allocates from the lowest bid, as Algorithm 3.1.2 shows; the value matrix is recomputed after every allocation. Let v^{min} be the minimum of the values except the winning value in that round, the payment for the winning agent is v^{min}. So the NaiveGreedy mechanism's *payment policy* at a round r is simply $\wp_r = \min\limits_{i,j} v_j^i$ with $j \neq \chi_r^i(\hat{v})$.

Definition 3.1.6. *The value matrix V is a $m \times n$ matrix, assuming there are m agents and n tasks in the market; and v^i_j is the value of the agent i for the task j.*

$$V = \begin{bmatrix} v^1_1 & \cdots & v^1_n \\ v^2_1 & \cdots & v^2_n \\ & \vdots & \\ v^m_1 & \cdots & v^m_n \end{bmatrix}, \text{ where } v^i_j = \begin{cases} v, \ v > 0; \\ 0, \ if\ the\ agent\ i\ is\ uninterested\ in\ the\ task\ j. \end{cases}$$

Example 3.1.6.1. *There are three agents and three tasks. The agents' bids are {A1 (T1, 3), (T2, 5)}, {A2 (T1, 2), (T3, 4)}, {A3 (T1, 3), (T2, 6), (T2, 3)}. So the value matrix is*

$$V = \begin{bmatrix} 3 & 5 & 0 \\ 2 & 0 & 4 \\ 3 & 6 & 3 \end{bmatrix}$$

Algorithm 3.1.2 NaiveGreedy allocation scheme(pseudocode)

```
$i \leftarrow size\ of\ agents,\ j \leftarrow size\ of\ tasks$
$v \leftarrow value\ matrix$
**while** $i\ != 0\ \ \&\&\ \ j\ != 0$ **do**
    1) $m, n \leftarrow \underset{m,n}{argmin}\ v^m_n$, and break ties randomly
    2) allocate task $n$ to agent $m$
    3) remove $m^{th}$ row and $n^{th}$ column from $v$
    4) $i \leftarrow i - 1,\ j \leftarrow j - 1$
**end while**
```

The mechanism has multiple rounds. Because at least one task is allocated at a round, the computational cost of the mechanism is $m \times n + (m-1) \times (n-1) + \cdots = O(mn(m+n))$ $= O(m^2n + mn^2)$ in the worst case, m is the number of agents and n is the number of tasks. The NaiveGreedy mechanism's computational complexity is better than that of the VCG mechanism.

Example 3.1.6.2. *Here is an example of applying the NaiveGreedy mechanism. The first table in 3.1 is a value matrix of four agents {A,B,C,D} and three tasks {1,2,3}. The allocation is as following.*

- *First round: The auctioneer finds the value of agent B on task 1 is the lowest, so it allocates the task 1 to agent B. The payment for agent B is 1. Then the value matrix is reduced to the second table in 3.1.*

	1	2	3
A	5	6	3
B	1	1	2
C	3	2	1
D	4	5	7

(1)

	2	3
A	6	3
C	2	1
D	5	7

(2)

	2
A	6
D	5

(3)

Table 3.1: Value matrices for the NaiveGreedy allocation scheme

- *Second round: the auctioneer finds the value of agent C on task 3 is the lowest and allocates task 3 to agent C. The payment for agent C is 2, and the value matrix changes into third table in 3.1.*
- *Third round: Agent D obtains the task 2 because it has the lowest value in the matrix, and the payment for agent D is 6.*

We now prove some properties for the mechanism.

Proposition 3.1.7. *The NaiveGreedy mechanism is* **individually rational**.

The allocation algorithm would let the auctioneer choose the bid with smallest value v_j^i among the remaining bids, so any value from another agent in the value matrix would not be larger the the bid's value chosen; so the payment is not smaller than the value.

Proposition 3.1.8. *The NaiveGreedy mechanism is not* **strategy-proof**.

In the example above, the agent B can submit its cost values as 2 for both task 1 and task 2, then it would obtain the task 1 with the payment as 2. Its original utility is $1 - 1 = 0$, but the utility afterwards is $2 - 1 = 1$; so the agent B can obtain a larger utility by submitting fake private information.

This mechanism does not always maximize the social welfare as well. Example 3.1.6.2 is also a counterexample, because the optimal allocation is (Agent $A \leftrightarrow$ task 3, Agent B $\leftrightarrow$ task 1, Agent $C \leftrightarrow$ task 2).

3.1.9 A Strategy-proof Multi-round Mechanism

The NaiveGreedy mechanism is not strategy proof, but using the multiple rounds can reduce the computational complexity compared to the standard VCG mechanism. In this section we propose a new mechanism which is more complicated than the NaiveGreedy mechanism, but is incentive compatible while keeping the computational complexity reduced.

The Multi-round mechanism would complete all allocations in multiple rounds. At every round r, the allocation scheme χ and the payment scheme $\wp$ for any remaining agent and any remaining task is shown in Algorithm 3.1.3.

Algorithm 3.1.3 The Multi-round mechanism(pseudocode)

Input: The value matrix v
Output: The allocation result χ and the payment $\wp$

```
$m \leftarrow size\ of\ agents,\ n \leftarrow size\ of\ tasks$
$\chi \leftarrow \{\chi_1 \dots \chi_m\},\ \chi_i \leftarrow NULL,\ \forall i = 1 \dots m$
$\wp \leftarrow \{\wp_1 \dots \wp_m\},\ \wp_i \leftarrow NULL,\ \forall i = 1 \dots m$
$h \leftarrow \{h_1 \dots h_n\},\ h_i \leftarrow NULL,\ \forall i = 1 \dots n$

$allocateagents \leftarrow 0,\ allocatetasks \leftarrow 0$
while $allocateagents\ != m\ \&\&\ allocatetasks\ != n$ do
  $allocate \leftarrow \{allocate_1 \dots allocate_m\},\ allocate_i \leftarrow NULL\ \forall i = 1 \dots m$
  $payment \leftarrow \{payment_1 \dots payment_m\},\ payment_i \leftarrow NULL\ \forall i = 1 \dots m$

  for $t = 1 \rightarrow n$ do
    if $\chi_{h_t}\ != t$ then
      $winagent \leftarrow argmin_{a, \chi_a == NULL}\ \{v_t^a\}$
      $paymentemp \leftarrow min_{a \neq winagent,\ \chi_a == NULL}\ \{v_t^a\}$
      if $h_t\ != NULL$ then
        $temphp \leftarrow \wp_{h_t} - v_{\chi_{h_t}}^{h_t} + v_t^{h_t}$
        if $paymentemp < temphp$ then
          $paymentemp \leftarrow temphp$
        end if
      end if
      if $allocate_{winagent}\ != NULL$ then
        $u_1 \leftarrow payment_{winagent} - v_{allocate_{winagent}}^{winagent}$
        $u_2 \leftarrow paymentemp - v_t^{winagent}$
        if $u_2 > u_1$ then
          $allocate_{winagent} \leftarrow t$
          $payment_{winagent} \leftarrow paymentemp$
```

```
            end if
        else
            $allocate_{winagent} \leftarrow t$
            $payment_{winagent} \leftarrow paymentemp$
        end if
        $h_t \leftarrow winagent$
    end if
end for

for $i = 1 \rightarrow m$ do
    if $allocate_{winagent} \ != NULL \ \&\& \ payment_{winagent} \geq v^{winagent}_{allocate_{winagent}}$ then
        $\chi_{winagent} \leftarrow allocate_{winagent},\ \wp_{winagent} \leftarrow payment_{wingent}$
        $allocateagents \leftarrow allocateagents + 1,\ allocatetasks \leftarrow allocatetasks + 1$
    end if
end for
end while

return $\chi$, $\wp$
```

According to the allocation scheme, the auctioneer chooses an agent with the lowest cost for every task. The payment for the winners in the first round is the second lowest value; in other words, the payment for an agent i winning the task t is $\wp^i_t = \underset{a}{min}\ v^a_t$ with $a \neq i$ and $\chi_a = \emptyset$. After the first round, every remaining task j is allocated to a remaining agent i with the smallest cost value, and the utility of the agent winning a task in the previous round would be an upper bound of the winner's utility in the current round. The mechanism breaks ties randomly; and the allocated items and winning agents are removed after allocating. If an agent has the smallest values for multiple remaining tasks, a self-interested agent is assumed to choose the task with the largest profit; the agent would choose a task $(t_j,\ payment_j)$ from $\{(t_1,\ payment_1),\ (t_2,\ payment_2)\ ...\ (t_n,\ payment_n)\}$, with $payment_j - cost_j \geq payment_k - cost_k$ for $\forall k \neq j$. The auctioneer checks if there are unallocated tasks and available agents, and moves to next round if there are at least one agent and one task left in the market.

Example 3.1.9.1. *Here is an example in Table 3.2 using the mechanism to allocate four tasks to four agents.*

In the first round, the auctioneer finds that the agent B wins on task 1 and task 2, with payment 3 for task 1 and payment 2 for task 2; and also the agents C wins task 3 and task

	1	2	3	4
A	5	2	6	4
B	1	1	2	3
C	3	2	1	2
D	4	5	7	3

Table 3.2: A value matrix for the Multi-round mechanism

4, with payment 2 for task 3 and payment 3 for task 4. The auctioneer asks agent B with the two tasks and their payments, and similarly asks the agent C. The agent B prefers task 1 and the agent C prefers task 3. Then the auctioneer allocates task 1 to the agent B with payment 3, task 2 to the agent A with payment 3, task 3 to the agent C with payment 2, and task 4 to the agent D with payment 4.

3.1.9.1 Analysis of the Multi-round Mechanism

Lemma 3.1.10. *If an agent k obtains two tasks n and j in some round r_1 and chooses the task j, the task n is allocated to an agent m in a later round r_2, with $r_2 > r_1$, then the agent k's utility is not smaller than that of the agent m, i.e. $u^k \geq u^m$.*

Proof. The agent m wins a task in a round $r > 1$, so its payment is $\wp^m = \wp^k - v_j^k + v_n^k$. Then agent m's utility

$$\begin{aligned} u^m = \wp^m - v_n^m \ &\leq \ \wp^k - v_j^k + v_n^k - v_n^m \\ &= \ \wp^k - v_j^k - (v_n^m - v_n^k) \\ &\leq \ \wp^k - v_j^k \ (as\ v_n^m \ \geq \ v_n^k) \\ &= \ u^k. \end{aligned}$$

□

Theorem 3.1.11. *The Multi-round mechanism is incentive compatible.*

Proof. First, we prove that a winning agent cannot submit fake information to obtain the same task at a later round for a larger utility.

Let an agent i win multiple items and choose an item n with the largest utility at the round r_1, when i submit its true type to the auctioneer. The agent i may try to submit false values to obtain n at a later round r_2; and consequently n would be allocated to an agent m with a larger cost at r_1. The agent m does not choose n at r_1 so that n is left after the round r_1.

Let sv_n^r denote the second lowest value for a task n at any round r. At the round r_1, $u^m = min\{u_n^k,\ sv_n^{r_1} - v_n^m\}$, k is the winner of n before r_1. Because the agent i is still in the market, its value $\hat{v}_i(n)$ for task n revealed to the auctioneer should satisfy $\hat{v}_i(n) \geq sv_n^{r_1}$. In a later round r_2 where i wins the item n, i's payment would be $\wp_n^i \leq u_n^m = min\{u_n^k,\ sv_n^{r_1} - v_n^m\}$, so $\wp_n^i \leq \hat{v}_n^i - v_n^m$; then $\hat{u}_n^i = \wp_n^i - \hat{v}_n^i \leq \hat{v}_n^i - v_n^m - \hat{v}_n^i < 0$ at the round r_2. The agent i cannot obtain the item n according to the allocate scheme. Therefore the agent i would not lie, otherwise either the agent cannot obtain the item again, or the utility of obtaining the same item in a later round would be smaller, according to Lemma 3.1.10.

We now prove that an agent i would not submit false values to obtain a different task. Suppose an agent i has a bid for the task j, but the task j is supposed to be allocated to the agent m. There are four possible circumstances where the agent i is looking for a strategy to obtain the task j.

Agent	i	m
Allocated Task	n	j

case 1

Agent	i	m	k
Allocated Task	n	j	q

case 2

Agent	i	m
Allocated Task	∅	j

case 3

Agent	i	m	k
Allocated Task	∅	j	q

case 4

Table 3.3: Four circumstances where an agent i is possible to submit false values to obtain a different task j

1. In the case 1, the agent i is supposed to obtain the task n, but since $u_j^i > u_n^i$, the agent i may submit false information to obtain the task j. The task j is supposed to be allocated to the agent m.

   As the agent m should obtain the task j when all agents submit true values, $v_j^m \leq v_j^i$. Thus the agent i should submit a value $\hat{v}_j^i < v_j^m$ to win. Then the payment for $\hat{v}_j^i$ is v_j^m, and the agent i's utility of the task j is changed to be $u_j^i = v_j^m - v_j^i < 0$. Therefore, the agent i cannot obtain a larger utility by submitting false values.

2. In the case 2, the agent i prefers the item j than its allocation n. The last winner of the task j before m is the agent k; and the agent k is supposed to obtain the task

q. So the agent i should submit a false value $\hat{v}_j^i < v_j^m$ to win. Because the payment scheme would choose the smaller value from $\wp_q^k - v_q^k + v_j^k$ and v_j^m, the agent i would receive the payment $\wp_j^i \leq v_j^m$. Its utility would be negative; so the agent i cannot obtain a larger utility by submitting false values.

3. In the case 3 and 4, the agent i does not receive any allocation if it submits true values. For any desired item j, the agent i should submit a smaller value than v_j^m to win, and then receives a payment that is smaller than agent i's cost value for task j. Therefore, the utility of the agent i is always negative if the agent makes a false declaration for winning a task.

Therefore, an agent without any allocation would not submit false information to obtain an allocation, and an agent with an allocation also would not submit false values to switch for another allocation.

□

Proposition 3.1.12. *The Multi-round mechanism is individually rational.*

Since the allocate scheme would allocate an item with the payment larger than the cost, the payment given to the agent would always be larger than the winner's cost value. The IR property of this mechanism is straight forward.

Proposition 3.1.13. *The Multi-round mechanism is not group strategy-proof.*

A counterexample for the proposition is a game with an auctioneer and two agents. When there are only two agents, they can always cheat in a group as the payment to an agent depends on the other agent's cost. The VCG mechanism is not strategy-proof as well, because the payment of an agent depends on the other agents' values and all agents can form a group to improve their payments. When all agents in the game would like to form a coalition, they can cheat as the payment depends on the others' cost values.

Proposition 3.1.14. *If an agent has the lowest or second lowest cost value for a task and is not in any group, the allocation of that task would not be affected by any non-truthful groups.*

Proof. But when an agent with the lowest cost for a task is not in any group and submits its true information, other agents cannot make a lower offer for this task which would decrease their utility to be non-positive; other agents would not submit fake values higher than their true values for this task as the fake values do not make them obtain the task. Therefore, the allocation and the payment are not changed by any non-truthful agents in this case.

When an agent with the second lowest cost for a task is not in any group and submits its true information, the payment of that task would not be changed by fake reports; this is because that the group which the winning agent belongs to cannot provide a higher payment. The allocation would not be changed as well, because the agents with higher cost would not lie to receive a payment lower than their cost. Therefore, if an agent has the lowest or second lowest cost value for a task, and it is not in any group, the allocation of that task would not be affected by any non-truthful groups. □

The payment scheme is important for the mechanism's incentive compatibility. If the payment scheme changes to the second lowest value, the mechanism is no longer strategy-proof, even if there is a bonus payment. In such a case, if an agent i wins a task j in the first round, the payment of i is $\wp_j^i = min_{a \neq i} v_j^a$; and if agent i wins in a round later, the payment is $\wp_j^i = min_{a \neq i}\ v_j^a + 1$. Although the agents cannot obtain an allocation by submitting false information, an agent can obtain some additional payment by being non-truthful.

Suppose there are four tasks $1 - 4$ and four agents $A - D$ with the value matrix in Table 3.4. Each of the agents B and C has the right to choose its allocation from two tasks. Let tasks 1 and 4 are chosen by agent B and C. Then the agent A obtains task 3 with payment 3. But the agent A could submit its cost value to be 3 for task 1, such that it would receive a larger payment 4. This circumstance violates the truthful property that agents would obtain the best payment by submitting their true information.

	1	2	3	4
A	5	3	2	6
B	4	2	4	7
C	6	3	1	3
D	6	4	4	5

Table 3.4: A value matrix for the second lowest price payment

Let m be the number of agents and n be the number of tasks. The computational complexity of the allocation scheme is $O(m)$ in every round, and the payment scheme's computational complexity is $O(1)$. There is at least one allocation in every round, so the number of rounds is $O(n)$. Thus the Multi-round mechanism's computational complexity is $O(mn)$.

3.2 The Dynamic Model and Solutions

In the previous section, we looked at a setting where agents and tasks are all in the market at the same time. In this section we look at a dynamic setting where tasks arrive and depart. We also assume that when allocating a task, the winning agent must process the task which takes a finite amount of time. During this processing period, the agent is not available to take on new tasks.

The allocation problem has the assumptions:

1. If an agent is processing a task, it cannot be assigned any other task until the allocated task is completed, i.e. the number of uncompleted tasks for an agent is ≤ 1.
2. The private information of agents includes the value and the processing time.
3. The auctioneer prefers that the winning agents have the maximum values, and the agents would like to receive the payment that is not larger than the value.

Example 3.2.0.1. *Here is an example for the model within three time periods in Figure 3.1. There are an auctioneer, two agents $A1, A2$ and five tasks $T1 - T5$.*

A1	{T1, value 3, 2}, {T2, value 4, 3}	T1 [0, 5)	Time period 0
A2	{T1, value 5, 2}^{allocated,processing}	T2 [0, 3)	
		T1 [0, 5)	
A1	{T2, value 4, 3}, {T3, value 6, 3}^{allocated,processing}	T2 [0, 3)	Time period 1
A2	{T1, value 5, 2}^{allocated,processing}	T3 [1, 4)	
		T2 [0, 3)	
A1	{T2, value 4, 3}, {T3, value 6, 3}^{allocated,processing}	T3 [1, 4)	Time period 2
A2	{T1, value 5, 2}^{completed}, {T5, value 2, 1}	T4 [2, 6)	
		T5 [2, 4)	

Figure 3.1: An example within the restricted outcome space

In the time period 0, tasks $T1$ and $T2$ arrive. The agent $A1$ submits value 3 and processing time 2 for $T1$, which means its value of $T1$ is 3 and its processing time of $T1$ is *two periods; $A1$ also announces that its value of $T2$ is 4 and its processing time of $T2$ is 3. The agent $A2$ submits its value of $T1$ is 5 and the processing time is 2. $T1$ will depart at the time period 5, so if it is allocated, it should be completed before the time period 5. $T2$ will depart at the time period 3, so any winning agent need to complete $T2$ at the time period 2 at the latest. In the example, $T1$ is allocated to $A2$, and $A2$ begins to process $T1$ from period 0.*

In the next period, task $T3$ arrives, and it will depart at period 4. $A1$ submits its value and processing time for $T3$; and the auctioneer allocates $T3$ to $A1$. The agent $A2$ continues to process T1 at this time period, and does not know the new task.

In the time period 2, tasks $T4$ and $T5$ arrive. Because $T1$ is completed, the agent $A2$ submits its value and processing time for $T5$, while agent $A1$ continues to process $T3$.

3.2.1 Service Providing Agents

The agents stay in the auction market for the whole time horizon, and bid for any interested available task. Besides the value, an agent has a *processing time* in its private information for a task. When the agent obtains a task, it begins to process the task. During the processing, it is assumed that the agent would not submit new bids to the auctioneer. After the *processing time* periods have passed, the agent returns to the market.

Because the tasks in the dynamic environment arrive at different time periods, the agents would like to maximize their total expected utility. An assumption, made when analyzing the dynamic environment, is that every agent can only obtain one task in a time period, and only after the specified departure time period, the agent can have a chance to obtain a task again. Every agent knows the number of time periods for processing a task, but the agents do not have preferences on which specific time periods to obtain a task.

As the static case, an agent's type is its private information, which includes the values and the processing time. A bid b_j^i of an agent i contains i's value on j, and i's processing time on j. An agent has a set of bids, including the previous allocated bids, the currently processing bid and other bids for tasks that have not departed.

3.2.2 Tasks

The tasks arrive and depart at some specific time periods. A task's type includes the time period it arrives and the time period it departs. The task exists from the arrival time period and expires at the departure time period. A task should be finished before the departure time period when it is allocated to an agent. We consider two cases of new tasks to the auctioneer and the agents; in the first case, the auctioneer and agents do not know the arrival and departure time of any future task; the auctioneer and the agents do know some future tasks in the second case.

3.2.3 The Auctioneer

The auctioneer knows which agents are busy and when the busy agents will be available again. At every time period, the auctioneer only collects bids from the accessible agents. New tasks are revealed to agents by the auctioneer. Then the accessible agents submit their values and processing time periods for those tasks to the auctioneer.

The auctioneer also has all tasks' information. Since there are two cases of the tasks, the tasks arrived before the agents submit bids for them in the first case; in the second case, the tasks known by the auctioneer can be divided into the arrived tasks and other tasks that are not available but will arrive at a future time period.

3.2.4 Algorithms for Allocation Schemes

We first look at optimal algorithms for allocation that the auctioneer could use. Complete mechanisms are presented later.

As an agent could have bids for different time periods, some tasks can be allocated to the same agent if they exist in different time durations. Besides, a task can only be allocated to a single agent. The tasks which can not be allocated to the same agent are considered to be in conflict; and tasks that can be allocated to a single agent are considered to be non-conflicts.

We look at the bipartite graph again, which includes two sides of nodes and weighted edges. In a bipartite graph for the allocation problem, every node on one side is for an agent and every node on the other side is for a task; an edge linking two nodes a and b represents the private information of an agent for a task; and the weight of an edge is the value of an agent to a task. The matching in the bipartite graph is a set of edges where the edges do not share neither of their vertices, and these edges are considered non-conflicts.

Lemma 3.2.5. *An allocation problem could be transformed into a matching problem in the bipartite graph, if in every agent bids' set, either the bids do not conflict with each other, or every bid conflict with the other bids. In the graph, the nodes at one side represent agents and the other side's nodes represent tasks, and the weight of an edge is the value of a bid.*

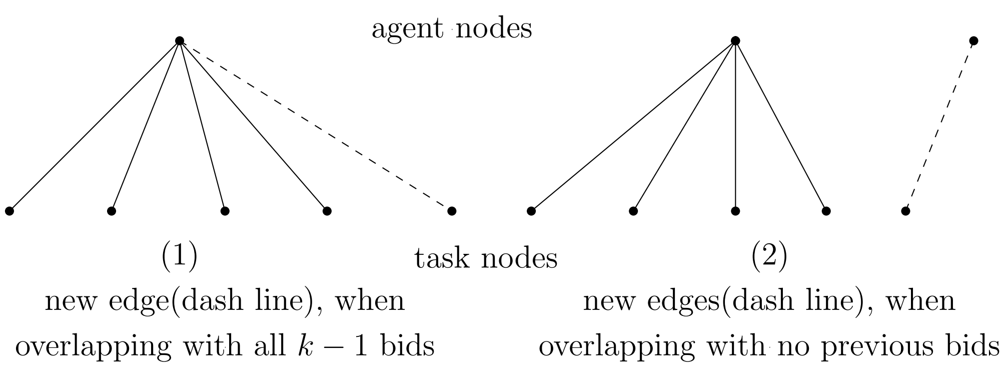
agent nodes
task nodes
(1)
new edge(dash line), when
overlapping with all $k-1$ bids
(2)
new edges(dash line), when
overlapping with no previous bids

Figure 3.2: An appending graph for the k^{th} new bid

Proof. A bid is transformed into an edge of the graph in the following recursive way. Let k be the size of bids' set for an agent i at a time period.

Case 1 $k = 1$: A node in the bipartite graph represents the agent i, and there is an edge from this node to a node representing a task. The task is allocated to the agent i in the optimum allocation whenever the edge is in the optimal matching.

Case 2 $k > 1$: Suppose the $k-1$ bids of the agent i have been transformed into the bipartite graph. Then there are two situations. New edges for the k^{th} bid can be added into the graph as Figure 3.2 shows.

situation 1 The k^{th} bid overlaps with all of the other $k-1$ bids on time. Then the new edges for the bid are those that link every node representing the agent to the task node, because a matching should not contain edges for both of the k^{th} bid and any of the other $k-1$ bids, and the matching should not select multiple edges for the k^{th} bid as well. So whenever any of the k^{th} bid's edges is in the optimal matching, the k^{th} bid is in the optimal allocation.

situation 2 The k^{th} bid does not overlap with any of the other $k-1$ bids on time. Then a new node is added, with an edge to the task node; and this node is added into the set of nodes that represent the agent i. So whenever the new edge is in the optimal matching, the k^{th} bid is in the optimal allocation.

The k^{th} bid can be combined into the bipartite graph when previous $k-1$ bids are transformed. In the transforming, no extra task nodes are added. Thus the bids of other agents can adopt the same task nodes. According to the above method, any bid from an

agent: \ *task:*	1	2	3
a	5	3	4
b	4	6	$\emptyset$
c	$\emptyset$	5	2

Table 3.5: A value matrix for the transformation algorithm

agent is able to be represented in a bipartite graph; and the agents and tasks are on the different sides.

Let a, b be two edges in the graph and i, j be the corresponding bids.

If an allocation mechanism should not allocate i and j at the same time, then either i and j are bids for the same task from different agents, or i and j are from the same agent and overlap on time. When i and j are bids from the same task, edges a and b share the same task vertex; so a and b cannot be in the same matching. When i and j are from the same agent but overlap on time, all bids from that agent conflict, as the situation 1 defines. Thus a and b share a single agent node; they cannot be in the same matching as well. So if two bids conflict, the corresponding edges cannot be in a matching at the same time.

When two edges a and b are not able to be in the same matching, they share a node in the graph. If the edges share a task node, the bids are from different agents for the same task; if the edges share an agent node, the bids overlap on time and are from the same agent. So, if two edges cannot be in the same matching, the corresponding bids conflict.

Therefore, any set of bids, where the bids can be allocated at the same time, can be transformed into a matching in the constructed bipartite graph. The problem of computing the optimal allocation is transformed to an optimal matching problem. □

Example 3.2.5.1. *Here is an example of applying the transformation algorithm. There are three tasks* $\{1, 2, 3\}$ *and three agents* $\{a, b, c\}$*, and the value matrix is in Table 3.5.*

The transformation algorithm produces a bipartite graph G in Figure 3.3; the optimal matching is $\{(a, 3),\ (b, 2),\ (c, 1)\}$.

The transformation algorithm could simply transfer an allocation problem when the agents do not submit a bid overlapping with a part of the other bids. It could be used in the situation where the agents do not know any task arriving in the future.

When the agents do not know the information of future tasks, this transformation algorithm constructs a bipartite graph with $2 \times max(m, n)$ nodes for an allocation problem where there are m agents and n tasks. Let k be the number of the agents' bids and k_i be the

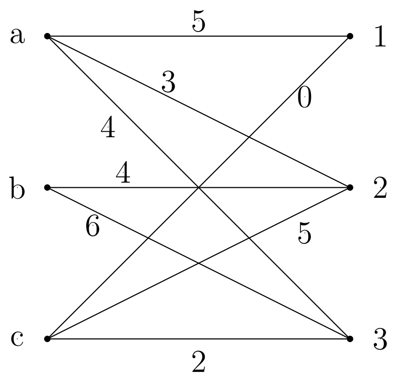

Figure 3.3: A bipartite graph generated from the allocation problem

number of an agent i's bids. The computational cost of appending a bid for an agent i is the number of the previously appended bids at most, so the computational cost of constructing the agent i's graph part is $T(k_i) = T(k_i - 1) + k_i = O(k_i^2)$. Then constructing a graph for the allocation problem costs $O(k_1^2 + k_2^2 + \cdots + k_m^2) = O(k^2)$, where $k_1 + k_2 + ... + k_m = k$. The transformation algorithm's computational complexity is $O(k^2)$. As $k = O(mn)$, the computational complexity is also $O(m^2n^2)$. Because solving the optimal matching problem costs $O((m + n)^3)$, the computational complexity of computing the optimal allocation is $O((m + n)^3 + m^2n^2)$.

There is another transformation algorithm that can convert any allocation problem at a single time period to a matching problem in a bipartite graph, if the bipartite graph exists. This second transformation algorithm does not require that every agent cannot submit a bid not overlapping with the other bids of the same agent; the allocation problem with overlapping bids from the same agent can also be transformed.

The second transformation algorithm constructs a bipartite graph where the nodes do not represent agents or tasks any more and an edge includes the information of a bid's value from an agent to a task and which agent submitting the value for which task for the bid. Let i, j be two bids, l_i, l_j are the vertices of i, j on the left side and r_i, r_j are the vertices of i, j on the right side. The conflict between i, j can be transformed into the condition that the edges share a vertex; $l_i = l_j \ \wedge \ r_i = r_j$ if i conflicts with j. If two bids do not conflict, their edges do not share a vertex; $l_i \neq l_j$ & $r_i \neq r_j$ if i does not conflict with j.

Theorem 3.2.6. *An allocation problem in a time period can be transformed into the optimal matching problem in a bipartite graph, if there exists a bipartite graph where every* *matching corresponds to an allocation outcome.*

Proof. Suppose there are k bids at a time period t.

If $k = 1$, the allocation problem could be transformed into a graph in Figure 3.4. There is an edge $[1, 1]$ with value v to represent the bid. If an allocation scheme chooses this bid, the matching result would be $\{[1, 1], [2, 2]\}$; and if the allocation scheme does not choose the bid, the matching result is $\{[1, 2], [2, 1]\}$.

If $k > 1$, we first assume that previous $k-1$ bids have been transformed into a bipartite graph with nodes' size less than an integer s. So each vertex of the edge for the k^{th} bid is in the set $N = 1, 2 \ldots s + 1$. The previous $k - 1$ bids can be divided into two groups: $S1$ and $S2$. $S1$ contains the bids which conflict with the k^{th} bid, and $S2$ contains the others which do not conflict with the k^{th} bid.

Then the vertices l_e, r_e of the k^{th} bid's edge e are chosen according to the two conditions:

- $\forall j \in S1$, either $l_j = l_e$ or $r_j = r_e$, but e and j cannot have both of their vertices be the same.
- $\forall j \in S2$, $l_j \neq l_e$ and $r_j \neq r_e$.

A new edge $[l_e, r_e]$ with the weight equivalent to the k^{th} bid's value is added into the graph, where l_e, r_e satisfy the two conditions and the edge $[l_e, r_e]$ does not already exist in the graph for previous bids.

No matter whether the $k - 1$ bids conflict with the k^{th} bid, the allocation scheme can exclude the k^{th} bid. Therefore after all bids are transformed into edges, a mask edge with no value is added for every vertex v so that there can be a matching containing an edge not related to any bid for this vertex. As a mask edge should only conflict with the edges on the vertex v, the other vertex of the mask edge should be a new node with this single mask edge.

In the matching result, $l_a \neq l_a$ and $r_a \neq r_a$ for any edge a and b, so the two conditions make sure that the edges in any matching are derived from a set of non-conflicting bids. Any matching can be a valid allocation outcome.

A valid allocation outcome allocates all bids in the set U, which is a subset of the set S that consists of all bids. An edges' set E is constructed as $\forall i \in U$, adding the edge $[l_e, r_e]$ to E, and $\forall j \in S \backslash U$, adding the mask edge for j to E. Because the bids for the allocation should not conflict, and a mask edge does not conflict with any of the other bids' edges, every edge $[a, b] \in E$ should have $l_a \neq l_b$ and $r_a \neq r_b$. Therefore, the set E is a matching

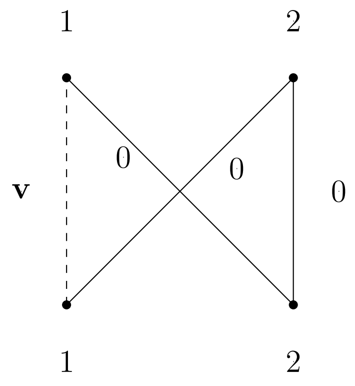

Figure 3.4: A bipartite graph for one bid

in the graph. As there is only one edge in the graph representing a bid, any matching containing E is equivalent to the allocation outcome.

□

Example 3.2.6.1. *This is an example of the second transformation algorithm. There are three bids b_1, b_2 and b_3 at a time period; b_1 conflicts with b_2 and b_3, b_2 conflicts with b_1, b_3 conflicts with b_1. Then the allocation problem for the three bids could be transformed into the graph in Figure 3.5. The real lines are the edges for the bids, and the broken lines are the mask edges. The edges $[1,1]$ and $[1,2]$ share a vertex because b_1 conflicts with b_2; and the edges $[1,1]$ and $[2,1]$ share a vertex because b_1 conflicts with b_3.*

If the agents have information for any future time period, the bids from the same agent may not conflict. The second transformation algorithm constructs a graph so that every possible allocation can be a matching. The algorithm can construct a graph with at most $4k$ nodes, k is the number of agents' bids in total. Let m be the number of agents and n be the number of tasks, the number of nodes in the transformed graph is $O(4k) = O(mn)$.

This transformation algorithm finds a pair (l_e, r_e) among $O(k^2)$ alternatives, and for every alternative, the algorithm checks the conflict status with the previous $k-1$ edges. The computational complexity of the transformation algorithm is $O(k^3) = O(m^3n^3)$. Because computing the optimal matching in the graph costs $O(m^3n^3)$, the computational complexity of the algorithm is $O(m^3n^3)$.

The bipartite graph does not always exist. For a bid i, it is possible that there is no l_e, r_e satisfying the two conditions. For example, there are four bids $\{1,2,3,4\}$. The bid 4

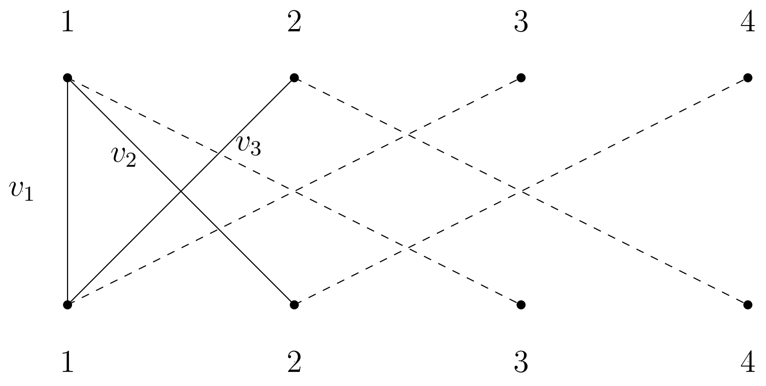

Figure 3.5: A bipartite graph for three bids

conflicts with all other bids, and the bids 1, 2, 3 do not conflict with each other. According to the conditions,

$$l_1 \neq l_2, l_1 \neq l_3, l_2 \neq l_3, r_1 \neq r_2, r_1 \neq r_3, r_2 \neq l_3 \tag{3.1}$$

and

$$(l_4 = l_1 \wedge r_4 = r_1) \ \&\ (l_4 = l_2 \wedge r_4 = r_2) \ \&\ (l_4 = l_3 \wedge r_4 = r_3). \tag{3.2}$$

Because of Equation 3.1, the vertices of bids 1, 2, 3 are set as $l_1 = r_1 = 1,\ l_2 = r_2 = 2,\ l_3 = r_3 = 3$; then Equation 3.2 changes to

$$(l_4 = 1 \wedge r_4 = 1) \ \&\ (l_4 = 2 \wedge r_4 = 2) \ \&\ (l_4 = 3 \wedge r_4 = 3). \tag{3.3}$$

There do not exist l_4 and r_4 satisfying Equation 3.3.

We looked at allocation algorithms since we want to incorporate them into a mechanism. There is another optimal algorithm besides the algorithms above using the bipartite optimal matching.

Because the allocation scheme should be optimal if we want to use the VCG mechanism, we would look at the space of the possible optimal allocations. For any bid i, the other bids could be divided into two sets $S1$ and $S2$. $S1$ contains the bids that conflict with the bid i, and $S2$ contains the remaining bids. So we could compute the optimal allocation in $S1$ and $S2$ respectively, then add the value of the bid i to $S2$'s result. The maximal social welfare is the larger one of $S1$'s result and $S2$'s result; and the optimal allocation is obtained at the same time. Algorithm 3.2.4 computes the optimal allocation at a time period without traversing all possible allocations. The computational complexity of the algorithm with k bids is $C(k) = C(k-1) + C(S2) \leq 2C(k-1)$, so the computational complexity is $O(2^{k\times(k-1)/2})$.

Algorithm 3.2.4 The third optimal allocation algorithm for one time period

```
Input: A set of bids $S$
Output: The optimal allocation($OA$) in $S$ and the maximal social welfare $W$
  $i \leftarrow S[0],\ S \leftarrow S \backslash i$
  $S2 \leftarrow \emptyset$
  for all $j \in S$ do
    if $j$ does not conflict with $i$ then
      $S2 = S2 \bigcup j$
    end if
  end for
  $OA1 = OA(S)$, $W1 = W(S)$
  $OA2 = OA(S2)$, $W2 = W(S2)$
  $OA2 \leftarrow OA2 \bigcup i,\ \ W2 \leftarrow W2 + value(i)$
  if $W1 > W2$ then
    return $\{OA1, W1\}$
  end if
  return $\{OA2, W2\}$
```

3.2.7 The Dynamic Mechanisms

In the dynamic setting, the auctioneer computes the expected values for agents to compete in the mechanisms. Suppose an agent i has a value v_j^i for a task j at a time period t. If the task j exists, the expected value of allocating it is $ev_j^i = v_j^i$; if the task j is supposed to arrive at a future time period te, the expected value is $ev_j^i = \gamma^{te-t} v_j^i$, γ is a discount factor. The allocation's uncertainty about future tasks can be transferred into the fixed expected values, and the optimal allocation could be computed using the expected values.

Using the expected values, an agent i would like to maximize its own utility that is $\sum_{t=t'}^{\infty} ev_i(\chi(\hat{ev})) - \wp_i(\chi(\hat{ev}))$, where $(\chi, \wp)$ is the mechanism used by the auctioneer. The social welfare at a time period t' is $\sum_{t=t'}^{\infty} \sum_{i\in A} \hat{ev}_i(\chi(\hat{ev}))$, A is the set of agents. Using algorithms in last section, the expected optimal allocation for every time period can be computed. We present the dynamic VCG mechanism.

Definition 3.2.8. *The **dynamic VCG mechanism**$(\chi, \wp)$ for the agents A under the* *model is*

$$\chi(\hat{v}) \;=\; \arg\max_{\chi} \sum_{t=t'}^{\infty} \sum_{i \in A} \hat{ev}_i(\chi(\hat{ev}));$$

the payment scheme is

$$\wp_i(\hat{v}) \;=\; \sum_{t=t'}^{\infty} \sum_{j \in A_{-i}} \hat{ev}_j(\chi(\hat{ev}_{-i})) - \sum_{t=t'}^{\infty} \sum_{j \in A_{-i}} \hat{ev}_j(\chi(\hat{ev})).$$

Theorem 3.2.9. *The dynamic VCG mechanism is truthful for agents.*

Proof. At every time period, the allocation scheme checks the available agents and constructs a bipartite graph. Then it computes the optimal allocation using the graph. The allocation scheme considers every existing task no matter when it would be allocated. So the allocation result maximizes the total utility from the current time period; and the mechanism is the optimal solution for the class of mechanisms maximizing *ex ante* efficiency. The agents cannot lie as the mechanism maximizes their declared utilities. □

The auctioneer can also consider waiting for tasks such that a larger social welfare may be achieved, especially when there is a task brings a large value to its winner while other tasks bring much smaller values to agents. Then the social welfare may be improved by keeping the other tasks for next period. Waiting may also help the auctioneer allocate more tasks in the same amount of time.

Agents turn to consider how long to wait for better tasks, we call it the agent patience. Agents can have different patience for tasks, which is the agents' expectation about how long a better task may arrive. If the current time period passed the patience, an agent does not wait any longer. Let p_i denote the patience information of an agent i. In our setting, p_i is a percentage value so that an agent would like to spend p_i of the available time duration to wait.

After every allocation, the patience of the winning agents are updated, according to wasting time percentage of the obtained allocation. We combine the patience information into the expected value and then the allocation result is influenced by the agents' patience. An agent i with a patience p_i has the patience-combined expected value $ev_j^i = \gamma^{(expire_j - arrive_j)*p_j^i + (arrive_j - t)} v_j^i$, at the time period t for a task j. A major difference between our mechanism and the dynamic VCG mechanism is that the expected value turns to consider the patience information besides the time.

Definition 3.2.10. *A dynamic **Patience-based VCG mechanism**$(\chi, \wp)$ for agents A and tasks T at a time period t' under the model is*

$$\chi(\hat{v}) \;=\; \arg\max_{\chi} \sum_{t=t'}^{\infty} \sum_{i \in A} ev_i(\chi(\hat{v}));$$

the payment for an agent i is

$$\wp_i(\hat{v}) \;=\; \sum_{t=t'}^{\infty} \sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}_{-i})) - \sum_{t=t'}^{\infty} \sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}));$$

where the expected value of an agent i to a task j is $ev^i_j = \gamma^{(expire_j - arrive_j) * p^i_j + (arrive_j - t')} v^i_j$.

Theorem 3.2.11. *The dynamic Patience-based VCG mechanism is truthful for the patience combined expected values.*

Proof. For any agent i, its utility on winning a task is $ev_i - \wp_i = ev_i - (\sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}_{-i})) - \sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}))) = ev_i + \sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}))$-$\sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}_{-i})) = ev_i + \sum_{j \neq i} \hat{ev}_j(\chi(\hat{v})) - h(-i)$, where $h(-i)$ is $\sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}_{-i}))$ and is irrelative to the agent i's reported value.

The mechanism chooses an outcome that maximizes $\sum_{i \in A} \sum_{t' \in T} ev_i(t', t) \chi_t(i, t') = \hat{ev}_i + \sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}))$, so the outcome generated by the mechanism when the agent i submit its true information would be the dominant strategy. In other words, it is a dominant strategy for the agent i that the processsing time is truthfully reported as well as the value, because the processing time is combined in ev_i and the outcome based on the true processing time and the true value would provide the maximal $ev_i + \sum_{j \neq i} \hat{ev}_j(\chi(\hat{v}))$. Therefore, any agent would submit its true information for tasks to maximize its expected utility, no matter whether other agents submit their true information. □

Chapter 4

Implementation and Evaluation

The mechanisms introduced in the chapter 3 are compared with other mechanisms. In the static setting, the VCG mechanism is compared to the Multi-round mechanism; and in the dynamic setting, the dynamic VCG mechanism and the Patience-based VCG mechanism are implemented and compared.

4.1 Evaluation of the Multi-round Static Mechanism

We choose to compare our Multi-round mechanism to the VCG mechanism. Both of the mechanisms' implementations first compute the cost value matrix. For the Multi-round mechanism, its implementation does the following things.

1. For every available task t, compute a list L_t of agents with lowest cost value; and $\wp(t) = min\{\wp(t),\ \textit{the second lowest value}\}$, $\wp(t) = MAXIMUM$ initially;

2. For every agent a, compute the task t such that a is the winner and $t = argmax_t($ $\wp(t) - v_t^a)$, then update the payment upper bounds of a's refused tasks; and remove the winning agents and the allocated tasks from the market;

3. If there is a new allocation after the steps 1 and 2, go back to step 1; otherwise, return the result.

In the experiments, the number of agents M can vary from 3 to 102 and the number of tasks is in the range $[1, 2M + 1)$. There can be more agents than the tasks, and there also can be more tasks than the agents. These numbers are generated randomly in every experiment. The cost value v_t^i of the agent i to a task t is generated from a normal distribution $\mathcal{N}(h, r^2)$, where $h \sim \mathcal{N}(3, 0.5^2)$ and $r = Math.random() \times agentSize$.

Additionally in the VCG mechanism, we compute a virtual value for every bid which is $200 - v_j^i$. 200 is larger than the maximal cost generated in our experiments. The optimal allocation scheme using the virtual value is implemented by Algorithm 2.4.1. The VCG mechanism matches a task or none to every agent, which would be the optimal matching. Before applying the algorithm, the method of constructing the bipartite graph is introduced in the subsection 3.2.4, and the construction uses the first transformation algorithm.

The experiment involves 100 different instances of agents and tasks; and it is repeated 300 times. Because the efficiency is relative to the tasks' size as well as the agents' size, the *Agent Size* in the figures is the sum of the number of agents and the number of tasks. The experimental results of the same *Agent Size* are averaged.

First we present the allocation efficiency results. As the standard optimal efficiency would be the maximum, we use the virtual values to compare. Figure 4.1 shows the experimental allocation efficiency results. The VCG mechanism always allocates a task to an agent; then the social efficiency increases along with the agents' size. As the utility is applied as an upper bound of following rounds' allocation in the Multi-round mechanism, not every agent could obtain a task. So the Multi-round mechanism performs worse than the VCG mechanism on efficiency as the agents' size increases.

We evaluate the mechanisms on the runtime besides efficiency. The experimental result for the runtime is in Figure 4.2; the runtime of the Multi-round mechanism can be almost ignored compared to the VCG mechanism's runtime, although the theoretic computational complexity of the Multi-round mechanism is $O(n^2)$ in the worst case and the VCG mechanism's computational complexity is $O(n^4)$. Empirically, the Multi-round mechanism's runtime is so small that it appears as though the worst case is rare.

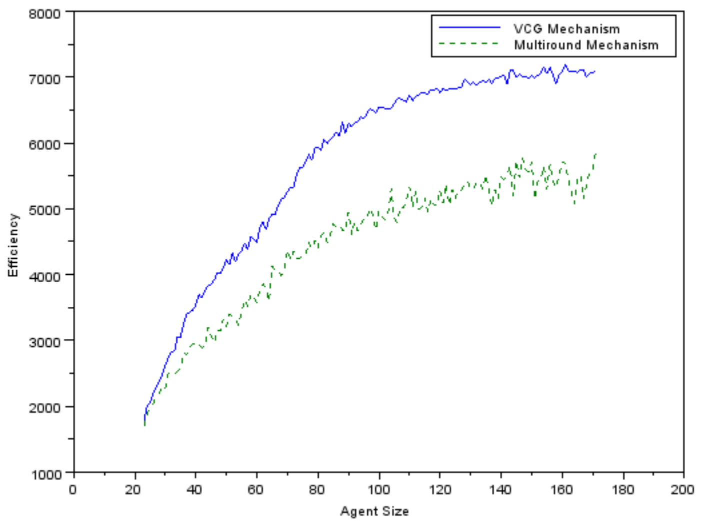

Figure 4.1: Allocation efficiency of the static mechanisms

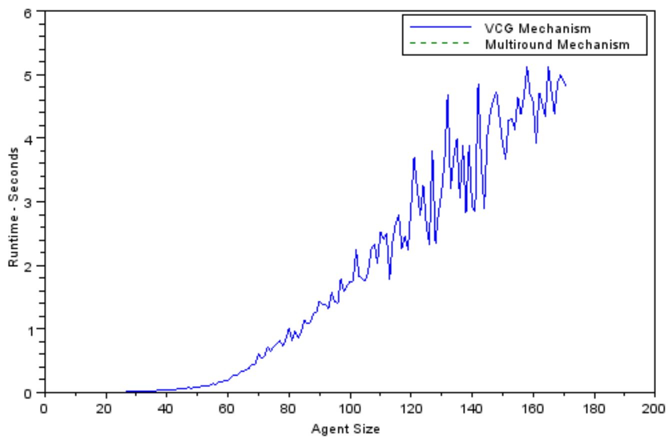

Figure 4.2: Runtime of the static mechanisms

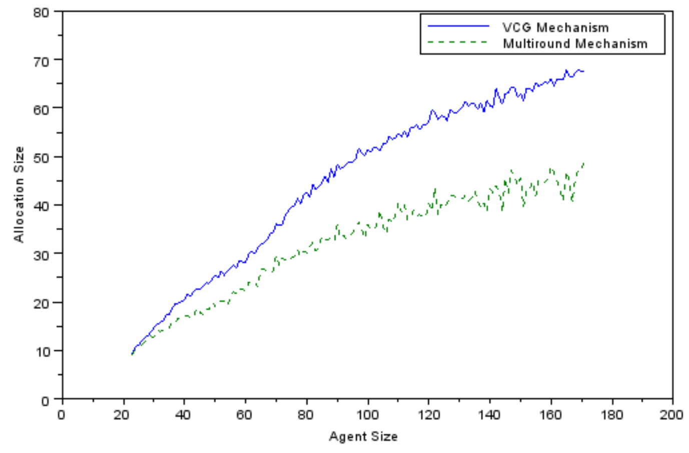

Figure 4.3: Allocation size of the static mechanisms

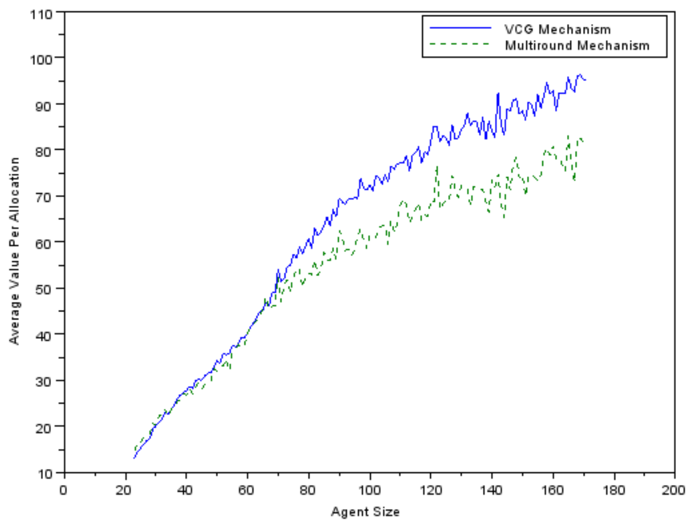

Figure 4.4: Average cost of an allocated task

The experimental result on the average cost per allocation is in Figure 4.4; and Figure 4.3 shows how many allocations are committed on average by both mechanisms. These empirical results tell that the Multi-round mechanism always allocates less tasks than the VCG mechanism, and the difference between them is enlarged by increasing the agents' size. However, the average cost of an allocation is not the same way. The Multi-round mechanism achieves a smaller average cost for allocations. The Multi-round mechanism cannot achieve a better efficiency than the VCG mechanism because the allocation scheme is not optimal; but the Multi-round mechanism can improve the average allocation cost.

4.2 Evaluation of the Patience-based VCG Mechanism

There is an auctioneer, multiple agents and multiple tasks in the dynamic setting. We use a discount factor for the time horizon. The auctioneer and agents always exist; and there are new tasks generated at every time period. Figure 4.5 is a brief description of the allocation's implementation.

A Task		The Auctioneer		An Agent
generated		remove the expired tasks		remove the expired bids
	→	new tasks arrive		update the new tasks
		ask for new bids	←	submit new bids
		compute the decision and the payment	→	receive the allocation
		compute the agents' utility	←	
expire				

Figure 4.5: A dynamical allocation problem

A new task is generated with a start time and a depart time. We implement two variation of the dynamic setting. In the first setting, the auctioneer knows the new tasks as soon as they arrive. In the second setting, new tasks known by the auctioneer are set with their arriving periods randomly generated from $[t, t+5)$ at every period t; the departing period is then determined by the arriving period and the existing duration. A task exists from 1 to 10 time periods. Then the auctioneer can know some future tasks, but cannot predict all tasks within the nearest five periods. The number of new tasks in a period is a random integer from $[0, n/2)$, n is the number of agents.

4.2.1 Implementation of Agents

An agent consists of the bids which are submitted to the auctioneer and the information for its allocated tasks. An agent i has a value and a processing period for every interested task; the value v^i_j for a task j is a random integer from [1,10), the processing period is also at most 9.

The information of an agent's allocation includes an array of completed tasks and the payments for the completed tasks, a variable stating whether the agent is processing a task, the current processing task and the value and payment of the agent for that task, the starting time period and completing time period for that task. When the auctioneer knows some future tasks, the auctioneer can allocate a task to an agent before the task arrives; so the agent also includes an array of the waiting allocations. As the Patience-based mechanism considers the agent's patience, a variable for the patience is also included in the agent's information.

A bid of an agent expires when the agent cannot finish processing the task before the task departs. At every time period, an agent removes its expired bids first, and then receives the new tasks from the auctioneer. If the agent is not busy with computing a task, it generates the values and the processing periods for the new tasks and submits this information to the auctioneer.

The auctioneer notifies an agent if the agent obtains a new allocation. The winning agent then removes the relevant bid and starts computing. If the auctioneer knows the future tasks, there is another situation about the allocation. An agent cannot start computing when it obtains a task which will arrive in the future; so the agent stores the allocation in an array, and checks that array at every time period to begin the computing as soon as the task arrives.

4.2.2 Implementation of the Auctioneer and the Mechanisms

The auctioneer has a list of tasks which are not expired and not allocated, and the auctioneer collects values and processing time periods from the agents for the tasks. Because an agent may not submit bids for all existing tasks, the auctioneer sets a unique index for every task. The task's index is used in the multiple agents' competition for the same task, and in specifying which task an agent's bid is for.

We implement the Patience-based VCG mechanism and the dynamic VCG mechanism. The auctioneer applies two mechanisms on the same agents and tasks, and the Patience-based VCG mechanism is evaluated by the comparison to the dynamic VCG mechanism. The major difference between the two mechanisms is the Patience-based VCG mechanism adopts the patience variable to compute the expected value. The initial patience of every agent is set as 0, which means the agents prefer to obtain an allocation when a task arrives; the discount factor of expected values is set as 0.8.

In the case that the agents simply care for the arrived tasks, the auctioneer does not need to know the future tasks. All tasks for an agent's bids have arrived and have not departed, so the bids of the agent conflict with each other. The allocation scheme employs the first transformation algorithm in Lemma 3.2.5 to construct a bipartite graph and computes the optimal allocation of the current period by this graph.

In the case that the agents know the information of some future arriving tasks, an agent's bid does not conflict with the bids for the current existing tasks when the future task arrives after the current tasks expired. A width-first traversing algorithm 3.2.4 is implemented to compute the optimal allocation. The allocation of the future tasks are delivered to the agents before the tasks arrives, but the agents wait until the first time period of the future task to start processing.

4.2.3 Evaluation Results

We evaluate the performance of the Patience-based VCG mechanism and the dynamic VCG mechanism on the efficiency and the number of completed tasks. Both of the mechanisms compute the outcome maximizing the expected social welfare from every time period t, which is $\sum_t^\infty \sum_{i\in A} v^i(\chi(\hat{v}))$; they might not maximize the social welfare already achieved, which is $\sum_0^t \sum_{i\in A} v^i(\chi(\hat{v}))$, assuming the time horizon starts at 0. An optimal algorithm for dynamical allocation indeed maximizes $\sum_0^t \sum_{i\in A} v^i(\chi(\hat{v}))$ at a time period t. But since we do not have such an algorithm, we use the dynamic VCG mechanism to evaluate our Patience-based mechanism.

Figure 4.6 is the experimental result on efficiency when the auctioneer does not know any future task; and Figure 4.7 shows the result of the completed tasks by the mechanisms.

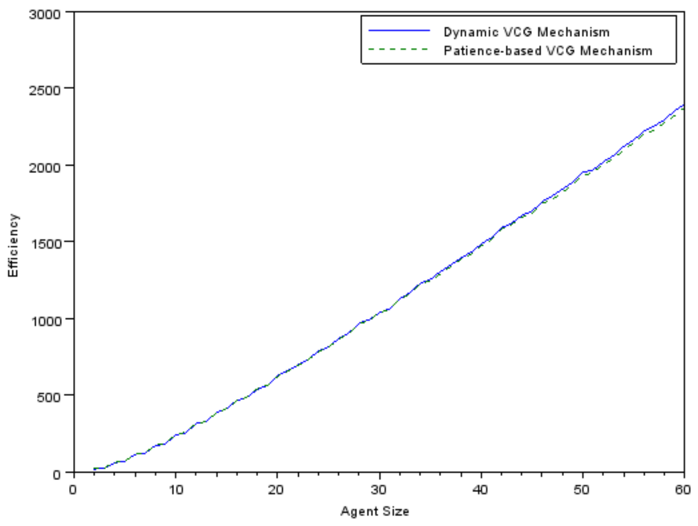

Figure 4.6: Allocation efficiency of the multiple agent sizes

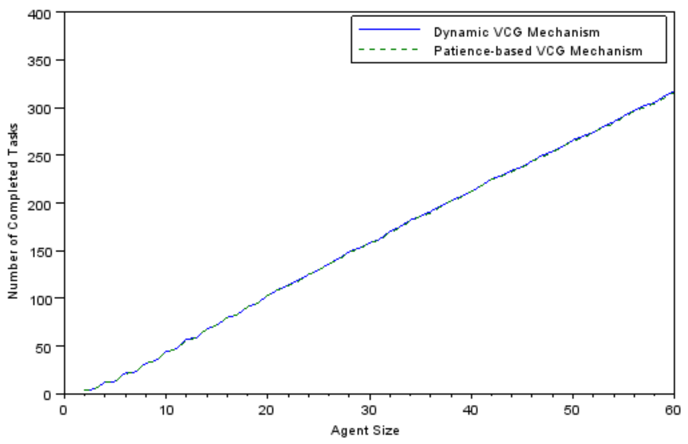

Figure 4.7: Number of completed tasks of the multiple agent sizes

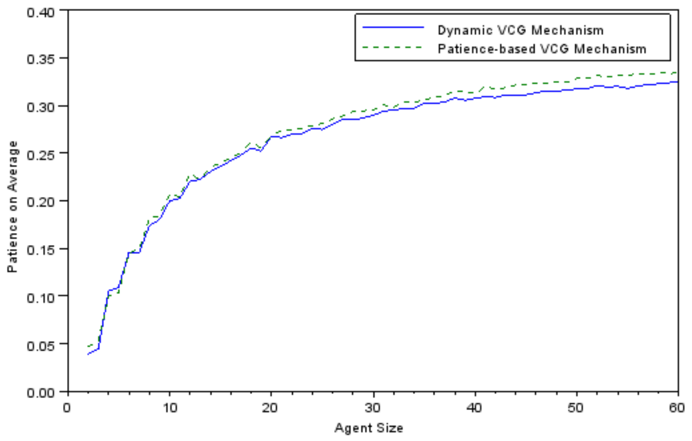

Figure 4.8: Patience of the agents on average

The experiment tests multiple allocation problems with an increasing agent size. The agent size varies from 4 to 62. There are 20 time periods for every allocation problem as the time horizon. The efficiency is the sum of agents' values on allocated task after the last period. This experiment has been repeated for 300 times.

We also compute the average agents' patience of both mechanisms. Although the dynamic VCG mechanism does not care for the patience, a task may not be allocated to an agent immediately when the task arrives. Figure 4.8 is the experimental result of the average patience value for all agents at the final time period. The agents wait longer in the Patience-based VCG mechanism than in the dynamic VCG mechanism.

The experimental results show that the Patience-based mechanism's efficiency is almost the same as the dynamic VCG mechanism; but it completes slightly fewer tasks if there are more than 40 agents. The ratio of the Patience-based VCG mechanism to the dynamic VCG mechanism is 0.9997 on the efficiency; and the ratio on the number of completed tasks is 0.9963.

Our experiments also test the mechanisms' performances on multiple time horizons. The agent size is fixed at 30 in this experiment and the time horizon increases from 20 periods to 79 periods. The results are in Figure 4.9 and Figure 4.10; and the average agents' patience result is in Figure 4.11. The ratio of the Patience-based VCG mechanism to the dynamic VCG mechanism is 1.0005 on efficiency; and the ratio on the number of completed tasks is 1.01. The Patience-based VCG mechanism achieves a slightly better outcome and the agents' waiting time before an allocation is longer than the dynamic VCG mechanism.

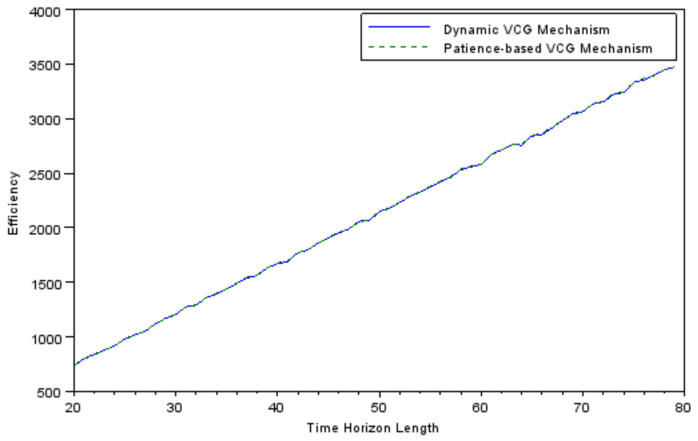

Figure 4.9: Allocation efficiency of the multiple time horizons

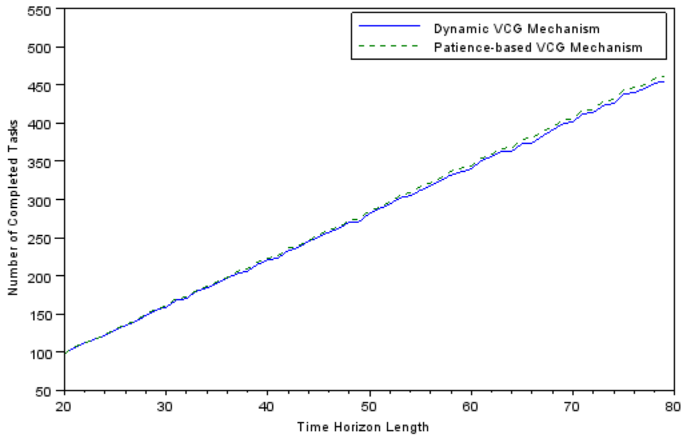

Figure 4.10: Number of completed tasks of the multiple time horizons

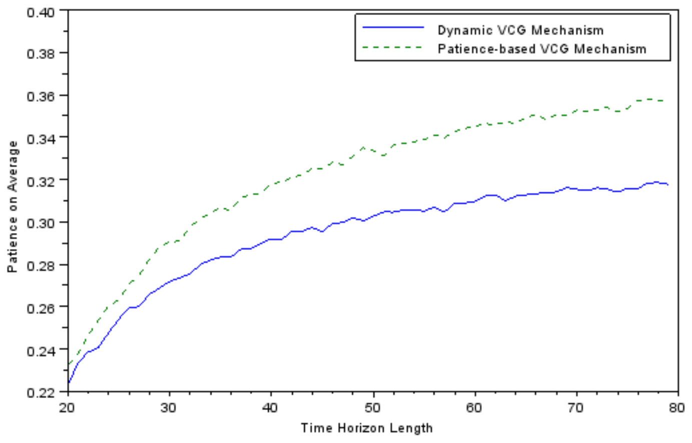

Figure 4.11: Patience of the agents of the multiple time horizons

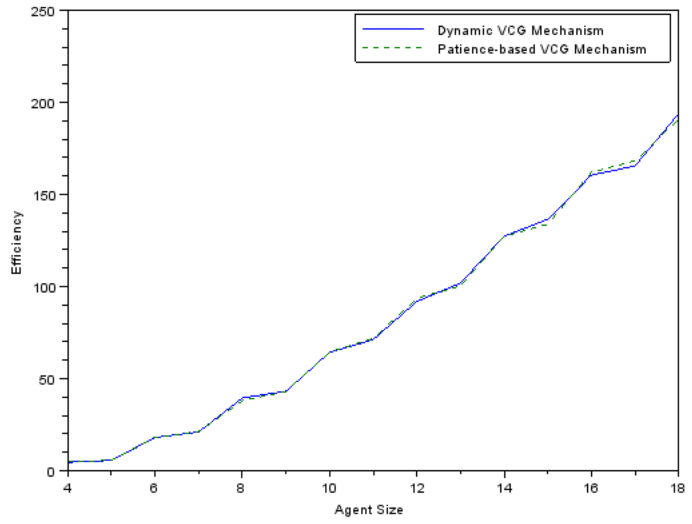

Figure 4.12: Allocation efficiency with future tasks of the multiple agent sizes

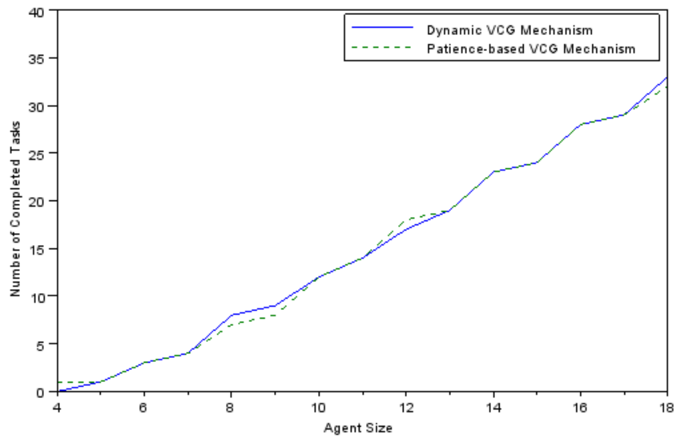

Figure 4.13: Number of Completed Tasks of the multiple agent sizes

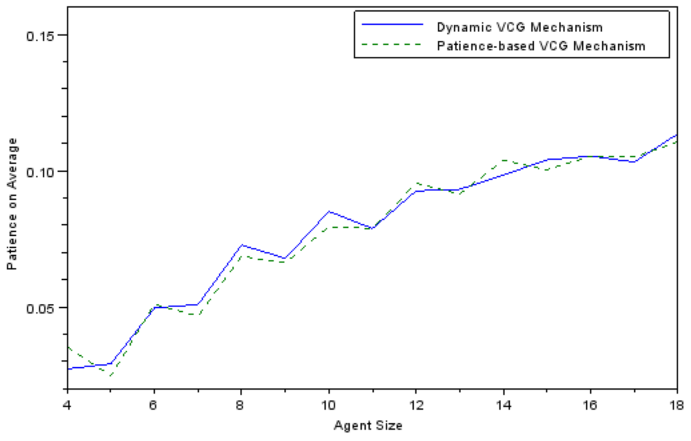

Figure 4.14: Patience of agents when there are future tasks

The agents may obtain an allocation before the task arrives, if the auctioneer knows the future tasks. The auctioneer can learn some tasks arriving within the nearest five periods in the experiment. The Patience-based mechanism's performance is evaluated in the same way as there are no future tasks.

Figure 4.12 is the result for the allocation efficiency when the agent size increases from 4 to 18 and the time horizon's length stays at 18 periods; Figure 4.13 shows the number of completed tasks under the two mechanisms, and Figure 4.14 is the average patience value of the agents. The ratio on the allocation efficiency is 1.003, so the Patience-based mechanism performs slightly better in practical. But as the agent size increases, the Patience-based mechanism does not always produce a better outcome as the figure 4.12 shows.

There is also an experiment for the increasing time horizon, when the auctioneer has the information of some future tasks. In this experiment, the agent size is fixed at 12 and the time horizon's length ranges from 20 to 69. The experimental results are in Figure 4.15, Figure 4.16 and Figure 4.17. The efficiency ratio of the Patience-based mechanism to the dynamic VCG mechanism is 1.0026. When the auctioneer knows some tasks coming in the near future, the efficiency achieved by the Patience-based mechanism is slightly better on average.

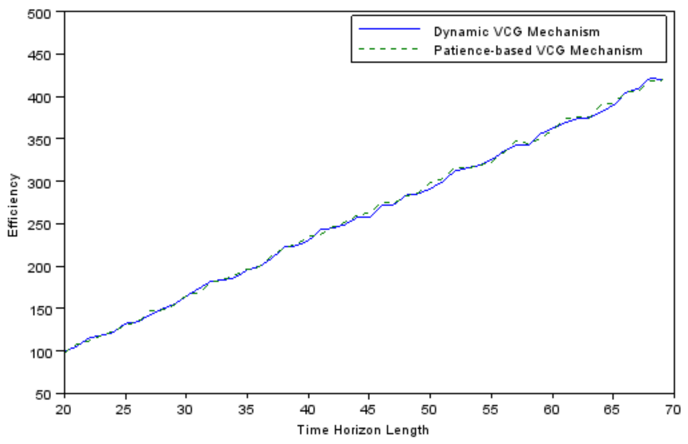

Figure 4.15: Allocation efficiency with future tasks

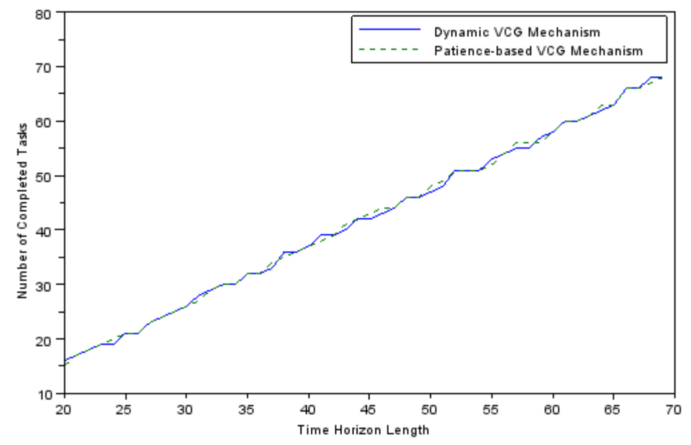

Figure 4.16: Number of Completed Tasks of the multiple time horizons

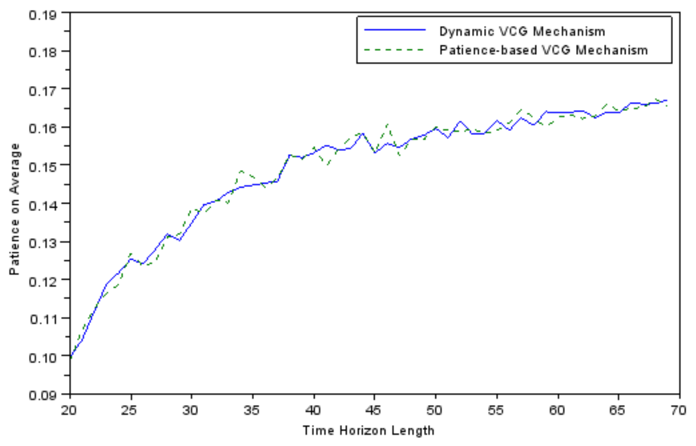

Figure 4.17: Patience of agents when there are future tasks

Chapter 5

Related Work

Mechanisms in the dynamic setting have been widely studied in both the economics and the computer science literature. In this chapter we discuss alternative mechanisms and the problems they were designed to solve.

5.1 Dynamic VCG Mechanisms and Models with Restrictions

In Bergemann and Valimaki's[2] work, only a single item exists for a dynamic auction; every agent i does not know its exact value w_i for the item. An agent would receive a signal about w_i if the agent wins in the current time period. Any agent could approximate its value for the next auction using the signals received in history. A dynamic VCG mechanism for this model is presented and is incentive compatible.

That model is for the single-unit allocation and describes a case where the agents are restricted to compute the approximation of their true valuations. To ensure that the agents would submit their true signals, the auctioneer is required to focus on the approximated values instead of exploring the true values. In other words, the auctioneer need capture the agents' expected utilities from the approximation knowing the signals. In the analysis of the paper, the authors ignore whether the agents could know their exact values after the final allocation, and the agents make decisions only upon the expected values.

The concept *flow marginal contribution* is proposed. The flow marginal contribution m_i of an agent i is the agent i's contribution for a single time period t, while its marginal contribution M_i includes the contribution for the allocation period t and the contribution on the allocations after the time period t. $M_i = m_i + \sigma(M_i(h^t, \chi^i) - M_i(h^t, \chi^k))$, where k is the agent that would win if i does not exist and σ is the discount factor. Under the mechanism's payment policy, an agent i's utility at a time period t is exactly m_i^t; thus although the social welfare includes agents' expected utilities of the future, the payment only depends on the utility of the current time period. As an allocation helps the winning agent obtain the signal of its valuation at a time period t, any allocation scheme depending on agents' signal history would be influenced by the winning agent at the next time periods. The flow marginal contribution computes the agents' contributions which is just for their winning time periods. The flow marginal contribution is essential in the payment policy and the mechanism's truthfulness.

Cavallo[5] provides a further analysis of dynamic VCG Mechanisms. The type of agents is modeled by an MDP without the requirement that the value of an agent is positive or negative. According to the MDP model, the allocation scheme could not depend on historical allocations. Cavallo[5] empirically discovered that only 10% to 20% of the social welfare is distributed to the agents. A dynamic mechanism is proposed to redistribute the social welfare through the payment policy and the mechanism makes a significant increase of every agent's utility.

Parkes and Duong[19] propose an ironing technique for dynamic mechanisms. The ironing technique can improve any general stochastic optimal policy to be incentive compatible and individually rational. The ironing technique is applicable where the agents have *Single Value Preferences*(SVP)[1]. Every agent with SVP is restricted to submit a single value v_i for a set of outcomes, and the agent obtains the value v_i for any outcome in that set. An agent has a single value for its interesting outcomes. The paper gives an example where the ironing technique is not used that, a bid does not win but decreasing the value can let the same bid win; this example shows that an agent can lie to make profit in the SVP setting.

A mechanism is strongly truthful and IR with no-early arrival and no-late departure misreport if and only if the allocation policy is monotonic. The ironing technique can remove all non-monotonic outcomes. Besides the value, the ironing can also be performed on other aspects in the agents' type, such as the arrival and departure time. This property relaxes the condition on the agents' time related features. In fact, the combination of the ironing policy and a stochastic optimal policy would be a specific extension of dynamic VCG mechanism within the restriction of SVP. While the ironing policy achieves the truthful and IR property, it results in strong monotonic property. So it is possible that the ironing may remove some decisions which could be retained. A further research[8] analyzes whether the ironed space could be relaxed and the allocation efficiency would then be improved.

Hajiaghayi, Kleinberg, Mahdian and Parkes[12] look at the online truthful mechanism design, where agents compete for a single re-usable resource in the finite time horizon. No early arrival and no late departure reports are required. Their work classifies the arrival and departure models into the asynchronous model and the synchronous model. In the asynchronous model, the arrival and departure time are real numbers, and an early allocation would be revoked when a better bid arrives.

The paper also analyzes the competitive ratio of the online mechanisms, which is the ratio of its performance to that of the offline mechanism. It is proved that there exist truthful online mechanisms with bounded competitive ratio; and those mechanisms can be generalized with the same ratio bounds into the multiple re-usable resources environment. The paper also analyzes the lower bound of the mechanisms' revenue ratio contrast with the VCG mechanism. A truthful online mechanism exists when any agent could ask for the item with more than one unit, and its revenue ratio to VCG is relative to the length of the units.

Cavallo and Parkes[6] propose a model where agents are uncertain about their final values on the resources at the first time, and their researching processes for the values are modeled by MDPs; the researching for an agent's final value is utility costly. The auctioneer computes an outcome for multiple MDPs which are privately reported. For an MDP where an agent's utility depends on the current state, a self-interested agent can choose to perform a redetermining action for a larger final value. The redetermining action is called deliberation.

The model uses the expected utility, and the auctioneer would like to achieve the maximum social welfare. Therefore, the dynamic VCG mechanism is applicable in this model. Furthermore, under the uncertainly improvable values condition, which could be derived from the assumption of the model, the agents' MDPs can be reduced into Markov Chains. Then the computation of the allocation scheme in the dynamic VCG mechanism is simplified through computing Gittins Index[10]. Therefore the allocation scheme's computation would be linear in the number of agents.

Seuken, Cavallo and Parkes[21] present a partially DEC-MDP for the agents' strategies. The agents have their private actions, values and states modeled as DEC-MDPs, and the agents can be inaccessible to the auctioneer sometimes. The auctioneer keeps the agents' last reports about their types and accessibleness. The authors analyzed the computational complexity for this kind of models; in more details, they analyzed the computational complexity of solving the DEC-MDPs where the time horizon is finite or not finite, and where there is a limit on the agents' inaccessible or not. It is proved that only when there is limitation on the agents' inaccessibility, that computational complexity is P-Complete.

Their work also proposes a dynamic Groves mechanism on partially DEC-MDP agents' model; the payment scheme for the inaccessible agents would be discounted on how long they have been inaccessible. The proposed mechanism is efficient and Bayes-Nash incentive compatible.

Cavallo, Parkes and Singh[7] present an analysis for mechanisms in multiple dynamic models. Their paper concerns about both the dynamic population of agents and the dynamic types of agents. Agents can be periodically inaccessible; for any agent that is not accessible in some time period, the auctioneer uses the Bayesian inference to compute a belief about the agent's state. The agents' types are also modeled by the MDP and the dynamic VCG mechanism is generalized for the model of dynamic agents' types.

The paper proposes a concept within-period *ex post* incentive compatibility(wp-EPIC). The dynamic VCG mechanism is proved to be wp-EPIC upon the conditional independency on arrival property, which is that the types of the agents arriving at a period t do not depend on those of the agents arrived earlier, but do depend on the auctioneer's actions. An online VCG mechanism is also presented for the agents' model with dynamic population and static types.

There is a class of mechanisms which is closely related to the VCG mechanism. This class is called the VCG-based mechanism[18]. The mechanisms in this class replace the optimal algorithm in the allocation scheme with another algorithm k, and keep the payment scheme as the same. The algorithm k may be not optimal but should take much less computational time, thus the VCG-based mechanisms are applicable when the computation is hard and the optimal allocation is not strictly required.

There has been a lot of work on VCG-based mechanisms and on understanding when it is possible to replace an optimal allocation algorithm with a non-optimal one. For example, Nisan and Ronen[18] have showed that one required property for the allocation algorithm is to be maximal-in-range, which means that the algorithm is optimal for a restricted set of valuations. Other classes of mechanisms similar to VCG-based mechanisms such as the second-chance mechanism[18] and affine-based mechanisms[14, 20] have also been studied in order to understand how computational constraints can influence the mechanism properties.

5.2 Dynamic Mechanisms without Payment Policy

In a situation where the auctioneer needs to know agents' willingness but cannot provide any payment to agents, the mechanism only contains the allocation policy.

Zou, Gujar and Parkes[23] propose a model and a scoring rule mechanism for this circumstance. Their model uses the preferences of agents on resources to capture the agents' willingness. Every agent i's type is defined as $(\alpha_i, \beta_i, \phi_i)$, where α_i is the agent's arrival time period, β_i is the departure time period, and ϕ_i is the agent's preference on all interested items. The efficiency of a mechanism in the model is similar as fairness.

The authors proved that the online dictatorship is necessary for any deterministic, strategy-proof, neutral and non-bossy online mechanism. The Arrival-Priority Serial Dictatorship(APSD) mechanism assigns an item to an agent when the agent arrives, and commits the allocation on that agent's departure. The scoring rule mechanism computes an outcome with a larger *ex ante Pareto* efficiency than the APSD mechanism if a part of the agents are not rational; and the scoring rule mechanism would reduce to the APSD mechanism when all the agents are strategic. Every agent cares for the resources existing between its arrival time and its departure time. The agents would like to obtain the best allocation at the departure time; and those mechanisms compute the outcomes maximizing agents utility of the departure time period.

Gujar and Parkes[11] also analyzed a model without payment. The agents are divided into two sides in their model; one side is dynamic, the other side is static and is always in the market. Every agent on the static side has a strict preference on the other side's agents. A static agent j can be re-allocated to a better agent after it is matched with a dynamic agent i, even if the agent i has left; and in that case the agent i would be matched to a substitute of the agent j. Using the substitutes for the static agents is called the *fall-back* option; this option can provide a dynamic side's agent another matching which is equivalent with its previous matching, although there is an additional cost for the auctioneer. The paper proposed a mechanism that is strategy-proof for the static side's agents.

5.3 Double Dynamical Auction

The resource-providing agents and resource-bidding agents can both dynamically involve in an allocation problem, this is the Double Auction(DA). However, usually every agent can be allocated once.

A *CHAIN* framework[4] is discovered to provide a truthful and IR dynamic mechanism if agents have restricted staying duration. The CHAIN framework is defined by an admission policy, a substitutable matching rule for every time period and a price-out policy. The admission policy is for new incoming bids; the price-out policy would reject some current alive bids permanently; and the matching rule produces the paired winners from the existing non-price-out bids. Any dynamic DA is strong-truthful if and only if the payment for a winner is independent and monotonic-increasing with time[12]. Therefore, the payment policy can depend on the historical bids, or losing bids and their final status, and those could be a reason for rejecting some bids.

As this is the framework, the social efficiency is hard to analyze. A concept *well-defined* is used to correlate the matching rule and the dynamic mechanism by CHAIN. A well-defined mechanism satisfies four requirements: truthful, IR, non-deficit and strong-feasible. To ensure the well-defined property of the dynamic CHAINed mechanism, the matching rule should be a well-defined static allocation mechanism.

Bredin and Parkes[3] focus on the online double auction, and they present a general method to design the truthful and weak budget balanced mechanism. There are no arrival misreports; the agents' patience is bounded; and the allocations are committed at the winners' departure time and no false-name bids.

Myerson and Satterthwaite[17] proved that maximizing efficiency would not result in a voluntary participation and a no-deficit mechanism. The no-deficit and truthfulness properties can be achieved by decreasing the efficiency. The paper proposes a price schedule policy $f_i(t, r_i, v_{-i})$ to price out some agent i with $f_i(t, r_i, v_{-i}) > w_i$. The price schedule should be implemented to check the agents' dependence, as the payment scheme would be rely on the price schedule function: $\tilde{ps}_i(t) = max_{\tau \in d_i - k, \ldots, t}\{f_i(t, r_i, v_{-i})\}$. ($k$ is the patience bound of the agent i.)

A practically designed mechanism also extends the static McAfee double auction Rule[15] as the valid price schedule policy, which helps to achieve the no-deficit property. The mechanism using McAfee rules is proved to be truthful and no-deficit.

5.4 Dynamic Mechanisms for Interdependent Agents

The agents' values could be dependent on other agents' information. If that information is public to all agents, the auctioneer can model every agent's value by a specific function $V_i(s_i)$ from the agent's private information s_i. But if an agent's value depends on other agents' private information, the agents are interdependent. In the Interdependent Values Model(IDV), the specific value function used by the auctioneer is modeled as $v_i(s_i, s_{-i})$.

There could be restrictions on the v_i function for the incentive compatible mechanisms. In an incentive compatible mechanism, v_i should have the monotonicity property with s_i, and the influence of s_i on v_i is not smaller than that of any other agent and is also larger than that of any other agent with the same signal value.

An MIP solution[9] is provided to maximize the allocation efficiency for a finite number of agents with IDV. The solution extends the concepts of the unconditioned and conditioned *critical signal* properties, which are used to analyze the agents' strategies. The critical signal is a value that is the smallest possible signal for an agent to win, and in the dynamic model, the unconditioned critical signal property may not exist; while the conditioned critical signal property is that an agent can find its critical signal and there would be an item available at the agent's departure time.

In a no arrival and departure misreport model, the agents' arrival and departure time can be modeled as $X_{i,1}X_{i,2}\dots X_{i,i-1}X_{i,i+1}\dots X_{i,n}$ in the finite time horizon, where the total number of agents is n, $X_{i,j}$ means whether the agent i arrives before the agent j. Considering the continuity of an agent's existence, some scenarios of the $X_{i,j}$ can be removed. A mechanism of this model is incentive compatible if and only if the agents submit true signals. Therefore, with the unconditioned critical signal property, the incentive compatibility problem is to check the constraints on the critical signals and bid signals, which is called inter-temporal incentive compatibility constrains(ITIC).

5.5 Summary

Our dynamic model for the allocation problem includes multiple agents and multiple resources. Every agent has an individual value for a task; there is not an internal model such as SVP that describes agents' values on some set of the outcomes. Besides the value, an agent also has the processing time in its private information, which is required by an allocation problem where the allocation needs to be processed. There are models for dynamic agents and static resources, and models for dynamic agents and dynamic resources; we assume the agents are static and the resources have dynamic duration. Our model is assumed to be a no early arrival and no late departure model like some researchers'.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We focus on a dynamic model where the allocation's outcome space is restricted. The outcome space is restricted because an agent cannot work on some tasks at the same time; the restricted outcome space helps agents with different capabilities obtain the tasks more fairly. The auctioneer chooses to collect private information from the available agents.

The static model with the same restricted outcome space is studied first. The VCG mechanism in this model is presented and its computational cost is biquadratic in the tasks' size. A truthful and individually rational Multi-round mechanism is proposed for this model, with quadratic computational complexity in the worst case. Empirically the Multi-round mechanism achieves 67% of the VCG mechanism's efficiency on average. But on the allocation's average cost, the cost of the Multi-round mechanism is 89.79% of that of the VCG mechanism; the Multi-round mechanism reduces the average cost by about 10 percent.

We propose the dynamic VCG mechanism and a truthful Patience-based VCG mechanism for the dynamic model. The Patience-based mechanism allows agents to wait for better tasks to arrive. The efficiency of the dynamic VCG mechanism and the Patience-based mechanism is about the same empirically. When the future tasks are not considered by agents, the Patience-based mechanism's efficiency is 99.9% of the VCG mechanism's efficiency; when the future tasks are considered by agents, the ratio turns to 1.003. The Patience-based mechanism performs slightly better in the experiments where future tasks are considered by agents.

6.2 Future Work

Though the proposed mechanisms are truthful, there are some shortcuts in the mechanisms. For example, the assumption that all agents can be allocated only one item at a time is quite strict. We also note that the computational complexity of our Patience-based mechanism is still quite high. We would like to do future work on these aspects.

1. To improve the allocation scheme of the Patience-based mechanism.

   The computation of the allocation scheme would make the mechanism's computational cost high. An alternative allocation scheme could be derived by ignoring some bids if the agents are patient and would wait for better tasks, and computing the optimal outcome as those bids did not exist.

   The new allocation scheme would find the optimal allocation in a range where only the bids for the agents that would not wait are considered. The outcome is certainly different from the current allocation scheme's outcome, because of the conflicts among the agents' bids. By reducing the information considered, the size of the current allocation problem is reduced; thus the computational cost would be lower. But whether there is a truthful mechanism with this allocation scheme and whether the mechanism could improve the efficiency remain to be seen.

2. To study the auctioneer's computational cost and the weak budget balanced property.

   As we said in section 2.2, the auctioneer is not concerned whether it needs resources to do computation and how much $\sum_i v_i(\chi(\hat{v})) - \sum_i \wp_i(\chi(\hat{v}))$ should be. It may be not fair to some agents that $\sum_i v_i(\chi(\hat{v})) - \sum_i \wp_i(\chi(\hat{v}))$ is very large and some agents' utilities are very small; the mechanism's computational cost can be a reference for the value of $\sum_i v_i(\chi(\hat{v})) - \sum_i \wp_i(\chi(\hat{v}))$. The weak budget balanced property is also important because there would be no single agent's effect in the VCG mechanism when the VCG mechanism achieves the weak budget balanced property.

3. To improve the model for the case where the agents have different working abilities and the tasks are different.

   Since every agent can be different and every task can also be unique, an agent may be able to work on two tasks at the same time. The agent would return to the market when it can work on another task. Therefore, the agent's private information includes the values, the processing time and the maximal amount of tasks that it can simultaneously process. The auctioneer needs to consider whether this extra information would be truthfully declared by agents.

Bibliography

[1] M. Babaioff, R. Lavi, and E. Pavlov. Mechanism design for single-value domains. In *National Conference on Artificial Intelligence*, pages 241–247, 2005.

[2] Dirk Bergemann and Juuso Valimaki. Efficient dynamic auctions. Cowles Foundation Discussion Papers 1584, Cowles Foundation for Research in Economics, Yale University, October 2006.

[3] J. Bredin and D. C. Parkes. Models for truthful online double auctions. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI'05)*, pages 50–59, 2005.

[4] J Bredin, David C. Parkes, and Q Duong. Chain: A dynamic double auction framework for matching patient agents. *Journal of Artificial Intelligence Research*, 30:133–179, 2007.

[5] R. Cavallo. Efficiency and redistribution in dynamic mechanism design. In *Proc. 9th ACM Conf. on Electronic Commerce (EC'08)*, pages 220–229, Chicago, IL, 2008.

[6] R. Cavallo and David C. Parkes. Efficient metadeliberation auctions. In *Proc. 23rd AAAI Conference on Artificial Intelligence (AAAI'08)*, pages 50–56, Chicago, IL, 2008.

[7] R. Cavallo, David C. Parkes, and S. Singh. Efficient mechanisms with dynamic populations and dynamic types. Technical report, Harvard University, 2010.

[8] F. Constantin and David C. Parkes. Self-Correcting Sampling-Based Dynamic Multi-Unit Auctions. In *10th ACM Electronic Commerce Conference (EC'09)*, pages 89–98, 2009.

[9] Florin Constantin, Takayuki Ito, and David C. Parkes. Online auctions for bidders with interdependent values. In *Autonomous Agents & Multiagent Systems Agent Theories, Architectures, and Languages*, 2007.

[10] J. C. Gittins. *Multi-armed Bandit Allocation Indices*. Wiley, Chichester, NY, 1989.

[11] S. Gujar and D. C. Parkes. Dynamic matching with a fall-back option. In *Proceedings of the 19th European Conference on Artificial Intelligence (ECAI'10)*, pages 263–268, 2010.

[12] M. T. Hajiaghayi, R. Kleinberg, M. Mahdian, and D. C. Parkes. Online auctions with re-usable goods. In *Proceedings of the 6th ACM Conference on Electronic Commerce (EC)*, pages 165–174, 2005.

[13] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics*, 2:83–97, 1955.

[14] R. Lavi, N. Nisan, and A. Mualem. Towards a characterization of truthful combinatorial auctions. In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, 2003.

[15] R. P. Mcafee. A dominant strategy double auction. *Journal of Economic Theory*, 56:434–450, 1992.

[16] James Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5:32–38, 1957.

[17] Roger B. Myerson and Mark A. Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 29:265–281, 1983.

[18] N. Nisan and A. Ronen. Computationally feasible VCG mechanisms. In *ACM Conference on Electronic Commerce*, pages 242–252, 2000.

[19] David C. Parkes and Q. Duong. An ironing-based approach to adaptive online mechanism design in single-valued domains. In *Proc. 22nd National Conference on Artificial Intelligence (AAAI'07)*, pages 94–101, 2007.

[20] K. Roberts. The characterization of implementatable choise rules. *Aggregation and Revelation of Preferences*, pages 321–349, 1979.

[21] S. Seuken, R. Cavallo, and D. C. Parkes. Partially synchronized DEC-MDPs in dynamic mechanism design. In *AAAI'08*, pages 162–169, 2008.

[22] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[23] J. Zou, S. Gujar, and D. C. Parkes. Tolerable Manipulability in Dynamic Assignment without Money. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, pages 947–952, 2010.