Diploma Thesis

Frugality and Truthfulness in Approximate Mechanism Design

Patrick Briest

Diploma Thesis
Department of Computer Science
University of Dortmund

November 2004

Supervisors:

Dr. Piotr Krysta
Prof. Dr. Berthold Vöcking

Patrick Briest
Saarlandstr. 49
44139 Dortmund
patrick.briest@uni-dortmund.de

Student Registration Number: 84 86 8
Department of Computer Science
University of Dortmund

Frugality and Truthfulness in Approximate Mechanism Design

PATRICK BRIEST

Contents

Chapter 1

Introduction

Recently, much interest has been paid to questions concerning the application of algorithms in game theoretic settings as given, for example, by large and heterogenous networks in which autonomous systems compete for available resources. In these settings one is faced with the problem that different parts of the input might be given by different subsystems, each acting according to their personal interest rather than some common social objective. It then becomes desirable to have protocols that allow algorithms to compute output of a certain social quality and at the same time motivate the selfish participants to cooperate. While the roots of *mechanism design* are found in theoretical economics and results date back to as early as the 1960's [17], it has only recently become a field of interest for computer scientists. From the point of view of economics, focus has been laid on the design of protocols for various types of auctions that would ensure good expected revenue for the seller by motivating the participating bidders to truthfully report their interests. As computer scientists become involved, this focus shifts to questions of computational complexity. Since the number of goods and bidders in electronic auctions for resource allocation can become very large, it is clear that the resulting allocation of goods must be computed efficiently. *Algorithmic mechanism design* is concerned with the question whether – and how – these different objectives can be combined [11].

The following paragraphs give a short outline of the main parts of this work. They point out the main results and techniques and briefly discuss how they relate to previous work. For precise definitions of any mechanism design specific notions that are used in this discussion please refer to the respective chapters.

Chapter 2 introduces the field of algorithmic mechanism design formally and gives an overview of some of the most important previous results. We describe the VCG mechanism [17, 4, 6] and point out that its payment scheme cannot be combined with approximate algorithms. We then introduce some general notation that captures any utilitarian mechanism design (optimization) problem and adopt the characterization of truthful approximate mechanisms due to Lehmann et al. [9] and Mu'alem and Nisan [10] for this type of problems and known single–minded agents. Especially, we point out the fact that the design of truthful approximate mechanisms in the single–minded case is essentially equivalent to the construction of monotone approximation algorithms for the underlying optimization problem. Finally, we briefly mention the problem of measuring the quality of a mechanism's payment behavior and point out why comparing to the payments of a VCG mechanism appears reasonable and has become a well established technique.

Chapter 3 derives a bound on the difference between the payments of an exact VCG mechanism and the payments of any approximate mechanism for the same utilitarian minimization problem that depends only on the approximation ratio. We make use of the fact that the exact algorithm used in the VCG mechanism is a monotone algorithm and, thus, we can compare agents' critical values on a one–to–one basis. Except for a few remaining technical difficulties, this is essentially the main step towards a comparison of the total payments that mechanisms make to possibly different sets of selected agents. Applying some further argumentation due to Talwar [15] we obtain a bound that turns out to be quite similar to the only previous result of this kind presented by Calinescu [3]. In fact it can be seen as a worst case variant of this previous bound – a line of argumentation that will be picked up again later on.

Chapter 4 deals with reverse multi–unit auctions, in which a single auctioneer wants to buy a number of identical goods from a set of potential suppliers. The underlying NP–hard optimization problem is a minimization variant of the Knapsack Problem and has been studied, e.g., by Kothari et al. [8]. It is a representative example for the minimization problems among the class of utilitarian optimization problems we are faced with. We present three different approximation algorithms for this problem and prove that they are monotone and, thus, suitable for the design of truthful mechanisms. First, we adopt the simple greedy strategy presented by Mu'alem and Nisan [10] for forward multi–unit auctions to obtain a 2–approximation, which represents the best previously known result for this type of auctions. We then add an enumeration technique due to Sahni [13] in order to get a PTAS and show that this technique is monotonicity preserving. We point out that both algorithms have strongly polynomial running time and can be applied to the more general scenario of unknown single–minded agents, as well. Following this we present a monotone FPTAS with weakly polynomial running time which is limited to known single–minded agents. While the well known approach of scaling the input appropriately before applying an exact pseudo–polynomial algorithm fails to be monotone, since some rounding has to be done based on the input, this problem can be avoided by applying another enumerative method. We specify our algorithm as enumerating over an infinite set of possible ways to round the scaled input. To obtain a polynomial time implementation we then limit the number of possibilities that can actually lead to good solutions.

For the greedy 2–approximation and the PTAS we present examples that show that the bound presented in Chapter 3 is tight in an asymptotic sense. This qualifies reverse multi–unit auctions as a possible near worst case example in terms of overpayment among all utilitarian mechanism design minimization problems. Applying the bound to the FPTAS, we see that the resulting mechanism truly approximates the VCG mechanism with respect to both cost and payment, since it shows that we can make the difference in payment any polynomially small fraction of the VCG payment by choosing the (polynomial) approximation ratio with respect to the cost accordingly. To further motivate this type of approximation, we prove that the computation of VCG payments for reverse multi–unit auctions is itself NP–hard.

Chapter 5 deals with other utilitarian problems considered in the context of mechanism design. We first show that the algorithms presented for reverse multi–unit auctions in Chapter 4 can easily be adopted for the forward variant. In the context of maximization problems we point out that the enumerative method from Chapter 4 can be used to obtain monotone FPTASs for a whole class of utilitarian problems, if an appropriate pseudo–polynomial algorithm exists. We show that this is the case for forward multi–unit auctions, Job Scheduling with Deadlines and the Constrained Shortest Path Problem. We obtain the first monotone FPTASs for all these problems. Finally, we show that a modified version of the results from Chapter 3 also applies to maximization problems and point out the main differences.

Chapter 2

Algorithmic Mechanism Design

The field of *algorithmic mechanism design* is concerned with the application of various algorithms in scenarios where input is presented by autonomous individuals, called *agents*. In this model of social interaction, agents are considered to be driven by some sort of private interest, i.e., having quite individual preferences as far as the algorithm's output is concerned. As to ensure that all agents can be satisfied by the algorithm, we allow that, besides presenting some output, additional payments are handed out. We term this combination of algorithm and additional payments a *mechanism*. Since mechanism design is closely related to many aspects of classical game theory, its roots also lie in the study of economics. From the view of computer science, interest focuses on questions of computational complexity that arise in this context.

As an example, look at the following scenario. The Internet connects an enormous number of resources (e.g. CPU–time, storage space etc.), making them accessible from anywhere around the world. Imagine that we are going to design some new operating system, which takes advantage of this fact. Of course, any distributed operating system faces the problem of allocating available resources in an efficient manner among the jobs that need to be processed (and, especially, choosing a subset of all available resources that will be sufficient to process all jobs), but wouldn't it be nice if this could be done among all the computers running our system, maybe even on the Internet as a whole, making lack of resources a problem of the past? Obviously, we are faced with a major challenge here, but for the moment let us just look at the following aspect: Clearly, we will have to deal with the fact, that different resources on the Internet are owned by different people or companies (let us call them agents). All these agents might have more or less interest in making their resources available to our operating system. Since, nevertheless, they all have stated their general will to participate in our resource sharing by buying our operating system, their interest in making a certain resource available is expressed by the price at which it is offered. (Little interest in making a noisy needle printer accessible to the public results in a correspondingly high price.) Hence, there is a simple protocol for the interaction between agents and the mechanism. Agents specify prices for the resources they own, the mechanism decides which resources will be used and how much each agent will be paid. (Note, that for every resource we must pay at least the specified price, but are free to pay any larger amount.) Depending on the mechanism's selection, each agent then decides whether to be satisfied by the result or not.

The major interest of the field is in the design of so–called *truthful* mechanisms, i.e., mechanisms that are robust against manipulation by individual agents. In our example above, it is each agent's interest to maximize the profit made from sharing her resources. To achieve this, depending on the mechanism that is used, it might not necessarily be the best strategy to specify for each resource its true price. Rather, it could be helpful to declare a higher price and maybe be paid more, accordingly. Hence, we are faced with the possibility that agents have an incentive to lie about the true price of their resources in order to increase their personal profit. We say that a mechanism is truthful, if no agent can gain anything by lying, i.e., presenting false input to the mechanism.

Further examples of the application of algorithmic mechanisms are given by various routing problems (where, e.g., connection lines are owned by agents) or different types of auctions.

In the next section, mechanisms and the notion of *truthfulness* are defined formally.

2.1 Model

We present the formal definition of mechanisms and truthfulness as introduced by Nisan and Ronen [11]. We consider a set of n agents. Agent i has given as her private input $t_i^* \in T$, termed her *type*. Everything else in the scenario is public knowledge. Input to the mechanism is given as a type vector $t = (t_1, \ldots, t_n)$. An *output specification* maps each type vector t to a set of feasible outputs $O(t) \subseteq O$. Usually, we consider mechanisms for optimization problems. In these cases, the output specification ensures that a given objective function is optimized.

Depending on her type, agent i values the output presented by the mechanism. Formally, preferences are given as a real valued *valuation function* $v : T \times O \to \mathbb{R}$. We consider this valuation to be done in some common currency. In addition to producing an output, a mechanism may hand out payments, which are also measured in this currency, to the participating agents. If the mechanism's output is o and agent i is handed out an additional payment of p_i, then we say that $u_i = p_i + v(t_i^*, o)$ is agent i's *utility*. Agents are interested in maximizing their personal utility. The definition above is generally referred to as "quasi-linear". We will limit ourselves to this type of utility.

Definition 1. ([11]) *A* mechanism design problem *is defined by an output specification and a set of agents' valuation functions. If the output specification is given by an objective function $g(t, o)$ and a set of feasible outputs $O(t)$ and we require an output o that optimizes $g(t, o)$, we speak of a* mechanism design optimization problem.

Just as algorithms are formally defined for classical optimization problems, we can now introduce *mechanisms* based on this problem definition.

Definition 2. ([11]) *A* mechanism $m = (o, p)$ *consists of an* output function $o : T^n \to O$ *(specifying an algorithm) and an n–tuple of payments* $(p_1, \ldots, p_n)$.

1. *For each agent i, the mechanism defines a set of possible types $T_i \subseteq T$. Each agent i can report any type $t_i \in T_i$ to the mechanism.*
2. *Depending on the reported types $t = (t_1, \ldots, t_n)$, the mechanism produces an output $o = o(t) \in O(t)$.*
3. *Depending on the reported types, the mechanism hands out a payment $p_i = p_i(t)$ to each agent i.*

We now have a formal description of the interaction between a mechanism and the participating agents. The mechanism produces its output o based on the reported types t_i. Agents aim to maximize their personal utility $u_i = p_i + v(t_i^*, o)$, which is based on their true types t_i^*. As was already mentioned before, this might give agents an incentive to lie about their types in order to increase their utility. We are interested in mechanisms ensuring that this does not happen, i.e., mechanisms in which the participating agents' utility is maximized if they report their true types.

Notation: Given a vector $t = (t_1, \ldots, t_n)$, we will denote $(t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n)$ by t_{-i}. (t_i, t_{-i}) refers to the vector $t = (t_1, \ldots, t_n)$.

Definition 3. ([11]) *Let agent i have true type t_i^*. A mechanism $m = (o, p)$ is called* truthful*, if for all reported types t_{-i} and any $t_i \neq t_i^*$:*

$$p_i(t_i^*, t_{-i}) + v(t_i^*, o(t_i^*, t_{-i})) \geq p_i(t_i, t_{-i}) + v(t_i^*, o(t_i, t_{-i}))$$

Hence, agent i cannot increase her utility by untruthfully reporting some type t_i instead of her true type t_i^*.

In the definition of truthfulness above, we require that agents' utility cannot be increased by reporting any false type to the mechanism. We say that truth-telling is a *dominant strategy*. If we require that agents' utility strictly decreases when a false type is reported, truth-telling becomes the only dominant strategy. In these cases, we speak of *strongly truthful* mechanisms. We will, however, work with the weaker notion of truthfulness as defined above and assume that agents have no incentive to lie about their types as long as no real gain in terms of utility can be expected.

We have defined truthful mechanisms by their property of making truth–telling a dominant strategy for all participating agents. The next section addresses the question of how to construct truthful mechanisms in a very general setting.

2.2 Utilitarian Problems and the VCG Mechanism

One of the most important positive results in mechanism design is certainly what is usually referred to as the generalized *Vickrey–Clark–Groves (VCG) Mechanism* [4, 6, 17]. We will first describe the scenario in which the VCG mechanism can be applied, then define the mechanism itself formally and finally show that the requirement of truthfulness as defined in the former section is met.

As already mentioned, we are interested in mechanisms for optimization problems. Hence, we are given an objective function $g(t, o)$ depending on the reported types t and the presented output o. The mechanism's output must be chosen to optimize g.

To describe the VCG mechanism formally, we need to decide whether we are facing a minimization or maximization problem. There seems to be a tradition of describing general VCG mechanisms only for the case of maximization. We will, of course, follow this tradition, which is most likely a consequence of the standard definition of agents' utility as found in the former section. Remember, that we stated that agents' incentive is to maximize their personal utility and that agents' valuations contribute to utility as a positive summand. From the following definition it becomes clear, why it is convenient to assume a maximization problem.

To be able to apply the VCG mechanism, we require the objective function to be simply the sum of all agents' valuations.

Definition 4. ([11]) *A (maximization) mechanism design problem is called* utilitarian *if its objective function g satisfies $g(t, o) = \sum_{i=1}^{n} v(t_i, o)$.*

Given a utilitarian problem, we can now define the VCG mechanism. Obviously, the mechanism has no alternative to choosing its output to maximize the objective function. Hence, the main question is how to choose the payments handed out to the agents.

Definition 5. *Let t be the vector of declared types and v valuation function. We say that $m = (o, p)$ is a* VCG mechanism *if:*

1. $o(t) \in \mathrm{argmax}_o(\sum_i v(t_i, o))$
2. $p_i(t) = \sum_{j \neq i} v(t_j, o(t)) - \sum_{j \neq i} v(t_j, o(t_{-i}))$

What is the intuition behind this quite technical definition? Let us, for example, assume that agents' valuations are payments that the agents are willing to make to the mechanism for being included in the solution. (This is the well known setting of several types of auctions.) By participating in the mechanism, agent i can only increase the value of the presented optimal solution, often referred to as *social welfare*. (This is due to the fact that the mechanism is free to ignore agents that do not contribute to the improvement of the solution.) The idea of the VCG mechanism's payment scheme is to give this increase in social welfare as a bonus to the agent who caused it. Hence, if agent i causes the social welfare to increase by c, her payment to the mechanism will be $v(t_i, o) - c$ instead of $v(t_i, o)$. In this example $o(t_{-i})$ denotes the mechanism's output, if agent i does not participate and, thus, cannot be included in the solution. Note, that in terms of this example the payment defined above has a negative sign and, thus, an agent's utility corresponds to the bonus given by the mechanism.

We observe that the VCG mechanism for an optimization problem does not have to be unambiguous, since for some problem instances the optimal solution might not be well defined. Additionally, we have some freedom in defining $o(t_{-i})$. In fact, $o(t_{-i})$ can be chosen to be any feasible output, as long as it does not depend on v_i. For our purposes, we abstract away from this problem and let "the VCG mechanism" refer to the mechanism obtained by fixing any exact deterministic algorithm for the optimization problem and some output $o(t_{-i})$ for the payment scheme. In Chapter 3 we define a more special version of the VCG mechanism.

Remark 1. *It is important to note that in the VCG mechanism, the payment for agent i is defined independently of agent i's reported type, thus $p_i = p_i(t_{-i})$. As we will see later, this is a necessary qualifier for any truthful mechanism.*

It now remains to be shown that the VCG mechanism is truthful. The proof is included here, because it is usually done in slightly different settings, resulting in the fact that proofs don't match the notation used here. Besides this, the proof points nicely towards the key problem that is addressed in the next section.

Theorem 1. ([11]) *The VCG mechanism is truthful.*

Proof: Consider declarations $t = (t_1, \ldots, t_n)$, where t_i is agent i's true type. Now let agent i untruthfully declare some type $t'_i \neq t_i$. Let t' denote the vector t with t'_i instead of t_i. We note that

$$\sum_j v(t_j, o(t)) \geq \sum_j v(t_j, o(t')),$$

since $o(t) \in \mathrm{argmax}_o(\sum_j v(t_j, o))$. Now let $u_i(t_i)$ denote agent i's utility when declaring type t_i. Then

$$\begin{aligned}
u_i(t_i) &= p_i(t) + v(t_i, o(t)) \\
&= \sum_j v(t_j, o(t)) - \sum_{j \neq i} v(t_j, o(t_{-i})) \\
&= \sum_j v(t_j, o(t)) - \sum_{j \neq i} v(t'_j, o(t'_{-i})), \text{ since } t'_j = t_j \text{ for } j \neq i \\
&\geq \sum_j v(t_j, o(t')) - \sum_{j \neq i} v(t'_j, o(t'_{-i})), \text{ as noted above} \\
&= p_i(t') + v(t_i, o(t')) \\
&= u_i(t'_i).
\end{aligned}$$

Hence, agent i cannot increase her utility by declaring any $t'_i \neq t_i$. This proves the claim. □

2.3 Approximation Algorithms and Monotonicity

We return to our introductory example from the beginning of the chapter, where the selection of a subset of available resources that are connected through the Internet by a common distributed operating system was presented as a possible application of algorithmic mechanisms. Since even quite limited cases of the underlying optimization problem are known to be NP–hard (e.g. Set Cover), we cannot hope to be able to find exact solutions to our problem in a large scale scenario as the Internet. So what does this mean for our operating system?

Instead of giving up the complete project, we could certainly consider applying some approximation algorithm to the problem. Computing only an approximately efficient resource selection would primarily lead to a number of resources not working to full capacity, a drawback that we would probably be able to cope with. So approximation algorithms seem to be an acceptable compromise for our system. But what about the second important quality of our mechanism: truthfulness? What would be the result if we didn't require the mechanism to be truthful, or maybe only required it to be *approximately truthful*? The effect would be an increase of the payments made to agents for the use of their resources. A bound for this increase has been presented by Kothari, Parkes and Suri [8], who show that the overpayment to an individual agent can become quite large. In our function as operating system developers, however, we are of course very much obliged not to cause any unnecessary cost for our customers. Hence, our interest is in embedding approximation algorithms in mechanisms that are still strictly truthful.

An initial idea could now be to add the VCG payment scheme to an approximation algorithm to obtain truthfulness. If we take a look at the proof of truthfulness for the VCG mechanism, however, we note that the application of an exact algorithm to the optimization problem is a necessary assumption in the proof. Hence, the VCG payment scheme cannot be used in combination with approximation algorithms. But the truth looks even worse. Lehmann, O'Callaghan and Shoham [9] showed that there are optimization problems and approximation algorithms, such that there exists no payment scheme at all that leads to a truthful mechanism.

In the remainder of this section, we will show how truthful approximate mechanisms can be constructed for a limited type of participating agents and what sort of approximation algorithms can be used for this. The cited results are due to Mu'alem and Nisan and were presented in [10] in the setting of (single–item) *multi–unit auctions*. Proofs are included as to match the slightly more general notation used here.

We start by giving a formal definition of an algorithm's approximation ratio and some related notation.

Definition 6. *Let* Π *be an optimization problem,* D_Π *the set of instances of* Π *and* A *an approximation algorithm for* Π*. For* $I \in D_\Pi$ *let* $A(I)$ *denote* A*'s solution,* $Opt(I)$ *the* *optimal solution on instance I, $w(A(I))$ and $w(Opt(I))$ the respective objective function values. We define:*

$$R_A(I) := \begin{cases} w(A(I))/w(Opt(I)), & \textit{if } \Pi \textit{ is minimization problem} \\ w(Opt(I))/w(A(I)), & \textit{if } \Pi \textit{ is maximization problem} \end{cases}$$

We say that A has approximation ratio

$$R_A := \sup\{R_A(I) | I \in D_\Pi\}.$$

Convention: In contrast to the previous section, we will from now on assume minimization problems when considering optimization.

We will first define a limited type of agent, for which we then characterize truthful approximate mechanisms. As a first step we want to consider agents that are usually referred to as being *single–minded*. In most cases this means that agents' valuations only differentiate between being part of the mechanism's solution or not. In our operating system example, we could imagine that each agent owns exactly one resource. Now some agent's resource might be chosen as a part of the allocation or it might not – there is, however, no "in between". Formally, each agent's valuation function partitions the set of possible outcomes into two disjoint sets.

Definition 7. ([9]) *Consider agent i with type t_i. We say that agent i is* single–minded*, if $t_i = (O_i^+, v_i)$, $O = O_i^+ \dot\cup O_i^-$ disjoint and*

$$v(t_i, o) = \begin{cases} v_i, & \textit{if } o \in O_i^+ \\ 0, & \textit{if } o \in O_i^-. \end{cases}$$

If $o \in O_i^+$, we say that agent i is selected*. By $S(o)$ we denote the set of selected agents in output o.*

For our intention single–minded bidders are still too complex to deal with. Look at our operating system once again. An agent's preferred output O_i^+ might include the information, what sort of resource the agent owns. Since agents' types are private knowledge, an agent might lie to our mechanism not only about the price of her resource, but about the resource's type, as well. To eliminate this uncertainty we introduce a further limitation by defining the so–called *known* single–minded agent.

Definition 8. *We say that a single–minded agent i is* known*, if her preferred output O_i^+ is known to the mechanism.*

Notation: For simplicity of notation, we drop the notions of "type" and "valuation function" when considering known single–minded agents. Instead, each agent i has a single positive value v_i^* as her private input. The values that are reported to the mechanism are denoted by $v = (v_1, \ldots, v_n)$. We will assume that $v_i^*, v_i > 0$. Hence, v_i describes the minimum price to be paid to agent i if she is chosen for the solution. As was already mentioned earlier, we now need to adapt our definition of utility to this setting. Since the mechanism's payments can now be seen as a compensation for the cost caused by the mechanism's output, utility is now defined as follows.

Definition 9. *Let $m = (A, p)$ be a mechanism for known single–minded agents. Agent i's utility is defined as*

$$u_i = \begin{cases} p_i - v_i^*, & \text{if agent } i \text{ is selected} \\ p_i, & \text{else.} \end{cases}$$

We will now characterize algorithms that can be embedded in truthful mechanisms for known single–minded agents. We consider mechanisms $m = (A, p)$, where A is some (approximation) algorithm for the underlying optimization problem.

Definition 10. *Algorithm A is* monotone *if for any agent i and declarations v_{-i}:*

$$o(v_i, v_{-i}) \in O_i^+ \Rightarrow o(v_i', v_{-i}) \in O_i^+ \text{ for all } v_i' \leq v_i.$$

This definition appears quite natural. If v_i is a winning declaration for agent i, then we require that this is also true for any smaller declaration $v_i' \leq v_i$. The following is a straight–forward observation.

Lemma 1. ([10]) *Let A be a monotone algorithm. Then for any v_{-i} there exists a single critical value $\theta_i^A = \theta(A, v_{-i}) \in (\mathbb{R}_0^+ \cup \infty)$, such that $o(v_i, v_{-i}) \in O_i^+$ for all $v_i < \theta_i^A$ and $o(v_i, v_{-i}) \in O_i^-$ for all $v_i > \theta_i^A$.*

Proof: First consider the case that no critical value θ_i^A exists. Then there must be declarations $v_i^1 < v_i^2$, such that $o(v_i^1, v_{-i}) \in O_i^-$ and $o(v_i^2, v_{-i}) \in O_i^+$. This contradicts A's monotonicity and, thus, cannot be the case. If, on the other hand, $\theta_i^1 \neq \theta_i^2$ are distinct critical values, we let $\varphi = (\theta_i^1 + \theta_i^2)/2$ and observe that $o(\varphi, v_{-i}) \in O_i^+ \cap O_i^- = \emptyset$, i.e., φ is winning and loosing declaration simultaneously. This contradiction finishes the proof. □

For agent i, the critical value θ_i^A is the supremum value that she can declare and still be selected. It is important to note that θ_i^A does not depend on the value declared by agent i. Critical values turn out to be the key to the construction of a truthful mechanism around algorithm A.

Definition 11. *The* critical value payment scheme *p^A associated with the monotone algorithm A is defined by $p_i^A = \theta_i^A$ if agent i is selected and $p_i^A = 0$ otherwise.*

It now remains to be shown that adding the associated critical value payment scheme to a monotone approximation algorithm will indeed result in the desired truthful approximate mechanism.

Lemma 2. ([10]) *Let A be a monotone algorithm and p_A the associated critical value payment scheme. Then $m = (A, p^A)$ is a truthful mechanism for known single–minded agents.*

Proof: Fix declarations v_{-i} and consider agent i's utilities u_i^*, u_i as results of declaring v_i^* or $v_i \neq v_i^*$, respectively.

1. $o(v_i^*, v_{-i}), o(v_i, v_{-i}) \in O_i^+$. Then $p_i^A(v_i^*, v_{-i}) = p_i^A(v_i, v_{-i})$ and, thus, $u_i^* = u_i$.
2. $o(v_i^*, v_{-i}), o(v_i, v_{-i}) \in O_i^-$. Then $u_i^* = p_i^A(v_i^*, v_{-i}) = 0 = p_i^A(v_i, v_{-i}) = u_i$.
3. $o(v_i^*, v_{-i}) \in O_i^+$, $o(v_i, v_{-i}) \in O_i^-$. Obviously $v_i^* < \theta_i^A < v_i$ and, hence,
$$u_i = 0 < \theta_i^A - v_i^* = p_i^A(v_i^*, v_{-i}) - v_i^* = u_i^*.$$
4. $o(v_i^*, v_{-i}) \in O_i^-$, $o(v_i, v_{-i}) \in O_i^+$. Obviously $v_i < \theta_i^A < v_i^*$ and, hence,
$$u_i = p_i^A(v_i, v_{-i}) - v_i^* = \theta_i^A - v_i^* < 0 = u_i^*.$$

We note that $u_i \leq u_i^*$ in each case. It follows that truthfully declaring v_i^* is a dominant strategy for agent i. □

Lemma 2 shows that we have indeed found a way to construct truthful mechanisms for a limited type of agents. It turns out, however, that this characterization even captures essentially all truthful mechanisms. A mechanism $m = (A, p)$ is called *normalized*, if agents that are not selected pay 0, i.e., $p_i = 0$ for $o \in O_i^-$.

Lemma 3. ([10]) *Let $m = (A, p)$ be a truthful normalized mechanism for known single–minded agents. Then A is monotone and p is the associated critical value payment scheme.*

Proof: Assume that A is not monotone. Then there are declarations $v_i < v_i'$, such that $o(v_i, v_{-i}) \in O_i^-$ and $o(v_i', v_{-i}) \in O_i^+$. Now assume that $v_i = v_i^*$. If agent i is selected, the payment p_i made to agent i is no less than the declared value v_i'. (Otherwise the agent could increase her utility by untruthfully declaring ∞.) For the resulting utilities u_i' and u_i^*, we have:
$$u_i' = p_i(v_i', v_{-i}) - v_i^* \geq v_i' - v_i^* > 0 = u_i^*$$
This contradicts A's truthfulness.

We have already seen that $p_i \geq v_i$ for any v_i that results in agent i being selected. Thus, $p_i \geq \theta_i^A$. Now assume that $p_i > \theta_i^A$ and consider the case of $\theta_i^A < v_i^* < p_i$. Untruthfully declaring $\theta_i^A - \epsilon$ obviously results in positive utility, while truthfully declaring v_i^* causes utility to be 0. This, again, contradicts A's truthfulness. It follows that $p_i = \theta_i^A$. □

Theorem 2. ([10]) *A normalized mechanism $m = (A, p)$ for known single–minded agents is truthful if and only if A is monotone and p is the associated critical value payment scheme.*

2.4 Frugality Considerations

Once more, let us return to our operating system example for known single–minded agents and have a closer look at the optimization goal. We are dealing with the combinatorial problem of choosing a subset $S \subseteq \{1, \ldots, n\}$ of agents to supply us with a set of required resources, S satisfying $S \in \operatorname{argmin}_{S'} \sum_{i \in S'} v_i$. Remember, that the v_i's are the prices declared by the agents. Thus, we are minimizing the sum of agents' declarations (usually referred to as *cost to society*) and not the sum of payments actually made to the agents, which (as we have seen in the previous section) can be written as $\sum_{i \in S} \theta_i$.

It is easy to substantiate the claim that for practical purposes, minimization of this second objective appears more reasonable, since it is obviously good to minimize the actual payment that has to be made, while it is not necessarily clear why we should be interested in something like cost to society. (When considering auctions, such an interest might result from government acting as auctioneer, but the question of a government's role in operating system design has not yet been finally answered by the courts.) There are, however, no known ways of constructing mechanisms for payment optimization in the generality of our scenario. But can we say anything about the payments made by a mechanism that minimizes (or approximately minimizes) cost to society?

When we speak of an algorithm that approximates cost to society, we measure the algorithm's approximation ratio by comparing the value of the computed output to the value of an optimal solution. We now need to decide how we are going to measure the payments made by an approximate mechanism. Since there is always an optimal objective function value that can be reached with respect to the underlying (minimization) problem, one could obviously try to define the "optimal payment" for a given problem instance analogously. This immediately leads to the following question. Given a set of declarations $(v_1^*, \ldots, v_n^*)$, what is the smallest payment that can be achieved by any truthful mechanism? The answer is given by the following observation.

Observation 1. *Assume declarations $(v_1^*, \ldots, v_n^*)$. If the set S^* of agents with*

$$c(S^*) := \sum_{i \in S^*} v_i^*$$

minimizes the cost to society (and there is sufficient competition), then there is a truthful mechanism that makes a total payment of exactly $c(S^*)$.

How does such a mechanism look like? Consider the following mechanism $m = (A, p^A)$, where p^A is the critical value payment scheme and A is defined in a way that guarantees just the critical values we need for our argumentation.

In Observation 1 we require "sufficient competition". As becomes clear from the following definition of A, this means the existence of at least two disjoint feasible solutions, which is required for the else–part of the algorithm. It is important to note that values $v_1^*, \ldots, v_n^*$ are used in the definition, i.e., the algorithm's definition is based on a fixed problem instance.

Algorithm 1: A

Input: Declarations $(v_1, \ldots, v_n)$.

1. If $v_i \leq v_i^*$ for all $i \in S^*$, then set $S = S^*$.
2. Else choose the best possible solution that does not select agents from S^*. Formally, let $S \subseteq \{1, \ldots, n\} \backslash S^*$ have minimal cost.

Output: Subset of agents S.

We observe that the critical value for any agent $i \in S^*$ and an arbitrary vector of declarations can be written as

$$\theta_i^A = \begin{cases} v_i^*, & \text{if } v_j \leq v_j^* \, \forall \, j \in S^* \backslash \{i\} \\ 0, & \text{else.} \end{cases}$$

Applying this to declaration vector $(v_1^*, \ldots, v_n^*)$ immediately implies the total payment claimed in Observation 1.

Clearly, there is no way to further improve the payment of $c(S^*)$ in this situation, since each agent must be paid no less than her declared value. Thus, we see that on every set of declarations, optimal cost to society and smallest possible payment are identical. But is it fair to compare other mechanisms' payments to this optimum? The obvious answer is no. The above mechanism shows outstanding performance in terms of payment on a small number of declaration vectors, but one can easily construct other examples in which payment and cost to society become arbitrarily bad.

How do we then judge the payment made by some (approximate) mechanism? The established way of measuring a mechanism's payment behavior (or *frugality*, as to include the case of agents being charged instead of paid, as well) is to compare it to the payments made by the VCG mechanism. This seems reasonable, since (although there are no such results for the general scenario we are interested in) it has been shown [2, 1] that VCG is optimal in terms of frugality in certain economical settings (assuming, for example, some possibility of resale between agents). Chapter 3 deals with the payment behavior of approximate mechanisms for utilitarian problems.

Chapter 3

Payment of Approximate Utilitarian Mechanisms

We have seen that mechanisms for optimization problems consist of two parts. An algorithm produces output that optimizes some given objective function depending on agents' declarations. A payment scheme defines additional payments intended to ensure agents' interest to cooperate with the mechanism. Utilitarian optimization problems were distinguished by the fact that their objective function is simply the sum of agents' declarations. Mechanisms for this sort of optimization problem are also termed utilitarian. For utilitarian mechanisms and known single–minded agents we have seen how payments that will ensure truthfulness can be derived directly from the algorithm that produces the output by applying the associated critical value payment scheme.

This chapter deals with approximate mechanisms and the relation between the approximation ratio of the algorithm in use and the resulting payments. We will prove that overpayment decreases as the approximation ratio with respect to the objective function is improved. More formally, we will compare the payments made by a truthful approximate mechanism for some utilitarian problem to the payments made by the corresponding VCG mechanism on identical input.

The current chapter will again be focused on minimization problems only. This is due to the fact that, on one hand, this is a necessary requirement for one of the conclusion we are going to make and, on the other hand, it immediately relates to the problem of *reverse multi–unit auctions* that we are going to consider later on. Nevertheless, similar results also apply to maximization problems and are briefly discussed in Section 5.4.

We consider a utilitarian (minimization) problem defined by the output specification that is given by feasible solutions O (and the usual utilitarian objective) and mechanisms $m_A = (A, p^A)$ and $m_{Opt} = (Opt, p^{Opt})$. We assume that Opt is an exact algorithm for the problem, while A is an approximation algorithm with approximation ratio $(1 + \varepsilon)$.

Notation: Let $A(v)$ and $Opt(v)$ denote the output computed by A and Opt, respectively, on declaration vector v, $S(A(v))$ and $S(Opt(v))$ the sets of selected agents, $c(A(v))$ and $c(Opt(v))$ the corresponding objective function values (or *cost*). By $Opt(v \mid i)$ we refer to an optimal solution under the condition that agent i must be selected. Formally:

$$Opt(v \mid i) \in \mathrm{argmin}_{o \in O_i^+} \sum_{j \in S(o)} v_j$$

Analogously, $Opt(v|\neg i)$ denotes an optimal solution under the condition that agent i cannot be selected:

$$Opt(v \mid \neg i) \in \mathrm{argmin}_{o \in O_i^-} \sum_{j \in S(o)} v_j$$

3.1 A normalized VCG mechanism

Let us have a look at the VCG mechanism $m_{Opt} = (Opt, p^{Opt})$. In Section 2.2 VCG payments were defined for maximization problems and the most general type of agents. Remember, that in our current model agent i's type is reduced to her preferred output O_i^+ and a single corresponding valuation v_i. The definition of VCG payments from Section 2.2 is:

$$p_i^{Opt}(v_{-i}) = \sum_{j \neq i} v(t_j, Opt(v_{-i})) - \sum_{j \neq i} v(t_j, Opt(v))$$

Note, that the two sums appear in inverted order compared to the original definition. This is due to the fact that we are now considering minimization problems. The left hand side of the equation is adapted as to represent the fact that only the v_i's are declared by the agents and the observation, that agent i's payment does not depend on v_i. As has already been mentioned in Section 2.2, there is a certain degree of freedom about the definition of $Opt(v_{-i})$, since we have only required $Opt(v_{-i})$ to be any output produced independently from agent i's declaration v_i. We now close this gap and define $Opt(v_{-i}) := Opt(v \mid \neg i)$.

Lemma 4. *Let $m_{Opt} = (Opt, p^{Opt})$ be VCG mechanism for a utilitarian problem and known single–minded agents with payment scheme p^{Opt} defined as above. Then:*

$$p_i^{Opt}(v_{-i}) = \begin{cases} c(Opt(v \mid \neg i)) - (c(Opt(v)) - v_i), & \textit{if } i \in S(Opt(v)) \\ 0, & \textit{else} \end{cases}$$

Proof: By definition, we have that:

$$\begin{aligned} p_i^{Opt}(v_{-i}) &= \sum_{j \neq i} v(t_j, Opt(v \mid \neg i)) - \sum_{j \neq i} v(t_j, Opt(v)) \\ &= \sum_{j \in S(Opt(v \mid \neg i)), j \neq i} v_j - \sum_{j \in S(Opt(v)), j \neq i} v_j \\ &= c(Opt(v \mid \neg i)) - \sum_{j \in S(Opt(v)), j \neq i} v_j. \end{aligned}$$

Assume that $i \in S(Opt(v))$. Then:

$$\sum_{j \in S(Opt(v)), j \neq i} v_j = \sum_{j \in S(Opt(v))} v_j - v_i = c(Opt(v)) - v_i$$

This proves the first part of the claim. On the other hand, if $i \notin S(Opt(v))$:

$$\sum_{j \in S(Opt(v)), j \neq i} v_j = \sum_{j \in S(Opt(v \mid \neg i))} v_j = c(Opt(v \mid \neg i))$$

This gives the second part. □

We note that this VCG mechanism is normalized. (Agents that are not selected pay 0.) We then have that the VCG mechanism $m_{Opt} = (Opt, p^{Opt})$ as defined above is truthful and normalized. From Theorem 2 it follows that Opt is monotone and p^{Opt} the associated critical value payment scheme. The following observation is a direct consequence of Lemma 4.

Observation 2. *The VCG mechanism is monotone with critical values*

$$\theta_i^{Opt} = c(Opt(v \mid \neg i)) - (c(Opt(v \mid i)) - v_i). \tag{3.1}$$

Somehow, it is intuitively clear that any exact algorithm must be monotone. Consider declarations v, such that agent i is not selected for the optimal solution. Now start to decrease the cost v_i declared by agent i. Obviously, the cost of the optimal solution $Opt(v \mid i)$ that is forced to select i decreases as well. At some point, this solution must become globally optimal. From this point on, agent i is selected for the solution. Hence, θ_i^{Opt} is the infimum value that agent i must declare, such that she is a part of the globally optimal solution.

3.2 Comparing critical values

In the next step we will now fix some agent i and declarations v_{-i}. We compare agent i's critical value θ_i^A in algorithm A to her optimal critical value θ_i^{Opt} and show a bound on the distance between those two. This bound will then be used to compare the total payment handed out by the mechanisms.

Lemma 5. *Let Opt be an optimal algorithm for some utilitarian problem and let A be monotone approximation algorithm for the same problem with approximation ratio $(1+\varepsilon)$. Then the following inequality holds for all declarations v_{-i}:*

$$\theta_i^{Opt} - \frac{\varepsilon}{1+\varepsilon} \cdot c(Opt(v \mid \neg i)) \leq \theta_i^A \leq \theta_i^{Opt} + \varepsilon \cdot c(Opt(v \mid \neg i))$$

Proof: We start by showing the upper bound. Assume that there are declarations v_{-i} for which the inequality does not hold. Then there is a declaration v_i, such that

$$\theta_i^{Opt} + \varepsilon \cdot c(Opt(v \mid \neg i)) < v_i < \theta_i^A.$$

Since $v_i < \theta_i^A$, agent i must be selected by A. We conclude that

$$\begin{aligned} c(A(v)) &\geq c(Opt(v \mid i)) \text{ , since } i \in S(A(v)) \\ &= v_i + (c(Opt(v \mid i)) - v_i) \\ &> \theta_i^{Opt} + (c(Opt(v \mid i)) - v_i) + \varepsilon \cdot c(Opt(v \mid \neg i)), \text{ by assumption} \\ &= (1+\varepsilon) \cdot c(Opt(v \mid \neg i)), \text{ by (3.1)} \\ &\geq (1+\varepsilon) \cdot c(Opt(v)). \end{aligned}$$

This contradicts A's approximation ratio. Hence, $\theta_i^A \leq \theta_i^{Opt} + \varepsilon \cdot c(Opt(v \mid \neg i))$.

We now show the lower bound. Once again, assume that the inequality does not hold for declarations v_{-i} and consider v_i with

$$\theta_i^A < v_i < \theta_i^{Opt} - \frac{\varepsilon}{1+\varepsilon} \cdot c(Opt(v \mid \neg i)).$$

Since $v_i < \theta_i^{Opt}$, agent i is selected for the optimal solution. Thus, we have:

$$\begin{aligned} c(Opt(v)) &= c(Opt(v \mid i)), \text{ since } i \in S(Opt(v)) \\ &= v_i + (c(Opt(v \mid i)) - v_i) \\ &< \theta_i^{Opt} + (c(Opt(v \mid i)) - v_i) - \frac{\varepsilon}{1+\varepsilon} \cdot c(Opt(v \mid \neg i)), \text{ by assumption} \\ &= c(Opt(v \mid \neg i)) - \frac{\varepsilon}{1+\varepsilon} \cdot c(Opt(v \mid \neg i)), \text{ by (3.1)} \\ &= \frac{1}{1+\varepsilon} \cdot c(Opt(v \mid \neg i)) \end{aligned}$$

We note that $v_i > \theta_i^A$ and, hence:

$$\begin{aligned} c(A(v)) &\geq c(Opt(v \mid \neg i)), \text{ since } i \notin S(A(v)) \\ &> (1+\varepsilon) \cdot c(Opt(v)), \text{ by the inequality above} \end{aligned}$$

Once again, this contradicts algorithm A's approximation ratio and it immediately follows that $\theta_i^{Opt} - \frac{\varepsilon}{1+\varepsilon} \cdot c(Opt(v \mid \neg i)) \leq \theta_i^A$. This completes the proof. □

Figure 3.1 displays the central idea of the proof of Lemma 5. If agent i declares a value of exactly $v_i = \theta_i^{Opt}$, then there exist at least two optimal solutions that minimize cost to society, one that selects agent i and one that does not. This is our starting point. Now consider what happens if we start to increase v_i. The value of any solution that selects i behaves like a positive linear function in v_i and increases accordingly. At some point of time even the value

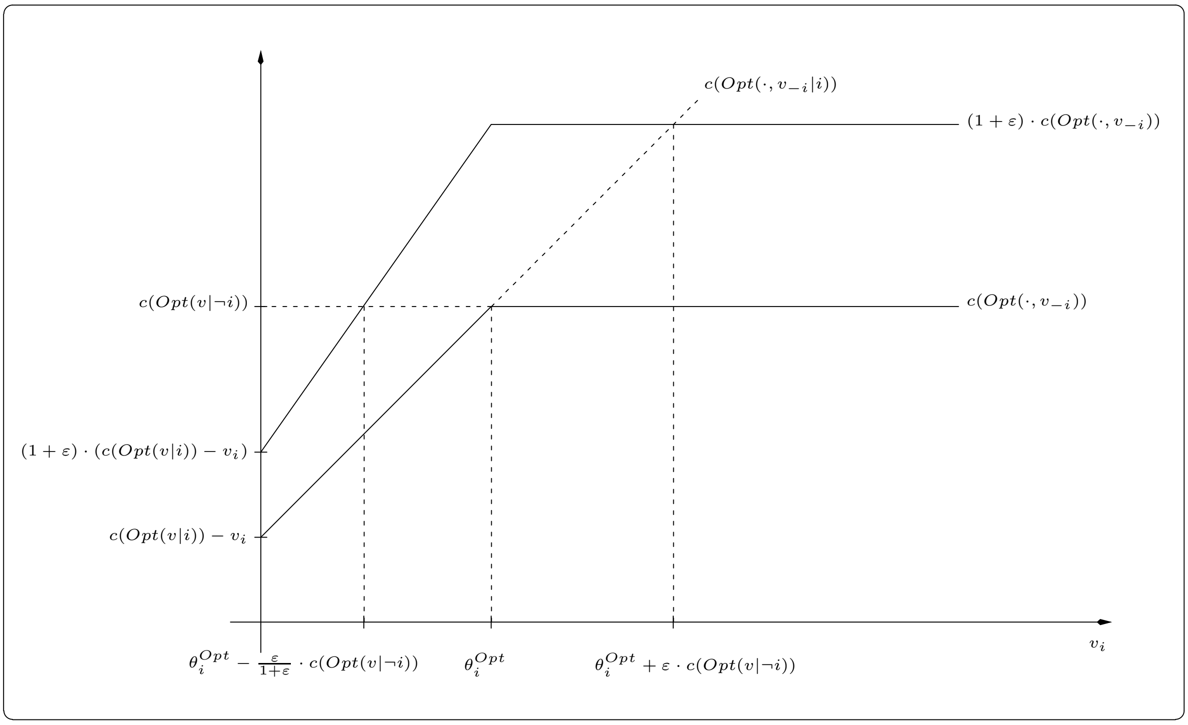

Figure 3.1: Range for the critical values θ_i^A of a monotone approximation algorithm A.

of the best solution selecting i will become larger than $(1+\varepsilon)$ times the value of an optimal solution not selecting i. At this point an algorithm A with approximation ratio $(1+\varepsilon)$ has no choice but to present a solution that does not select i, as well. Thus, we have found an upper bound on agent i's critical value in A. Just the same happens if we start to decrease v_i. The value of any solution selecting i decreases linearly until it becomes smaller than $(1+\varepsilon)^{-1}$ times the value of any solution not selecting i. This is the point when A must begin to select i as well to guarantee its approximation ratio. This gives the lower bound.

3.3 Comparing total payment

Using Lemma 5, we will now derive a bound on the total payment made by mechanism m_A. We have seen that the payments of both m_A and m_{Opt} are simply the sum of the critical values of all agents that are selected in each output. We have also shown how a single agent's critical values in the two mechanisms are related. Thus, the last problem we are still faced with is the fact that the sets of selected agents must, of course, not be identical for both mechanisms. Accordingly, the main challenge for the remainder of this section will be to describe the relation between the payments made to agents that are selected by only one of the mechanisms. Again, Lemma 5 will be helpful in accomplishing this task.

Notation: For mechanism m_A's output $A(v)$ and a set $T \subseteq S(A(v))$, let

$$c(T) := \sum_{i \in T} v_i \text{ and } p^A(T) := \sum_{i \in T} \theta_i^A$$

denote the cost to society of the partial selection T and the payment made by m_A to agents in T, respectively.

Theorem 3. *Let m_{Opt} and m_A be the mechanisms defined above. Fix some arbitrary declaration vector v and let $Opt = Opt(v)$ and $A = A(v)$ denote the computed outputs. Then the following inequalities hold:*

$$p^A(S(A)) \leq (1+\varepsilon) \cdot p^{Opt}(S(Opt)) + \varepsilon \cdot (|S(A)| + 1) \cdot c(Opt) \tag{3.2}$$

$$p^A(S(A)) \geq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot p^{Opt}(S(Opt)) - \frac{\varepsilon}{1+\varepsilon} \cdot |S(Opt)| \cdot c(Opt) \tag{3.3}$$

Proof: We start with the proof of (3.2). Let A_1, A_2 be disjoint sets with $S(A) = A_1 \,\dot{\cup}\, A_2$, $A_1 \subseteq S(Opt)$ and $A_2 \cap S(Opt) = \emptyset$. A_1 is the maximal set of agents selected by both m_A and m_{Opt}. We define $R := S(Opt) \backslash A_1$ and, thus, have $S(Opt) = A_1 \cup R$. Obviously, we have $c(A) = c(A_1) + c(A_2)$ and $c(Opt) = c(A_1) + c(R)$. Algorithm A has approximation ratio $(1+\varepsilon)$. Hence:

$$\frac{c(A_1) + c(A_2)}{c(A_1) + c(R)} \leq 1 + \varepsilon$$

Solving this inequality for $c(R)$ gives:

$$\begin{aligned} c(R) &\geq c(A_1) + c(A_2) - (1+\varepsilon) \cdot c(A_1) - \varepsilon \cdot c(R) \\ &= c(A_2) - \varepsilon \cdot (c(A_1) + c(R)) \\ &= c(A_2) - \varepsilon \cdot c(Opt) \end{aligned}$$

We now compare the payments made to agents $i \in A_1$ and agents $i \notin A_1$ separately. For the payment made by m_A to agents in A_1 we observe:

$$\begin{aligned} p^A(A_1) &= \sum_{i \in A_1} \theta_i^A \\ &\leq \sum_{i \in A_1} \theta_i^{Opt} + \varepsilon \cdot \sum_{i \in A_1} \underbrace{c(Opt(v \,|\, \neg i))}_{\leq c(Opt) + \theta_i^{Opt}}, \text{ by Lemma 5} \\ &\leq (1+\varepsilon) \cdot \sum_{i \in A_1} \theta_i^{Opt} + \varepsilon \cdot |A_1| \cdot c(Opt) \\ &= (1+\varepsilon) \cdot p^{Opt}(A_1) + \varepsilon \cdot |A_1| \cdot c(Opt) \end{aligned}$$

For the second part of our proof we look at the payment made by m_{Opt} to agents in R. First consider some agent $i \in A_2$. Since agent i is not selected by Opt, it must clearly be the case that $v_i \geq \theta_i^{Opt}$. We observe that $Opt = Opt(v \mid \neg i)$ in this case. From Lemma 5 it follows that $\theta_i^A \leq \theta_i^{Opt} + \varepsilon \cdot c(Opt)$. We conclude:

$$v_i \geq \theta_i^{Opt} \geq \theta_i^A - \varepsilon \cdot c(Opt) \tag{3.4}$$

Applying this, we get for the payment $p^{Opt}(R)$:

$$\begin{aligned}
p^{Opt}(R) &\geq c(R) \geq c(A_2) - \varepsilon \cdot c(Opt), \text{ as noted above} \\
&= \sum_{i \in A_2} v_i - \varepsilon \cdot c(Opt) \\
&\geq \sum_{i \in A_2} (\theta_i^A - \varepsilon \cdot c(Opt)) - \varepsilon \cdot c(Opt), \text{ by (3.4)} \\
&= p^A(A_2) - \varepsilon \cdot (|A_2| + 1) \cdot c(Opt)
\end{aligned}$$

Putting together what we have so far, we obtain:

$$\begin{aligned}
p^A(S(A)) &= p^A(A_1) + p^A(A_2) \\
&\leq (1+\varepsilon) \cdot (p^{Opt}(A_1) + p^{Opt}(R)) + \varepsilon \cdot (|A_1| + |A_2| + 1) \cdot c(Opt) \\
&= (1+\varepsilon) \cdot p^{Opt}(S(Opt)) + \varepsilon \cdot (|S(A)| + 1) \cdot c(Opt)
\end{aligned}$$

This completes our proof of (3.2).

The proof of (3.3) is analogous. We omit the details of this part of the proof and only point out the main steps. As a direct consequence of Lemma 5 we note that

$$p^A(A_1) \geq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot p^{Opt}(A_1) - \frac{\varepsilon}{1+\varepsilon} \cdot |A_1| \cdot c(Opt).$$

For the second part, note, that obviously $c(A_2) \geq c(R)$, since otherwise Opt would not be optimal. For an agent $i \in R$ (which is not selected by A) we have:

$$v_i \geq \theta_i^A \geq \theta_i^{Opt} - \frac{\varepsilon}{1+\varepsilon} \cdot c(Opt(v \mid \neg i))$$

As before we can then conclude:

$$\begin{aligned}
p^A(A_2) &\geq c(A_2) \geq c(R) = \sum_{i \in R} v_i \\
&\geq \sum_{i \in R} \theta_i^{Opt} - \frac{\varepsilon}{1+\varepsilon} \cdot \sum_{i \in R} \underbrace{c(Opt(v \mid \neg i))}_{\leq c(Opt) + \theta_i^{Opt}} \\
&\geq \left(1 - \frac{\varepsilon}{1+\varepsilon}\right) \cdot p^{Opt}(R) - \frac{\varepsilon}{1+\varepsilon} \cdot |R| \cdot c(Opt)
\end{aligned}$$

Putting both results together completes the proof of (3.3). $\square$

Theorem 3 shows that any mechanism that approximates the (optimal) VCG mechanism in terms of cost to society also approximates the payment handed out by VCG. The following corollary uses the results from Theorem 3 to derive something similar to an approximation ratio for these payments. We do not speak of an approximation ratio in the usual sense here, because the approximate payments need not be strictly larger (or smaller) than the VCG payments, but may deviate in both directions on different problem instances.

Corollary 1. *Let mechanisms m_{Opt}, m_A be defined as above and let v be declaration vector of length n. Then:*

$$\left(1 - \frac{\varepsilon}{1+\varepsilon}(n+2)\right) \leq \frac{p^A(S(A(v)))}{p^{Opt}(S(Opt(v)))} \leq (1 + \varepsilon(n+2))$$

Proof: The bounds are direct consequences of (3.2) and (3.3), since it is clearly the case that $c(Opt(v)) \leq p^{Opt}(S(Opt(v)))$. With $|S(Opt)|, |S(A)| \leq n$ the claim follows. $\square$

An interesting observation is the fact that Corollary 1 makes a strong statement about a certain class of approximation algorithms. If A is an FPTAS (*fully polynomial time approximation scheme*) in terms of cost to society, then A is an FPTAS in terms of payment, as well. More formally, if we want a $(1+\delta)$–approximation in terms of payment, i.e.,

$$(1+\delta)^{-1} \leq \frac{p^A(S(A(v)))}{p^{Opt}(S(Opt(v)))} \leq (1+\delta),$$

we use A to compute a $(1+\varepsilon)$–approximation to the optimal cost to society with ε being small enough. To obtain approximation ratio $(1+\delta)$, we need that

$$(1 + \varepsilon(n+2)) \leq (1+\delta)$$

and

$$\underbrace{\left(1 - \frac{\varepsilon}{1+\varepsilon}(n+2)\right)}_{\geq(1-\varepsilon(n+2))} \geq (1+\delta)^{-1}.$$

A simple calculation shows that choosing ε, such that

$$\varepsilon^{-1} \geq (n+2) \cdot (1+\delta) \cdot \delta^{-1},$$

ensures that both requirements are met. Hence, we have that $\varepsilon^{-1} = O(poly(n, \delta^{-1}))$[1] and can compute the desired $(1+\delta)$–approximation in time $O(poly(n, \delta^{-1}))$.

We want to give a short discussion on how (3.2) and (3.3) relate to other bounds that can be found in literature. Talwar [15] analyzes the overpayment of the VCG mechanism on a variety of utilitarian minimization problems. Given an instance of some minimization

[1] By $poly(x_1, \ldots, x_n)$ we refer to any polynomial in $x_1, \ldots, x_n$ with constant degree $k \in \mathbb{N}$.

problem Π, let Opt' denote a disjoint second best solution, i.e., $S(Opt) \cap S(Opt') = \emptyset$. Payment is then bounded by some multiplicity $r_\Pi \cdot c(Opt')$ of the cost of this second best solution, where r_Π is referred to as a problem's *frugality ratio*. It turns out that $r_\Pi = O(n)$ for any utilitarian optimization problem.

For the Set Cover Problem, assuming that each set is of size k, Talwar proves a frugality ratio of k. Calinescu [3] analyzes the payment of truthful approximate mechanisms for the same problem. For a mechanism that is based on a simple greedy approximation algorithm with constant approximation ratio k, payment is bounded by

$$k \cdot c(Opt') + (k-1) \cdot c(Opt) = \Theta(k \cdot (c(Opt) + c(Opt'))).$$

From the results of Talwar we know, that there are problems such that $r_\Pi = n$. Now assume that we have an approximate mechanism with constant approximation ratio c for such a problem. Then (3.2) gives

$$c \cdot (n \cdot c(Opt')) + (c-1) \cdot (n+1) \cdot c(Opt) = \Theta(n \cdot (c(Opt) + c(Opt')))$$

as an upper bound on the payment. We note that (3.2) is in a sense of the same type as the bound given by Calinescu, if we assume the underlying optimization problem to be a worst case as far as its frugality ratio is concerned. Such an example will be presented in the next chapter.

The lower bound (3.3) can simply be viewed as being symmetric to (3.2) and bounding "underpayment". Since, however, from an economical point of view it is usually of more interest to show that a mechanism's payment is not too large (as opposed to showing that payment will not become too small), the quality of this bound is hard to measure due to the lack of previous results. Nevertheless, we note that (3.3) is certainly of interest if we have an FPTAS for the optimization problem, as we have already explained.

Finally, it shall be mentioned that the presented results also apply to the more general scenario of *unknown* single–minded agents [9]. To see this, note, that the notion of truthfulness as defined in Section 2.3 can be extended to capture any larger number of parameters per agent (see, e.g., Section 4.5 or [9, 10]). There is, however, no simple and exhaustive characterization of truthful mechanisms for such agents in the full generality of our model. Nevertheless, it can easily be shown that any optimal algorithm for the abstract type of problems we consider must be monotone and, thus, critical values θ_i^{Opt} exist. Given an approximate truthful mechanism for this more general type of agents we then have to assume monotonicity and the existence of critical values θ_i^A, since these do not follow from truthfulness alone anymore. We note that this assumption is true for all the mechanisms we consider later on.

Chapter 4

Reverse Multi–Unit Auctions

In the previous chapter we have seen an upper bound on the overpayment of approximate mechanisms for utilitarian problems. We will now have a closer look at approximate mechanisms for a fixed utilitarian optimization problem.

The optimization problem we are going to consider is referred to as *reverse multi–unit auctions*. A single buyer needs to purchase a number of identical goods from a set of possible suppliers (or *bidders*). Each bidder i declares the number of goods $q_i \in \mathbb{N}$ she can supply and the price $v_i \geq 0$ she charges for these. The buyer then selects a subset of bidders to supply him with (at least) the required number of goods. We assume that the number of goods owned by each bidder is known to the mechanism. Note, that bidders are then known single–minded according to our definition.

Definition 12. *A* reverse multi–unit auction *among n single–minded bidders is defined by bidders' declarations $(q_1, v_1), \ldots, (q_n, v_n)$, a number of items m and feasible solutions*

$$O(q_1, \ldots, q_n) = \{S \subseteq \{1, \ldots, n\} | \sum_{i \in S} q_i \geq m\}.$$

The output S must be chosen to minimize $c(S) = \sum_{i \in S} v_i$.

Clearly, we could consider the above problem in a purely algorithmic setting as well. If we do so, we are dealing with a problem that can be viewed as a reverse version of the well known *Knapsack Problem*, which in turn corresponds to the more popular scenario of *forward multi–unit auctions* in mechanism design (see Section 5.1). We will present truthful approximate mechanisms for reverse multi–unit auctions. As we have seen in Section 2.3, we need to find approximation algorithms for the optimization problem that are monotone. These are then turned into truthful mechanisms by adding the associated critical value payment scheme. In addition to proving that the presented mechanisms are truthful, we will again consider their payment behavior. Formally, we will show that we can construct problem instances on which the mechanisms' overpayment is asymptotically equal to the upper bound we have seen in the previous chapter. In a sense, this qualifies reverse multi–unit auctions as a near worst–case example among all utilitarian optimization problems in terms of overpayment. By presenting a truthful FPTAS for reverse multi–unit auctions, we will then be able to apply the results from Chapter 3 to obtain an FPTAS for the approximation of VCG payments. As to further motivate this approximation result, we will then show that even the computation of VCG payments for reverse multi–unit auctions turns out to be NP–equivalent. The final section of this chapter shows how some of our mechanisms can also be applied to the more general scenario of (unknown) single–minded bidders.

Notation: In addition to $c(S) := \sum_{i \in S} v_i$ let $q(S) := \sum_{i \in S} q_i$ refer to the total declared number of items for any set S of bidders.

4.1 A greedy approach

We start by considering a greedy algorithm that has a constant approximation ratio of 2. Algorithm RA_G is based on the same idea as the well known 2–approximation algorithm for the Knapsack Problem.

4.1.1 The mechanism

Algorithm 2: RA_G

Input: Number of items m and bids $(q_1, v_1), \ldots, (q_n, v_n)$.

1. Choose π to order bids by increasing *price per item*

$$v_{\pi(1)}/q_{\pi(1)} \leq \cdots \leq v_{\pi(n)}/q_{\pi(n)}$$

and define $G := \emptyset$, $Best := \emptyset$, $num := 0$, $val := 0$, $best := +\infty$.

2. Repeat Steps 3 and 4 for $i = 1, \ldots, n$:

3. If $num + q_{\pi(i)} < m$, then add $\pi(i)$ to G. Formally $G := G \cup \{\pi(i)\}$, $num := num + q_{\pi(i)}$ and $val := val + v_{\pi(i)}$.

4. If $num + q_{\pi(i)} \geq m$, then compare to the best solution found so far. If $val + v_{\pi(i)} < best$, then $Best := G \cup \{\pi(i)\}$ and $best := val + v_{\pi(i)}$.

Output: Subset of bidders $Best$.

The proof of correctness is analogous to the proof given by Kothari et al. [8] for a more general type of bidder.

Theorem 4. *Algorithm RA_G has approximation ratio 2 and running time $O(n \cdot \log n)$, where n is the number of participating bidders.*

Proof: We define a slight modification of RA_G. Precisely, we change Step 4 by adding an early termination condition for the case that we come upon a bidder that cannot be added to G but offers a price that is small enough for our purposes.

1. ...
2. ...
3. ...
4. If $num + q_{\pi(i)} \geq m$, then consider the following cases:
   (a) $v_{\pi(i)} > val$. If $val + v_{\pi(i)} < best$, then $Best := G \cup \{\pi(i)\}$ and $best := val + v_{\pi(i)}$.
   (b) $v_{\pi(i)} \leq val$. If $val + v_{\pi(i)} < best$, then $Best := G \cup \{\pi(i)\}$. Return $Best$ and stop.

It is clear that the above modification can only lead to a worse approximation ratio than before, since a smaller number of feasible solutions is taken into account. Hence, it is sufficient to consider the modified version of RA_G.

We will look at bidders whose declarations are treated in Step 4a. Let S be the solution returned by the modified version of RA_G.

Case 1: Assume that in Step 4a we consider bidder j who is also a part of the optimal solution Opt with value $c(Opt)$, hence $v_j \leq c(Opt)$. Let G' and val' denote the values of G and val at the time when j is considered. Since we store $G' \cup \{j\}$ in $Best$, the algorithm returns a solution S with value at most $val' + v_j$. Furthermore, we know that $val' < v_j$ and conclude

$$c(S) \leq val' + v_j \leq 2 \cdot v_j \leq 2 \cdot c(Opt).$$

Case 2: All bidders treated in Step 4a are not part of the optimal solution Opt. Assume that we modify the algorithm's input by eliminating all bidders that are treated in Step 4a. Obviously, this does not change the optimal solution's value $c(Opt)$. At the same time, the value of the solution presented by RA_G can only increase. This is due to the fact that the possible solutions that were previously considered in Step 4a do not exist anymore. The remaining considered solutions are the same as before. Hence, it is sufficient to look at the solution RA_G computes on the modified input.

Let j be the first bidder that is not treated in Step 3 and val' and $best'$ denote the values of val and $best$ at the time when j is considered. Since bids are treated in order of increasing price per item it must be the case that $val' < c(Opt)$. When considering j, RA_G will terminate by its early termination condition because $best' = +\infty$ and return a solution S with

$$c(S) = val' + v_j \leq 2 \cdot val' < 2 \cdot c(Opt).$$

Hence, $c(S) \leq 2 \cdot c(Opt)$ in both cases. □

It remains to be shown that RA_G is a monotone algorithm.

Theorem 5. *Algorithm RA_G is monotone.*

Proof: Consider bidder j with winning declaration v_j. Assume that j decreases her declaration to $v'_j < v_j$. We need to show that v'_j is also winning. Let $Best$ denote the solution returned by RA_G if j declares v_j and $best$ its value. We will show that, if j declares v'_j, the following two claims hold:

Claim 1 During its computation RA_G finds a feasible solution S that includes j and has cost at most $best$.

Claim 2 RA_G finds no solutions without j that it does not find with j declaring v_j.

It is then clear that v'_j is winning, because RA_G returns the best solution that it finds during its computation. If RA_G, however, finds a better solution than S (as defined in Claim 1), it has to be new. But then it must contain j as well, because of Claim 2.

RA_G processes bids in order of increasing price per item. Let us assume that bidder j's position in the list of bids is p_j when declaring v_j and p'_j when declaring v'_j. We say that bidder j is considered in step p_j, p'_j. Clearly, it must be $p'_j \leq p_j$, since the price per item of bidder j's offer decreases. For any $t \in \{1, \ldots, n\}$, let $G(t)$, $G'(t)$, $num(t)$, $num'(t)$, $val(t)$ and $val'(t)$ denote the content of RA_G's variables after step t with bidder j declaring v_j or v'_j, respectively.

We have already seen that decreasing her bid will let bidder j move up the list of bids to be processed. The main question is how j becomes part of RA_G's solution before and after her declaration changes. On one hand, it might be the case that j is added to G when being processed by the algorithm. The best solution is then completed in some later step. On the other hand, j might not be added to G. Then the best solution is found in the step in which j is being processed. Due to this, we differentiate between the following cases that are depicted in Figure 4.1.

Case 1: *When declaring v_j, bidder j is added to G.*

In this case bidder j is still added to G when declaring v'_j. To be precise, exactly the same bidders as before are added to G. This is formalized in the following claim.

Claim: We have $G(p_j) = G'(p_j)$.

Proof: Since j is added to G originally, we have that $q_j < m - num(p_j - 1)$. Declaring v'_j lets j move to position $p'_j \leq p_j$. RA_G's first $p'_j - 1$ steps are independent of j's declaration, thus

$$num'(p'_j - 1) = num(p'_j - 1) \leq num(p_j - 1).$$

Hence, j is again added to G.

Now consider a bidder k on position $p_k \in \{p'_j, \ldots, p_j - 1\}$. If k is not added to G when j declares v_j, then $q_k > m - num(p_k - 1)$. On the other hand, $p'_k = p_k + 1$ and $num'(p_k) = num(p_k) + q_j$, because $j \in G'(p_k)$. Hence, $q_k > m - num'(p'_k - 1)$ and k will again not be added to G.

If, on the other hand, k is added to G when j declares v_j, then it left enough space for j, formally $num(p_k - 1) + q_k + q_j < m$. With j declaring v'_j, the situation becomes:

$$m - num'(p'_k) = m - (num(p_k - 1) + q_j) > q_k$$

Hence, k is again added to G and the claim follows. □

Originally, the best solution is found in some step $t > p_j$. (Otherwise j could not be part of it.) Since $G(p_j) = G'(p_j)$, this solution is still found when j declares v'_j. This proves Claim 1. It remains to be shown that RA_G does not find any new solutions not including j. In steps $1, \ldots, p'_j - 1$ the considered solutions are the same as before. In all remaining steps, j has already been added to G and, thus, is part of every considered solution.

Case 2: *When declaring v_j, bidder j is not added to G.*

In this situation, we cannot be sure how j is treated after decreasing her bid. We split this case up accordingly.

Case 2.1: *When declaring v'_j, bidder j is not added to G.*

With j declaring v_j, the best solution is found in step p_j. (Note that this is the only considered solution that contains j at all.) As in Case 1 the set of bidders added to G does not change and we have $G(p_j) = G'(p_j)$. Thus, if j declares v'_j, no new solutions are found in steps t with $t < p'_j$ or $t > p_j$. In steps $p'_j + 1 \leq t \leq p_j$, RA_G considers the solutions that were previously considered in steps $p'_j \leq t \leq p_j - 1$. The only changed solution is the one containing j. Since j has moved up the list, it now appears in a solution with value

$$val'(p'_j - 1) + v'_j = val(p'_j - 1) + v'_j \leq val(p_j - 1) + v_j.$$

The inequality is due to $val(p'_j - 1) \leq val(p_j - 1)$ and $v'_j \leq v_j$. Hence, we have found the solution required for Claim 1.

Case 2.2: *When declaring v'_j, bidder j is added to G.*

Let B denote the set of bidders on positions in $\{p'_j, \ldots, p_j - 1\}$ that are added to G when j declares v_j. In this situation the best solution is constructed in step p_j as $G(p'_j - 1) \cup B \cup \{j\}$.

When declaring v_j', j is added to G before bidders from B are processed. Let k be the first bidder in B (according to the list kept by RA_G) that is not added to G now, B' the set of bidders in B that stand before k. (Note, that bidder k must exist. Otherwise, j had been added to G when declaring v_j.) Then, in step p_k', RA_G finds the possible solution $G(p_j'-1) \cup \{j\} \cup B' \cup \{k\}$. Since $B' \cup \{k\} \subseteq B$, this solution must have cost no higher than the previously best solution and, thus, satisfies the requirement of Claim 1. Again, the solutions found in steps $t < p_j'$ are the same as before, solutions found in steps $t \geq p_j'$ necessarily contain j. This gives Claim 2.

We still need to consider RA_G's running time. Finding the ordering π in Step 1 takes time $O(n \cdot \log n)$. After this, the main loop is repeated n times. Steps 3 and 4 of the algorithm can obviously be performed in constant time. This finishes the proof. □

We have seen that algorithm RA_G is monotone and has approximation ratio 2. As shown in Theorem 2, we can then define the truthful approximate mechanism $m_G = (RA_G, p^G)$, where p^G denotes the associated critical value payment scheme, for reverse multi–unit auctions among known single–minded bidders.

4.1.2 Payment Considerations

We will now discuss the payments made by our mechanism. According to the notation used in Section 3.3, RA_G's approximation ratio is characterized by $\varepsilon = 1$. From Corollary 1 we conclude that

$$\frac{p^G(RA_G(q,v))}{p^{Opt}(Opt(q,v))} \leq (n+3)$$

for any bids $(q_1, v_1), \ldots, (q_n, v_n)$. But is this bound tight? Let us consider the following example of bidders' declarations and look at the resulting payments of VCG and our approximate mechanism.

Example 1. *Let $m = n-1$, $\delta > 0$ and consider bids*

$$\underbrace{(1,1), \ldots, (1,1)}_{n-1}, (n-1, n-1+\delta).$$

How does the optimal (VCG) mechanism behave on this input? We note that the optimal solution is obtained by choosing the first $n-1$ identical bids from the list, resulting in total cost $n-1$. Selecting only the last bidder in the list leads to a disjoint second best solution with cost $n-1+\delta$. For any bidder k that is among the first $n-1$ in the list we observe that

$$\begin{aligned} \theta_k^{Opt} &= c(Opt(v \mid \neg k)) - (c(Opt(v \mid k)) - v_k) \\ &= (n-1+\delta) - ((n-1)-1) = 1+\delta \end{aligned}$$

describes her critical value in algorithm Opt (see Observation 2).

For the total payment handed out by the VCG mechanism we conclude:

$$p^{Opt}(Opt(q,v)) = (n-1)\cdot(1+\delta)$$

We now turn to our approximate mechanism m_G. We first note that the list of bids is already ordered by increasing price per item. It is easy to see that RA_G will return the optimal solution on this input. The first $n-2$ bidders will be added to G when they are processed by the algorithm. Bidder $n-1$, who completes the optimal solution, will then be considered in Step 4 of RA_G. But what about its total payment? Again, let k be one of the first $n-1$ bidders in the list. What happens if bidder k increases her bid to some value c? At some point of time, k moves down the list of bids, resulting in the following arrangement:

$$\underbrace{(1,1),\ldots,(1,1)}_{n-2},(n-1,n-1+\delta),(1,c)$$

In this situation RA_G will add the first $n-2$ bidders to its set G and then consider the solutions obtained by adding one of the last two bidders. Obviously, adding the last bidder will result in smaller total cost as long as $c < n-1+\delta$. Hence, we can write bidder k's critical value θ_k^G in algorithm RA_G as

$$\theta_k^G = n-1+\delta.$$

Since the critical values of all $n-1$ selected bidders are obviously identical, the total payment handed out by m_G then is

$$p^G(RA_G(q,v)) = (n-1)\cdot(n-1+\delta).$$

We can now compare the two payments:

$$\begin{aligned}\frac{p^G(RA_G(q,v))}{p^{Opt}(Opt(q,v))} &= \frac{(n-1)\cdot(n-1+\delta)}{(n-1)\cdot(1+\delta)}\\ &= \frac{n-1+\delta}{1+\delta}\rightarrow n-1,\ \text{for } \delta\rightarrow 0\end{aligned}$$

Hence, we see that our bound on the payment approximation is "almost" tight in this example. We especially note that our example shows tightness in an asymptotic sense. We indeed loose a factor of $\Theta(n)$ compared to RA_G's approximation ratio in terms of cost to society. The following theorem states this result.

Theorem 6. *For reverse multi–unit auctions among known single–minded bidders, it exist a truthful approximate mechanism $m=(A,p^A)$, A having constant approximation ratio, and bidders' declarations (q,v), such that*

$$\frac{p^A(A(q,v))}{p^{Opt}(Opt(q,v))} = \Theta(n).$$

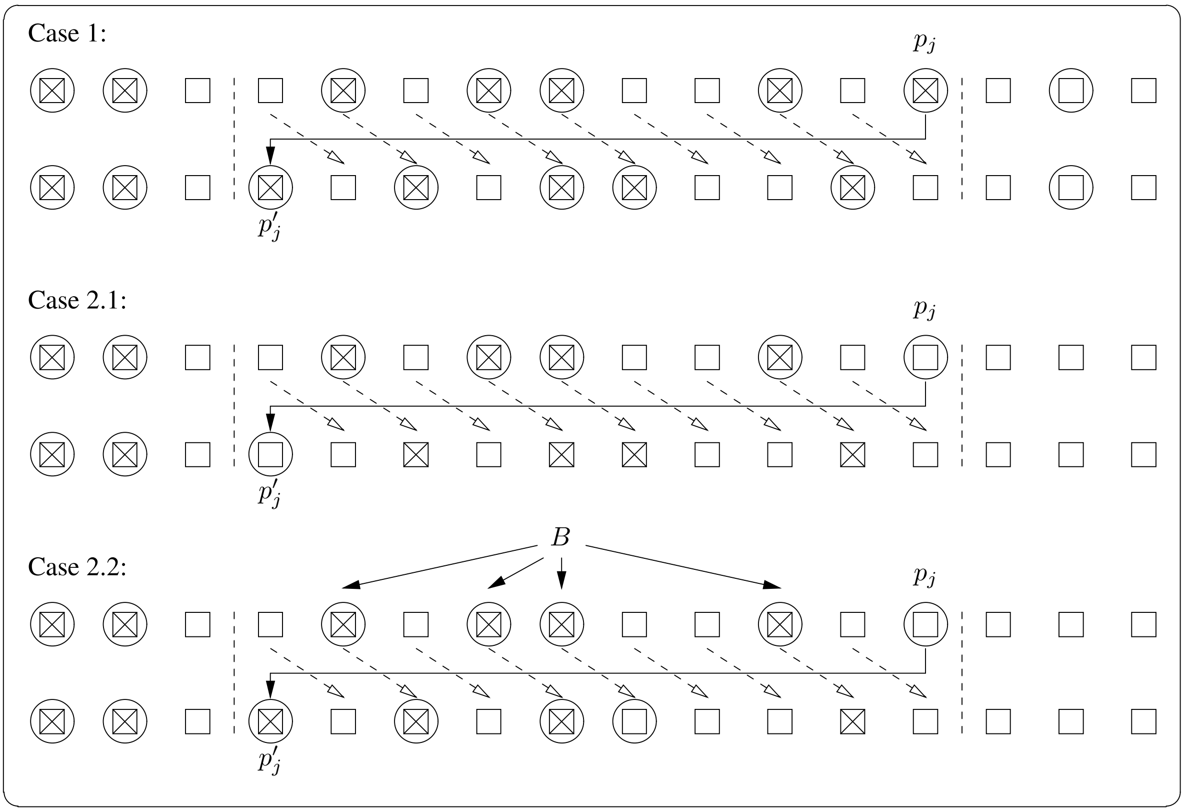

Figure 4.1: Cases from the proof of Theorem 5. Each box represents a bidder in the ordered list of bids kept by RA_G, price per item increasing from left to right. Boxes that are marked with a cross stand for bidders that are added to G when processed by the algorithm. In each case, the first line displays the situation with bidder j declaring v_j. Encircled boxes correspond to the returned solution $Best$. The second line displays the changes appearing when v'_j is declared. Encircled boxes correspond to solution S defined in Claim 1 of the proof. Arrows between the two lines show how bids are moved when bidder j changes her declaration.

4.2 Adding enumeration: a PTAS

In this section we will present a polynomial time approximation scheme (PTAS) for the Reverse Multi–Unit Auction Problem. The algorithm, which was first presented for the Knapsack Problem by Sahni [13] in 1975, is based on the well known technique of *partial enumeration*. A parameter k is introduced to control the approximation ratio. Given $k \in \mathbb{N}$, the algorithm enumerates over all possible sets of at most k bidders. If the chosen subset of bidders cannot supply the required number of items, additional bidders are selected according to the greedy approach of algorithm RA_G from Section 4.1. In contrast to some quite similar techniques used in the design of approximation algorithms (e.g., considering some number of sufficiently large objects separately from the rest) it turns out that partial enumeration is monotonicity preserving and, thus, suitable for truthful mechanism design.

4.2.1 The mechanism

Algorithm 3: RA_{PTAS}

Input: Number of items m, bids $(q_1, v_1), \dots, (q_n, v_n)$ and parameter k.

1. Consider all subsets of bidders with cardinality at most k. Formally, let $Best := \emptyset$, $best := +\infty$ and repeat the following steps for all sets $S \subseteq \{1, \dots, n\}$, $|S| \le k$:
2. Use algorithm RA_G to complete the partial solution. Formally, let
$$q(S) = \sum_{i \in S} q_i \text{ and } c(S) = \sum_{i \in S} v_i.$$
If $m_S < m$, then run RA_G on the input given by number of items $m - q(S)$ and bids $\{(q_i, v_i) | i \notin S\}$ and let $S' := S \cup S_G$, where S_G is the solution computed by RA_G. If $q(S) \ge m$, then $S' := S$.
3. Compare S' to the best solution found so far. If $c(S') < best$, then
$$Best := S' \text{ and } best := c(S').$$

Output: Subset of bidders $Best$.

Theorem 7. *For any given parameter $k \in \mathbb{N}$, algorithm RA_{PTAS} has approximation ratio $1 + \frac{1}{k+1}$ and running time $O(k \cdot n^{k+1})$, where n is the number of participating bidders.*

Proof: We start by proving the approximation ratio. Consider a given set of bids $(q_1, v_1), \dots, (q_n, v_n)$ and optimal solution $Opt \subseteq \{1, \dots, n\}$. If $|Opt| \le k$, then $S = Opt$ in Step 1 at some point of time and RA_{PTAS} returns an optimal solution. Hence, let $|Opt| = l > k$ and $Opt = X \cup Y$, where X contains the k bidders with highest valuations in Opt, thus, $|X| = k$ and $v_i \ge v_j$ for all $i \in X, j \in Opt \backslash X$.

At some point of time, we will have that $S = X$ in Step 1. Since $Y \neq \emptyset$, algorithm RA_G is then applied to the set of remaining bidders $R = \{1, \dots, n\} \backslash X$ to complete the solution. RA_G processes all bidders from R in order of increasing price per item, trying to add them to its set G. Let $j \in Y$ be the first bidder from Y that is not added to G and G' denote set G at the time when j is considered.

Claim: We have $c(G') < c(Y)$.

Proof: Let $G' \cap Y = Y_1$, $Y = Y_1 \,\dot{\cup}\, Y_2$ and $G' = Y_1 \,\dot{\cup}\, T$. It's obviously sufficient to show that $c(T) < c(Y_2)$.

All bidders from T have already been added to G by the time j is considered. Since RA_G orders the bidders by increasing price per item, we have that $v_i/q_i \le v_j/q_j \ \forall\ i \in T$. By assumption, bidder j is the first bidder that is not added to G. Hence, the remaining bidders from Y_2 are processed at a later time and we conclude that $v_i/q_i \ge v_j/q_j \ \forall\ i \in Y_2$.

The number of items to be acquired by RA_G is $m - m_S$. We note that

$$\sum_{i\in T} q_i < m - m_S - \sum_{i\in Y_1} q_i \text{ and } \sum_{i\in Y_2} q_i \ge m - m_S - \sum_{i\in Y_1} q_i.$$

The first inequality obviously holds for T at any point of time, the second inequality is due to the fact that Opt must be a feasible solution. We conclude that

$$\begin{aligned} c(T) &= \sum_{i\in T} q_i \cdot \frac{v_i}{q_i} \le \frac{v_j}{q_j} \cdot \sum_{i\in T} q_i \\ &< \frac{v_j}{q_j} \cdot \sum_{i\in Y_2} q_i \le \sum_{i\in Y_2} q_i \cdot \frac{v_i}{q_i} \\ &= c(Y_2). \end{aligned}$$

This proves the claim. □

RA_G considers $G' \cup \{j\}$ as a possible solution. Thus, it returns a solution with cost at most $c(G') + v_j$. The resulting solution S' found by RA_{PTAS} then has total cost of at most $c(X) + c(G') + v_j$. Since X contains the k bidders with highest valuations from Opt and $j \in Opt \backslash X$, it must be the case that $v_j \le c(Opt)/(k+1)$. It follows that

$$\begin{aligned} c(X) + c(G') + v_j &< c(X) + c(Y) + v_j \\ &\le c(Opt) + c(Opt)/(k+1) \\ &= \left(1 + \frac{1}{k+1}\right) \cdot c(Opt) \end{aligned}$$

is an upper bound on the cost of the solution computed by RA_{PTAS}. This gives the approximation ratio.

Now consider RA_{PTAS}'s running time. The main loop is repeated for

$$\sum_{i=1}^{k} \binom{n}{i} \le \sum_{i=1}^{k} n^i \le k \cdot n^k$$

sets of size at most k. Each run of RA_G can be performed in linear time, if bids are ordered in advance. Thus, the running time is bounded by $O(k \cdot n^{k+1})$. □

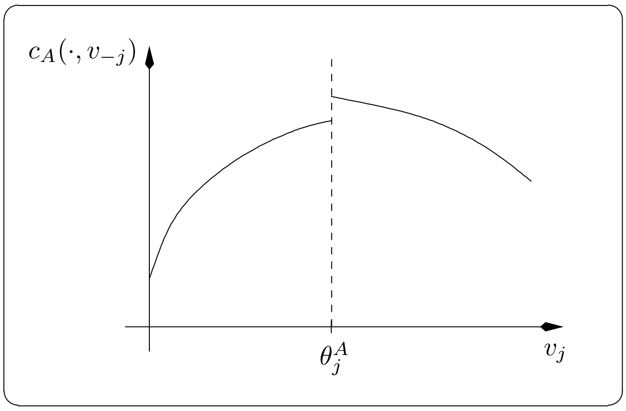

Figure 4.2: Cost to society of a monotone bitonic algorithm.

As in the previous section it now remains to be shown that RA_{PTAS} is monotone. We have seen, however, that proving monotonicity has been a somewhat lengthy and technical task even for the much simpler approximation algorithm RA_G. While there has been no way to avoid going through a direct proof in the former section, we can now make use of RA_{PTAS}'s characteristic structure to obtain a simpler proof of monotonicity. Mu'alem and Nisan [10] have developed a toolkit for the construction of truthful approximate mechanisms for forward multi–unit auctions among known single–minded bidders. Their results, which can be applied to reverse auctions as well, include a nice way of concluding RA_{PTAS}'s monotonicity from its structure and observations we have already made. We start by defining a special class of monotone algorithms, called *bitonic*.

Definition 13. **([10])** *Let A be a monotone approximation algorithm for reverse multi–unit auctions. We say that A is* bitonic *if for every bidder j and declarations v_{-j}, one of the following conditions holds:*

1. *The total cost to society $c_A(v_j, v_{-j})$ of the solution computed by A is a non–decreasing function of v_j for $v_j \leq \theta_j^A$ and a non–increasing function of v_j for $v_j > \theta_j^A$.*
2. *The total cost to society $c_A(v_j, v_{-j})$ of the solution computed by A is a non–decreasing function of v_j for $v_j < \theta_j^A$ and a non–increasing function of v_j for $v_j \geq \theta_j^A$.*

The two (almost identical) conditions in the above definition are required, because from the definition of critical values we do not know whether θ_j^A is the largest value at which j is selected or the smallest value at which this is not the case. As a consequence, it is not clear in which way the intervals that c_A is defined on should be chosen.

The key concept of the toolkit we are going to use is given by so–called *operators*. Operators are used to combine simple monotone approximation algorithms into more complex ones. We will now present the most natural of these operators and discuss how it is related to bitonicity. Given two algorithms A_1 and A_2, we define the MIN–operator as follows.

Why are we interested in bitonic algorithms? Operators that are to be used in the construction of truthful mechanisms must ensure that monotonicity is preserved, since we combine monotone algorithms and expect that the result is monotone again. For the MIN–operator this is not always the case for general monotone algorithms. Theorem 8 shows that bitonicity is exactly the technical qualifier that is needed to overcome this difficulty.

Operator $MIN(A_1, A_2)$

Input: Number of items m and bids $(q_1, v_1), \ldots, (q_n, v_n)$.

1. Run algorithm A_1. Let the returned solution be S_1.
2. Run algorithm A_2. Let the returned solution be S_2.
3. If $c(S_1) \leq c(S_2)$ then $S := S_1$, else $S := S_2$.

Output: Subset of bidders S.

Theorem 8. **([10])** *Let A_1 and A_2 be monotone bitonic algorithms for reverse multi–unit auctions. Then $MIN(A_1, A_2)$ is monotone and bitonic.*

By induction, it is clear that the MIN–operator can be used to combine any finite number of algorithms. It is, however, important that these algorithms are given in some fixed order, since this ensures that Step 3 in the definition above can be viewed as some deterministic tie breaking rule. For a finite ordered set $\mathcal{S}$ of algorithms, let $MIN(\mathcal{S})$ denote the application of the MIN–operator to all algorithms in $\mathcal{S}$ in the given order.

Algorithm 4: RA_S

Input: Number of items m and bids $(q_1, v_1), \ldots, (q_n, v_n)$.

1. Select all bidders from S. Formally, define $R := \{1, \ldots, n\} \backslash S$ and $m_R := m - \sum_{i \in S} q_i$.
2. Use RA_G to complete the solution. If $m_R > 0$, then run RA_G with number of items m_R and bids R, let S_G denote the returned solution and define $S' := S \cup S_G$. Else define $S' := S$.

Output: Subset of bidders S'.

We will now use the presented results for the proof of RA_{PTAS}'s monotonicity. Fix some $S \subseteq \{1, \ldots, n\}$ and consider the simple algorithm RA_S defined above.

Lemma 6. *For any $S \subseteq \{1, \ldots, n\}$ algorithm RA_S is monotone and bitonic.*

Proof: We show monotonicity first. Consider bidder j with winning declaration v_j. If $j \in S$, any declaration is winning and RA_S is monotone with critical value $\theta_j^S = +\infty$ for bidder j.

Now assume that $j \notin S$. Since v_j is nevertheless winning, it must be the case that $m_R > 0$ and j is selected by RA_G. From Theorem 5 we already know that RA_G is monotone. This completes the first part of the proof.

What remains to be shown is bitonicity. Let $c_S(v_j, v_{-j})$ denote the cost to society of the solution computed by RA_S and consider the two cases described above. If $j \in S$, then the returned solution does not depend on v_j. Formally, $\theta_j^S = +\infty$ and

$$c_S(v_j, v_{-j}) = \underbrace{(c(RA_S(q, v)) - v_j)}_{\text{constant}} + v_j$$

is an increasing linear function of v_j on $[0, \infty)$.

If $j \notin S$ we must look at algorithm RA_G. In the proof of Theorem 5 we have already shown that $c_S(v_j, v_{-j})$ is a non–decreasing function of v_j for $v_j \leq \theta_j^S$. More precisely, Claim 1 stated that decreasing a winning declaration v_j leads to non–increasing cost to society.

We now only have to take care of the second (and much easier) part of the proof. Assume that $v_j > \theta_j^S$ is loosing. We need to show that increasing the declaration to $v'_j > v_j$ leads to non–increasing cost to society. If bidder j declares v_j and RA_G finds the best solution before j is processed (in the ordered list of bidders), then increasing v_j cannot lead to increasing cost to society, because the part of the list up to bidder j's original position remains unchanged and, thus, the previously best solution will still be found.

Hence, we can concentrate on the case that, if j declares v_j, the best solution is found after j has been processed. Since j is not selected, this implies that j is not added to G. (Otherwise she were a part of any further solution.) Just as in Case 2.1 in the proof of Theorem 5, we note that j is still not added to G when declaring v'_j. It follows that the set G remains unchanged at all times and, thus, the previously best solution is still found by RA_G. For an illustration see Figure 4.3.

This completes the proof of bitonicity. □

Putting together these results we can now show that RA_{PTAS} is monotone. Obviously, RA_{PTAS} can be viewed as an application of the MIN–operator to a set of algorithms of the same type as RA_S.

Theorem 9. *Algorithm RA_{PTAS} is monotone and bitonic.*

Proof: For any fixed $k \in \mathbb{N}$ let

$$\mathcal{S}_k = \{S | S \subseteq \{1, \ldots, n\}, |S| \leq k\}$$

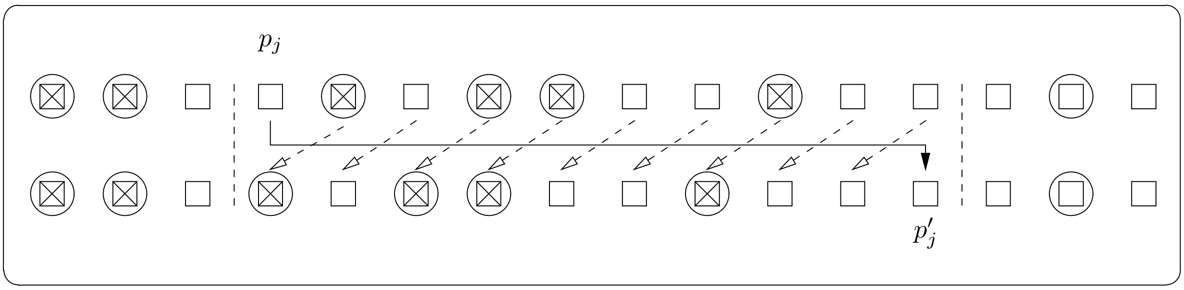

Figure 4.3: Considered case in the proof of bitonicity in Lemma 6. Notation as in Figure 4.1.

be the ordered set of subsets of $\{1, \ldots, n\}$ with cardinality at most k. The ordering can be chosen arbitrarily, it only needs to be fixed for each k, such that a deterministic tie breaking rule is defined for the MIN–operator's application. It is then clear that

$$RA_{PTAS} = MIN(\{A_S | S \in \mathcal{S}_k\}).$$

The claim then follows directly from Lemma 6 and Theorem 8. □

It is clear that we can obtain a truthful approximate mechanism $m_{PTAS} = (RA_{PTAS}, p^{PTAS})$ by adding the associated critical value payment scheme to RA_{PTAS}.

4.2.2 Payment Considerations

Once again we want to have a closer look at the payments handed out by our mechanism. From Theorem 3 we know that

$$\begin{aligned} p^{PTAS}(RA_{PTAS}(q,v)) \;\; &\leq \;\; \left(1 + \frac{1}{k+1}\right) \cdot p^{Opt}(Opt(q,v)) \\ &\qquad + \frac{1}{k+1} \cdot (|RA_{PTAS}(q,v)| + 1) \cdot c(Opt(q,v)) \\ &\leq \;\; \left(1 + \frac{1}{k+1} \cdot (|RA_{PTAS}(q,v)| + 2)\right) \cdot p^{Opt}(Opt(q,v)) \end{aligned}$$

for any bids $(q_1, v_1), \ldots, (q_n, v_n)$. We will show that we can again find problem instances, such that the payment generated by RA_{PTAS} almost equals this bound.

Example 2. *For an arbitrary number of items m with $(k+1) \mid m$ consider bids*

$$(m,m), \underbrace{\left(\frac{m}{k+1}, \frac{m}{k+1}\right), \ldots, \left(\frac{m}{k+1}, \frac{m}{k+1}\right)}_{k+1}, \left(\frac{m}{k+1} - 1, \frac{m}{k+1} - 1 - \delta\right)$$

for some $\delta > 0$.

Let us first consider the (optimal) VCG mechanism. We note that there are two disjoint optimal solutions with cost m. We can either select only the first bidder in the list or her $k+1$ identical successors. In both cases it is easy to see that a total payment of m will result.

But what about our approximate mechanism? Since $k \geq 1$, RA_{PTAS} will at some point of time choose the first bidder in the list as the only element of its set S. It follows that RA_{PTAS} returns the first of the two optimal solutions as its output. The mechanism's total payment is then simply the critical value θ^{PTAS} of the first bidder in the list (which we will refer to as bidder 1).

Before we determine the actual payment, we will have another look at the above bound. Due to $|RA_{PTAS}(q,v)| = 1$ and $p^{Opt}(Opt(q,v)) = m$, we can write that

$$p^{PTAS}(RA_{PTAS}(q,v)) \leq \left(1 + \frac{3}{k+1}\right) \cdot m.$$

Now assume that bidder 1 starts to increase her bid to some (m,c), $c > m$. RA_{PTAS} iterates over all sets of size at most k and uses RA_G in each iteration to complete the solution. Bidder 1 will drop from the returned solution, when for some set S, $1 \notin S$, a solution with cost smaller than c is constructed by RA_G.

We refer to the last bidder in the above list of bids as bidder $(k+3)$. This bidder plays an important role in determining the total payment.

Observation 3. *Consider any iteration of algorithm RA_{PTAS} for some set S. If $1 \notin S$, it follows that $(k+3) \in S$ or $(k+3)$ is selected by RA_G when completing the solution.*

To see this assume that $1 \notin S$. From $|S| \leq k$ it follows that $\sum_{i \in S} q_i \leq (1 - \frac{1}{k+1}) \cdot m$. Bidder $(k+3)$ offers the lowest price per item of all participating bidders and, thus, is the first bidder to be processed by RA_G. Since space of $\frac{m}{k+1}$ is still to be filled, bidder $(k+3)$ is added to G and necessarily becomes a part of the solution.

The only feasible solutions including bidder $(k+3)$ are obtained by adding either bidder 1 or the subsequent $k+1$ identical bidders. Each of these solutions has cost $\left(1 + \frac{1}{k+1}\right) \cdot m - 1$. From the argumentation above we conclude that

$$p^{PTAS}(RA_{PTAS}(q,v)) = \theta^{PTAS} = \left(1 + \frac{1}{k+1}\right) \cdot m - 1.$$

This shows that the given bids are indeed a near worst case example in terms of overpayment. The following theorem formulates this result.

Theorem 10. *For reverse multi–unit auctions among known single–minded bidders and for any $\varepsilon > 0$, it exist a truthful approximate mechanism $m_A = (A, p^A)$, A having approximation ratio $(1+\varepsilon)$, and bidders' declarations (q,v), such that $|S(A(q,v))| = 1$ and*

$$\frac{p^A(A(q,v))}{p^{Opt}(Opt(q,v))} = (1+\varepsilon) - \frac{1}{m},$$

where m is the total number of items to be acquired.

4.3 More enumeration: an FPTAS

Finally, we will now present a fully polynomial time approximation scheme (FPTAS) for the Reverse Multi–Unit Auction Problem. We start by transferring the well known FPTAS for the Knapsack Problem to this scenario. For the Knapsack Problem an FPTAS can be constructed by scaling the profits presented in the input in an appropriate way and then using dynamic programming to find an exact solution on the modified instance. A few changes are sufficient to obtain a corresponding approximation scheme for reverse auctions.

Algorithm RA_{dyn} is an exact algorithm for reverse auctions that uses the technique of dynamic programming and has pseudo–polynomial running time.

Algorithm 5: RA_{dyn}

Input: Number of items m and bids $(q_1, v_1), \ldots, (q_n, v_n)$.

1. For $i \in \{1, \ldots, n\}$, $V := \max_i v_i$ and $v \in \{0, \ldots, nV\}$ let
$$A(i, v) := \max\{q(S) | S \subseteq \{1, \ldots, i\} \wedge c(S) = v\}$$
denote the largest number of items that has a declared cost of exactly v, if only the first i bidders are considered. Define $A(i, v) = 0$ if v cannot be reached. Especially, let $A(1, v_1) = q_1$, $A(1, v) = 0$ for all $v \neq v_1$ and $A(i, v) = 0$ for all $v < 0$.
2. For $i = 2, \ldots, n$ and $v = 0, \ldots, nV$ compute
$$A(i, v) := \max\{A(i-1, v), A(i-1, v-v_i) + q_i\}.$$
3. Let S be the set of bidders that results in $\min\{v | A(n, v) \geq m\}$.

Output: Subset of bidders S.

Correctness of RA_{dyn} can easily be shown by induction. We are interested in the subset of bidders with the smallest sum of declarations that supplies us with at least m items. This is just the set S as described in the algorithm's last step. Note, that this set can be easily found if some additional information is stored in the table used for dynamic programming. Correctness of the equality used in Step 2 is intuitively clear. Considering bidder i, the total value v can be reached either selecting or not selecting bidder i, there are obviously no further choices. To see that running time is pseudo–polynomial, note, that the algorithm is calculating the entries of a $(n \times nV)$–matrix and that each entry can be found in constant time. A formal proof for the following lemma is omitted.

Lemma 7. *Algorithm RA_{dyn} returns an exact solution to the Reverse Multi–Unit Auction Problem in time $O(n^2V)$, where V denotes the value of the largest declaration.*

This dynamic programming approach can now be used in the construction of the desired FPTAS. As was already mentioned, the key idea lies in scaling the given declarations before applying RA_{dyn}. Obviously, if declarations are scaled to a range that is polynomially bounded in n and ε^{-1}, applying RA_{dyn} will result in running time $poly(n, \varepsilon^{-1})$. Hence, we only need to find a range to scale to that will guarantee the desired approximation ratio.

In contrast to the Knapsack Problem, scaling becomes a little bit more difficult for reverse auctions. As will be motivated in the proof of correctness, we first need to find a "good" lower bound on the optimal solution's cost $c(Opt)$ that we can then use in the scaling process. We can use the greedy algorithm RA_G from Section 4.1 to find this bound. If RA_G returns a solution with cost $c(A_G)$, then we simply take $c^* = c(A_G)/2$ as a lower bound on $c(Opt)$. It immediately follows by Theorem 4 that

$$\frac{1}{2} \cdot c(Opt) \leq c^* \leq c(Opt).$$

Depending on c^* it might then be necessary to remove extremely large declarations from the input to ensure polynomial running time. The remaining computation can be done in analogy to the Knapsack case.

Algorithm 6: RA_{FPTAS}

Input: Number of items m, bids $(q_1, v_1), \ldots, (q_n, v_n)$ and $\varepsilon > 0$.

1. Use algorithm RA_G to find a lower bound c^* on the optimal solution's cost $c(Opt)$, such that $c(Opt)/2 \leq c^* \leq c(Opt)$. Then remove all declarations larger than $2 \cdot c^*$ from the input. For the ease of notation, assume that no such declarations exist.

2. Let
$$\alpha := \frac{n}{\varepsilon \cdot c^*}$$
and scale the remaining declarations to range $\{1, \ldots, \lceil \frac{2n}{\varepsilon} \rceil\}$:
$$v_i' := \lceil \alpha \cdot v_i \rceil \text{ , } i = 1, \ldots, n$$

3. Run algorithm RA_{dyn} on declarations
$$(q_1, v_1'), \ldots, (q_n, v_n')$$
and let S be the returned subset of bidders.

Output: Subset of bidders S.

The following proof of correctness is done analogously to the proof given for the Knapsack case, e.g., in [16]. It is included here to show that the lower bound computed in Step 1 of the algorithm is indeed what has to be used for the scaling process.

Theorem 11. *For any $\varepsilon > 0$, algorithm RA_{FPTAS} has approximation ratio $1+\varepsilon$ and running time $O(n^3\varepsilon^{-1})$, where n is the number of participating bidders.*

Proof: We first argue about running time. After removing large declarations and scaling the remaining ones, the largest declaration is bounded above by $\lceil \frac{2n}{\varepsilon} \rceil$. From Lemma 7 we conclude that RA_{dyn}'s running time on the scaled declarations is bounded by $O(n^3\varepsilon^{-1})$. Calling RA_G in Step 1 results in running time $O(n \cdot \log n)$, Step 2 can obviously be performed in linear time. The claim follows.

We now show the approximation ratio. Let S denote the solution returned by RA_{FPTAS}, Opt an optimal solution. We first note that Opt is not changed in step 1. If bidder i is removed from the input, then

$$v_i > 2 \cdot c^* \geq c(Opt)$$

and i is obviously not part of Opt. Hence, w.l.o.g., we assume that no bidder is removed. We now scale declarations back to their original range. Formally:

$$v_i'' = \frac{v_i}{\alpha}, i = 1, \ldots, n$$

Due to the fact that declarations were rounded in the algorithm's scaling step, declarations are not scaled back to their exact former values. We note that, for all $i \in \{1, \ldots, n\}$,

$$v_i'' - v_i = \frac{\lceil \alpha \cdot v_i \rceil}{\alpha} - v_i \leq \frac{\alpha \cdot v_i + 1}{\alpha} - v_i = \frac{1}{\alpha}.$$

By $c(S)$ and $c''(S)$ we denote the cost of solution S when assuming declarations v_i or v_i'', respectively. We observe that

$$\begin{aligned} c''(Opt) - c(Opt) &\leq |Opt| \cdot \frac{1}{\alpha} \leq \frac{n}{\alpha} \\ &= \varepsilon \cdot c^* \leq \varepsilon \cdot c(Opt). \end{aligned}$$

We know that RA_{dyn} computes a solution S that is optimal with respect to declarations v_i'. It follows that S is also optimal with respect to declarations v_i''. We conclude:

$$c(S) \leq c''(S) \leq c''(Opt) \leq (1 + \varepsilon) \cdot c(Opt)$$

The last inequality is due to the above observation. This completes the proof. □

This finishes the first part of this section. As promised, we have adopted the well known Knapsack FPTAS for our reverse auction scenario. It turns out, however, that this approach cannot directly be used for the design of a truthful approximate mechanism, because

RA_{FPTAS} is not a monotone algorithm. We want to have a closer look at why this is the case. Clearly, RA_{dyn} is monotone, since it computes exact solutions. Hence, monotonicity is lost because of our scaling procedure.

Consider some bidder i and assume that i is selected by RA_G and by RA_{FPTAS}. Then decreasing declaration v_i will obviously have an influence on the solution computed by RA_G and, thus, will cause $c(A_G)$ and c^* to decrease. What can we then say about the scaled declarations v'_j? We know that all declarations will slightly increase, because we are dividing by a smaller value c^*. At the same time, we cannot control how much each v_j will increase, because declarations are rounded after scaling. Now assume that RA_{FPTAS} selects i as part of a solution, where decreasing v_i causes every scaled and rounded declaration v'_j of the other selected agents to increase by 1. If there exists a second best disjoint solution, where rounded declarations do not increase, then clearly this solution might now become optimal. This would cause i not to be selected anymore and, thus, contradict RA_{FPTAS}'s monotonicity.

Algorithm 7: RA_k^{dyn}

Input: Number of items m, bids $(q_1, v_1), \ldots, (q_n, v_n)$ and $\varepsilon > 0$.

1. Set $v'_i := \min\{v_i, 2^{k+2}\}$ for all $i \in \{1, \ldots, n\}$.
2. Let
$$\alpha_k := \frac{n}{\varepsilon \cdot 2^k}$$
and scale the modified declarations:
$$v''_i := \lceil \alpha_k \cdot v'_i \rceil, \, i = 1, \ldots, n$$
3. Run algorithm RA_{dyn} on declarations
$$(q_1, v''_1), \ldots, (q_n, v''_n)$$
and let S be the returned subset of bidders.

Output: Subset of bidders S.

We note that the main problem with monotonicity is caused by the fact that declarations have a direct influence on the algorithm's scaling procedure. If we could avoid this, we might be able to reestablish monotonicity. On the other hand, it is clear that scaling independently of actual declarations will not allow us to generally guarantee a good approximation ratio. In the next step, we will consider a set of algorithms RA_k^{dyn} as defined above which scale declarations independently of their actual values, thereby ensuring monotonicity. They also guarantee a good approximation ratio if we assume that the optimal solution's cost falls into a fixed range depending on k. Once more applying the toolkit presented by Mu'alem and Nisan [10], these algorithms can then be used as building blocks for a monotone FPTAS.

Lemma 8. *Let $0 < \varepsilon < 1$ and $c(Opt(q, v))$ denote the value of an optimal solution for a given set of declarations (q, v). Algorithm RA_k^{dyn}'s running time is bounded by $O(n^3\varepsilon^{-1})$. If $2^k \leq c(Opt(q, v)) < 2^{k+1}$, then RA_k^{dyn} computes a $(1 + \varepsilon)$–approximation to $c(Opt)$ on declarations (q, v).*

Proof: After Step 1 no declaration with value higher than 2^{k+2} exists. Hence, in Step 2 all declarations are scaled to range $\{1, \ldots, \lceil\frac{4n}{\varepsilon}\rceil\}$. The claimed running time is then a direct consequence of Lemma 7.

The approximation ratio can be shown as for algorithm RA_{FPTAS} in the proof of Theorem 11. Let again S denote the solution returned by RA_k^{dyn}, Opt an optimal solution. As before, Step 1 does not change the value of Opt, since, if declaration v_i is changed, we have that

$$v_i > 2^{k+2} > 2 \cdot c(Opt)$$

and, thus, i is obviously not part of Opt. We note that, as a consequence of $\varepsilon < 1$, also RA_k^{dyn} will not select any bidder i whose declaration is changed in Step 1. Hence, we have $v'_i = v_i$ for every bidder i in Opt and S. Now the proof of Theorem 11 can be applied. We again define

$$v'''_i = \frac{v''_i}{\alpha_k}, i = 1, \ldots, n$$

and note that

$$v'''_i - v_i = \frac{\lceil\alpha_k \cdot v'_i\rceil}{\alpha_k} - v_i = \frac{\lceil\alpha_k \cdot v_i\rceil}{\alpha_k} - v_i \leq \frac{\alpha_k \cdot v_i + 1}{\alpha_k} - v_i = \frac{1}{\alpha_k}$$

as mentioned above for any bidder i in Opt. Using the same notation as before, it follows that

$$\begin{aligned} c'''(Opt) - c(Opt) &\leq |Opt| \cdot \frac{1}{\alpha_k} \leq \frac{n}{\alpha_k} \\ &= \varepsilon \cdot 2^k \leq \varepsilon \cdot c(Opt). \end{aligned}$$

The solution computed by RA_{dyn} is optimal with respect to declarations v''_i and v'''_i. We conclude

$$c(S) = c'(S) \leq c'''(S) \leq c'''(Opt) \leq (1 + \varepsilon) \cdot c(Opt),$$

as claimed. □

It remains to be shown that RA_k^{dyn} is monotone. In addition to showing this, we will once more also prove bitonicity, which will again be required for the application of a generalized version of the MIN–operator presented in the previous section.

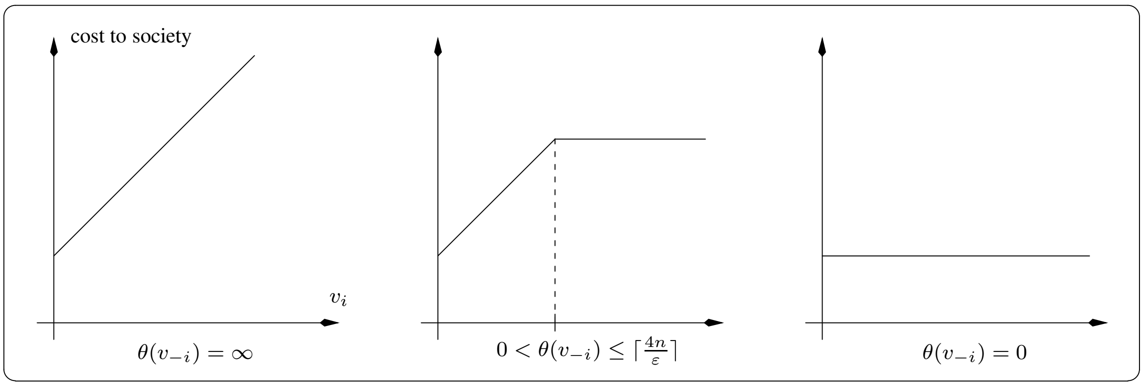

Figure 4.4: Cost to society of algorithm RA_k^{dyn} as a function of declaration v_i.

Lemma 9. *For any fixed $k \in \mathbb{N}_0$, algorithm RA_k^{dyn} is monotone and bitonic.*

Proof: Given a declaration v_i, let $f(v_i)$ denote the value of v_i' after Step 1 of the algorithm, $g(f(v_i))$ the value of v_i'' after Step 2. We observe that both $f(\cdot)$ and $g(\cdot)$ are non–decreasing functions on $\mathbb{R}$. For any declarations v_i, v_i' we note:

$$\begin{aligned} v_i' \leq v_i &\Rightarrow f(v_i') \leq f(v_i) \\ f(v_i') \leq f(v_i) &\Rightarrow g(f(v_i')) \leq g(f(v_i)) \end{aligned}$$

It was already mentioned that obviously RA_{dyn} is monotone, since it computes exact solutions. Let $\theta(v_{-i})$ denote bidder i's critical value with respect to her scaled declaration $g(f(v_i))$. Thus, for a fixed set of declarations v_{-i}, bidder i is selected whenever declaring v_i, such that $g(f(v_i)) < \theta(v_{-i})$.

Now assume that v_i is a winning declaration for bidder i and let $v_i' < v_i$. We immediately conclude that

$$g(f(v_i')) \leq g(f(v_i)) < \theta(v_{-i})$$

and, thus, v_i' is a winning declaration. This completes the proof of monotonicity.

Bitonicity can also be derived from the monotonicity of $f(\cdot)$ and $g(\cdot)$ and the fact, that RA_{dyn} finds exact solutions. Let again v_i be winning and $v_i' \leq v_i$. Now look at the resulting cost to society. For bidder i's scaled declaration we have that $g(f(v_i')) \leq g(f(v_i))$, while the scaled declarations of all other bidders remain unchanged. As a consequence, RA_{dyn} will compute the same solution as before and cost to society can only decrease.

If, on the other hand, v_i is a loosing declaration, then increasing v_i will also increase i's scaled declaration and RA_{dyn} will still compute the same solution not containing i. As a consequence the cost to society remains constant. $\square$

Since k is a fixed parameter for every algorithm RA_k^{dyn}, it is clear that the gap between 2^{k+2} and bidders' actual declarations can become arbitrarily large. Since nevertheless some solution is computed, it can obviously happen that a certain bidder is selected although her declaration is larger than 2^{k+2}. Such a bidder is then selected for all larger declarations, as well, because the algorithm treats all declarations larger than 2^{k+2} as having the same value. Just in the same way, it might be the case that some bidder is never selected, no matter how small her declaration becomes. The possible cases are depicted in Figure 4.4. We note that, of course, in every case the requirement of bitonicity is fulfilled.

We will now proceed with the construction of our monotone FPTAS. We will again make use of the MIN–operator defined in the previous section to combine algorithms RA_k^{dyn} into the desired algorithm. We define the ordered set

$$\mathcal{A} := \{RA_k^{dyn} | 0 \leq k < \infty\},$$

where a natural ordering is obviously given by index k. By $MIN(\mathcal{A})$ we refer to the application of the MIN–operator to $\mathcal{A}$, i.e.,

$$MIN(\mathcal{A})(q, v, \varepsilon) := \text{argmin}\{c(RA_k^{dyn}(q, v, \varepsilon)) | RA_k^{dyn} \in \mathcal{A}\},$$

where $RA_k^{dyn}(q, v, \varepsilon)$ denotes the solution computed by RA_k^{dyn} on declarations (q, v) for any parameter $0 < \varepsilon < 1$ and $c(RA_k^{dyn}(q, v, \varepsilon))$ the corresponding cost. We assume that ties are broken based on the ordering described above, i.e., if algorithms RA_j^{dyn} and RA_k^{dyn} with $j < k$ produce solutions with identical cost, then RA_j^{dyn} will be given preference. Note, that $MIN(\mathcal{A})$ is well–defined, since there is only a finite number of feasible solutions for any given set of declarations. For the moment, we will ignore the question of $MIN(\mathcal{A})$'s running time. Obviously, running an infinite number of algorithms to find a minimum will result in unacceptable running time. It is, however, clear that we in fact do not have to run every RA_k^{dyn}, since the number of different feasible solutions is limited. We will return to this problem later. We start by showing that $MIN(\mathcal{A})$ defines a monotone algorithm and ensures the desired approximation ratio.

Lemma 10. *For any $0 < \varepsilon < 1$, algorithm $MIN(\mathcal{A})$ is monotone and has approximation ratio $1 + \varepsilon$.*

Proof: We start by showing monotonicity. Assume that $MIN(\mathcal{A})$ is not monotone. Then there exist a bidder i and declarations $v_i^1 < v_i^2$, such that v_i^1 is loosing and v_i^2 winning declaration. Let $RA_k^{dyn}(v_i) = RA_k^{dyn}(q, (v_i, v_{-i}), \varepsilon)$ denote the solution returned by RA_k^{dyn} as a function of v_i.

With bidder i declaring v_i^1 let RA_j^{dyn} be the algorithm that computes the solution with minimum cost $c(RA_j^{dyn}(v_i^1))$, formally

$$c(RA_j^{dyn}(v_i^1)) \leq c(RA_k^{dyn}(v_i^1)) \;\; \forall \;\; 0 \leq k < \infty.$$

From the bitonicity of RA_j^{dyn} it follows that

$$c(RA_j^{dyn}(v_i^2)) \leq c(RA_j^{dyn}(v_i^1)).$$

With bidder i declaring v_i^2 let RA_l^{dyn} be the algorithm resulting in a solution with minimal cost, thus,

$$c(RA_l^{dyn}(v_i^2)) \leq c(RA_k^{dyn}(v_i^2)) \ \forall \ 0 \leq k < \infty.$$

From the bitonicity of RA_l^{dyn} we conclude that

$$c(RA_l^{dyn}(v_i^1)) \leq c(RA_l^{dyn}(v_i^2)).$$

Putting these together closes the cycle and shows that

$$c(RA_j^{dyn}(v_i^1)) = c(RA_j^{dyn}(v_i^2)) = c(RA_l^{dyn}(v_i^1)) = c(RA_l^{dyn}(v_i^2)).$$

If $j = l$, then RA_j^{dyn} cannot be a monotone algorithm. Hence, we can assume that $j \neq l$. We now look at our tie breaking rule. From the fact that RA_j^{dyn} is given preference when bidder i declares v_i^1, it follows that $j < l$. Analogously, the situation when declaring v_i^2 implies $l < j$, a contradiction. We conclude that $MIN(\mathcal{A})$ is monotone.

The approximation ratio remains to be shown. Consider declarations (q, v) and let $c(Opt)$ be the cost of an optimal solution for these declarations. Define k^*, such that

$$2^{k^*} \leq c(Opt) < 2^{k^*+1}.$$

The claimed approximation ratio is a direct consequence of $RA_{k^*}^{dyn} \in \mathcal{A}$ and Lemma 8. □

Algorithm 8: RA^*_{FPTAS}

Input: Number of items m, bids $(q_1, v_1), \ldots, (q_n, v_n)$ and $\varepsilon > 0$.

1. Let $V := \max_i v_i$, $c_{min} := +\infty$ and $S := \emptyset$. Then repeat the next step for all $k = 0, \ldots, \lceil \log \left(\frac{n}{\varepsilon} V\right) \rceil$:
2. Run RA_k^{dyn} on the given declarations and let S_k be the returned solution. If $c(S_k) < c_{min}$, then set $S := S_k$ and $c_{min} := c(S_k)$.

Output: Subset of bidders S.

The last step in the construction of our FPTAS is now to find an implementation of $MIN(\mathcal{A})$ that guarantees polynomial running time. This implementation will be based on the following idea. Each algorithm RA_k^{dyn} starts by scaling the declarations. This is basically done by multiplying with some constant and then dividing by 2^k. As k gets larger, the range that declarations are scaled to clearly becomes smaller and finally will be $\{1\}$. Any further increase of k will not have an effect on the algorithms outcome, since RA_{dyn} is always run on identical input. This idea immediately leads to algorithm RA^*_{FPTAS} as defined above.

Theorem 12. *RA^*_{FPTAS} is a monotone algorithm for reverse multi–unit auctions among known single minded bidders. For any $0 < \varepsilon < 1$, RA^*_{FPTAS} has approximation ratio $1 + \varepsilon$ and running time $O(n^3\varepsilon^{-1}\log(n\varepsilon^{-1}V))$, where n is the number of participating bidders and V denotes the value of the largest declaration.*

Proof: Assume declarations (v, q) and let $V = \max_i v_i$. Now define $k^* = \lceil\log\left(\frac{n}{\varepsilon}V\right)\rceil$, fix some $k \geq k^*$ and consider algorithm RA_k^{dyn}. In Step 2 of RA_k^{dyn} declarations are scaled. For any declaration v_i we observe that

$$v_i'' \leq V'' = \lceil \underbrace{\alpha_k \cdot V}_{\leq \alpha_{k^*} V \leq V^{-1} V} \rceil \leq 1.$$

It follows that $RA_k^{dyn} = RA_{k^*}^{dyn}$ for every $k \geq k^*$ and, hence,

$$RA^*_{FPTAS} = MIN(\mathcal{A}).$$

We can then conclude monotonicity and the claimed approximation ratio directly from Lemma 10.

The running time is a straightforward observation. The main loop is repeated $\lceil\log\left(\frac{n}{\varepsilon}V\right)\rceil$ times. Each call to some RA_k^{dyn} needs time $O(n^3\varepsilon^{-1})$ as shown in Lemma 8. □

We see that RA^*_{FPTAS} is indeed a monotone FPTAS for the Reverse Auction Problem. We again point out the fact that with this algorithm, we have also found an FPTAS for the approximation of VCG payments for reverse auctions as shown in Corollary 1 in Section 3.3. We note that this implies a sort of trade–off as the choice of an optimization goal is concerned. If we want a truthful mechanism that gives a good approximation to the optimal cost to society, then this mechanism must also approximate VCG payment. If we are, however, interested in a truthful mechanism that improves on VCG payments, we cannot expect such a mechanism to approximate optimal cost to society really well.

4.4 Hardness of payment computation

We have seen that the FPTAS developed in the previous section can also be used as an FPTAS for the approximation of VCG payments in reverse auctions. As to further motivate our interest in such an approximation, we will now show that, apart from the design of truthful mechanisms with payment close to VCG, even the computation of VCG payments is already a hard problem.

Definition 14. *Consider languages L_1 and L_2 over alphabet Σ. We say that L_1 is* polynomial time reducible *to L_2, denoted as $L_1 \leq_p L_2$, if there exists a transformation f that can be computed by a deterministic Turing machine in polynomial time, such that, for all $w \in \Sigma^*$,*

$$w \in L_1 \Leftrightarrow f(w) \in L_2.$$

Now consider a search problem Π and a language L. We say that Π is Turing reducible *to L, denoted as $\Pi \leq_T L$, if there exists a deterministic Turing machine that solves Π in polynomial time and uses an oracle for L. Analogously we define $L \leq_T \Pi$. If $L \leq_T \Pi$ for some NP–complete language L, then Π is said to be* NP–hard. *If $\Pi \leq_T L$ for some $L \in NP$, then Π is* NP–easy. *Search problems that are NP–hard and NP–easy are referred to as* NP–equivalent.

Due to the fact that we are not primarily concerned with questions from complexity theory, the above definition's way of treating search problems is somewhat sloppy. Formally, a Turing machine does not "solve" a search problem Π. Rather, it computes a function that realizes some relation $R_\Pi \subseteq (\Sigma^*)^2$, which in turn corresponds to Π itself. We omit the precise definitions needed in this context.

Remember, that the Reverse Multi–Unit Auction Problem is defined by a set of bidders' declarations $(q_1, v_1), \ldots, (q_n, v_n)$ and a number m of items to be acquired. Then

$$L_{RA} := \{((q_1, v_1), \ldots, (q_n, v_n), m, k) \mid \exists S \subseteq \{1, \ldots, n\} : \sum_{i \in S} q_i \geq m \wedge \sum_{i \in S} v_i \leq k\}$$

defines the language that describes the corresponding decision problem. Given a set of declarations $(q_1, v_1), \ldots, (q_n, v_n)$, we need to decide whether there is a subset of bidders that offer at least m items at a total price of at most k. We note that this definition looks very similar to the decision variant L_{KP} of the Knapsack Problem. It is easy to see that $L_{KP} \leq_p L_{RA}$. For the Knapsack Problem, we need to decide if there is a subset of goods with sufficient profit that will fit into the knapsack. Obviously, we can just as well ask whether there is a subset of items with sufficient size and small enough profit to leave out of the knapsack. Formally, we consider the polynomial time computable transformation f with

$$f((q_1, v_1), \ldots, (q_n, v_n), m, k) := ((q_1, v_1), \ldots, (q_n, v_n), \sum_{i=1}^{n} q_i - m, \sum_{i=1}^{n} v_i - k)$$

for instances $((q_1, v_1), \ldots, (q_n, v_n), m, k)$ of the Knapsack Problem. This immediately gives the following lemma.

Lemma 11. *L_{RA} is NP–complete.*

By $VCG(RA)$ we denote the problem of computing the total payment handed out by the normalized VCG mechanism on an instance of the Reverse Multi–Unit Auction Problem RA. If we can show that this problem is hard, this certainly implies that this is also true for the computation of an agent's individual payment.

Theorem 13. *$VCG(RA)$ is NP–equivalent.*

Proof: We start by showing that $L_{RA} \leq_T VCG(RA)$. Consider an instance

$$I = ((q_1, v_1), \ldots, (q_n, v_n), m, k)$$

of L_{RA}, let $v_{min} = \min_i v_i$ and consider a modified instance

$$I' = ((q_1, v_1), \ldots, (q_n, v_n), (q_{n+1}, v_{n+1}), m, k),$$

where $q_{n+1} = m$ and $v_{n+1} = v_{min} - \varepsilon$ for some small $\varepsilon > 0$. Run an algorithm for $VCG(RA)$ on this input. We know that in our modified instance the only optimal solution to the Reverse Multi–Unit Auction Problem is selecting only the newly added bidder $n + 1$. It then follows by Lemma 4 and Observation 2 that the total VCG payment on this instance corresponds to the critical value of bidder $n + 1$ and can be written as

$$\begin{aligned} p^{Opt}(Opt(I')) &= \theta_{n+1}^{Opt} \\ &= c(Opt(I' \mid \neg(n+1))) - \underbrace{(c(Opt(I' \mid (n+1))) - v_{n+1})}_{=0} \\ &= c(Opt(I' \mid \neg(n+1))) = c(Opt(I)). \end{aligned}$$

This is exactly the value of an optimal solution to the Reverse Multi–Unit Auction Problem on the original instance I. By comparing it to k it can immediately be decided whether $I \in L_{RA}$. It is clear that all modifications can be done in polynomial time. It follows that $VCG(RA)$ is NP–hard.

We proceed by showing that $VCG(RA) \leq_T L_{RA}$. We note that, if we have an algorithm A that decides L_{RA} in polynomial time, then we can find the cost of an optimal solution on our instance in polynomial time, as well. Given declarations $(q_1, v_1), \ldots, (q_n, v_n)$ and a number of items m, we know that the optimal solution's cost falls into the interval $[0, \sum_{i=1}^{n} v_i]$. Hence, applying binary search, $\log (\sum_{i=1}^{n} v_i)$ calls to algorithm A are sufficient to find the desired value. We can use this idea to compute the VCG payment for each bidder i explicitly by removing i's bid from the list and then computing the value of two optimal solutions supplying m or $m - q_i$ items, respectively. We conclude that $VCG(RA)$ is NP–easy. $\square$

This finishes our discussion of the complexity of computing VCG payments in reverse multi–unit auctions. We especially note that the computation turns out to be NP–hard. It then follows that the FPTAS as presented in Section 4.3 is the best possible type of polynomial time approximation algorithm for the problem.

4.5 Unknown Single–Minded Bidders

The last section of this chapter is devoted to the slightly more general model of *unknown single–minded bidders*, as opposed to the case of known single–minded bidders that we considered so far. Remember, that in our reverse multi–unit auction scenario, bidder i is considered known if her offered number of items q_i is known to the mechanism. In this case, the only private information of any bidder is her true cost of supplying these items. In Section 2.3 truthful mechanisms for known single–minded bidders were characterized.

We now cite a corresponding characterization of truthful mechanisms for reverse multi–unit auctions among unknown single–minded bidders which is due to Lehmann et al. [9] and Mu'alem and Nisan [10]. Essentially, we need to adapt the definition of monotonicity to meet the requirements of this more general setting.

Definition 15. *Let A be an algorithm for reverse multi–unit auctions among single–minded bidders. A is said to be* monotone*, if for any bidder i and fixed declarations (q_{-i}, v_{-i}) of the remaining bidders, whenever (q_i, v_i) is a winning declaration for i, so is any bid (q_i', v_i') with $q_i' \geq q_i$ and $v_i' \leq v_i$.*

Given this very natural generalization of the notion of monotonicity, the following theorem states the desired characterization of truthfulness.

Theorem 14. *A normalized mechanism $m = (A, p)$ for reverse multi–unit auctions among single–minded bidders is truthful if and only if A is monotone and p is the associated critical value payment scheme.*

We will now show that the first two mechanisms presented in this chapter are also truthful when applied to the case of single–minded bidders. As we have just seen, it is enough to show that the presented approximation algorithms RA_G (Section 4.1) and RA_{PTAS} (Section 4.2) are monotone as defined above. For both algorithms we have already seen that they are monotone with respect to the declared prices, i.e., if (q_i, v_i) is a winning declaration for bidder i, so is (q_i, v_i') for any $v_i' \leq v_i$. It is then sufficient now to prove monotonicity with respect to the offered numbers of items, i.e., if (q_i, v_i) is a winning declaration for bidder i, so is (q_i', v_i) for any $q_i' \geq q_i$. Clearly, monotonicity as in the above definition then follows.

We start by considering RA_G. As before, the results obtained for RA_G will then be useful for our argumentation about RA_{PTAS}. The proof of the next theorem is very close to that of Theorem 5. Nevertheless, there are important differences about the arguments given for some of the considered cases.

Theorem 15. *Algorithm RA_G is monotone with respect to the declared numbers of items.*

Proof: We use the notation introduced in the proof of Theorem 5. Assume that (q_i, v_i) is a winning declaration for bidder i and that i's position in the ordered list of bids is p_i. Then consider a declaration (q_i', v_i), $q_i' > q_i$, that lets i move to position $p_i' \leq p_i$ in the list. By $Best$ and $best$ we refer to the solution computed by RA_G if i declares q_i and its cost. Furthermore, for any $t \in \{1, \ldots, n\}$ let $G(t)$, $G'(t)$, $num(t)$, $num'(t)$, $val(t)$ and $val'(t)$ denote the content of RA_G's variables after step t with bidder i declaring q_i or q_i', respectively. Again, it is clearly sufficient to show that the following two claims hold with bidder i declaring q_i':

Claim 1 During its computation RA_G finds a possible solution S that includes i and has cost at most $best$.

Claim 2 RA_G finds no solution without i that it does not find with bidder i declaring q_i.

We assume that i is a part of the solution returned by RA_G when declaring q_i. It is, however, not clear how i becomes part of this solution. Depending on this we differentiate between the following cases. In all cases, let B denote the set of bidders on positions in $\{p'_i, \ldots, p_i - 1\}$ that are added to G when bidder i declares q_i.

Case 1: When declaring q_i, bidder i is not added to G.

Case 1.1: When declaring q'_i, bidder i is not added to G.

This is analogous to Case 2.1 in the proof of Theorem 5. Bidder i's declaration moves up the list of bids and appears in a new solution that corresponds to the previously best solution without the bidders in B and obviously has cost no higher than before. The other feasible solutions found by RA_G remain unchanged.

Case 1.2: When declaring q'_i, bidder i is added to G.

This is analogous to Case 2.2 in the proof of Theorem 5. Bidder i is added to G in step p'_i. After this, only bidders from B are added to G. The first bidder from B that cannot be added completes the solution S required in Claim 1. Claim 2 follows, since all solutions considered after step p'_i must contain i.

Case 2: When declaring q_i, bidder i is added to G.

Case 2.1: When declaring q'_i, bidder i is added to G.

This case is quite similar to Case 1 in the proof of Theorem 5. Here, however, we must pay attention to the fact that it is the number of offered items that causes bidder i's price per item to decrease. As a result, we cannot be sure which bidders are still added to G when q'_i is declared. If all bidders from B are still added, then the proof is in fact analogous to that of Case 1 in the proof of Theorem 5. Otherwise, assume that k is the first bidder from B that is processed and not added to G. This case is depicted in Figure 4.5.

The best solution found with i declaring q_i can be written as $G(p'_i - 1) \cup B \cup \{i\} \cup R$, where R consists of some bidders on positions larger than p_i. Let $B' \subset B$ denote the bidders from B on positions smaller than that of k. Since the list of bids up to position $p'_i - 1$ remains unchanged when i declares q'_i, we have that $G'(p'_i - 1) = G(p'_i - 1)$. With bidder i declaring q'_i, RA_G finds the feasible solution $G(p'_i - 1) \cup \{i\} \cup B' \cup \{k\}$. Clearly, this can be used as the solution required by Claim 1. Again, no new solutions without i will appear, since all solutions found after step p'_i contain i and the list up to position $p'_i - 1$ has not been changed.

Case 2.2: When declaring q'_i, bidder i is not added to G.

We did not have to consider this case in Theorem 5. We start by showing the solution S required for Claim 1. Since i is not added to G when declaring q'_i, we know that RA_G finds the feasible solution $G'(p'_i - 1) \cup \{i\}$. With $G'(p'_i - 1) = G(p'_i - 1)$ it immediately follows that we have found an appropriate solution, since we are dealing with a subset of $Best$, and set $S = G(p'_i - 1) \cup \{i\}$. By $c(S)$ we denote the total cost of solution S. Figure 4.5 shows this situation.

Unfortunately, Claim 2 need not necessarily hold in this case. Instead of showing Claim 2, we will prove directly that S as defined above is the best solution that is ever found with i declaring q'_i. Clearly, this will also suffice. We start by observing the following fact:

Claim: It is $G'(p_i) = G(p_i)\backslash\{i\}$.

Proof: We have already noted that $G'(p'_i - 1) = G(p'_i - 1)$ and, thus, $num'(p_i - 1) = num(p'_i - 1)$. By declaring q'_i bidder i is moved from position p_i to p'_i. By assumption i is not added to G. It is a straightforward observation that

$$G'(t) = G(t-1) \text{ and } num'(t) = num(t-1)\, \forall\, t \in \{p'_i, \dots, p_i\},$$

since every bidder k on position $p_k \in \{p'_i, \dots, p_i - 1\}$ is moved to position $p_k + 1$ when i declares q'_i. We conclude that

$$G'(p_i) = G(p_i - 1) = G(p_i)\backslash\{i\}.$$

This finishes the proof of the claim. □

The above proof uses the fact that, with bidder i declaring q'_i, in steps $p'_i + 1, \dots, p_i$ algorithm RA_G finds the solutions it previously found in steps $p'_i, \dots, p_i - 1$. Hence, all solutions found in these steps have higher cost than the previously best solution $Best$, which was found in a later step. Since our new solution S has cost at most $best$, no better solutions than S are found until step p_i.

Now consider any solution found in a later step. We can write this solution as $G'(p_i) \cup R$, where R is a set of bidders that are on positions larger than p_i. Let

$$q_R = \sum_{j \in R} q_j \text{ and } v_R = \sum_{j \in R} v_j.$$

Since all bidders in R are on positions larger than p_i, it follows that $v_R/q_R \geq v_i/q_i$. When declaring q_i, i is added to G. It follows that $m - num(p_i - 1) > q_i$ and, since $G'(p_i) \cup R$ is a feasible solution, we have that $q_R \geq m - num'(p_i) = m - num(p_i - 1) > q_i$. We conclude that $v_R > v_i$ and, thus, solution $G'(p_i) \cup R$ has total cost

$$\begin{aligned} val'(p_i) + v_R &= val(p_i - 1) + v_R \\ &> val(p'_i - 1) + v_i \\ &= c(S). \end{aligned}$$

This completes the proof of Case 2.2. □

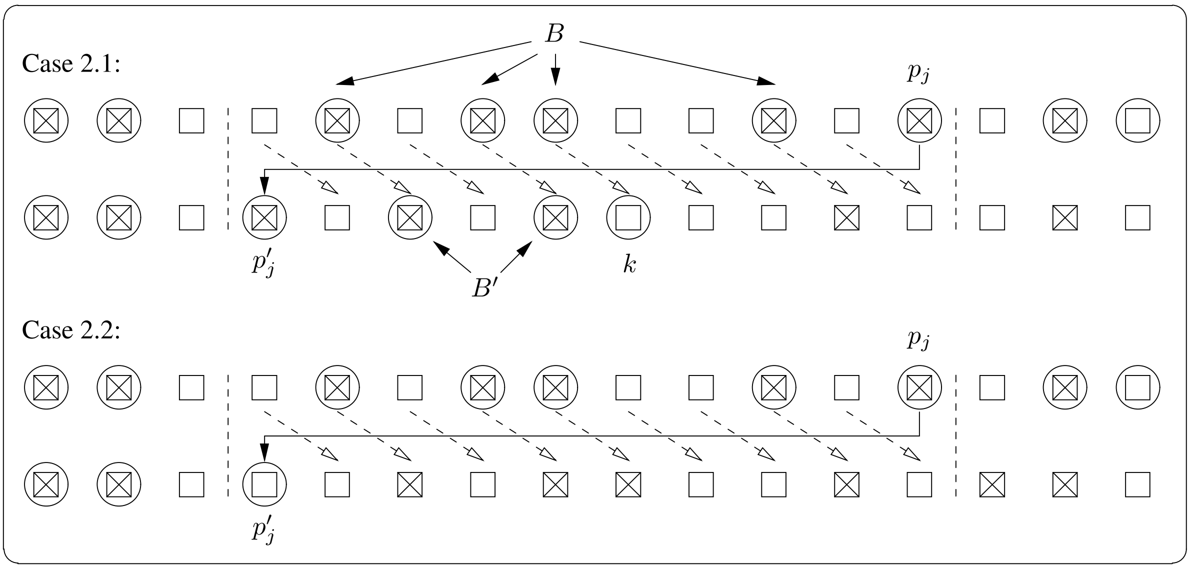

Figure 4.5: Cases from the proof of Theorem 15. Notation as in Figure 4.1.

This finishes our discussion of algorithm RA_G. Next, we will turn our attention to RA_{PTAS}. Remember that, in our previous analysis, we considered RA_{PTAS} as being the result of the MIN–operator's application to a set of simpler building blocks RA_S, which could themselves be seen as slight modifications of RA_G. Once again, this will be the way of proving RA_{PTAS}'s monotonicity.

Theorem 16. *Algorithm RA_{PTAS} is monotone with respect to the declared numbers of items.*

Proof: For algorithm RA_S and given declarations $(q_1, v_1), \ldots, (q_n, v_n)$ let $c_S(q_i)$ denote the cost to society resulting from the solution computed by RA_S as a function of the number of items q_i declared by bidder i. In analogy to the definition of bitonicity we state the following two claims about RA_S:

Claim 1 Let (q_i, v_i) be a winning declaration. Then $c_S(q'_i) \leq c_S(q_i)$ for all $q'_i \geq q_i$.

Claim 2 Let (q_i, v_i) be a loosing declaration. Then $c_S(q'_i) \leq c_S(q_i)$ for all $q'_i \leq q_i$.

Furthermore, we have to require that algorithm RA_S is monotone with respect to our generalized definition and, thus, add another claim:

Claim 3 Algorithm RA_S is monotone with respect to the declared numbers of items.

Now assume that all three claims hold for RA_S and that (q_i, v_i) is a winning declaration with respect to RA_{PTAS}. Increasing q_i to q'_i then causes the cost to society of any RA_S that selects (and continues to select) i to decrease or stay constant, while the cost to society of any RA_S that does not select i behaves non–decreasing. Since RA_{PTAS} is just an application of the MIN–operator, it immediately follows that (q_i', v_i) is winning. This proves the claim.

We now have to show that our claims will indeed hold. We first argue about Claims 1 and 3. Consider some algorithm RA_S, assume that (q_i, v_i) is winning with respect to RA_S and $q_i' \geq q_i$. If $i \notin S$, then i is added by RA_G in Step 2 of the algorithm. In this case it follows directly from the proof of Theorem 15 that i is still selected when declaring q_i' and that $c_S(q_i') \leq c_S(q_i)$. We then assume that $i \in S$. Now consider what RA_G does after q_i is increased to q_i'. Bidders that were previously not added to G will still not be added, because the number of items left to be acquired has decreased. Bidders that were previously added, might again be added or not. The first one that is not added, however, completes a feasible solution with cost no higher than that of the previously best solution. Obviously, there is no question about the fact that i is still selected. Claims 1 and 3 follow.

We then argue about Claim 2. We fix some RA_S and assume that (q_i, v_i) is loosing with respect to RA_S and $q_i' \leq q_i$. We need to show that $c_S(q_i') \leq c_S(q_i)$. Clearly, since (q_i, v_i) is a loosing declaration, we have that $i \notin S$ and, thus, it is enough to show that Claim 2 is true for RA_G.

We now argue about RA_G. Decreasing q_i to q_i' lets i move from position p_i to $p_i' \geq p_i$ in the ordered list of bids. If the best solution is originally found in a step before p_i, then this is not changed by declaring q_i' and the claim follows. Hence, we can assume that the best solution is originally found in a step $p_k > p_i$. Since i is not part of this solution, we know that i is not added to G when declaring q_i. The following is the central observation for this proof:

Claim: If, by declaring q_i', bidder i moves to position $p_i' < p_k$, then i is again not added to G.

Proof: Assume that i is added to G when declaring q_i'. Using the notation from the proof of Theorem 15, we observe that

$$G(t) = G'(t-1)\, \forall\, t \in \{p_i + 1, \dots, p_i'\}.$$

The best solution with i declaring q_i can be written as $G(p_i') \cup R$, where R is a set of bidders on positions larger than p_i'. Again, let

$$q_R = \sum_{j \in R} q_j \text{ and } v_R = \sum_{j \in R} v_j.$$

From the fact that i is added to G when declaring q_i' and that $G(p_i') \cup R$ is a feasible solution, we conclude that

$$\begin{aligned} q_i' &< m - num'(p_i' - 1) \\ &= m - num(p_i') \\ &< q_R. \end{aligned}$$

We also note that $v_i/q_i' \leq v_R/q_R$, since all bidders in R are on positions larger than p_i'. It then follows that $v_i < v_R$. When bidder i declares q_i, RA_G takes into account the feasible solution $G(p_i - 1) \cup \{i\}$, since i is originally not added to G. With $G(p_i - 1) \subseteq G(p'_i)$ it follows from the above observation that this solution has smaller cost than $G(p'_i) \cup R$. Hence, (q_i, v_i) must be a winning declaration, a contradiction. □

We can now finish our proof of Claim 2 for algorithm RA_G. First, assume that i indeed moves to a position $p'_i < p_k$. Then, since i is again not added to G, RA_G finds exactly the same solutions not including i as before. This immediately gives the claim. If, on the other hand, i moves to position $p'_i \geq p_k$, then we know that at least the previously best solution $Best$ is still found in step $p_k - 1$. Thus, it is not important what happens in the algorithm's remaining steps, since we can only improve on this solution's cost. This finishes the proof of Theorem 16. □

We have seen that the greedy approximation algorithm (Section 4.1) and the more sophisticated PTAS based on partial enumeration (Section 4.2) for reverse multi–unit auctions are monotone with respect to the declared numbers of items and, thus, suitable for the design of truthful approximate mechanisms for unknown single–minded agents. We note that a similar result cannot be obtained for the FPTAS presented in Section 4.3. In this case bitonicity of the building blocks with respect to the declared numbers of items is lost, because the applied dynamic programming approach computes optimal solutions based on rounded declarations. These, however, need of course not be optimal with respect to the original declarations. With this negative result we finish our discussion of unknown single–minded bidders.

Chapter 5

Other Utilitarian Optimization Problems

In Chapter 4 we have constructed truthful approximate mechanisms for reverse multi–unit auctions. We will now consider further utilitarian optimization problems and see how previous results can be applied here.

As a natural continuation of our discussion so far, we will first look at the popular scenario of *forward multi–unit auctions* in Section 5.1. It will turn out that all the mechanisms that were presented for reverse multi–unit auctions can easily be adopted for the forward variant.

In Section 5.2 we will then introduce a generalization of the Knapsack Problem, referred to as *Job Scheduling with Deadlines* (JSD). As for the Knapsack Problem, an FPTAS based on scaling the profits associated with each job before applying a dynamic programming approach with pseudo–polynomial running time is known for this problem. Again this FPTAS is not monotone by itself. We will show that we can use our enumeration technique from Section 4.3 to obtain a monotone FPTAS, which can then be used in the construction of a truthful approximate mechanism.

As an example for several network and routing problems that are interesting from the point of view of mechanism design, Section 5.3 deals with the *Constrained Shortest Path Problem*. Once more, we will be able to apply our technique to obtain a monotone FPTAS.

In contrast to the previous chapter we are now faced with both minimization and maximization problems. In Section 5.4 we will show how the results from Chapter 3 can be transferred to the maximization case and what the main differences are.

5.1 Forward Multi–Unit Auctions

The setting of *forward multi–unit auctions* is similar to that introduced for the reverse case. A single auctioneer wants to sell a number of identical goods to a set of potential buyers, or bidders. Each bidder i reports the number q_i of items she is interested in and the maximum price v_i she is willing to pay. The auctioneer then chooses the subset of bidders who will be supplied with (at least) the requested number of items. Again we assume that the numbers of items are known to the mechanism and, thus, bidders are known single–minded.

Definition 16. *A* forward multi–unit auction *among n single–minded bidders is defined by bidders' declarations $(q_1, v_1), \ldots, (q_n, v_n)$, a number of items m and feasible solutions*

$$O(q_1, \ldots, q_n) = \{S \subseteq \{1, \ldots, n\} | \sum_{i \in S} q_i \leq m\}.$$

The output S must be chosen to maximize $\sum_{i \in S} v_i$.

In case of maximization problems, the sum of the prices declared by the selected subset of bidders is referred to as *social welfare* (instead of cost to society). We will adopt our notation to reflect this difference.

Notation: For any subset of bidders S, the social welfare is denoted as $w(S) := \sum_{i \in S} v_i$.

The first mechanism for reverse multi–unit auctions that was introduced in Section 4.1 was obtained using a simple greedy technique based on ordering bids by the offered price per item. A similar algorithm FA_G for forward multi–unit auctions is given by the well known greedy approach for the Knapsack Problem. This algorithm has been analyzed by Mu'alem and Nisan [10] in the context of mechanism design.

Algorithm 9: FA_G

Input: Number of items m and bids $(q_1, v_1), \ldots, (q_n, v_n)$.

1. Choose π to order bids by decreasing *price per item*

   $$v_{\pi(1)}/q_{\pi(1)} \geq \cdots \geq v_{\pi(n)}/q_{\pi(n)}$$

   and define $G := \emptyset$, $num := 0$.
2. Repeat Step 3 for $i = 1, \ldots, n$:
3. If $num + q_{\pi(i)} \leq m$, then $G := G \cup \{\pi(i)\}$, $num := num + q_{\pi(i)}$.
4. Let $v_k = \max_i v_i$. Set $S := \text{argmax}\{w(G), w(\{k\})\}$.

Output: Subset of bidders S.

Theorem 17. ([10]) *Algorithm FA_G for forward multi–unit auctions among known single–minded bidders has approximation ratio 2. Furthermore, algorithm FA_G is monotone.*

We can then continue with the construction of more sophisticated approximate mechanisms by again applying the toolkit due to [10], that was explicitly developed for forward multi–unit auctions. Algorithm FA_{PTAS} is exactly the algorithm introduced by Sahni ([13]) for the Knapsack Problem in 1975. Monotonicity can be shown as for the reverse variant presented in Section 4.2 in analogy to the proof of Theorem 9. FA_{PTAS} can be seen as the application of a MAX–operator to a set of algorithms that are basically modified versions of FA_G. FA_{PTAS}'s monotonicity is then concluded from the monotonicity and bitonicity of these simpler building blocks, where an appropriate formal definition of bitonicity for the case of maximization problems and the MAX–operator can be found in [10].

Algorithm 10: FA_{PTAS}

Input: Number of items m, bids $(q_1, v_1), \ldots, (q_n, v_n)$ and parameter k.

1. Consider all subsets of bidders with cardinality at most k. Formally, let $Best := \emptyset$, $best := 0$ and repeat the following steps for all sets $S \subseteq \{1, \ldots, n\}$, $|S| \leq k$:
2. Use algorithm FA_G to complete the partial solution S. Formally, let
$$q(S) = \sum_{i \in S} q_i \text{ and } w(S) = \sum_{i \in S} v_i.$$
Run FA_G on the input given by number of items $m - q(S)$ and bids $\{(q_i, v_i) | i \notin S\}$ and let $S' := S \cup S_G$, where S_G is the solution computed by FA_G.
3. Compare S' to the best solution found so far. If $w(S') > best$, then
$$Best := S' \text{ and } best := w(S').$$

Output: Subset of bidders $Best$.

Theorem 18. ([13]) *For any given parameter $k \in \mathbb{N}$, algorithm FA_{PTAS} has approximation ratio $1 + \frac{1}{k}$ and running time $O(k \cdot n^{k+1})$, where n is the number of participating bidders.*

Theorem 19. *Algorithm FA_{PTAS} is monotone and bitonic.*

The proofs of the two preceding theorems are analogous to the proofs of Theorems 7 and 9 in Section 4.2. In Section 4.5 the more general model of unknown single–minded bidders was considered. Two of the mechanisms presented for reverse multi–unit auctions turned out to be truthful in this scenario as well. Similar results can be obtained for forward auctions.

Both FA_G and FA_{PTAS} meet the requirements described in Section 4.5. This is stated by the following theorem. The proof, which is analogous to the proofs of Theorems 15 and 16, is omitted.

Theorem 20. *Algorithms FA_G and FA_{PTAS} are monotone with respect to the declared numbers of items.*

It remains to be shown that the FPTAS presented in Section 4.3 can also be adopted. In fact, we will see that the technique that was used for the construction of a monotone FPTAS for reverse multi–unit auctions can be applied to a whole class of problems. The following results are stated for maximization problems only. They can, however, easily be transferred to the minimization case in analogy to the example given in Section 4.3.

Let Π be some utilitarian maximization problem and assume single–minded agents. Then according to the notation that was introduced in Section 2.3, input is given by declarations $(O, v) = ((O_1^+, v_1), \ldots, (O_n^+, v_n))$ and Π is fully specified by its set $O(O_1^+, \ldots, O_n^+)$ of feasible solutions. For any given set of declarations and any agent i, we require that there is at least one feasible solution $o \in O(O_1^+, \ldots, O_n^+)$, such that agent i is selected, i.e., $i \in S(o)$. Now let $V = \max_i v_i$ and assume that there is an exact algorithm A_Π for Π with pseudo–polynomial running time $O(poly(n, V))$.

Algorithm 11: A_k^Π

Input: Declarations $(O_1^+, v_1), \ldots, (O_n^+, v_n)$ and $\varepsilon > 0$.

1. Set $v_i' := \min\{v_i, 2^{k+1}\}$ for all $i \in \{1, \ldots, n\}$.
2. Let
$$\alpha_k := \frac{(1+\varepsilon) \cdot n}{\varepsilon \cdot 2^k}$$
and scale the modified declarations:
$$v_i'' := \lfloor \alpha_k \cdot v_i' \rfloor, i = 1, \ldots, n$$
3. Run algorithm A_Π on declarations
$$(O_1^+, v_1''), \ldots, (O_n^+, v_n'')$$
and let o be the returned solution.

Output: Solution o.

We can then define algorithms A_k^Π, $k \in \mathbb{N}$, as presented for reverse multi–unit auctions. A_k^Π works by scaling declarations depending on parameter k and then running algorithm A_Π on the modified input. To ensure polynomial running time, declarations that are too large will be cut appropriately. Compared to minimization, scaling becomes somewhat simpler now, since by the above assumption, a trivial lower bound on the welfare of an optimal solution is given by the maximum declared value V, which appears in at least one feasible solution. Precisely, we have that $V \leq w(Opt(O,v)) \leq nV$.

Analogously to Lemmas 8 and 9 the following are the central observations on the way towards a monotone FPTAS.

Lemma 12. *Let $\varepsilon > 0$, $k \in \mathbb{N}$, $w(Opt(O,v))$ denote the welfare of an optimal solution for a given set of declarations (O,v) and $V = \max_i v_i$. Algorithm A_k^Π's running time is bounded by $O(poly(n,\varepsilon^{-1}))$. If $2^k \leq V < 2^{k+1}$, then A_k^Π computes a $(1+\varepsilon)$–approximation to $w(Opt(O,v))$ on declarations (O,v).*

Lemma 13. *For any $k \in \mathbb{N}_0$, algorithm A_k^Π is monotone and bitonic.*

For the proof of Lemma 12, note, that running time follows from the range that declarations are scaled to. If $2^k \leq V < 2^{k+1}$, then no declaration is changed in Step 1 of algorithm A_k^Π before scaling and V is a lower bound on $w(Opt(O,v))$. Analogous to Lemma 8 one can then show that $w(A_k^\Pi(O,v)) \geq (1-\varepsilon/(1+\varepsilon)) \cdot w(Opt(O,v))$. As in Lemma 9, the claims of Lemma 13 follow from the monotonicity of the scaling process and the fact that A_Π computes an optimal solution on the scaled declarations.

As in Section 4.3, we now characterize the desired FPTAS as an application of the MAX–operator to an (infinite) set of algorithms. Formally, let $\mathcal{A}_\Pi := \{A_k^\Pi | 0 \leq k < \infty\}$ and

$$MAX(\mathcal{A}_\Pi)(O,v,\varepsilon) := \operatorname{argmax}\{w(A_k^\Pi(O,v,\varepsilon)) | A_k^\Pi \in \mathcal{A}_\Pi\}$$

for any declarations (O,v) and $\varepsilon > 0$. As in Lemma 10 we observe the following fact.

Algorithm 12: A^Π_{FPTAS}

Input: Declarations $(O_1^+, v_1), \ldots, (O_n^+, v_n)$ and $\varepsilon > 0$.

1. Let $V := \max_i v_i$, $w_{max} := 0$ and $o := \emptyset$. Then repeat the next step for all $k = 0, \ldots, \lceil \log\left(\frac{n}{\varepsilon}V\right)\rceil$:
2. Run A_k^Π on the given declarations and let o_k be the returned solution. If $w(o_k) > w_{max}$, then set $o := o_k$ and $w_{max} := w(o_k)$.

Output: Solution o.

Lemma 14. *For any $\varepsilon > 0$, $MAX(\mathcal{A}_\Pi)$ is monotone and has approximation ratio $1+\varepsilon$.*

In the last step we now need to describe a polynomial time implementation of $MAX(\mathcal{A}_\Pi)$.

From the fact that, for $k > \log\left(\frac{n}{\varepsilon}V\right)$, algorithm A_k^Π scales all declarations to 0, it immediately follows that $A_{FPTAS}^\Pi = MIN(\mathcal{A}_\Pi)$. The following theorem states that A_{FPTAS}^Π is a monotone FPTAS for Π.

Theorem 21. *Let Π be a utilitarian maximization problem among known single–minded agents as defined above. For any $\varepsilon > 0$, algorithm A_{FPTAS}^Π has approximation ratio $1 + \varepsilon$ and running time $O(poly(n, \varepsilon^{-1}, \log V))$, where n is the number of participating agents and V denotes the value of the largest declaration. Furthermore, A_{FPTAS}^Π is monotone.*

To finish this section, it remains to be shown that forward multi–unit auctions (FA) are among the problems to which our technique applies. This is, however, a direct consequence of the fact that an appropriate pseudo–polynomial algorithm is obtained by a dynamic programming approach. In analogy to the discussion in Section 4.3, we now let $A(i, v)$ denote the minimum number of items that has a declared total value of exactly v, if only bidders $\{1, \ldots, i\}$ are considered. Formally, we let

$$A(i, v) := \min\{q(S) | S \subseteq \{1, \ldots, i\} \wedge w(S) = v\}$$

and observe the recurrence

$$A(i, v) = \min\{A(i-1, v), A(i-1, v - v_i) + q_i\},$$

which can easily be verified by induction. Since we are interested in a subset of bidders S that results in $\max\{v | A(n, v) \leq m\}$, it is obviously sufficient to compute the entries of a $(n \times nV)$–matrix, V denoting the largest declared value. The desired pseudo–polynomial running time is then a straightforward observation.

5.2 Job Scheduling with Deadlines

Another class of utilitarian problems that is of interest from the point of view of mechanism design is given by a variety of scheduling problems. A set of agents presents jobs that need to be processed on some machines that are kept by the mechanism. Agents declare the time requirements of their jobs, any constraints that have to be taken into account and a price that they are willing to pay if their jobs are processed and no constraints are violated. Based on this information, the mechanism then decides which jobs are to be processed on each machine and in which order this is going to happen.

In this Section we will look at one of these scheduling problems, termed *Job Scheduling with Deadlines*. We will show that our technique described in Section 5.1 can be used to obtain a monotone FPTAS for this problem, which can then be used to construct a truthful approximate mechanism as described in Section 2.3.

We start by describing the problem. We assume that every agent presents a single job to the mechanism and that all jobs are to be processed on a single machine. An agent declares the time that her job will take on the machine, a deadline at which it should be finished and a price that she is willing to pay if the deadline is not exceeded. The mechanism then decides in which order the jobs are processed on the machine. We will assume that running times and deadlines for all jobs are known to the mechanism and, thus, agents are known single–minded. We now define the problem formally.

Definition 17. Job Scheduling with Deadlines *(JSD) for single–minded agents is defined by a set of reported jobs* $(t_1, d_1, v_1), \ldots, (t_n, d_n, v_n)$*, job* i *having running time* t_i*, deadline* d_i *and profit* v_i *if finished before its deadline. Feasible output is given by any permutation* $\pi \in S_n$*. Corresponding to previous notation, we define*

$$i \in S(\pi) \Leftrightarrow \sum_{j=1}^{\pi(i)} t_{\pi^{-1}(j)} \leq d_i,$$

i.e., $i \in S(\pi)$ *if and only if job* i *is finished before its deadline when jobs are processed in the order defined by* π*. The output* π *must then be chosen to maximize* $\sum_{i \in S(\pi)} v_i$*.*

Notation: For any $S \subseteq \{1, \ldots, n\}$, let $w(S) = \sum_{i \in S} v_i$ and $t(S) = \sum_{i \in S} t_i$.

As we have seen in the previous section, our technique for the construction of a monotone FPTAS is applicable if we can find an algorithm for JSD with running time that is polynomial in the number of jobs n and the largest declared price $V = \max_i v_i$. Such an algorithm for JSD has first been described by Sahni [14] in 1976. Although we will present a different version here, it is still based on the same central observation.

Definition 18. *Let* $J \subseteq \{1, \ldots, n\}$ *be a subset of jobs with* $J = \{\sigma(1), \ldots, \sigma(|J|)\}$ *and* $d_{\sigma(1)} \leq \ldots \leq d_{\sigma(|J|)}$*. We say that* J *is* processable *if* $\sum_{i=1}^{l} t_{\sigma(i)} \leq d_{\sigma(l)}$ *for all* $l = 1, \ldots, |J|$*, i.e., if every job is finished before its deadline when jobs are processed in order of increasing deadlines. For a given instance* (t, d, v) *of JSD, let* $P(t, d)$ *denote the set of all processable subsets of jobs.*

Lemma 15. ([14]) *Given an instance* (t, d, v) *of JSD, we have that*

$$\max_{\pi \in S_n} w(S(\pi)) = \max_{J \in P(t,d)} w(J).$$

Proof: It is clear that $\max_{J \in P(t,d)} w(J) \leq \max_{\pi \in S_n} w(S(\pi))$, since, for a given processable set of jobs J, we can define a permutation $\pi_J \in S_n$, such that $J = \{\pi_j^{-1}(1), \ldots, \pi_j^{-1}(|J|)\}$. It follows that $J \subseteq S(\pi_J)$.

It remains to be shown that $\max_{\pi \in S_n} w(S(\pi)) \leq \max_{J \in P(t,d)} w(J)$. We fix a permutation $\pi \in S_n$ and consider the set of jobs $S(\pi)$. All jobs in $S(\pi)$ are finished before their deadlines if they are processed in the order given by π. We observe that this can also be obtained by processing in order of increasing deadlines. To see this, assume that there exist $i, j \in S(\pi)$, such that $\pi(i) < \pi(j)$ and $d_i > d_j$. Obviously, both jobs must then be finished before d_j. Hence, jobs i and j can as well be processed in the opposite order. It follows that $S(\pi)$ is processable. This completes the proof. □

Lemma 15 gives a new characterization of JSD. In the original definition, the problem's search space was given as the set of permutations S_n. By Lemma 15, a corresponding search space is defined by all processable subsets of jobs. Applying the technique of dynamic programming, this fact can be used in the construction of the desired pseudo–polynomial algorithm.

Assume that jobs are ordered by increasing deadlines, i.e., $d_1 \leq \cdots \leq d_n$. By $J(i, v)$ we denote a subset of jobs with minimum total running time that results in total profit of exactly v and contains only jobs from $\{1, \ldots, i\}$. Formally, let

$$J(i, v) := \operatorname{argmin}\{t(J) \mid J \subseteq \{1, \ldots, i\} \text{ is processable and } w(J) = v\}.$$

For ease of notation we let $t(i, v)$ refer to the total running time $t(J(i, v))$ of jobs in $J(i, v)$. We set $J(i, v) = \emptyset$ and $t(i, v) = +\infty$ for any $i, v < 0$, $J(0, v) = \emptyset$ and $t(0, v) = +\infty$ for all $v \neq 0$, $J(0, 0) = \emptyset$ and $t(0, 0) = 0$. The following lemma states the recurrence required for a dynamic programming approach.

Lemma 16. *Let* $V = \max_j v_j$, $i \in \{1, \ldots, n\}$ *and* $v \in \{0, \ldots, nV\}$.

If $t(i-1, v-v_i) + t_i \leq d_i$ *and* $t(i-1, v-v_i) + t_i \leq t(i-1, v)$ *then*

$$J(i, v) = J(i-1, v-v_i) \cup \{i\}$$

else

$$J(i, v) = J(i-1, v).$$

Clearly, the claim can easily be verified by induction. We are interested in a subset of jobs $J(n, v^*)$, such that $v^* = \max\{v \mid t(n, v) < +\infty\}$. To find this subset, we need to compute the entries of a $(n \times nV)$–matrix. From Lemma 16 it follows that each entry can be found in constant time. This gives the desired pseudo–polynomial algorithm JSD_{dyn} with running time $O(n^2 V)$. From our previous discussion it is immediately clear how a monotone FPTAS based on JSD_{dyn} can be constructed. This finishes our discussion of JSD.

5.3 The Constrained Shortest Path Problem

Finally, we will now look at a problem from the area of networks and routing, which is another central field of interest in the context of mechanism design. We assume that we have a network that is defined by a given graph in which each edge is owned by a different agent.

Every agent declares the properties of her edge (or communication network connection) and the price at which it is available. What properties have to be declared obviously depends on the respective problem.

The example of this kind that we will study is a minimization problem. Without going through the proofs again, we note once more that our enumeration technique can be used to obtain monotone FPTASs for general utilitarian minimization problems in analogy to Section 4.3. As for reverse multi–unit auctions we require approximation ratio $(1+\varepsilon)$ with $\varepsilon < 1$.

Definition 19. *The* Constrained Shortest Path Problem (CSP) *with single–minded agents is defined by a graph $G = (V, E)$ with agents declaring length $l(e)$ and cost $c(e)$ for each edge $e \in E$, two special vertices $s, t \in V$ and a maximum total length L. We want to find a path P from s to t that satisfies $l(P) = \sum_{e \in P} l(e) \leq L$ and has minimal cost $c(P) = \sum_{e \in P} c(e)$.*

It has been shown that CSP is weakly NP–hard. Pseudo–polynomial algorithms have been proposed by Hassin [7] and Phillips [12]. We will present a slight modification of the approach presented by Phillips to obtain the required pseudo–polynomial algorithm with running time that is polynomial in the size of the graph and edges' costs. This algorithm can then be turned into a monotone FPTAS.

We modify the so–called *search graph* described in [12] and encode total cost of any path rather than its length.

Definition 20. *Given a graph $G = (V, E)$, $|V| = n$, $|E| = m$, with edge costs and lengths as defined above and $C = \max_{e \in E} c(e)$, we construct the* search graph $G' = (V', E')$ *by defining*

$$V' := \{(v, c) \mid v \in V, c \in \{0, \dots, nC\}\}$$

and

$$E' := \{\{(u, c), (v, c + c(e))\} \mid \{u, v\} = e \in E, c \in \{0, \dots, nC - c(e)\}\}.$$

Edges are given the length of the original edges in G, i.e., $l(\{(u, c), (v, d)\}) = l(\{u, v\})$.

We can then use Dijkstra's algorithm [5] to solve the *Single Source Shortest Paths Problem* on G'. We are interested in finding a shortest path from s to t in G with length at most L. In G' we compute paths from vertex $(s, 0)$ to vertices (t, c) and let $l(c)$ denote their lengths. We observe that $l(c)$ denotes the length of a shortest path from s to t in G that has cost exactly c. Now let $c^* = \min\{c \mid l(c) \leq L\}$. Then the path from $(s, 0)$ to (t, c^*) in G' obviously corresponds to the desired minimum cost path in G.

The search graph G' can be constructed in time $O(n^2C + mnC)$, Dijkstra's algorithm has running time $|V'| \log |V'| + |E'| = O(n^2C \cdot \log(nC) + mnC)$ on G'. This gives the desired pseudo–polynomial running time.

It should be observed that our enumerative construction method has another advantage when applied to CSP. Using the standard technique of scaling and rounding (i.e., without enumerating) we first need to find a good lower bound on the optimal solution's cost that can be

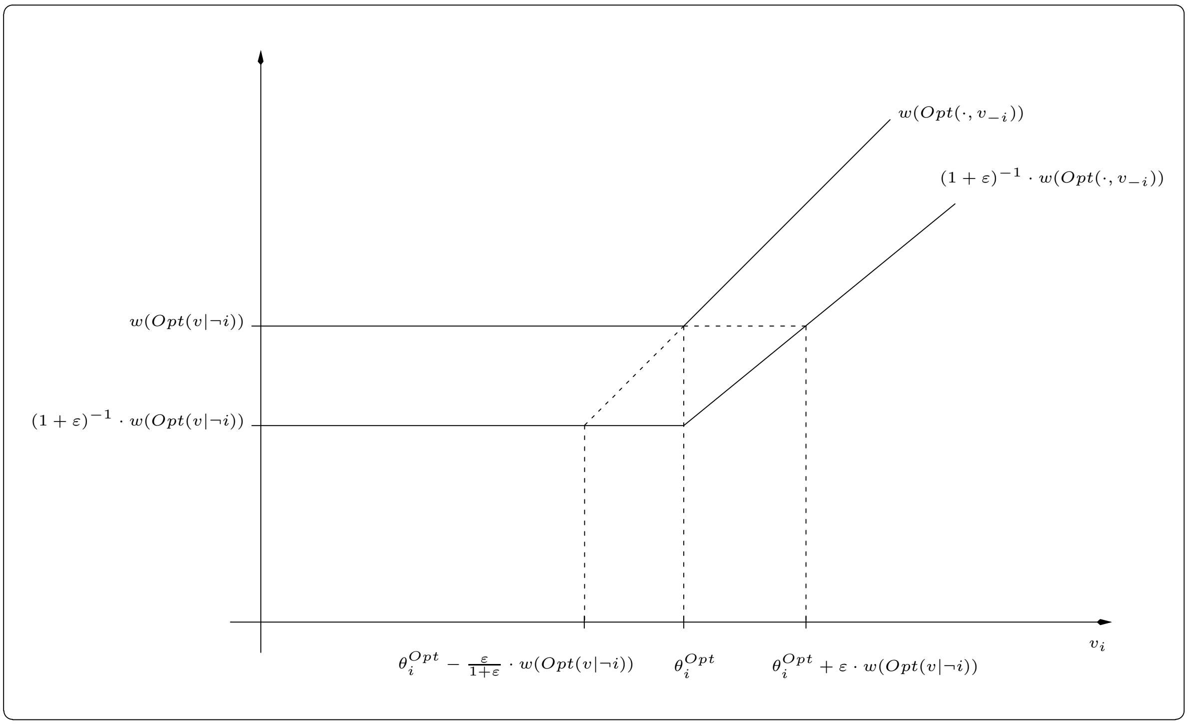

Figure 5.1: Range for the critical values θ_i^A of a monotone approximation algorithm A for a maximization problem.

used in the scaling process. For CSP, however, finding this bound is non–trivial. Hassin [7] solves this problem by first applying a pseudo–polynomial algorithm to "false" scalings until paths of a certain quality are found. Using our technique we do not have to compute good lower bounds explicitly, since they will be found implicitly at some point of time during the enumeration. This might turn out as an important advantage also for other problems where pseudo–polynomial algorithms are known but the computation of sufficiently good lower bounds on the optimal solution's cost is rather time consuming.

5.4 Payment and Maximization Problems

Chapter 3 presented a bound on the overpayment of truthful approximate mechanisms for utilitarian minimization problems and single–minded agents. We will now see how these results can be modified to apply to the maximization case and what the main differences are. We will use the notation that was introduced in Chapters 2 and 3, except that we will again let $w(S(o)) = \sum_{i \in S(o)} v_i$ refer to the social welfare obtained by output o to reflect the fact that we are now considering maximization.

For a fixed utilitarian maximization problem, we consider a normalized VCG mechanism $m_{Opt} = (Opt, p^{Opt})$ and a truthful approximate mechanism $m_A = (A, p^A)$, algorithm A having approximation ratio $(1 + \varepsilon)$. We start by making an observation about the VCG mechanism.

Observation 4. *The VCG mechanism $m_{Opt} = (Opt, p^{Opt})$ is monotone with critical values*

$$\theta_i^{Opt} = w(Opt(v|\neg i)) - (w(Opt(v|i)) - v_i)$$

and p^{Opt} is the associated critical value payment scheme.

This observation is analogous to Lemma 4 and Observation 2. As before, we can then continue by fixing a single agent i and comparing her critical values in algorithms Opt and A. Similarly to Lemma 5 we obtain the following result.

Lemma 17. *Let Opt be an optimal algorithm for some utilitarian maximization problem and let A be a monotone approximation algorithm for the same problem with approximation ratio $(1 + \varepsilon)$. Then the following inequality holds for all declarations v_{-i}:*

$$\theta_i^{Opt} - \frac{\varepsilon}{1+\varepsilon} \cdot w(Opt(v|\neg i)) \leq \theta_i^A \leq \theta_i^{Opt} + \varepsilon \cdot w(Opt(v|\neg i)) \tag{5.1}$$

The idea of the proof, which is analogous to that of Lemma 5, is depicted in Figure 5.1. We again start with agent i declaring $v_i = \theta_i^{Opt}$ and observe the existence of at least two optimal solutions, one that selects agent i and one that does not. Now assume that we decrease v_i. Then the value of any solution selecting i will decrease by the same amount and finally have a value smaller than $(1 + \varepsilon)^{-1}$ times the value of an optimal solution not selecting i. At this point of time, algorithm A must stop to select i because of its approximation ratio. This gives the lower bound. On the other hand, assume that we increase v_i. Obviously, the value of any solution selecting i will increase. Finally, one of these solutions will become larger than $(1 + \varepsilon)$ times the value of an optimal solution not selecting i. From this point on, A has no choice but to select agent i.

Having these inequalities, we can then again derive bounds on the total payment (also termed *revenue* in case of maximization problems) collected by our approximate mechanisms. The stated results are analogous to Theorem 3.

Theorem 22. *Let m_{Opt} and m_A be the mechanisms defined above. Fix some arbitrary declaration vector v and let $Opt = Opt(v)$ and $A = A(v)$ denote the computed outputs. Then the following inequalities hold:*

$$p^A(S(A)) \leq p^{Opt}(S(Opt)) + \varepsilon \cdot |S(Opt)| \cdot w(Opt) \tag{5.2}$$

$$p^A(S(A)) \geq p^{Opt}(S(Opt)) - \frac{\varepsilon}{1+\varepsilon} \cdot (|S(A)| + 1) \cdot w(Opt) \tag{5.3}$$

The proof of this theorem is analogous to that of Theorem 3. We note, however, that both inequalities appear to be somewhat simpler than in the minimization case. This is due to the fact that for maximization problems $w(Opt(v))$ can be used as an upper bound on $w(Opt(v|\neg i))$ for any $i \in \{1, \ldots, n\}$, while in Theorem 3 another inequality had to be used to bound the difference between those two.

Chapter 3 closed with a corollary that used the results from Theorem 3 to derive an approximation ratio for the approximate mechanism's payment. This could be done because the difference in payment was bounded by some multiplicity of the optimal solution's value and an upper bound on this value was given by the payment of the VCG mechanism. We note that an analogous result for maximization problems is not possible, because now we have that $p^{Opt}(Opt) \leq w(Opt)$. Additionally, there is no way of bounding $p^{Opt}(Opt)$ below by some polynomial fraction of $w(Opt)$. To see this, one can look at forward multi–unit auctions and consider a scenario where all items are allocated to a single agent declaring some astronomically high bid and whose payment will then be determined by the (much) smaller bids of the remaining agents. It follows that our previous remark on fully polynomial time approximation schemes does not apply to maximization problems. Hence, an FPTAS for a utilitarian maximization problem need not be an FPTAS with respect to the generated revenue. With this negative result, we finish our discussion of maximization problems.

Chapter 6

Summary and Open Problems

Finally, we will now briefly summarize the presented main results. Rather than keeping to the order in which results were presented in the main part of this work, focus will now be laid on how single results relate to each other, what implications can be observed and what questions have been left unanswered.

We have presented two enumerative techniques that lead to monotone approximation algorithms for forward and reverse multi–unit auctions among known single–minded bidders and improve over the best previously known algorithm that was 2–approximate. The first technique due to Sahni [13] is based on the idea of enumerating over all partial solutions up to a fixed size and completing each partial solution by some monotone procedure. The resulting algorithm is monotone as it can be seen as an application of the toolkit developed by Mu'alem and Nisan in [10]. For multi–unit auctions we obtain a PTAS with strongly polynomial running time that can also be used in the more general case of unknown single–minded bidders. A second enumerative method even leads to monotone FPTASs not only for multi–unit auctions, but for a whole class of utilitarian mechanism design problems. While the standard technique of rounding the declared prices based on data that has to be taken from the input is inherently non–monotone, this problem is avoided if we enumerate over all rounding possibilities and then pick the best found solution. Since the number of rounding possibilities is potentially infinite, we now have to distinguish between the algorithm's specification (i.e., enumerating over some infinite set) and its actual implementation (which must guarantee polynomial running time). Compared to the PTAS described before, the resulting FPTAS has two important disadvantages. First, its running time depends in a logarithmic way on the declared prices and, thus, is only weakly polynomial. Second, it cannot be applied to the more general scenario of unknown single–minded bidders.

A natural question is whether these two disadvantages can be removed. Weakly polynomial running time is due to the fact that the number of rounding possibilities that have to be checked depends on the declared prices. In order to regain strongly polynomial running time, one would have to further limit the number of promising possibilities and make it independent of the actual declarations. To see the problem with unknown bidders for reverse multi–unit auctions, consider algorithm RA_k^{dyn} from Section 4.3 that scales input with respect to parameter k and then applies the exact algorithm RA_{dyn} based on dynamic programming. To obtain a monotone FPTAS, we would need (among others) that RA_k^{dyn} is bitonic with respect not only to the declared prices, but also the declared numbers of items. Now assume that bidder i is selected by the algorithm and increases the number of items she offers. Then consider the solution computed by RA_{dyn} based on the rounded prices. Since RA_{dyn} is an exact algorithm, the new solution's cost with respect to rounded prices can only be smaller. But how do these two solutions relate to each other with respect to the original prices? Here the situation may be opposite due to the rounding and the new solution might potentially have higher cost. It follows that RA_k^{dyn} is not bitonic with respect to the declared numbers of items. It is not clear whether – or how – this problem could be solved.

We also presented a first general bound on the overpayment of approximate utilitarian mechanisms in comparison to the respective VCG mechanism. Combining this result with our monotone FPTASs for utilitarian mechanism design problems shows that the resulting truthful mechanisms truly approximate the VCG mechanism both with respect to the objective function value and payment. It is not clear, however, that a VCG mechanism's payments are optimal among all possible truthful mechanisms. Hence, one could be interested in designing mechanisms that improve on VCG in this aspect. Our results imply, however, that such a mechanism cannot approximate VCG well with respect to the objective function value. Thus, we can observe a certain tradeoff here. If we want to improve on the VCG payments, we necessarily have to accept higher deviation from the optimal objective function value. Certainly, an interesting question is whether such mechanisms exist and can be constructed.

In Sections 4.1 and 4.2 we have seen that our bound on the overpayment from Chapter 3 is tight in an asymptotic sense for reverse multi–unit auctions and the two considered approximation algorithms. Similar results can be obtained for forward multi–unit auctions. Clearly, this suggests that multi–unit auctions might be a worst case example in terms of overpayment among all utilitarian optimization problems. This speculation is supported by the results of Talwar [15], who shows that multi–unit auctions are indeed a worst case example if we look at the payments generated by the VCG mechanism. It would be nice if this could be generalized to capture also approximate truthful mechanisms. It seems to be difficult to obtain such a result in full generality, i.e., for all possible approximate mechanisms for multi–unit auctions. An idea, however, could be to find an appropriate subclass of approximation algorithms that allow such results.

These questions for further research finish our discussion of approximate mechanisms and their frugality.

Bibliography

[1] L. M. Ausubel and P. Cramton. The optimality of being efficient, 1999. Technical report, University of Maryland.

[2] L. M. Ausubel and P. Cramton. Vickrey auctions with reserve pricing, 1999. Technical report, University of Maryland.

[3] G. Calinescu. Bounding the payment of approximate truthful mechanisms. In *International Symposium on Algorithms and Computation (ISAAC)*, 2004. To appear.

[4] E. Clarke. Multipart pricing of public goods. *Public Choice*, 8:17–33, 1971.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press and McGraw–Hill, 1990.

[6] T. Groves. Incentives in teams. *Econometrica*, 41(4):617–631, 1973.

[7] R. Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(1):36–42, 1992.

[8] A. Kothari, D. Parkes, and S. Suri. Approximately-strategyproof and tractable multi-unit auctions. In *ACM Conference on Electronic Commerce*, pages 166–175, 2003.

[9] D. Lehmann, L. O'Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. In *ACM Conference on Electronic Commerce*, pages 96–102, 1999.

[10] A. Mu'alem and N. Nisan. Truthful approximation mechanisms for restricted combinatorial auctions, 2002. In AAAI (poster). Also presented at Dagstuhl workshop on Electronic Market Design.

[11] N. Nisan and A. Ronen. Algorithmic mechanism design. In *ACM Symposium on the Theory of Computing (STOC)*, pages 129–140, 1999.

[12] C.A. Phillips. The network inhibition problem. In *ACM Symposium on the Theory of Computing (STOC)*, pages 776–785, 1993.

[13] S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, 22:115–124, 1975.

[14] S. Sahni. Algorithms for scheduling independent tasks. *Journal of the ACM*, 23(1):116–127, 1976.

[15] K. Talwar. The price of truth: Frugality in truthful mechanisms. In *Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 608–619, 2003.

[16] V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

[17] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *J. Finance*, 16:8–37, 1961.