Accelerated Implementations of the Ascending Proxy Auction

A thesis presented
by

John K. Lai

To
Computer Science and Mathematics
in partial fulfillment of the honors requirements
for the degree of
Bachelor of Arts
Harvard College
Cambridge, Massachusetts

April 5, 2005

Acknowledgements

This senior thesis would not have been possible without the advice, direction, and support of David Parkes. I would be hard pressed to find another mentor who is as caring, knowledgeable, and friendly as you. Whenever I was stuck or feeling down about the research, I would have my weekly meeting with you and be reenergized! I would also like to thank Adi Sunderam for going through this process with me. We made it! To Wendy, for always giving me something to look forward to – I love you. To Jimmy and Jerry who remind me to enjoy my youth. And finally, thank you Mom and Dad for your unconditional love and advice, for helping me focus on the bigger picture, and for always having confidence in me.

Abstract

Combinatorial auctions have important applications to resource allocation problems. These auctions allow bidders to submit bids for entire packages of goods, rather than on single items. This rich bidder interface allows bidders to specify their preferences more precisely, and as a result, allows for more efficient allocation of resources. However, these allowances to bidders come at a computational cost. This senior thesis addresses the computational issues that arise in a specific type of combinatorial auction, the Ascending Proxy Auction. We develop a novel, accelerated algorithm for computing the outcomes of the Ascending Proxy Auction. We also address numerous computational issues that can be generalized to other combinatorial auction algorithms. Finally, we perform extensive experimentation on the performance of our new algorithm. We show that our new algorithm outperforms prior approaches for certain test cases, and may prove to be a useful tool for making combinatorial auctions a practical choice.

Contents

Chapter 1

Introduction

Combinatorial auctions, or auctions where bidders can bid on packages of goods rather than single goods, have become increasingly important in resource allocation problems. These auctions are useful in settings where there are multiple goods being sold and the goods share complex relationships. As an example, consider an individual who is placing bids for a flight and hotel. Depending on the arrival city of the flight, her hotel preferences will change. In the single goods setting, in order to avoid the risk of mismatching the flight and hotel reservation, she would have to wait for the results of the flight auction before bidding on a hotel. However, once she wins the flight, her value for the hotel becomes inflated because the flight is useless without an accompanying hotel. As an alternative to single goods auctions, combinatorial auctions allow bidding on packages of goods. In the previous example, the individual would be able to specify bids for flight-hotel pairs whose cities correspond.

One setting where combinatorial auctions may prove useful is in the Federal Aviation Administration (FAA) allocation of take-off and landing slots at congested airports. Traditionally, individual airport committees have allocated take-off and landing slots based on proposals from individual airlines [4]. However, as airports like New York's LaGuardia become increasingly congested, a fair reallocation of take-off and landing slots must occur. A possible alternative to committee-based reallocation is to use a combinatorial auction where the individual slots are goods and each airline submits bids for the landing slots.

The combinatorial auction has many benefits in this setting. Instead of bidding

on single slots, airlines can be more specific about their needs and bid on entire sets of slots. The combinatorial auction also eliminates the subjectivity of the airport committees and helps maximize social surplus. The auction does so by basing the allocation on the valuations submitted by the bidders, and those who receive the most payoff from the slots will submit the highest bids. Some other applications include the Federal Communications Commission's auction of the wireless spectrum, the privatization of London's bus routes, negotiations in the trucking industry, and negotiations between industrial purchasers and suppliers [6, 7, 5].

Though combinatorial auctions have some very positive properties, allowing bidders to place bids on any package of goods increases the complexity exponentially. As a result, there are numerous computational issues that arise when attempting to implement these auctions.

We will discuss a specific type of combinatorial auction, the ascending proxy auction. This auction mechanism and its results have provable properties that make it a promising choice for the FAA's airport landing slots auction. However, the direct implementation (Pure Proxy) of this auction involves iteratively solving an NP-complete problem, and is feared to be intractable in practice.

1.1 Contributions

In this paper we provide both theoretical and experimental results for an accelerated algorithm that replicates the ascending proxy auction. On the theoretical side, we fully develop an accelerated algorithm for the ascending proxy auction. While related work has been done for accelerated implementations of similar auctions such as iBundle(2) [22], we offer the first accelerated algorithm for the ascending proxy auction. [1]

In addition, we develop novel ways to deal with the computational issues that arise in these exact accelerated implementations. These questions have not been addressed previously in the literature which only provide the main theoretical underpinnings of the accelerated algorithms. However, these computational issues prove to be the bottleneck of the accelerated algorithms in practice, and our work provides a way to

[1] Ascending Proxy Auction has the same dynamics as iBundle(3) with straightforward bidding. This is how iBundle(2) is related to APA

efficiently address these issues. The methods we develop are not algorithm specific and can be generalized to the accelerated implementations of iBundle(2) as well.

Another important contribution is the extensive experimentation we have performed. Previously, much of the work in this area has had a theoretical bias. The experiments that were performed were based on small hand-constructed test cases. To our knowledge, there has not been extensive experimentation using large, randomly generated test cases. Our work presents the first extensive comparison of the performance of accelerated algorithms with the direct Pure Proxy approach. We find that our accelerated algorithm offers certain advantages over the Pure Proxy algorithm and may prove to be useful in practice.

1.2 Outline

In Chapter 2, we define the combinatorial auction and important concepts. Chapter 3 develops the VCG mechanism which provides the backdrop and motivation for mechanisms like iBundle and the ascending proxy auction. We then formally define the ascending proxy auction, its properties, and related work in accelerated implementations in Chapter 5. Chapter 5 gives a full description of our accelerated algorithm with examples. Novel methods for dealing with various computational issues that arise can be found in Chapter 6, and we provide experimental results in Chapter 7.

Chapter 2

Preliminaries

2.1 Problem Specification

In this section, we offer a formal description of the combinatorial auction problem.

A combinatorial auction problem is described by the following:

- A set of agents $A = \{0, 1, \ldots, N\}$, where agent 0 denotes seller.
- A discrete set of goods G.
- Let S denote the set of all subsets of G. A valuation function $v : A \times S \rightarrow \mathbb{R}^+$ We denote this function $v_i(s)$, where $v_i(s)$ represents the value agent i has for the set of goods described by s in monetary terms.

  We assume that v satisfies:

  - Free-disposal: $v_i(s_1) \leq v_i(s_2)$ if $s_1 \subset s_2$.
  - Seller has 0 value for goods: $v_0(s) = 0 \; \forall \; s \in S$.

What differentiates a combinatorial auction from a single good auction is the fact that an agent's valuation function is across S, not just G. This formally represents the ability for agents to provide values on entire subsets of goods rather than just single goods. We notice also that the only requirement on the valuation function is free-disposal. In other words, the valuation function is not necessarily linear and in most cases is probably highly-nonlinear. Recalling the example from our introduction,

the traveler's value for the flight and hotel together is much greater than the sum of her values for each good on its own.

2.2 Definitions

Having defined the inputs to our problem, we now give some basic terminology and describe the notion of an outcome, or solution to the combinatorial auction problem.

Let a *bundle* of goods be any subset of the set of all goods, G. Notice that every bundle of goods is contained in the set S.

Definition 1. An *allocation* is a tuple $(s_1, \ldots, s_N)$. For a given allocation $(s_1, \ldots, s_N)$, agent i receives bundle s_i. A *feasible allocation* is an allocation $(s_1, \ldots, s_N)$ where $s_i \cap s_j = \varnothing \ \forall \ i \neq j$. Let X denote the set of all feasible allocations.

Since an allocation assigns each agent a bundle, in order for an allocation to be feasible, it must be the case that each good is only assigned to a single agent. This is guaranteed by the condition $s_i \cap s_j = \varnothing \ \forall \ i \neq j$.

Let a *coalition* be a subset of the agent set A.

Definition 2. For a given allocation $(s_1, \ldots, s_N)$, the *winning coalition* is the set of agents who receive non-empty bundles. A *winning agent* is any member of the winning coalition.

We will use p_i to denote the price that agent i pays in the auction. A solution to the the combinatorial auction problem consists of an allocation, and a specification of how much each agent pays for her bundle. Therefore, an *outcome* is an allocation along with a price vector $(p_1, \ldots, p_N)$, where p_i denotes the price agent i pays for bundle s_i.

For the purposes of this paper, we assume that agents have quasi-linear utility functions. In other words, an agent's net gain is the value of the bundle obtained minus the price paid for that bundle. In general it is plausible for agents to have alternate utility functions, but here we limit consideration to quasi-linear utility. In addition, we assume that agents use an XOR bidding language. Agents submit valuations $(v_1, s_1), (v_2, s_2), \ldots, (v_n, s_n)$ where these bids mean that the agent is willing to receive at most one of $s_1, \ldots, s_n$ for her given valuations. This allows the agent to

specify exact values for the s_i without worrying about how her value changes if she also receives some other s_j. If the agent has value for bundle $s_i \cup s_j$, she can just submit a value for that bundle as well. These assumptions are made in many of the real world applications for which the ascending proxy auction is being considered.

Definition 3. Given an outcome with allocation $(s_1, \ldots, s_N)$ and price vector $(p_1, \ldots, p_N)$, the payoff of agent i is defined as $\pi_i = v_i(s_i) - p_i$. The payoff for the seller (agent 0) is defined as $\sum_{i \in A, i \neq 0} p_i$.

Example 1. Suppose we have three agents and two goods, A, B. $v_1(A) = 5, v_2(B) = 5, v_3(AB) = 20$. A possible outcome would be allocation $(A, B, \varnothing)$ with price vectors $(3, 3, 0)$. Agent 1 pays 3 for bundle A, and agent 2 pays 3 for bundle B, and agent 3 does not receive any bundle. As a result, the payoff for agent 1 is $v_1(A) - p_1 = 5 - 3 = 2$, the payoff for agent 2 is $v_2(B) - p_2 = 5 - 3 = 2$, and the payoff for agent 3 is 0. Not to forget about the seller, her payoff is the sum of the payments which is 6.

2.3 Properties

One of the desirable properties of an outcome is that it is in the *core*. Intuitively, an allocation is in the core if there is no coalition other than the winning coalition that challenges the outcome. A coalition may challenge the outcome if it could offer more revenue to the seller than the current outcome, and if all agents in the new coalition were willing to move to the new outcome.

Before defining the core, we formalize the notions of being better off in a new outcome and of coalitions that may challenge a given outcome.

Definition 4. An agent i *weakly-prefers* outcome J_1 to outcome J_2 if $\pi_i(J_1) \geq \pi_i(J_2)$.

This is a formalization of the notion that an agent is willing to move to a new outcome. If agent i weakly prefers outcome J_1 to J_2, then she is willing to move to outcome J_1.

Definition 5. Suppose the winning coalition for an outcome is coalition K, and the winning payments are $(p_1, \ldots, p_N)$. A coalition L *blocks* this outcome if there is some other outcome that generates more revenue to the auctioneer and is weakly-preferred by all agents in L.

To illustrate these new ideas, we revisit the Example 1 considered above. We have three agents and two goods. $v_1(A) = 5, v_2(B) = 5, v_3(AB) = 20$. Suppose that we have the same outcome J_1, which consists of allocation $(A, B, \varnothing)$ and price vector $(3, 3, 0)$. In this case, the payoff for agent 3 is 0. As a result, she weakly prefers any outcome in which she receives AB and pays at most 20. But then the outcome where agent 3 pays 7 for bundle AB blocks outcome J_1 since the seller receives more revenue and the winning agents (in this case just agent 3) weakly prefers this outcome. But this new outcome is again blocked. Under the new outcome, agents 1 and 2 receive payoff 0, so they are willing to pay up to 5 for bundles A and B respectively. Therefore, the outcome where agents 1 and 2 pay 4 for A and B respectively blocks since the seller receives more revenue (8 rather than 7), and the winning agents (agents 1 and 2) weakly prefer this outcome.

Intuitively, we see that a blocking coalition has a valid objection to the given outcome. If there is a blocking coalition, then that coalition can argue that they should win since they could provide an outcome with more revenue for the seller. Also key to the notion of blocking coalition is that this is a credible claim since all winning agents in the new outcome are not worse off and therefore willing to switch.

Having noted that blocking coalitions are undesirable, we define the notion of the core accordingly.

Definition 6. A *core* outcome is an outcome that is unblocked by any coalition.

Continuing with our Example 1, we recall that $v_1(A) = 5, v_2(B) = 5, v_3(AB) = 20$. We see that coalition $\{1, 2\}$ will block any outcome in which they receive no bundles that generates revenue less than 10 (since they could generate revenue of 10 by paying their values for each bundle). However, they cannot block an outcome that generates revenue greater than 10 since to generate that much revenue, either agent 1 or agent 2 would have to bid more then her value, giving her a negative payoff. Notice that a negative payoff is worse than the zero payoff she receives when she is not winning. Likewise, coalition $\{3\}$ will block any outcome that does not include agent 3 and generates less than 20 revenue. With these two observations, we see that the only outcomes that are unblocked are outcomes where agent 3 receives bundle AB and pays at least 10. These are the core outcomes of Example 1.

In order to further investigate the notion of the core, we introduce the concept of the *coalitional value function*. For a given coalition K, this value is simply the

maximum revenue that could possibly be generated by an outcome in which K is the winning coalition. We notice that to generate the maximum revenue, all agents in the winning coalition will pay their values for their bundles. As a result, we can view this as finding the feasible allocation that maximizes the sum of the value of received bundles.

Alternatively, we can think of the coalitional value for coalition K in value terms. For a given allocation $s_1, \ldots, s_N$, the *value* to coalition K of this allocation is $\sum_{i \in K} v_i(s_i)$. We can think of this as summing over the values of the goods allocated, disregarding payments. The coalitional value for K is therefore the largest value agents in K can receive in any allocation.

Definition 7. For a given coalition K, $w(K)$ is the *coalitional value function.* If $0 \notin K$, then $w(K) = 0$. Otherwise:

$$w(K) = \max_{(s_1,\ldots,s_N) \in X} \sum_{i \in K} v_i(s_i) \tag{2.1}$$

Notice that the coalitional value function is defined for every coalition. In Example 1, if we take coalition $\{1, 3\}$, the coalitional value is 20. Any feasible allocation will only allocate bundles to either agent 1 or 3 since A and AB overlap. Therefore, since $v_1(A) = 5, v_3(AB) = 20$, the value maximizing allocation gives AB to agent 3 and yields value 20.

Proposition 1. For a core outcome with winning coalition K and payments $(p_1, ..., p_N)$, it must be the case $\sum_{i \in K} p_i \geq w(L) \ \forall \ L \cap K = \varnothing$.

Proof: If the winning coalition is K, then it must be the case that for any $i \notin K$, $\pi_i = 0$ since agent i does not receive any bundles in the outcome. As a result, any such agent would be willing to pay her entire value for any bundle to block the outcome. Therefore, in order for an outcome to be unblocked, it must be that the revenue generated is greater than the maximum feasible revenue generated by all coalitions L that consist only of non-winning bidders. The maximum feasible revenue generated by these coalitions is just $w(L)$.

Definition 8. Let an outcome J be in the core. J is *bidder-Pareto-optimal* if there is no other core outcome that is weakly preferred by all agents to J.

Returning to our example, we recall that $v_1(A) = 5, v_2(B) = 5, v_3(AB) = 20$. We reasoned that the only core outcomes were outcomes where agent 3 wins and

pays at least 10 for bundle AB. As a result, two possible core outcomes are agent 3 paying 15 for bundle AB, and agent 3 paying 16 for bundle AB. In this case, agent 3 prefers paying 15 since she receives payoff 5 instead of 4. Agents 1 and 2 do not care since in both cases, they each receive zero payoff. However, even the outcome where agent 3 pays 15 is not bidder-Pareto-optimal since paying 14 is also a core outcome. Therefore, we see that the only bidder-Pareto-optimal core outcome is the outcome where agent 3 pays 10 for bundle AB. She pays just enough so that the coalition $\{1, 2\}$ cannot entice the seller with more revenue.

2.4 The Winner Determination Problem

Having defined some theoretical properties of combinatorial auctions, we turn to a central computational problem. Intimately related to combinatorial auctions is the winner determination problem. The winner determination problem is very intuitive and has many applications. The winner determination problem asks, given a set of prices agents are willing to pay for packages, what is the feasible allocation that maximizes the auctioneer's revenue? We have already seen a function whose value is a solution to the winner determination problem. In defining core outcomes, we defined the coalitional value function for a given coalition. The value of this function for a given coalition is simply the solution to the winner determination problem where agents are willing to pay their entire values for packages.

More formally, for a coalition of agents $\{1, \ldots, N\}$, with agent i willing to pay at most $p_i(s_i)$ for bundle s_i, we seek:

$$\max_{(s_1,\ldots,s_N)\in X} \sum_{i\in\{1,\ldots,N\}} p_i(s_i) \tag{2.2}$$

A straightforward way to solve the winner determination problem is to construct an integer linear program (IP). The variables in the program are indicator variables $x_{is} \in \{0, 1\}$ which tell us whether agent i is allocated bundle s in the revenue maximizing allocation. In order to be a feasible allocation, it must be the case that each good is only allocated once. Additionally, each agent can only be allocated a single bundle. These constraints along with revenue maximization are captured in the following

integer linear program:

$$\max \sum_{i \in A} \sum_{s \in S} p_i(s) x_{is} \tag{2.3}$$

$$\text{subject to } \sum_{i \in A} \sum_{s:g \in s} x_{is} \leq 1, \ \forall g \in G \tag{2.4}$$

$$\sum_{s \in S} x_{is} \leq 1, \ \forall i \in A \tag{2.5}$$

$$x_{is} \in \{0,1\}, \ \forall i \in A, \ \forall s \in S \tag{2.6}$$

Constraint 2.4 ensures that no good is allocated more than once, and Constraint 2.5 ensures that no agent receives more than one bundle. Notice that by querying the values of the x_{is} that maximize the objective function, we can retrieve the revenue maximizing allocation. For a given agent, if all the x_{is} are set to 0, then that agent receives the null bundle. For all other agents, it must be the case that only one of the $x_{is} = 1$ (Constraint 2.5), and that agent receives the unique bundle for which $x_{is} = 1$.

While the solution method presented here is simple and easy to implement, we notice that this formulation requires the solution of an NP-complete problem, namely integer linear programming. In fact, the winner determination problem itself has been shown to be NP-complete [18], though much work has been done finding more efficient algorithms for the winner determination problem [19, 20, 1, 9, 10].

Chapter 3

The VCG Mechanism

Having defined the combinatorial auction problem and the notion of a core outcome, we proceed by discussing one of the first and most famous mechanisms for determining outcomes. The Vickrey-Clarke-Groves (VCG) mechanism is an extension of the Vickrey auction to the combinatorial setting [1] [12]. For auctions of single goods, a Vickrey auction asks bidders to submit the maximum amount they would be willing to pay for the good. The outcome of the Vickrey auction is that the bidder who submitted the highest valuation wins the item, but pays the second highest valuation. The Vickrey auction was a novel invention because the mechanism makes bidding truthfully a *dominant strategy* [21]. In other words, bidders cannot do better by lying about their true valuation, regardless of the bids of other bidders. This is because a bidder's submission only determines *whether she wins* and not the price she pays. In terms of the core for single item settings, the Vickrey outcome is trivially in the core since the winning bidder pays the price of the second highest bid. In order to generate more revenue to the seller, other bidders would have to pay more than this amount, but this would give the non-winning bidders a negative payoff.

In the single good setting, the winning bidder pays the second highest price. We can think of this as the value of the allocation if the winning bidder were excluded. In extending the mechanism to the combinatorial setting, we make use of the same notions. The winning coalition is the coalition that maximizes the total value of the allocation (the allocation that yields the coalitional value of the entire set of agents).

[1] Vickrey auctions are also referred to as second-price auctions

We now have to determine the prices that the winning agents pay. Similar to the single goods setting, we look at what happens if we exclude agent i. If we exclude agent i, then their is some alternative allocation that maximizes value. The maximizing value is equal to the coalitional value function over the coalition of all agents except for agent i. However, because we include agent i, the coalition that does not include agent i receives value that is less than this maximal value. The price agent i pays is the difference between the maximal value to all other agents when excluding agent i and the actual value to all other agents when including agent i.

Ausubel and Milgrom describe this as the opportunity cost of the items that each agent wins [3]. Because agent i wins on a given bundle, all goods in the bundle can no longer be allocated to other agents. Agent i has to pay the amount of value that is lost by all other agents due to the removal of the goods in the bundle she wins.

More formally, we can describe this using our notion of coalitional value described above.

Definition 9. (VCG Outcome) Suppose we have a value maximizing allocation $(s_1^*, \ldots, s_N^*)$. Then any winning agent i pays:

$$w(A - \{i\}) - \sum_{j \in A, j \neq i} v_j(s_j^*) \tag{3.1}$$

The left side of the expression is the maximal value to all other agents if we exclude agent i, and the right side is the value received by all other agents in the value maximizing allocation that includes agent i.

Example 2. Calculating the VCG outcome. Suppose we have two goods and three agents. $v_1(AB) = 3, v_2(A) = 2, v_3(B) = 2$. The allocation that yields the most value is $(\varnothing, A, B)$ with value 4. To calculate the prices that agents 2 and 3 pay, we need to look at the value maximizing allocation without agents 2 and 3. If we remove agent 2, then the value maximizing allocation is $(AB, \varnothing, \varnothing)$. This yields value 3. However, coalition $\{1, 3\}$ receives value 2 when including agent 2. So the agent 2 pays price 1. Symmetrically, agent 3 pays price 1.

3.1 Merits of VCG

Before addressing merits of VCG, we introduce two concepts that describe mechanisms for determining outcomes for combinatorial auctions.

Definition 10. (Allocative Efficiency) A mechanism is *allocatively efficient* if the final allocation maximizes coalitional value.

Having defined the coalitional value function, we know that for any coalition K, $w(K)$ is the maximum value any allocation can yield. As a result, a mechanism is efficient if the value of the allocation in the outcome is equal to $w(A)$ where A is the coalition of all agents. Notice that allocative efficiency is independent of the payments. As long as the allocation in the outcome maximizes value, it is efficient regardless of the payments.

Definition 11. (Strategyproofness) A mechanism is *strategyproof* if truthful revelation of valuations [2] is a dominant strategy assuming that the bidder only submits these valuations to the mechanism.

In our example of the single good Vickrey auction, we saw that bidding truthfully was a dominant strategy. However, implicit in this assumption was that the bidders did not know each other's bids. If this were the case, then this opens the room for collusion. Strategyproofness formalizes the notion of truthful reporting being a dominant strategy when we have no knowledge of other agent's valuations.

It turns out that the VCG mechanism is efficient and strategyproof. In terms of efficiency, the VCG mechanism is efficient since by definition, the allocation it chooses is the value maximizing allocation. Also, similar to the single good auction, the VCG mechanism has the property that reporting values truthfully is a dominant strategy for all bidders. This and other properties of the VCG mechanism are proved in [3].

Additionally, the VCG mechanism can be reduced to solving $O(|A|)$ winner determination problems. Though the winner determination problem itself is NP-complete, the VCG mechanism is a single shot method which generally requires much fewer winner determination problems than the iterative auctions we will later consider. To see that the number of winner determination problems required is $O(|A|)$, recall that we need to solve winner determination to find the value maximizing allocation. Then, since the price a winning agent i pays is defined by the coalitional value function of $A - \{i\}$, we need to solve the winner determination problem for each of the winning agents.

[2] We assume that the bidder's values for bundles are independent of other events in the world.

3.2 Faults of VCG

Despite the various positive properties of the VCG mechanism, the mechanism has numerous faults. Ausubel and Milgrom present the following as weaknesses of the VCG mechanism [3]:

- low (or zero seller revenues)
- non-monotonicity of the seller's revenues in the set of bidders and the amounts bid
- vulnerability to collusion by a coalition of losing bidders
- vulnerability to the use of multiple identities by a single bidder

Ausubel and Milgrom provide a set of examples that illustrate each of these deficiencies. We include these examples here as they quickly show why the VCG mechanism may not be a practical choice.

Example 3. Low seller revenues. Suppose we have two goods and three agents again. We slightly modify the valuations. $v_1(AB) = 2, v_2(A) = 2, v_3(B) = 2$. The allocation that maximizes value is $(\varnothing, A, B)$ which yields value 4. We now solve for the prices paid again. If we remove agent 2, then the value maximizing allocation is either $(AB, \varnothing, \varnothing)$ or $(\varnothing, \varnothing, B)$. In both cases, the value received is 2. However, when we include agent 2, coalition $\{1, 3\}$ receives total value 2. Therefore, the price paid by agent 2 is 2 - 2 = 0. Symmetrically, agent 3 also pays 0. As seen in this example, the VCG mechanism may yield 0 revenue to the seller. Because of free-disposal, the seller would have been better off selling goods A and B together since all agents would bid 2 for AB and the seller would receive revenue 2.

Example 4. Non-monotonicity in the set of bidders and the amounts bid. We expect that when we increase the number of bidders, the seller's revenue should increase. Also, when the valuations submitted by bidders increase, we expect that revenue should increase. However, this is not the case with the VCG mechanism. Suppose that in the Example 3 we remove agent 3. Then we have $v_1(AB) = 2, v_2(A) = 2$. One value maximizing allocation is $(AB, \varnothing)$. If we remove agent 1, then agent 2 would receive A and value 2. However, agent 2 receives no bundles and value 0 when we include agent 1. Therefore, agent 1 pays 2 - 0 = 2 for AB when we remove bidder 3,

and the seller receives revenue 2 rather than 0. This example shows how the VCG mechanism may counter-intuitively generate more revenue to the seller when there are fewer bidders. Similarly, if bidder 3 reduces her bid to 1, we see that the value maximizing allocation is still $(\varnothing, A, B)$. Removing agent 2 still yields maximal value 2 for agents 1 and 3, but now agents 1 and 3 only receive value 1 from the VCG allocation. Therefore, agent 2 pays 2 - 1 = 1 instead of 0. The seller's revenue has increased even though agent 3 has decreased her bid.

Example 5. Vulnerability to collusion by a coalition of losing bidders. Consider the same example as above, with modified values. $v_1(AB) = 2, v_2(A) = 0.5, v_3(B) = 0.5$. In this scenario, agents 2 and 3 do not win since the value maximizing allocation is $(AB, \varnothing, \varnothing)$. However, as seen from the example above, if $v_2(A) = v_3(B) = 2$, then agents 2 and 3 win the bundle A and B and pay price 0. Thus, if agents 2 and 3 have true valuations $v_2(A) = 0.5, v_3(B) = 0.5$, they can collude and agree ahead of time to bid $v_2(A) = 2, v_3(B) = 2$ and arrive at the better outcome.

Notice that this does not contradict the strategyproofness of the VCG mechanism. Given that bidders are planning individually, truthful bidding is a dominant strategy. However, if bidders decide to collude, they can manipulate the outcome in their favor.

Example 6. Vulnerability to the use of multiple identities by a single bidder. Closely related to the above examples, suppose we have two agents. $v_1(AB) = 2, v_2(AB) = 1, v_2(A) = 0.5$. The value maximizing allocation results in agent 1 winning. However, if agent 2 can pretend to be two bidders, then agent 2 could replicate the outcome of Example 3. Instead of being a losing agent, by pretending to be another bidder and submitting valuations $v_2(A) = 2, v_3(B) = 2$, agent 2 can manipulate the outcome so that she wins both bundles and pays 0.

As seen, the VCG mechanism has many faults which make it problematic when used in practice. Indeed, the VCG mechanism appears to be a mostly theoretical tool that is not used in actual applications [3].

Ausubel and Milgrom argue that most of the faults arise because the VCG outcome does not necessarily lie in the core [3]. Using the example where the seller receives 0 revenue, $v_1(AB) = 2, v_2(A) = 2, v_3(B) = 2$, we see that coalition $\{1\}$ will block since agent 1 is willing to offer up to 2 for bundle AB while the VCG outcome generates 0 revenue for the seller.

For the VCG outcome to lie in the core, a stronger condition on the valuations submitted by bidders must be satisfied. This condition is known as the *agents are substitutes* (AAS) condition [16].

Another aspect of the VCG mechanism that is troublesome is costly preference elicitation. Preference elicitation refers to the process of calculating preferences and submitting them in the auction. One of the issues that arises in preference elicitation setting is how bidders can communicate their preferences to the auctioneer. In the VCG mechanism, bidders need to submit their entire valuation functions. Since there are an exponential number of possible bundles on which to bid, this process can be intractable. Another reason for costly preference elicitation is related to the bidder. Many times, determining a bidder's absolute preferences may be very difficult. In the motivating example of airport slots, airlines may need to solve complicated optimization problems for routing and scheduling before knowing how much they value a certain set of slots. Because of these faults with the straightforward VCG mechanism, there have been attempts to find mechanisms for the combinatorial auction problem that do not suffer from these deficiencies.

Chapter 4

The Ascending Proxy Auction

In this chapter, we develop iterative combinatorial auctions as a lead-in to the defintion of the ascending proxy auction. Iterative combinatorial auctions were developed to address costly preference elicitation, and this work led to the development of the ascending proxy auction which focuses on producing core outcomes even when AAS is not satisfied. We then define the ascending proxy auction and cite some of the provable properties of its outcomes. Lastly, we discuss related work in the area of acceleration of the ascending proxy auction.o Therefore, in describing the ascending proxy auction, we first describe iterative combinatorial auctions as background to the ascending proxy auction.

4.1 Iterative Combinatorial Auctions

As stated previously, one of the problems with the VCG mechanism is costly preference elicitation. Parkes and Ungar developed iterative combinatorial auctions to help alleviate this problem of costly preference elicitation.

In iterative combinatorial auctions, the auction proceeds in rounds. In each round, the auctioneer declares a set of ask prices for bundles. Bidders then submit new bids that are at least the ask prices for the bundles. The auctioneer then performs some computation (usually the winner determination problem) and declares a new set of winners and new ask prices. The auction proceeds until the ask prices are such that no new bids are submitted.

In this way, instead of determining their entire valuations at the beginning of the auction, bidders can focus only on the bundles whose prices are changing. Additionally, instead of being locked into their valuation functions at the beginning of the auction, bidders can change their bids or focus based on feedback from the results of the rounds in the auction. In fact, bidders may not even need exact values for bundles in iterative combinatorial auctions [15, 17].

Other than addressing the preference elicitation issue, iterative combinatorial auctions also offer other advantages. One way to view iterative combinatorial auctions is as a distribution of the computational burden across bidders. Mechanisms like the VCG mechanism require information to be gathered at a centralized source at a single point in time. This centralized source then takes this information and performs a lot of computation to determine the answer. Iterative combinatorial auctions spread the computation burden across bidders. The auctioneer simply has to update the allocation and ask prices, while bidders can take on the burden of resubmitting and recalculating bids [15, 17].

Another advantage of iterative combinatorial auction is *transparency*. Because there are many rounds taking place, bidders have an idea of how the auction progresses, and how we arrive at the final allocation. This is not the case with the VCG mechanism, which is more like a black box which outputs the answer. Because of transparency, bidders can verify that the outcome is fair and correct. A third potential advantage is that if goods are correlated, then the interaction between bidders in the various rounds will allow for better outcomes [14, 15, 17].

4.2 iBundle

The particular iterative combinatorial auctions we will consider are *i*Bundle and its two variations. We discuss *i*Bundle here because it is closely tied to the ascending proxy auction. There are two versions of *i*Bundle, *i*Bundle(2) and *i*Bundle(3). They are very similar, but differ in that *i*Bundle(2) has *anonymous* prices while *i*Bundle(3) has *non-anonymous* prices. By anonymous prices, we mean that the ask prices for bundles are the same regardless of the bidder. For non-anonymous prices, the ask prices for a specific bundle can change depending on the bidder.

4.2.1 *i*Bundle(2)

*i*Bundle(2) consists of a number of rounds, indexed by $t \geq 1$. At the conclusion of each round, the auctioneer declares a provisional allocation. The provisional payments are assumed to be the current winning bids. At the start of the next round, the auctioneer declares new ask prices for each bundle. The bidders then resubmit their bids. A bid is *competitive* if it is at least the ask price, and a bidder is *competitive* as long as she submits a single competitive bid. The sole exception is for bidders who were winning in the provisional allocation. These bidders still need to resubmit their previous bids, but they do not need to worry about exceeding the ask price. Also, bidders can submit a "last and final" bid which is ϵ below the ask price. However, this means they cannot submit any more bids for the bundle in question.

Once bids are submitted, the auctioneer solves the winner determination problem to determine the provisional allocation. For the next round, the ask price increases by ϵ for every bundle on which a losing bidder submitted a bid. All other ask prices stay the same. The iterative process continues until all competitive bidders are in the provisional allocation (all non-winning bidders have not submitted any competitive bids). At this point, the provisional allocation becomes the final allocation, and the payments are the winning bids.

4.2.2 *i*Bundle(3)

*i*Bundle(3) is similar to *i*Bundle(2), except that the ask prices are for a (package, bidder) pair instead of simply a package. Each bundle can have a different ask price for each bidder. The auction proceeds in the same manner as *i*Bundle(2), except in updating ask prices. In *i*Bundle(3), the ask prices for each losing bidder increases by ϵ. Notice that this opens the possibility for bidder-specific ask prices. As a simple example, suppose bidders 1 and 2 both submit a bid on bundle AB, but bidder 1 is chosen to be in the provisional allocation. Then bidder 2 experiences a price increase on bundle AB but bidder 1 does not.

4.3 Ascending Proxy Auction

Notice that the iBundle auctions focus on the way the auction progresses but does not constrain what bidders can do in the course of the auction. Because of the numerous rounds in the auction, bidders can strategize and change their bids. They may not even know their valuations at the start of the auction. Interestingly, Ausubel and Milgrom observed that when iBundle is paired with specific bidding strategies, the outcomes produced have provably nice properties. This observation led to the creation of the ascending proxy auction, which can be thought of as iBundle(3) coupled with a specific bidding strategy. Working towards a definition of the ascending proxy auction, we now examine some simple bidding strategies for bidders in iBundle(2) and iBundle(3).

In *straightforward bidding*, the bidders have fixed valuations for the bundles in the auction. Given their valuations and ask prices, bidders can determine which bundle gives the highest payoff. Bidders submit bids at the ask prices for bundles that give the highest payoff. A bidder's *demand* set consists of all the bundles that give the highest payoff at the current ask prices.

Definition 12. (Demand Set) Let $p_i(s)$ denote the ask price bidder i faces for bundle s. A bidder's demand set is defined as:

$$\mathrm{DS}(i) = \{s \in S : v_i(s) - p_i(s) = \max_{s' \in S}(v_i(s') - p_i(s')\} \tag{4.1}$$

Similarly, we can relax this to *ϵ-straightforward bidding*. Instead of submitting bids only for the bundles with highest payoff, bidders submit bids for all bundles that are within ϵ of the highest payoff.

Definition 13. (Epsilon Demand Set) Let $p_i(s)$ denote the ask price bidder i faces for bundle s, A bidder's ϵ-demand set is defined as:

$$\epsilon\mathrm{DS}(i) = \{s \in S : v_i(s) - p_i(s) + \epsilon \geq \max_{s' \in S}(v_i(s') - p_i(s')\} \tag{4.2}$$

4.4 Ascending Proxy Auction (APA) Definition

Having defined straightforward bidding, we can now define the ascending proxy auction. Though not described in terms of ask prices, the ascending proxy auction is

equivalent to iBundle(3) with straightforward bidding. Since the bids for a given set of ask prices are now deterministic, we no longer need an active bidder to resubmit bids. Instead, we can have a *proxy agent* stand in for each bidder, as long as the proxy agent knows the bidder's valuations at the beginning of the auction. The ascending proxy auction can be thought of as iBundle(3) with ϵ-straightforward bidding implemented by proxy agents.

More formally, we can describe the ascending proxy auction as follows:

- Let each round be indexed by $t \geq 1$.
- Before the auction begins, each bidders submits a set of (value, bundle) pairs to the auctioneer. We can think of this as defining the valuation function for each bidder. Bidders cannot change their valuation functions once submitted. After submitting their bids, each bidder will be represented by a proxy agent.
- Let ϵ be a pre-defined bid increment.
- Let $b_i^t(s)$ denote the highest bid agent i has submitted for bundle s in round t.
- Let $\pi_i^t(s) = v_i(s) - b_i(s)$ denote the payoff agent i receives for bundle s in round t.
- Let the winning coalition in round t be the coalition that is winning in the provisional allocation. A winning agent in round t is any member of the winning coalition in round t.
- In round 0, the provisional allocation allocates the null set to every agent (no agent is winning).
- In round t, each agent who receives the null set in the provisional allocation at round t calculates the bundle with highest payoff s^*. Let the payoff of s^* be π^{t*}. Then the agent increases her bid by ϵ on all packages with payoff within ϵ of π^{t*}. (Note: Some definitions only increase bids on the bundles with highest payoff. This does not affect the properties of the outcome, and we will assume our definition).
- In round t, agents who receive non-null bundles in the provisional allocation do not change their bids.

- Given the set of bids, the auctioneer solves the winner determination problem and announces a new provisional allocation.
- This process repeats until we reach a round when no new bids are submitted.

4.5 Properties of the Ascending Proxy Outcome

The most important property of the ascending proxy outcome is that it is in the core. We saw earlier the numerous problems with the VCG mechanism, many of which were consequences of non-core outcomes.

Theorem 1. *[2] The outcome determined by the ascending proxy auction is in the core with respect to reported preferences.*

We will not give a formal proof here, but the intuition is that if there were a blocking coalition, then it must be the case that some losing agent has not finished submitting her bids. A blocking coalition suggests there is an alternate outcome that generates more revenue to the seller yet is weakly-preferred by all agents. Since agents submit bids only for the maximal payoff bundles, the winning agents will not be willing to pay more for any other bundles. Therefore, in order to generate more revenue, the alternate outcome must have the losing agents pay more than their current bids. But this contradicts termination of the ascending proxy because all losing agents should be bidding their valuations.

Another nice property of the ascending proxy outcome is that given certain conditions, the outcome is equivalent to the VCG outcome. While the VCG outcome in general can be very poor, we cited that when the AAS condition is satisfied the outcome will be in the core. With a slightly stronger condition on the coalitional value function, the *buyer-submodular* (BSM) condition, the ascending proxy outcome exactly replicates the VCG outcome (and is in the core since BSM is a stronger condition than AAS).

Theorem 2. *[2] The coalitional value function w is buyer-submodular iff for every coalition C that includes the seller, the (restricted) VCG payoff vector is in the core.*

Therefore, the ascending proxy outcome can be thought of as an improvement on the VCG mechanism. When the VCG outcome is not in the core (AAS is not

satisfied), the APA outcome is in the core. When the VCG outcome is in the core for every coalition of agents (not just the coalition of all agents), the APA outcome is equivalent to the VCG outcome. The only gap that exists is when we have AAS but not BSM. Then both the VCG and APA outcomes are in the core, but the APA outcome is not necessarily the same as the VCG outcome. While the APA outcome always produces a core outcome, it has the added property that under certain conditions when the VCG outcome is also a good outcome, APA matches VCG.

4.6 Potential Problems with the Ascending Proxy Mechanism

Having provided a description of the ascending proxy mechanism, there is a straightforward *direct implementation* of the mechanism. We can program the steps of the algorithm as described, and exactly simulate what happens in each round. We consider this approach the Pure Proxy approach. Just as the algorithm describes, bidders submit their valuations before hand, and our computer program simulates proxy agents submitting bids on their behalf. When all new bids are submitted, the winner determination problem is solved, a provisional allocation is declared, and new bids are submitted.

One issue with the Pure Proxy direct implementation is that the final result of the auction is dependent on the bid increment ϵ. If we choose a large bid increment, then there will be fewer rounds, but we may cause agents to stop bidding prematurely. If we choose a smaller bid increment, we can be more accurate but this causes prices to increase at a slower rate and requires more rounds. Thus, there is a tradeoff between speed and accuracy when choosing ϵ (see Example 8 in Chapter 5.8).

Another potential problem with the direct implementation is that each round necessitates the solving of the winner determination problem which is NP-complete. Small values of ϵ would further worsen the situation as more rounds are needed to reach the point when no new bids are submitted.

Because of the various drawbacks of direct implementation of the ascending proxy mechanism, there has been an attempt to develop faster, accelerated algorithms that replicate the ascending proxy results. These attempts can be separated into two

categories. There are algorithms which focus on producing outcomes with the same properties as APA, and there are algorithms which reproduce the exact results of APA.

4.7 Approximate Accleration of Ascending Proxy

First, we describe related work which looks to find more efficient algorithms that replicate the properties of the ascending proxy outcome, but not the exact outcome. Two major efforts by Hoffman et. al. and Day et. al. have appeared in this area. Hoffman et. al. modify the start conditions of the ascending proxy auction and iteratively run the auction with different bid increments. Day et. al. focus on generating outcomes with the specified properties and does not resemble or utilize the ascending proxy auction.

4.7.1 Hoffman et. al.

Safe Start One of the proposed algorithms involves starting at non-zero prices. While the direct implementation of the ascending proxy initializes the prices on all bundles to 0, the concept of safe start is to start at a conservative estimate for the prices on bundles. In other words, instead of forcing the proxy to start at a bid of 0 and incrementally increase by ϵ, we can start at a conservative bid and incrementally increase by ϵ from there.

Because the ascending proxy outcome is in the core, we are able to determine a lower bound for the winning bids. In order to be in the core, it must be the case that the winning agents and bundles are one of the solutions to the winner determination problem where the agents' bids are simply their valuations [11]. Therefore, we can obtain by solving the winner determination problem, the winning and non-winning agents along with the bundles that winning agents receive.

Having determined the winning agents and allocated bundles, Hoffman et. al. offer an observation for how to calculate a lower bound for the winning bids.

Proposition 2. If agent i is a winning agent who wins on bundle s, then $p_i(s) \geq v_j(s) \; \forall$ non-winning j.

Proof: Suppose not. Then there is some non-winning agent j who has value greater than the price agent i is paying for bundle s. But then the coalition which swaps i and j would block since j is willing to pay more for bundle s. We know this is feasible since j was a non-winning agent and therefore was not already winning on some other bundle. This contradicts the fact that the winning allocation is in the core.

For non-winning agents, it must be the case that their final bids on a given bundle equal their valuations on that bundle. The ascending proxy only ends when all bids have been submitted. If an agent is not winning but her bids are not equal to her valuations, she would still have unsubmitted bids and the ascending proxy would not terminate.

To summarize, we can solve the winner determination problem and then assign lower bounds to agents' bids by following:

- If agent i is winning and receives bundle s, then $p_i^*(s)$ is at least the maximal value any non-winning agent has for bundle s. For all other bundles s' that agent i bids on, set $p_i^*(s') = v_i(s) - p_i^*(s)$ (the payoff agent i receives on s when paying the conservative price).
- If agent j is not in the winning allocation, then her bids on any given bundle s must be equal to her value for s.

Given these observations, instead of starting at bids of zero for each agent, we start at these conservative estimates and run the ascending proxy from this initial point. Because the bids do not have to incrementally rise from 0, this algorithm goes through fewer rounds than the direct ascending proxy.

Incremental Scaling Another method proposed by Hoffman et. al. is incremental scaling. This approach arises from the observation that it may not necessarily be optimal to use the same bid increment in every round of ascending proxy. Instead, we can use a large bid increment in early rounds when bids are small and not close to agent's valuations, and use progressively smaller bid increments as we get closer to the termination point of the auction.

Based on this observation that varying the bid increment may be helpful, Hoffman et. al. propose that we run the ascending proxy with a very large bid increment at

first. Once we obtain the result, we can determine conservative prices for the final allocation. Having determined these conservative prices, we can start from this point and use a smaller increment to provide more accuracy. We continue this iterative process until we reach an acceptably small bid increment.

As with the safe start algorithm, the incremental scaling algorithm also depends on determining an estimated lower bound for the final prices agents pay. Suppose the bid increment is ϵ. We run ascending proxy and obtain a winning allocation. For non-winning agents, it must be the case that all bids submitted are within ϵ of their value. Otherwise, the ascending proxy would end with unsubmitted bids remaining. For the winning agents, it is possible that a large ϵ causes over-bidding to occur. As an estimated lower bound for the APA payments of winning agents, we scale back the winning bids by ϵ. Therefore, the non-winning agents do not change their prices while the winning agents decrease their winning bids by ϵ. We use these prices as a starting point and run ascending proxy with a smaller bid increment. This iterative process continues until we have reached a desirably small bid increment. We notice that in scaling back the winning bids by ϵ, we do not have a guarantee as with safe start that these prices are a lower bound. Hoffman et. al. entertain this possibility by proposing a *corrective rollback* method that deals with this case [11].

Safe Start with Incremental Scaling Hoffman et. al. ultimately combine their two insights into one algorithm. They apply safe start to determine a conservative estimate for the initial prices, and then they use incremental scaling when running ascending proxy from these initial prices.

Properties of Outcome Hoffmann et. al. show that the outcomes of their algorithm have the same properties as the outcomes of the ascending proxy auction. In addition, their outcomes mimic the VCG outcomes when AAS holds but BSM does not hold. Therefore, their outcomes mimic VCG when VCG is in the core, and produces outcomes that are in the core when VCG is not in the core. Hoffman et. al. also provide a theoretical bound on the number of integer linear programs to be solved. They find that the number of IPs they solve is polynomial in the digits of accuracy desired and the number of packages in the optimal allocation [11].

While the mechanism offered by Hoffman et. al. utilizes the ascending proxy as

a basis for their algorithm, the use of multiple bid increments and starting at the non-zero prices does not exactly mimic the progression and final payments of the direct implementation of the ascending proxy mechanism.

Results Hoffman et. al. provide results running their algorithm on examples from literature and other small, representative examples. They show that the safe start and incremental scaling methods significantly decrease the number of rounds required for termination. We will consider these test cases example further in Chapter 7 which describes experimental results of our algorithm and compares these with previous findings.

4.7.2 Day et. al.

Day et. al. propose another method for computing outcomes to the combinatorial auction problem. Instead of trying to mimic the specifics of the ascending proxy mechanism, Day et. al. focus on finding outcomes with the same properties as the ascending proxy outcome. Their work relates to the ascending proxy only in the types of outcomes they find, not in the way they arrive at these outcomes.

The two properties that Day et. al. focus on are core outcomes and bidder-Pareto-optimal outcomes. Their approach is to start at a provisional allocation equal to the VCG outcome and then ask whether this allocation is in the core. A straightforward definition of the core contains an exponential number of constraints (since we need to ensure that there are no blocked coalitions and there are 2^n possible blocking coalitions). However, the Day approach uses constraint generation to avoid enumerating this exponential number of constraints. They construct an integer linear program which given a provisional allocation, finds the most blocking coalition if there are any. However, this is more complicated than simply finding coalitions whose coalitional value exceeds the current provisional allocation. Any agent i who is winning in the provisional allocation will not be willing to join another coalition if her payoff decreases. Therefore, if an agent's current payoff is $\pi_i^t(s) = v_i(s) - p_i^t(s)$, she will only be willing to pay $v_i(s') - \pi_i^t(s)$ for any other bundle s'. Day et. al. adjust their IP to correctly account for these dynamics. The resulting IP has form similar to the winner determination IP. Let $(s_1, ..., s_N)$ denote the provisional allocation, $(p_1, ..., p_N)$ denote agent payments, and W denote the set of winning agents. The following IP finds the

most-blocking coalition, if there are any:

$$\max \sum_{i \in A} \sum_{s \in S} v_i(s) \cdot x_{is} - \sum_{i \in W} (v_i(s_j) - p_i^t(s_j)) \cdot \gamma_j \tag{4.3}$$

$$\text{subject to} \quad \sum_{g \in s} \sum_{i \in A} x_{is} \leq 1, \ \forall g \in G \tag{4.4}$$

$$\sum_{s \in S} x_{is} \leq 1, \ \forall i \in A \tag{4.5}$$

$$\sum_{s \in S} x_{is} \leq \gamma_j, \ \forall i \in A \tag{4.6}$$

$$x_{is} \in \{0, 1\}, \ \forall s \in S, \ \forall i \in A \tag{4.7}$$

$$\gamma_j \in \{0, 1\} \tag{4.8}$$

Just like the winner determination IP, constraints 4.4 and 4.5 ensure that each good is only allocated once and each agent only receives a single bundle. The extra term in the objective function represents the previously discussed notion that agents who are currently winning need to receive at least their current payoff. Therefore, instead of adding their entire value, we need to subtract their current payoff $v_i(s_i) - p_i^t(s_i)$. The γ_j ensure that we only subtract off for currently winning agents which are also winning in the outcome found by the IP.

After finding the most blocking coalition if one exists, Day et. al. define a new provisional allocation. The winning agents and bundles in the provisional allocation are determined by the IP in the same way the winner determination IP outputs winning agents and bundles. The only remaining issue is the payment vector. Using a linear program, payments are chosen to be the minimal payments that generate more revenue than previous provisional outcomes and are weakly preferred by all previously winning agents.

The iterative process of generating the most blocking coalition and new provisional allocations and payments stops when there are no blocking coalitions. The provisional allocation and provisional payment vector becomes the final outcome. Day et. al. prove that this outcome is always bidder-Pareto-optimal [8].

Results Like Hoffman et. al., Day et. al. provide the results of running their algorithm on a small example from the literature. They compare the number of IPs and mixed integer linear programs (MIPs) solved by various algorithms, and show

that their algorithm requires less work than other accelerated algorithms.

4.8 Exact Acceleration of Ascending Proxy

Other related work has focused on finding accelerated algorithms that exactly replicating the ascending proxy auction. This work has been pioneered by Wurman et. al. and Parkes, who suggest that iterative combinatorial auctions will proceed in stages. Within each stage, we can compute what prices will be at the end of the stage without directly implementing the rounds of the ascending proxy auction.

4.8.1 Wurman et. al.

Price Trajectories Wurman's work actually provides an accelerated implementation of iBundle(2) with straightforward bidding, which provides some insights into our algorithm for iBundle(3) with straightforward bidding and APA. We will assume straightforward bidding when we refer to iBundle(2) and iBundle(3) in the rest of this section. Wurman makes the observation that the auction proceeds in stage. Stages are separated by two different types of events.

The first event is when an agent begins to submit bids on a new bundle or drops out of the auction. An agent will start bidding on a new bundle when the agent's payoff on her current bundles fall below her value for a bundle on which she is not currently bidding. An agent drops out of the auction when she is bidding her values. The second way stages are separated is when a new coalition emerges as a potential winner. During both iBundle(2) and iBundle(3), the coalitions that win during a stage will be the coalitions that generate the most revenue during the stage. However, these coalitions are declared the winner in the provisional allocation some of the time. As a result, when these coalitions are winning, their member agents will not increase their bids on the winning bundles. Coalitions that win during a stage therefore might increase revenue at a slower rate due to the fact that they are winning [1]. Coalitions that never win may eventually catch up. When these *collisions* occur, we need to begin a new stage [22].

[1]Note that this is not always true. This just notes the possibility for coalitions that do not win to increase revenue at a higher rate and gives some intuition as to why this may occur.

After splitting iBundle(2) into a sequence of stages, Wurman calculates the *trajectory* of ask prices within each stage. The primary insight here is that each agent has one unit of *attention* she can allocate. This attention is divided between the bundles in her demand set, and the trajectory of the ask price of a bundle will be the sum of the attention being paid to that bundle [22].

Wurman also notices that within a stage, the coalition that is declared the winner alternates between a given set of coalitions that are actively competing. As with the allocation of attention, each of these coalitions will win for a certain fraction of time. From these fractions, we can determine how often a single agent is winning (an agent is winning whenever the winning coalition contains that agent) [22].

By combining the notion of attention with the observation about winning coalitions, Wurman formulates a mixed integer linear program that solves for the trajectory of bundle prices within a stage. Once these bundle trajectories are calculated, Wurman can solve for the points at which coalitions collide and agents begin bidding on new bundles. Thus, having determined the trajectory of prices within a stage, Wurman calculates when to throw out these trajectories and begin the next stage [22].

Results Wurman argues that his trajectory algorithm mimics the outcome of iBundle(2) with straightforward bidding. He also steps through a small example with his proposed algorithm.

Though Wurman's algorithm is tailored to iBundle(2) which has anonymous prices, many of the ideas carry over to iBundle(3) which has non-anonymous prices. We indeed find parallels to Wurman's concepts in our independently developed algorithm for iBundle(3)/APA. Specifically, the notions of stages and coalitional fractions reappear in the iBundle(3)/APA setting.

4.8.2 Parkes

Unlike Wurman's algorithm which focuses on iBundle(2) with straightforward bidding, Parkes began work on accelerated implementations of iBundle(3) with straightforward bidding (APA). Again we assume straightforward bidding when referring to iBundle(3) in the rest of the section. Parkes also observed that iBundle(3) could be

split into stages, with stages separated by the entrance of new bundles into an agent's demand set. Having defined his notion of stages, Parkes proposed using mixed integer linear programming techniques to describe the interaction between coalitions within each stage [16].

Another insight that Parkes offered was that of *interesting coalitions* and *interesting bundles*. Parkes noticed that during a run of the auction, only a small set of coalitions is actually ever declared to be the winner of a provisional allocation. Parkes proposed that we limit our attention to these interesting coalitions, and he offered an algorithm to calculate these coalitions before beginning the accelerated implementation.

Parkes's work served as the foundation of our accelerated algorithm and this senior thesis. By splitting the auction into stages, we were indeed able to formulate mixed integer linear programs to determine the behavior within each stage. Moreover, the notion of interesting coalitions proved to be very useful in dealing with the computational complexity of the accelerated implementation.

Chapter 5

Our Accelerated Algorithm

In this chapter, we offer a description of our new accelerated algorithm. As opposed to the Pure Proxy approach, our algorithm exactly replicates the progression of prices in APA, but does not directly implement the rules of APA. Instead, we exploit built-in patterns in APA that result from the rules of the auction. Using the previous insight of Parkes and Wurman, our algorithm makes use of the notion that the ascending proxy auction is a sequence of stages, with predictable behavior within each stage.

The chief advantages of our algorithm over Pure Proxy are:

- Outcomes and runtime are independent of the bid increment. Our outcome represents the outcome of Pure Proxy for an infinitesimally small bid increment.
- Savings in runtime due to solving fewer winner determination type problems.

Our chief advantages over Parkes's algorithm are:

- Defining and calculating coalitional fractions and agent fractions.
- Formalization of stage definition.
- Efficient calculation of interesting coalitions (Chapter 6).

Our chief advantages over Wurman's algorithm are:

- Extension to *i*Bundle(3) with straightforward bidding and APA.

	8	7	10	4	4	2
Iteration (t)	$p_1^t(AB)$	$p_2^t(CD)$	$p_3^t(CD)$	$p_4^t(BD)$	$p_5^t(AC)$	$p_5^t(C)$
1	(1)	(1)	1	1	1	0
2	1	1	2	(2)	(2)	0
3	(2)	2	(3)	2	2	0
4	2	3	3	(3)	(3)	1
5	(3)	(4)	4	3	3	1
6	3	4	5	(4)	(4)	2
7	(4)	5	(6)	4	4	2
8	(4)	(6)	6	4	4	2
9	(4)	6	(7)	4	4	2
10	(4)	(7)	7	4	4	2
11	(4)	7	(8)	4	4	2

Table 5.1: The first row contains the agents' values for the bundles in the respective columns. The parentheses around the bid value signifies that the given bid is winning. For example, in iteration 1, the winning coalition is $\{1, 2\}$. The breaks in the table represent changes in agents' demand sets. The first break is when agent 5 starts bidding on C, and the second break is when agents 4 and 5 bid their values of 4. The end of the chart occurs when agent 2 bids her value of 7.

- Efficient calculation of the length of each stage (Chapter 6).
- Efficient calculation of interesting coalitions (Chapter 6).
- Potential alternative to a mixed integer formulation (Chapter 6).

Experimental results can be found in Chapter 7.

5.1 A Motivating Example

Before proceeding to describe our algorithm, we provide a simple example and run through APA on this example. This will provide a concrete basis for the description of our algorithm.

Suppose we have five agents and four goods, $\{A, B, C, D\}$. $v_1(AB) = 8, v_2(CD) = 7, v_3(CD) = 10, v_4(BD) = 4, v_5(AC) = 4, v_5(BC) = 2$. The results of running APA with a bid increment of 1 can be summarized in Table 5.1.

Of note in the example is that agents that were previously winning do not increase *any* of their current bids. Pictorally, any agent who has a bid surrounded by a

parenthesis will not change any of her bids in the next round. For example, in iteration 4, the specific bid of agent 5 on bundle C is not winning, but since agent 5 is winning in the round on AC, she does not change her bid on bundle C. All non-winning agents increase all their bids by the bid increment of 1.

Also notice that agent 5 does not begin bidding on C until her payoff on C equals her payoff for AC. Since she has higher value for AC, AC starts out with higher payoff. Only when her bid for AC becomes 3 does she submit a bid of 1 for bundle C (at this point, $\pi_5^t(AC) = v_5(AC) - p_5^t(AC) = 4 - 3 = 1$ and $\pi_5^t(C) = v_5(C) - p_5^t(C) = 2 - 1 = 1$).

Also, the winning bids are always chosen to maximize the seller's revenue. We enforce feasibility by making sure that the bundles that win do not overlap (for example, agents 1 and 4 never win in the same iteration because their bundles both contain goods B). We will keep referring back to this example when discussing key concepts in our algorithm.

5.2 Preliminaries

Let an *active* bundle for agent i be a bundle on which the agent is actively bidding ($p_i(b) \neq 0$ and $p_i^t(b) \neq v_i(b)$). Recall that an agent's *demand set* consists of her active bundles. An agent is *active* if her demand set is not the null set.

Proposition 3. In the ascending proxy auction, the payoff an agent receives from each of her active bundles in a round is constant with a factor of the bid increment ϵ.

Proof: An agent only starts bidding on a new package s when the value of s is within ϵ of the maximal payoff on her current bundles. Therefore, when an agent first starts bidding on s, the payoff on s is within ϵ of her maximal payoff. Additionally, whenever the agent increases her bids by ϵ, she does so on all packages that have payoff within ϵ of her maximal payoff. By induction, the agent's payoffs for all of her active bundles must be within ϵ of her maximal payoff.

Because of Proposition 3, we can define the payoff of an agent to be independent of the current bundle within a factor of ϵ. For our accelerated algorithm, we assume infinitesimally small ϵ, so the payoffs for an agent on her active bundles will always be equal. Thus, we can denote the payoff of an agent without referring to specific

bundles by simply π_i^t, where t indexes the round.

Corollary 4. When an agent stops bidding on a single package, she stops bidding on all packages.

Proof: When the payoff for a single package becomes 0, it must be the case that the payoff for all packages becomes 0 because of Proposition 3.

If we look at Example 5.1, agent 5 is the only agent who bids on more than 1 bundle. However, when she is submitting bids on both bundles, it is the case that her payoff is the same on both bundles. She only begins to submit bids for C in round 4. At this point, her payoff for AC is 4 - 3 = 1, and her payoff for C is 2 -1 = 1. Similarly, as a demonstration of Corollary 4, agent 5 stops bidding on both of her bundles at the same time after round 7.

5.3 Interesting Coalitions and Interesting Bundles

Definition 14. A coalition C_i is *interesting* if at some point during the auction, C_i is declared the winner.

In our motivating example, we can find the interesting coalitions by looking across each row and finding the coalitions that correspond the winning bundles. In this case, there are three interesting coalitions, namely $\{1,2\}, \{1,3\}$ and $\{4,5\}$.

Definition 15. Let $B_i(j)$ denote the bundle agent j is allocated when coalition C_i wins at a given time. Then the *revenue generated* by coalition C_i with allocation B_i is defined as:

$$r^t(C_i, B_i) = \sum_{j \in C_i} p_j^t(B_i(j)) \tag{5.1}$$

Returning to our motivating example, in round 5, the revenue generated by coalition $\{1,2\}$ with allocation $\{AB, CD\}$ is 3 + 4 = 7. Similarly, in the same round, the revenue generated by coalition $\{4,5\}$ with allocation $\{BD, AC\}$ is 3 + 3 = 6. Alternatively, the revenue generated by coalition $\{4,5\}$ with allocation $\{BD, C\}$ is 3 + 1 = 4.

Proposition 5. If C_i is interesting, then C_i always wins on the same bundles. In other words, if C_i is winning, then the agents in C_i are always allocated the same packages.

Proof: Consider the first time that C_i is declared to be the winning coalition. Let $B_i(j)$ denote the package allocated to agent j when C_i is first declared the winner. Suppose at some later time, the agents are allocated some other packages, $B_i' \neq B_i$. By definition, this means B_i' generates more revenue than B_i. However, since B_i was allocated the first time C_i won, it must have been the case that at that time, the revenue generated by B_i must have been greater than the revenue generated by B_i'. Since agents increase their bids equally on all packages, B_i must always generate more revenue than B_i'. Thus, it cannot be the case that B_i' is the winning allocation at some later time.

As a concrete example, in our motivating example, the coalition $\{4, 5\}$ always wins on bundle BD and AC. Even though agent 5 submits bids for bundle C alone, coalition $\{4, 5\}$ never wins on bundles BD and C. The logic for this is that bundle C yield less revenue since agent 5 starts bidding on it at a later iteration. Since agent 5 increases her bids on all bundles whenever she increases her bids, bundle C will always yield less revenue than AC, and hence $\{4, 5\}$ can only win on bundles BD and AC.

Because C_i always wins on the same bundles, we can define $r^t(C_i)$ using the bundles B_i associated with C_i whenever C_i is winning. The revenue generated by coalition C_i is now only a function of the coalition C_i and the current round t:

$$r^t(C_i) = \sum_{j \in C_i} p_j^t(B_i(j)) \tag{5.2}$$

Proposition 5 shows that whenever C_i is winning, the agents in C_i win on the same packages.

Definition 16. We therefore define the *interesting bundles* associated with interesting coalition C_i to be the unique mapping between agents in C_i and bundles defined by the allocation whenever C_i is winning.

In the motivating example, the interesting bundles for coalitions $\{1, 2\}$ and $\{1, 3\}$ are $\{AB, CD\}$, and the interesting bundles for coalition $\{4, 5\}$ are $\{BD, CA\}$.

5.4 Overview

If we look at Example 5.1, we see some very interesting behavior in the ascending proxy. Before agents 4 and 5 stop bidding, coalitions $\{1, 2\}, \{1, 3\}$ and $\{4, 5\}$ each are winning some of the time. After agents 4 and 5 stop bidding, only coalitions $\{1, 2\}$ and $\{1, 3\}$ win. The observation of Parkes and Wurman is that these changes in dynamics can be predicted [16, 22]. In this case, the change occurs because agents 4 and 5 have stopped submitting new bids for their bundles. Without getting into the specifics which follow, we can therefore think of Pure Proxy APA as proceeding in a sequence of stages, where new stages are defined by a change in the dynamics of competition. When viewed this way, instead of simulating every single round as in Pure Proxy, we can perform some calculations to determine the dynamics in each stage and then simulate all the rounds within the stage at once. Our algorithm therefore consists of a definition of a stage, how to compute the dynamics within a stage, and how to determine when a new stage begins.

5.5 Stage Definition

We can think of stages as being defined by events which signal the beginning of a new stage. Within a stage, the coalitions which win do so in a cyclical manner. As seen in Example 5.1, during the first and second stages (marked by breaks in Table 5.1), the winning coalitions take turns winning. Therefore, the dynamics of a stage may change when new winning coalitions emerge or when winning coalitions drop out of competition. Two distinct types of events can cause this to occur:

- Changes in an agent's demand set.
- A non-winning coalition catches up to the winning coalitions.

The first event that may cause a change in dynamics is a change in an agent's demand set. As in Example 5.1, when the demand sets of agents 4 and 5 become the null set (they no longer submit new bids), the dynamics change as coalition $\{4, 5\}$ no longer wins. Alternatively, when an agent first begins to bid on a new bundle, it is possible that this new bid will cause the creation of a new winning coalition.

The second event that may separate two stages is more subtle. Within a stage, there are coalitions that generate the top-level of revenue and win some portion of time, while there are other coalitions that are never declared the provisional winner since they always generate lower revenue. The coalitions that generate the top-level of revenue compete and increase their prices together, but at a slower rate because they may win some of the time (and hence their agents will not increase their bids all of the time by the definition of ascending proxy). The coalitions that never win increase their bids at a potentially faster rate since they are never winning (and hence their agents might always be increasing their bids) [1] The second event that defines stage boundaries is when one of these low revenue coalitions catches up to the top-tier of revenue generating coalitions because of this difference in rates. Our motivating example does not contain this phenomena, but Example 9 in Section 5.8 provides such a case.

5.6 Calculating Behavior within the Stage (FRAC)

Having defined our notion of a stage, we now look to compute the dynamics within a given stage. We call this algorithm FRAC. Because of our definition of the stage, we can assume two facts when computing behavior within the stage:

- Agent's demand sets stay the same.
- All winning coalitions start the stage at the highest level of revenue.

We can assume that agent's demand sets stay the same because if not, we would be dealing with two different stages. Similarly, a winning coalition which starts out at a lower-level of revenue would contradict our stage definition. If a coalition did not start at the top-level of revenue but became a winning coalition at some point in the stage, this would mean that coalition had caught up to the competing coalitions. This contradicts our stage definition.

[1] Note that this may not always be the case. For example, if a lower revenue coalition contains an agent in common with a winning coalition, the lower revenue coalition also increases less when the winning coalition is winning. The argument here provides an explanation for why this phenomena might ever occur.

Therefore, in computing the dynamics of a given stage, we can restrict our attention to the coalitions that generate the highest amount of revenue at the start of the stage.

Definition 17. For a given stage T, the set of *T-interesting coalitions* is those interesting coalitions which generate the highest level of revenue at the start of the stage. We denote this set of coalitions by C^*.

These T-interesting coalitions will each take turns being named the winner. This is necessarily true because when a coalition is the named the winner, the other coalitions increase their prices while the winning coalition stays stagnant. This will eventually cause a new coalition to be named the winner. We want to figure out how often each coalition is declared the winner within each stage.

The key observation is that given any sequence of iterations during the APA, we can look back after the fact and say that each coalition was winning for a specific number of rounds out of the total number of rounds elapsed. When we sum these ratios over all coalitions, they must sum to one since only one coalition wins in a round, and some coalition wins in every round. When we consider infinitesimally small bid increments, these discrete ratios become fractions and the number of rounds becomes a continuous time interval. Therefore, if coalition C_i wins for a fraction f_{C_i}, this means that for a given time interval D, C_i is winning $f_{C_i}D$ of the time. As in the discrete case, these fractions must sum to 1.

If we have a variable f_{C_i} for each coalition C_i, this is equivalent to the expression:

$$\sum_{C_i \in C^*} f_{C_i} = 1 \tag{5.3}$$

Notice that we can restrict our attention to C^* since the stage definition allows us to assume that all winning coalitions within stage T are T-interesting. Also notice that the $f_{C_i} \geq 0$, but some of the f_{C_i} could be equal to zero, meaning coalition C_i never wins during the stage. We distinguish the coalitions that win from the coalitions that never win with the following definition.

Definition 18. Within a stage, a *competitive coalition* C_i is one for which $f_{C_i} > 0$.

Referring back to Example 5.1, in the first stage (before agent 5 starts bidding on C), the winning alternates between coalitions $\{1,2\}, \{1,3\}$ and $\{4,5\}$. These coalitions are the only competitive coalitions. Each of these coalitions wins 1/3 of the time during the first stage. Indeed, if we look at any set of rows of Table 5.1,

every coalition wins for a well-defined fraction of time. This is simply because only one coalition wins in each iteration. Also, due to the way we construct the fractions, they must sum to 1.

Given the f_{C_i}, we can then derive the fraction of time that a given agent is winning within the time interval in terms of the f_{C_i}. The amount of time a single agent is winning will simply be the sum of the f_{C_i} of the coalitions in which the agent takes part. An exception to this rule occurs when an agent is no longer active. An inactive agent never changes her bids. We can model this by saying that she is winning all of the time.

Definition 19. Formally, we can define the fraction of time that agent j is winning to be the *agent fraction*:

$$f_{A_j} = \sum_{C_i \in C^* \text{ s.t. } j \in C_i} f_{C_i} \ \forall \text{ active } j \tag{5.4}$$

$$f_{A_j} = 1 \ \forall \text{ non-active } j \tag{5.5}$$

Because of the structure of APA, we can use these agent fractions to calculate how much prices are changing for every coalition. In APA, an agent does not increase her bid when she is winning, and increases her bid by ϵ on all packages in her ϵ-demand set when she is not winning. In the limit as ϵ goes to 0, an agent who is winning a fraction f_{A_i} of the time will increase her bid $(1 - f_{A_i})D$ of the time for a given duration D. Similarly, since coalitions are just collections of agents, the revenue increase a coalition experiences is equal to the sum of the price increases of its agents. For a given time interval D, a coalition's generated revenue increases by:

$$\sum_{j \in C_i} (1 - f_{A_j})D \tag{5.6}$$

Definition 20. We define this multiplier of the duration as the *price increase* for a coalition. The price increase k_{C_i} of a coalition C_i is:

$$k_{C_i} = \sum_{j \in C_i} 1 - f_{A_j} \tag{5.7}$$

Applying these concepts to the first stage of our motivating example, we see that agent 1 wins 2/3 of the time since she is a member of coalitions $\{1, 2\}$ and $\{1, 3\}$ each of which wins 1/3 of the time. Agent 2,3,4,5 each win 1/3 of the time. Now

that we have these fractions, we can backtrack and calculate the price increase for $\{1,2\},\{1,3\}$ and $\{4,5\}$. $\{1,2\}$ consists of agents 1 and 2. Agent 1 wins 2/3 of the time, so she increases her bid 1/3 of the time. Agent 2 wins 1/3 of the time, so she increases her bid 2/3 of the time. Therefore, the price increase for $\{1,2\}$ is 1. Similarly, the price increase for $\{1,3\}$ is 1. For coalition $\{4,5\}$, the price increase is $2/3 + 2/3 = 4/3$.

The price increase is a very important concept for determining behavior within a stage. The key observation is that in the discrete case, in order for coalitions to take turns winning, they must generate about the same amount of revenue. That is, their revenue must rise at approximately the same rate. When we take the limit as the bid increment goes to 0, this forces coalitions to have the same price increase.

Proposition 6. For any pair of competitive coalitions C_i, C_j, we have $k_{C_i} = k_{C_j}$.

Proof: Suppose not. Then wlog, $k_{C_i} > k_{C_j}$. However, this means that for every positive duration D, C_j will generate more revenue than C_i. Thus, C_j will never be winning. But this contradicts the competitiveness of C_j. It must be the case that $k_{C_i} = k_{C_j}$.

Corollary 7. For any pair of coalitions C_i, C_j, if C_i is competitive and C_j is non-competitive, then $k_{C_i} \geq k_{C_j}$.

Thus, we see that we have a second set of conditions on the values of the f_{C_i}:

$$k_{C_i} = k_{C_j} \ \forall \ C_i, C_j \text{ with } f_{C_i}, f_{C_j} > 0 \tag{5.8}$$

$$k_{C_i} \geq k_{C_j} \ \forall \ C_i, C_j \text{ with } f_{C_i} > 0, f_{C_j} = 0 \tag{5.9}$$

Notice that in our motivating example, the price increases for the competitive coalitions are not equal. This is due to the non-trivial bid increment we chose to decrease the number of rounds in the example. When the bid increment is non-trivial, the price increases will be equal within a factor dependent on the bid increment. Since our algorithm represents the limit of APA as the bid increment goes to 0, the price increases for the competitive coalitions need to be exactly equal. When the bid increment goes to 0, the price increase represents the continuous rate at which the revenue generated by a coalition increases. In order to be competitive with other coalitions (be chosen as the winner a nonzero amount of time), it must be the case that the price increases are equal. Otherwise, the unequal price increase is manifested in every positive duration, and the coalition with smaller price increase will never win.

We are now ready to calculate the price trajectories for a given stage. We seek the f_i that satisfy Equations 5.3 and 5.8. Suppose we are considering n coalitions. For now we assume that they are all competitive, that is $f_{C_i} > 0$. Since they are all competitive, we have a system of n linear equations in the f_{C_i}. Equation 5.3 contributes one constraint, while Equation 5.8 contributes $n-1$ equations since we need that the k_{C_i} are all equal (once we set one, we fix the $n-1$ others). We have not been able to prove the linear independence of this system of equations. However, intuitively, increasing one coalition's winning fraction should affect the price increase of that coalition more than the other coalitions, which leads us to believe the solution is unique. In summary, our system of n-linear equations with n unknowns is (NSYS):

$$\sum_{C_i} f_{C_i} = 1 \tag{5.10}$$

$$k_{C_1} = k_{C_2} = \ldots = k_{C_n} \tag{5.11}$$

This system of equations is linear in f_{C_i} since the k_{C_i} are linear functions of the f_{C_i}. Notice, however, in order to solve the system of equations, we cannot place a constraint on the values of the f_{C_i}. If we enforce such a condition, then we are not guaranteed a solution. Indeed, the solution to the set of equations may result in a negative fraction for one of the coalitions. While this does not make sense in our setting, intuitively, this symbolizes the inability of a given coalition to compete within a stage. For instance, because of the way the ascending proxy is structured, larger coalitions will experience larger bid increments. Whenever a coalition is not winning, the size of its bid increase is proportional to the size of the coalition. Larger coalitions may overwhelm smaller coalitions, making it impossible for the smaller coalitions to satisfy Equation 5.8 (see Example 8 in Section 5.8).

Since negative fractions do not make sense in our setting (the minimum f_{C_i} hat is sensible is 0), we need to do some extra work in order to obtain the correct f_{C_i}. We cannot solve the n-dimensional linear equation while enforcing positivity on the f_{C_i} because some coalitions are non-competitive (they have $f_{C_i} = 0$). However, if we only considered the competitive coalitions, which by definition are those with $f_{C_i} > 0$, the solution to the linear equations would be valid (no $f_{C_i} < 0$). Thus, if we can find the correct subset of coalitions that are competitive, we could just solve NSYS restricted to those coalitions to obtain the correct f_{C_i}. All non-competitive coalitions would have $f_{C_i} = 0$. We can find the correct subset and the resulting fractions in one step

using a mixed integer linear program (FRAC-MIP):

$$\max \sum_{C_i \in C^*} k_{C_i} \tag{5.12}$$

$$\text{subject to} \sum_{C_i \in C^*} f_{C_i} = 1 \tag{5.13}$$

$$f_{A_j} = \sum_{C_i \text{ s.t. } j \in C_i} f_{C_i}, \ \forall \text{ active } j \tag{5.14}$$

$$f_{A_j} = 1 \ \forall \text{ non-active } j \tag{5.15}$$

$$k_{C_i} = \sum_{j \in C_i} 1 - f_{A_j} \tag{5.16}$$

$$k_{C_i} - k_{C_j} + N a_{C_j} \leq N \tag{5.17}$$

$$f_{C_i} \leq a_{C_i} \tag{5.18}$$

$$a_{C_i} \in \{0,1\}, f_{C_i} \geq 0, C_i \in C^*, N > k_{C_i} - k_{C_j} \ \forall \ i, j \tag{5.19}$$

Here, we have indicator variables a_{C_i} denoting whether or not C_i is competitive during the given round. $a_{C_i} = 1$ indicates that C_i is competitive, and $a_{C_i} = 0$ indicates that C_i is not competitive. If C_i is not competitive, then $f_{C_i} = 0$, which is enforced by constraint 5.18. If we do indeed have the correct competitive coalitions, then it must be the case that $k_{C_i} \leq k_{C_j}$ for all competitive coalitions C_j. We enforce this by using the Constraint 5.16. If $a_{C_j} = 1$, then Constraint 5.16 forces $k_{C_i} - k_{C_j} \leq 0$ or alternatively, $k_{C_i} \leq k_{C_j}$. However, if $a_{C_j} = 0$, this constraint is non-binding if we choose N to be large enough (greater than the largest possible difference between k_{C_i} and k_{C_j}). As a result, if we choose a coalition C_i to be competitive, it must be the case that all other coalitions have price increase less than or equal to the revenue generated by C_i.

We notice that the objective in FRAC-MIP has been chosen to be the sum of the price increase over all coalitions. We have included this objective because we do not have theoretical results on the uniqueness of the solution to this system of equations with constraints (if we knew the solution were unique, we would not need an objective). Intuitively, it seems that there should be a unique point at which the competitive coalitions experience the same price increase that is at least the price increase of the non-competitive coalitions. If we increase the coalitional fraction of one coalition, we need to decrease the coalitional fraction of some other coalition because of Constraint 5.13. This should cause the price increase of the first coalition

to decrease and the price increase of the second to increase, resulting in unequal price increases. However, when we look at changing more than two coalitional fractions, the effects are more complicated. Theoretical results on this uniqueness may be an area of future work.

5.7 Calculating the Length of a Stage (DUR)

By solving FRAC-MIP, we obtain coalitions for which $f_{C_i} > 0$ along with the exact values of the f_{C_i}. All other coalitions will have f_{C_i} set to 0. We now have a way to update agent's bids. By Equation 5.4 which gives the f_{A_i} in terms of the f_{C_i}, we can calculate the fraction of time agents are increasing their bids. Because we compute everything in fractions, to determine the actual price increase, we need a duration, D. Given a duration D, agent i's bids will increase by $(1 - f_{A_i})D$. We can think of the duration D as a unit of time, and the f_{A_i} as a rate.

We therefore need to determine the duration D for which the rates from the first part of our algorithm are valid. We call this portion of the algorithm DUR. Recall that there are two restrictions on our duration D, notably, the two events that define our notion of a stage. Ruling out these two events created assumptions that played key roles in our determination of the f_{A_i}. As a result, we can only run the stage while these two events do not occur.

5.7.1 Demand Set Change

The first event that does not occur within is stage is a change in agent's demand sets. This assumption could possibly be violated when an agent begins bidding on a new package or when an agent stops bidding. In the first case, an agent starts bidding on a new bundle when her payoff from her current bundle equals the value of the new bundle. Let π_i^T denote the payoff for agent i at the start of stage T. For each agent, we let the minimal price increase before a demand set change be δ_i^T. We have that:

$$\delta_i^T \leq \pi_i^T - \max_{s \text{ s.t. } p_i^T(s)=0} v_i(s) \tag{5.20}$$

For the case where agent i is already bidding on all of her bundles, we need to know when it will stop bidding (the prices on bundles reach the agent's value). In

this case, the agent just stops bidding when her payoff becomes 0:

$$\delta_i^T \leq \pi_i^T \tag{5.21}$$

We see that the second constraint is included in the first (since if the agent is bidding on all her packages, there will be no packages s.t. $p_i^T(s) = 0$). We therefore have that:

$$\delta_i^T \leq \pi_i^T - \max_{s \text{ s.t. } p_i^T(s)=0} v_i(s) \tag{5.22}$$

Since each agent bids up by $1 - f_{A_i}$ for a duration of 1, the time it takes to bid up by δ_i^T is $\frac{\delta_i^T}{1-f_{A_i}}$. Thus:

$$D \leq \min_{i \in A} \frac{\delta_i^T}{1 - f_{A_i}} \tag{5.23}$$

5.7.2 Catching Up

The second event that does not occur within a stage is the emergence of a new winning coalition. It is possible that while the winning coalitions are competing at the top-revenues (and therefore winning some portion of the time), coalitions which originally generated lower revenues may catch up (because they are losing all the time and hence may always increasing their bids) [2].

Therefore, we have to end the current stage if we ever find that some coalition which previously generated lower revenue has caught up to the top-tier of competing coalitions. Recall that a coalition C_i by definition yields revenue $r^T(C_i)$ at the start of stage T. Therefore, the amount of time D_j required for a lower revenue coalition C_j to catch up to a maximal revenue coalition C_i satisfies:

$$\begin{aligned} r^T(C_j) + D_j \cdot k_{C_j} &= r^T(C_i) + D_j \cdot k_{C_i} \\ \Rightarrow D_j &= \frac{r^T(C_i) - r^T(C_j)}{k_{C_j} - k_{C_i}} \end{aligned} \tag{5.24}$$

The D_j is only meaningful if $k_{C_j} > k_{C_i}$. If $k_{C_i} = k_{C_j}$, then we divide by zero, implying it will take an infinite amount of time to catch up. This makes sense since

[2] As noted previously, this is one possible scenario and is not true for every coalition that generates lower revenue

two coalitions that are increasing at the same rate but start at different revenues will never cross. Similarly, if $k_{C_i} < k_{C_j}$, then $D_j < 0$, which means it takes a negative amount of time to catch up. This is meaningless for our interpretation of the D_j.

Therefore, the duration D has to be less than minimal time required for coalitions C_j with k_{C_j} greater than the competitive k_{C_i} to catch up. If we let C_i be a representative of the competitive coalitions, we have the constraint:

$$D \leq \min_{C_j \notin C^*, k_{C_j} > k_{C_i}} \frac{r^T(C_i) - r^T(C_j)}{k_{C_j} - k_{C_i}} \tag{5.25}$$

5.7.3 Duration Calculation

Thus, we can now find the maximum duration for which we can use our fractions calculated by FRAC by taking the minimum of the two maximum durations above. Let C_i be any competitive coalition. Then the maximum duration for which our fractions from FRAC are valid will be:

$$D = \min(\min_{C_j \notin C^*, k_{C_j} > k_{C_i}} \frac{r^T(C_i) - r^T(C_j)}{k_{C_j} - k_{C_i}}, \min_{i \in A} \frac{\delta_i^T}{1 - f_{A_i}}) \tag{5.26}$$

After simulating the auction for this duration (updating the bids of each agent based on f_{A_i} and D), we can repeat the process for the next stage. We continue this staged implementation until there is only one competitive coalition, and there are no coalitions that generate lower revenue that could possibly catch up to this coalition.

5.8 Illustrative Examples

Example 7. Recall Example 1. We have three agents 1, 2, 3 bidding on goods A and B. Each agent values a single bundle. $v_1(A) = 5, v_2(B) = 5, v_3(B) = 20$. Intuitively, the result here will have agent 3 winning and paying a price of 10 for bundle AB. This is due to the fact that agents 1 and 2 will drop out once the prices of A and B reach 5. In order to be more appealing than the coalition of $\{1, 2\}$, agent 3 must bid at least 10 for AB.

We first notice that the interesting coalitions will be $\{1, 2\}$ and $\{3\}$. There cannot be coalitions consisting of 1 and 3 or 2 and 3 because their goods overlap. A coalition

consisting of either 1 or 2 by itself is non-competitive since $\{1, 2\}$ will always yield more revenue. Let $C_1 = \{1, 2\}$ and $C_2 = \{3\}$. Using Equations 5.4 and 5.8, we have the following expressions in terms of the f_{C_i}:

$$\begin{aligned} f_{A_1} &= f_{C_1} \\ f_{A_2} &= f_{C_1} \\ f_{A_3} &= f_{C_2} \\ k_{C_1} &= 1 - f_{A_1} + 1 - f_{A_2} = 2 - 2f_{C_1} \\ k_{C_2} &= 1 - f_{A_3} = 1 - f_{C_2} \\ f_{C_1} + f_{C_2} &= 1 \end{aligned}$$

Solving for the f_{C_i} subject to the constraint $f_{C_1} + f_{C_2} = 1$ and $k_{C_1} = k_{C_2}$, we obtain $f_{C_1} = 2/3, f_{C_2} = 1/3$. This means that coalition C_1 is winning 2/3 of the time and C_2 is winning 1/3 of the time. This makes sense since every time C_1 is losing, its generated revenue increases by 2ϵ while C_2 only increases its generated revenue by ϵ each time it is losing (C_1 has two members while C_2 has one member). Therefore, in order to generate the same revenue, C_2 must be losing twice as often, which is indeed the case.

Having computed these fractions, we now need to know when to stop the stage. In this case, there are no coalitions that started the stage with lower revenue, so we do not need to check for catching-up. We do need to compute the δ for each agent. (We drop the superscript for the stage since we know we are in the first stage). Since the agents only submitted bids for a single bundle, δ_i will just be the agent's values. $\delta_1 = 5, \delta_2 = 5, \delta_3 = 20$. For every time increment, agent 1 increases her bid by $1 - f_{A_1} = 1 - f_{C_1} = 1 - 2/3 = 1/3$. The same is true for agent 2. Agent 3 increases her bid by $1 - f_{A_3} = 1 - f_{C_2} = 1 - 1/3 = 2/3$. Therefore, agents 1 and 2 can no longer bid when the duration exceeds 5 / (1/3) = 15, and agent 3 can no longer bid when the duration exceeds 20 / (2/3) = 30. We therefore simulate the stage for a duration of 15, arriving at prices $p_1(A) = 5, p_2(B) = 5, p_3(AB) = 10$. At this time, agents 1 and 2 have submitted their valuations, and have no bids left to submit. As a result, coalition $\{3\}$ is declared the winner, paying a price of 10 for bundle AB. This example is the same as Example 1, and we showed in Chapter 2 that the core outcomes were those outcomes in which agent 3 receives AB and pays price 10. Here we see that our accelerated implementation arrives at a core outcome as expected (in

fact it is bidder-Pareto-optimal).

Example 8. An example of non-competitiveness. More concretely, suppose we have seven agents each with single bundle valuations. $v_1(A) = 10, v_2(B) = 1, v_3(C) = 1, v_4(A) = 10, v_5(B) = 1, v_6(C) = 1, v_7(ABC) = 50$.

We see that there will be 2^3 possible coalitions with 3 agents (two choices for agents bidding on A, two choices for agents bidding on B, two choices for agents bidding on C). There is also one coalition C_9 consisting of agent 7 alone. Suppose that C_9 never wins during the stage. Then the price increase of C_9 will be ϵ for each time increment (agent 7 is always increasing her bid since C_7 never wins). Looking at $C_1, ..., C_8$, the coalitions with three agents, we notice that as a whole they are symmetric. Therefore, $f_{C_1} = f_{C_2} = ... = f_{C_8} \Rightarrow f_{C_1} = ... = f_{C_8} = 1/8$. For a given agent in $\{1, 2, ..., 8\}$, that agent will be in 4 coalitions of three agents (if the agent bids on bundle b, then there are two choices of other agents for each of the two other bundles). Thus, $f_{A_j} = (1/8) \cdot 4 = 1/2$ for $1 \leq j \leq 6$. The price increase k_{C_i} for any of these eight coalitions k_{C_i} will be:

$$k_{C_i} = \sum_{j \in C_i} (1 - f_{A_j}) = 3(1 - 1/2) = 3/2$$

Therefore, we see that even if we assume C_9 never wins, $k_{C_i} = 3/2$ for $1 \leq i \leq 8$, and $k_{C_9} = 1$. Since C_9 increases its price most rapidly when it is never winning, this shows that it is impossible for C_9 to compete during this stage . Indeed, when we solve the system of linear equations to obtain the f_{C_i}, we see that $f_{C_9} = -1/5$, $f_{C_i} = 3/20$ for $1 \leq i \leq 8$. When we set these fractions, we see that $f_{A_i} = (3/20) \cdot 4 = 3/5$ for $1 \leq i \leq 6$ and $f_{A_7} = -1/5$, giving:

$$\begin{aligned} k_{C_1} &= \sum_{j \in C_1} (1 - f_{A_j}) = 3(2/5) = 6/5 \\ k_{C_9} &= \sum_{j \in C_9} (1 - f_{A_j}) = 1 - f_{A_7} = 1 - f_{C_7} = 1 - (-1/5) = 6/5 \end{aligned}$$

As we have seen in this example, there are times when coalitions cannot compete since the size of the increase in generated revenue is proportional to the number of agents in the coalition. In this case, a coalition with a single agent cannot compete with coalitions with three agents. As seen, if we just try to solve the system of n linear equations (NSYS, Equations 5.10 and 5.11) with C_9 included, then we get a

negative fraction for f_{C_9} which is meaningless. Instead, the output of FRAC-MIP will tell us that C_9 is not competitive and will give the fractions that equalize the price increases of the competitive coalitions $C_1, \ldots, C_8$. Because FRAC-MIP tells us C_9 is not competitive, we know to set $f_{C_9} = 0$.

However, even though C_9 starts out non-competitive, the coalitions with three agents actually quickly bid up the price and reach their valuations. In later rounds, C_9 will catch up and ultimately become the winner of the auction.

Example 9. An example of catching-up. Let us continue with Example 8. $v_1(A) = 10, v_2(A) = 10, v_3(B) = 1, v_4(B) = 1, v_5(C) = 1, v_6(C) = 1, v_7(ABC) = 50$. The first stage of the auction is as described in Example 8. Again, we have nine total coalitions that could possibly compete. There are eight coalitions consisting of three agents each, and there is one coalition consisting of only agent 7. Let $C_1, ..., C_8$ denote the eight coalitions consisting of three agents each, and C_9 denote $\{7\}$. In the first stage, we will have the same agent fractions as Example 8. Agents 1,2,...,6 will be winning 1/2 of the time, while agent 7 is never winning.

We stopped our analysis in Example 8 at this point since we saw that C_9 was not competitive. We now continue stepping through our algorithm. Moving into the DUR portion of the algorithm, we can calculate the δ_i for each agent. Recall that δ_i is the maximum duration for which agent i's active bundles do not change. In this example, since each agent only submits values for a single bundle, we calculate the maximum duration until the agent's prices exceed their values:

$$\begin{aligned}
\delta_1 &= v_1(A)/(1 - f_{A_1}) = 10/(1/2) = 20 \\
\delta_2 &= v_2(A)/(1 - f_{A_2}) = 10/(1/2) = 20 \\
\delta_3 &= v_3(B)/(1 - f_{A_3}) = 1/(1/2) = 2 \\
\delta_4 &= v_4(B)/(1 - f_{A_4}) = 1/(1/2) = 2 \\
\delta_5 &= v_5(C)/(1 - f_{A_5}) = 1/(1/2) = 2 \\
\delta_6 &= v_6(C)/(1 - f_{A_6}) = 1/(1/2) = 2 \\
\delta_7 &= v_7(ABC)/(1 - f_{A_7}) = 50/1 = 50
\end{aligned}$$

Since all coalitions start out generating 0 revenue, we cannot have the catching-up phenomena occurring in the first stage. Therefore, our duration will be the minimum of the δ_i, which is 2. After running for duration 2, we update the prices for each agent by multiplying the duration by the fraction of time the agent is increasing her

bid. $p_1(A) = 2 \cdot (1/2) = 1, p_2(A) = 1, p_3(B) = 1, p_4(B) = 1, p_5(C) = 1, p_6(C) = 1, p_7(ABC) = 2 \cdot 1 = 2$. We notice that at this point, agents 3,4,5, and 6 are no longer active since they have reached their valuations. However, since agents 1 and 2 are still active, they still compete on behalf of $C_1, ..., C_8$.

Notice that after running the first stage, C_9 only generates revenue 2 to the auctioneer, while the three agent coalitions generate revenue 3. As a result, in the second stage, C_9 is not T-interesting and we assume that $f_{C_9} = 0$. By symmetry again, we see that the $f_{C_1} = f_{C_2} = \ldots = f_{C_8}$. Since these fractions must sum to one, $f_{C_i} = 1/8$ for $1 \leq i \leq 8$.. Agent 1 participates in exactly half of these coalitions, so $f_{A_1} = 1/2$. By symmetry, $f_{A_2} = 1/2$. Since agents 3,4,5,6 have reached their valuations, $f_{A_3} = f_{A_4} = f_{A_5} = f_{A_6} = 1$ (we assume these agents are always winning and hence never increase their bids as required). $f_{A_7} = f_{C_9} = 0$ since within this stage, C_9 is never winning because it starts out generating lower revenue.

Having in hand our agent fractions, we can now calculate the deltas in order to determine the correction duration of this stage:

$$\begin{aligned}
\delta_1 &= \pi_1(A)/f_{A_1} = (v_1(A) - p_1(A))/(1/2) = 18 \\
\delta_2 &= \pi_2(A)/f_{A_2} = (v_2(A) - p_2(A))/(1/2) = 18 \\
{}^3\delta_3 &= \delta_4 = \delta_5 = \delta_6 = \pi_j(s)/(1 - f_{A_j}) = \pi_j(s)/0 = \infty \\
\delta_7 &= \pi_7(ABC)/f_{A_7} = (v_7(ABC) - p_7(ABC))/(1/2) = 96
\end{aligned}$$

Notice that $\delta_3 = \delta_4 = \delta_5 = \delta_6 = \infty$ since agents 3,4,5,6 are no longer active. If we take the minimum of the δ_i, we find that our minimum duration is 18. However, unlike the first stage, we now have to make sure the catching-up phenomena does not occur since C_9 starts the stage generating lower revenue than the competitive coalitions, $C_1, ..., C_8$. C_9 starts out generating revenue 2, while $C_1, ..., C_8$ generate revenue 3. However, any one of $C_1, ..., C_8$ only contains either agent 1 or agent 2 (they both bid on good A). Since agents 3,4,5,6, are no longer active, the price increase of $C_1, ..., C_8$ is simply the price increase of agent 1 or agent 2. They both win half the time, so $k_{C_1} = k_{C_2} = ... = k_{C_8} = 1/2$. On the other hand, C_9 never wins so agent 7 never wins and is always increasing her prices. Since C_9 consists solely of agent 7, this means the price increase of C_9 is also 1, so $k_{C_9} = 1$. Since C_9 is increasing its price faster than the competitive coalitions, it is possible that the revenue generated by C_9 catches up to the competitive coalitions before any change in agents' active bundles. To solve

for this duration D, we set up the following equation. We can choose C_1 to represent the competitive coalitions since they are symmetric.

$$\begin{aligned} r(C_1) &+ k_{C_1} D = r(C_9) + k_{C_9} D \\ \Rightarrow D &= (r(C_9) - r(C_1))/(k_{C_1} - k_{C_9}) \\ \Rightarrow D &= (2-3)/(0.5-1) = 2 \end{aligned}$$

We see that after a duration of 2, C_9 will catch-up to the competitive coalitions. This is smaller than the minimum of the δ_i which is 18. Therefore, our f_{A_i} are only valid for duration 2, and after updating the prices using the current f_{A_i} and a duration of 2, we need to recalculate the f_{A_i} since C_9 has become competitive.

Even though C_9 started the stage generating less revenue than the competitive coalitions, it was able to catch-up. In this case, C_9 had a larger price increase than the competitive coalitions and was able to make up the original difference in revenue before a change in agents' active bundles.

Chapter 6

Computational Issues

6.1 Computing the Interesting Coalitions

In the description of the Algorithm FRAC (Section 5.6), we assumed that there was some way to determine the T-interesting coalitions at the start of stage T. However, in practice, determining these T-interesting coalitions is non-trivial and can be the bottleneck of the indirect method. We provide several methods for computing these T-interesting coalitions.

6.1.1 Precomputation

After introducing the notion of interesting coalitions, Parkes suggests an algorithm for precomputing all of the interesting coalitions before we run the indirect algorithm [16]. Parkes proposes an iterative process for determining the possible winning coalitions. First we solve the winner determination problem, where agents' bids are simply their valuations. On the first solution to the winner determination problem, we will obtain some winning coalition. This coalition will be competitive. However, the fact that this coalition is the solution to the winner determination problem means that no coalition including all the agents in this coalition can be a winning coalition. We use the dynamics of the ascending proxy auction to prove this result.

Proposition 8. If a coalition K is the designated unique winner after solving the winner determination problem, then no proper superset of K is ever a provisional

winner.

Proof: Suppose K is the coalition that solves the winner determination problem where agent's bids are their valuations. Suppose that there is some strict superset $K' \supset K$ that wins at some point during the ascending proxy. Then consider the first time that coalition K' is declared the provisional winner. At this point, it must be the case that $r^t(K') \geq r^t(K)$ since K' is declared the provisional winner. By definition of payoffs and the coalitional value function, the coalitional value of a coalition, $w(K)$, is the current revenue generated plus the sum of current payoffs for agent in K. Thus, $w(K') = r^t(K') + \sum_{i \in K'} \pi_i^t$ and $w(K) = r^t(K) + \sum_{i \in K} \pi_i^t$. Subtracting, we see that $w(K') - w(K) = r^t(K') - r^t(K) + \sum_{i \in K'} \pi_i^t - \sum_{i \in K} \pi_i^t$. Since $r^t(K') \geq r^t(K')$, $w(K') - w(K) \geq \sum_{i \in K'} \pi_i^t - \sum_{i \in S} \pi_i^t = \sum_{j \in K', j \notin K} \pi_j^t$. Since payoffs are always non-negative and $K \subset K'$, $w(K') \geq w(K)$, meaning that K' yields at least as much value as K. This contradicts the choice of K as the unique winner of the winner determination problem.

Given this result, Parkes offers a conservative estimate for the interesting coalitions. If we find that K is the solution to the winner determination problem, we know that no strict superset of K can be interesting. Removing both K and all supersets of K from our search space, we again solve the winner determination problem. Like before, we remove the results K' and all supersets of K' from the search space. We do this exhaustively until we have eliminated all possibilities.

Once we have the set of all possible interesting coalitions, we no longer have to think about any other coalitions. While this approach may drastically reduce the number of coalitions we keep track of when running our algorithm, it may be intractable to enumerate all interesting coalitions at time 0. Without more knowledge about the problem, the number of interesting coalitions could be exponential in the number of agents.

6.1.2 Conservative Generation

Instead of enumerating all of the interesting coalitions at time 0, we can progressively compute the interesting coalitions as the staged mechanism proceeds. At the beginning of stage T, we need the T-interesting coalitions. As a conservative estimate for the T-interesting coalitions, we could compute all coalitions that generate the top

level of revenue at the start of stage T.

IP Generation One way to generate these T-interesting coalitions is to iteratively use an integer linear program to find new coalitions that generate the highest level of revenue. Our basic IP that finds coalitions that generate the maximal revenue:

$$\max \sum_{i \in A} \sum_{s \in S} p_i(s) x_{is} \tag{6.1}$$

$$\text{subject to} \sum_{s \in S, g \in s} \sum_{i \in A} x_{is} \leq 1, \ \forall g \in G \tag{6.2}$$

$$\sum_{s \in S} x_{is} \leq 1, \ \forall i \in A \tag{6.3}$$

$$x_{is} \in \{0, 1\} \tag{6.4}$$

This IP is the same as the winner determination IP, except that the objective function sums over the price of bundle s rather than the value of bundle s.

To obtain all of the coalitions that generate the highest level of revenue, we can solve this IP once to obtain the highest level of revenue R. After that, we can iteratively run the IP with added constraints to make sure we do not rediscover a previously found coalition. This can be done by adding the following constraint for each discovered interesting coalition C_j with corresponding interesting bundles B_j:

$$\sum_{i \in C_j} x_{j(B_j(i))} < |C_j| \tag{6.5}$$

With this constraint, we are guaranteed that the IP will not output C_j since not all of the $x_{j(B_i(j))}$ can be set to one (or else they would sum to at least $|C_j|$).

Therefore, to obtain a conservative estimate of the T-interesting coalitions, we iteratively grow the set of T-interesting coalitions and add the corresponding constraint to the IP until the IP outputs a coalition which generates less than R revenue. By adding in constraints that do not allow the IP to regenerate previously output coalitions, we are sure that this process will terminate.

Wurman's Approach Wurman suggests another way of keeping track of these coalitions. Namely, we know that from stage to stage that the only new winning coalitions come from events that separate the stages. Therefore, if a coalition catches

up, we just need to add that single coalition to the set of T-interesting coalitions. If an agent's demand set increases, then we need to add all new T-interesting coalitions and allocations that include the agent and her new bundle[22].

There are two issues with this approach. First, at the beginning of the algorithm, we need to enumerate all feasible coalitions since all coalitions start at revenue 0, and hence all coalitions are T-interesting. Second, it may take a lot of effort to generate all new coalitions that are formed when an agent's demand set changes.

Summary The advantage of both of these methods of conservative generation is that we will be guaranteed to have all of the T-interesting coalitions at the end of this process. Unfortunately, this can also be a disadvantage, as the entire set of T-interesting coalitions can still be exponential in size, and it is intractable to fully enumerate all of these coalitions.

6.1.3 Constraint Generation (FRAC-CG)

Recognizing the disadvantage of our first two methods, we suggest an alternative method which utilizes the constraint generation paradigm. Instead of explicitly calculating all of the T-interesting coalitions, we generate a small subset of T-interesting coalitions, run Algorithm FRAC on this subset to obtain agent fractions, and check to make sure the fractions generated are correct. The hope is that if we take a small subset of the T-interesting coalitions, we will be able to capture all of the competitive dynamics in the system without having to explicitly deal with every T-interesting coalition.

The motivating observation is that we can check if a given set of coalitional fractions are valid. A set of coalitional fractions f_{C_i} will define the agent fractions f_{A_i}. These agent fractions then define the price increase for every coalition. If the f_{A_i} are correct, then the competitive coalitions will all have equal price increase, and the non-competitive coalitions ($f_{C_i} = 0$) will not have greater price increase than the competitive coalitions (Proposition 6 and Corollary 7).

Because we have this verification process, we can use the constraint generation paradigm. We run algorithm FRAC on a subset Y of the T-interesting coalitions. This generates a set of fractions f_{C_i} for the coalitions in Y. We assume that all

coalitions outside of Y are non-competitive and have $f_{C_i} = 0$. We then use these fractions to compute the f_{A_i} and in turn, the price increases for *every coalition*. We check that these price increases satisfy Proposition 6 and Corollary 7. If they do, we have the correct set of agent fractions and we can move onto algorithm DUR. If not, we enlarge the set Y and try again. We iteratively do this until the agent fractions we obtain are valid.

We can use Algorithm FRAC to generate the f_{C_i} for some subset Y. In order to check the validity of the derived f_{A_i}, we can use an IP that is similar to the winner determination IP. As with Section 6.1.2, we are looking for coalitions that satisfy certain characteristics. In Section 6.1.2, the characteristic was the revenue generated. Here, we are concerned with the price increase defined by a given set of f_{A_i}. Let R be the level of revenue generated by the T-interesting coalitions. We can use a slightly modified IP to find the T-interesting coalition with maximal price increase:

$$\max \sum_{i \in A} \sum_{s \in S} x_{is}(1 - f_{A_i}) \tag{6.6}$$

$$\text{subject to } \sum_{i \in A} \sum_{s \in S} p_i(s) x_{is} = R \tag{6.7}$$

$$\sum_{s \in S, g \in s} \sum_{i \in A} x_{is} \leq 1, \ \forall \ g \in G \tag{6.8}$$

$$\sum_{s \in S} x_{is} \leq 1, \ \forall \ i \in A \tag{6.9}$$

$$x_{is} \in \{0, 1\}$$

Again, we have constraints that ensure each good is allocated once and each agent receives at most one bundle (Constraints 6.8 and 6.9). We have a new constraint 6.7 that ensures the coalition generated is T-interesting. The objective function maximizes the price increase of a coalition.

Suppose the IP has objective value J. J represents the maximal price increase of any T-interesting coalition. We can also calculate the maximal price increase K of any coalition in Y by just taking the price increase of any competitive coalition in Y. If $J \leq K$, then using Y to generate the f_{A_i} does not violate Proposition 6 or Corollary 7 and we can move on. However, if $J > K$, then using Y to generate the f_{A_i} violates Corollary 7 since there is some coalition with higher price increase than the competitive coalitions. Therefore, we need to add more T-interesting coalitions to Y.

In line with the constraint generation paradigm, we add the most-violated coalition. In this case, this will be the coalition with the maximal price increase. This can be retained from the solution of the IP by examining the value of the indicator variables x_{is}.

6.2 Avoiding the MIP (Acc-LP)

Another computational issue that arises is solving FRAC-MIP. This is a rather large mixed integer linear program, and can possibly be the bottleneck of our accelerated algorithm. MIPs are slow in practice, and we need to solve FRAC-MIP each time we derive the correct coalitional fractions and agent fractions. Here we propose a different method that iteratively solves a number of linear programs instead of solving the much harder MIP. Linear programs are known to be in P whereas mixed integer linear programs are NP-complete. As a result, we may be better off solving a number of LPs rather than a single MIP.

The observation that influenced this approach was that we can solve the n-dimensional set of linear equations (NSYS, Equations 5.10 and 5.11) to obtain a set of fractions that equalizes the price increase of any given set of coalitions. Solving an n-dimensional linear equation can be done with an LP rather than a MIP. However, the problem is that in order to guarantee a solution, we cannot place constraints on the fractions (we may obtain negative fractions). With the MIP, we are essentially able to find the set of coalitions that are positive, and then solve for the fractions on this restricted subset. Instead of using the MIP to do this, we can iteratively solve the LP. We use an LP to solve the n-dimensional linear equation. If this produces negative fractions, then we remove the coalitions that are assigned negative fractions. We continue this process of removing coalitions from our system of equations and resolving until we obtain all positive fractions.

Once we obtain these fractions, we use the constraint generation methods discussed in section 6.1.2 to check if these are the correct fractions. Notice that unlike the MIP approach, given a set of T-interesting coalitions, we are not guaranteed to find the fractions for these coalitions that satisfy Proposition 6 and Corollary 7. We label this algorithm FRAC-CG.

When we remove coalitions from the system of equations, we no longer enforce any

conditions on the price increase of the removed coalitions. Therefore, this approach can be thought of as a heuristic whereas the MIP approach will always be correct. We take the fact that coalitions are assigned negative fractions to be an indicator of the fact that they will not be competitive during the given stage. We then assume they are not competitive by removing them from our system of equations. However, this assumption may be incorrect since we have no theoretical guarantee that being assigned a negative fraction implies non-competitiveness. For instance, suppose we have a large set Y of coalitions. We solve for the fractions that equalize the price increase for all of these coalitions, and find that coalitions K_1 and K_2 are assigned negative fractions. Does this mean that both K_1 and K_2 are always non-competitive? It could be that K_1 would be competitive if we removed K_2 and vice versa.

Another possible issue is one of convergence. Because we remove coalitions from consideration, it is possible that in the constraint generation process, these removed coalitions reenter into our set. We have not been able to prove theoretical properties of this heuristic, but in practice, we see both very good convergence and a significant performance improvement over the MIP approach. More details on experimental results can be found in Chapter 7.

6.3 Computing the Duration (DUR-CG)

Another computational issue arises in Algorithm DUR. We observe that in computing the duration, we take the minimum across all coalitions $C_i \notin C^*$. Since C^* is a select subset of the entire space of coalitions, this expression is a minimum over an exponentially large set. Directly computing this minimum is intractable in practice.

Instead, we again use constraint generation. First, we compute the maximum duration D until an agent begins bidding on a new bundle or drops out of the bidding, as described by Expression 5.22. Given this duration, we can use an integer linear program to see if there are any violations. In other words, we suppose that the maximum duration calculated in Expression 5.22 is the correct duration, and we use a MIP to tell us whether this is the case. In addition, using the value of the variables in the MIP that maximize the objective, we can determine what the correct duration

should be. Consider the following MIP:

$$\max \sum_{i \in A} \sum_{s \in S} x_{is}(p_i^t(s) + (1 - f_{A_i})D) \tag{6.10}$$

$$\text{subject to } \sum_{i \in A} \sum_{s \in S} x_{is} \leq 1 \tag{6.11}$$

$$\sum_{g \in G} \sum_{s \in S} x_{is} \leq 1 \tag{6.12}$$

$$x_{is} \in \{0, 1\} \tag{6.13}$$

The purpose of this IP is to output the coalition which reaches the highest price level given as input a duration, the current prices, and the agent fractions (f_{A_i}). As with the winner determination IP, the variables x_{is} are indicator variables. The objective function maximizes the revenue generated by a coalition after running for duration D with the f_{A_i}.

If it is the case that the maximal revenue reached after duration D is equivalent to the revenue reached by the competitive coalitions, then we have the correct duration. If this is not the case, then some coalition must have caught up to the competitive coalitions. Otherwise, it would not be possible that after duration D there are coalitions that generate revenue more than the competitive coalitions. In this case, the IP will tell us what coalition caught up to the competitive coalitions so that we can refine our duration. We can use the calculation shown in Expression 5.24 to obtain the smaller duration that reflects when this violating coalition will catch up to the competitive coalitions. We can run this IP iteratively until there are no violating coalitions. At that point, we are sure that we have obtained the correct duration, without explicitly minimizing over the exponential number of all possible coalitions. We label this algorithm DUR-CG.

6.4 Epsilon Introduction

Having developed an alternate process by which we can arrive at the APA outcome, we try to introduce an ϵ into our algorithm that parallels the ϵ bid increment of the Pure Proxy algorithm. By introducing an ϵ, we can trade off the accuracy of our outcome with the runtime required. Our hope is that introducing ϵ into the accelerated implementation will allow us to outperform comparable ϵ bid increments

in Pure Proxy. In this manner, we can accelerate the computation of outcomes for both exact Pure Proxy (very small bid increments) and more relaxed Pure Proxy (larger bid increments).

6.4.1 Constraint Generation

ϵFRAC, ϵDUR One natural place to introduce an ϵ into the accelerated implementation is in the constraint generation phases. We have two constraint generation phases consisting of checking for the correct agent fractions (FRAC-CG) and checking for the correct duration (DUR-CG). In each of these cases, we can introduce an ϵ. In FRAC-CG, we we check to make sure no T-interesting coalitions have larger price increase than the competitive T-interesting coalitions. In DUR-CG, we check to make sure no coalitions end at higher revenue if we run the stage for the given duration. Therefore, for FRAC-CG we can introduce an ϵ which means "As long as the maximal price increase is within ϵ of the competitive coalitions, we assume this is not a violation." Similarly, for DUR-CG, we can introduce an ϵ which means "As long as the revenue reached after duration D is within ϵ of that of the competitive coalitions, we assume this is not a violation." As a first cut, we can have these ϵ values be constant. We label these methods ϵFRAC and ϵDUR for checking of the fractions and duration respectively.

per-ϵFRAC As hinted at above, it may not be the best choice to just take a constant ϵ. During some stages, it may be the case that the competitive coalitions are increasing at a slow rate, and therefore, a small constant ϵ might be meaningful. However, when the competitive coalitions are increasing at a high rate, a small constant ϵ may not result in a savings at all. Therefore, it may be better to have ϵ represent a fraction of the price increase of the competitive coalitions. However, we need to be careful here because we may get into a situation where the price increase of the competitive coalitions is very small (and hence a fraction of this price increase will also be very small). To cope with this, we combine the ϵ_1 fraction with a minimal ϵ_2. When we run constraint generation, if the competitive price increase is k^*, as long as the maximal price increase is within $\max(\epsilon_1 k^*, \epsilon_2)$, we assume there are no violations.

6.4.2 Stage Limit

An alternate way of introducing error into our exact accelerated algorithm is to place a cap on the number of constraint generation phases we enter when calculating the fractions for competitive coalitions. We keep entering constraint generation phases as long as we do not have the correct coalitional fractions, but intuitively, after a fixed number of iterations, we should have a good picture of what the correct fractions actually are. Therefore, placing a cap may give a good approximation for what the true fractions are and avoid costly iterations which only result in small perturbations to the fractions. We will refer to this method as STAGE-LIMIT.

Chapter 7

Results

In addition to developing the algorithm for an accelerated version of APA, we have implemented this algorithm in order to compare it with the direct implementation of *i*Bundle(3) with straightforward bidding / APA. To our knowledge, this has been the only extensive experimentation of accelerated implementations of APA with large, randomly generated test cases. Previously, the work of Hoffman et. al., Day et. al., and Wurman et. al. only considered small, hand constructed test cases.

7.1 Experimental Setup

For our experiments, we implemented the proposed accelerated algorithm using *i*Bundle(3) as a starting point. Our experiments were performed on the `deas` blades (`node-3` and `node-4`) servers. To solve the mixed integer linear programs that arose in our algorithm, we used CPLEX version 8.1. In order to maintain uniformity throughout, we switched *i*Bundle(3) to use CPLEX to solve its winner determination problems as well.

For our tests, we implemented two versions of the accelerated algorithm. For the first version, we use FRAC-MIP to determine the coalitional fractions for in Algorithm FRAC. We will label this approach Acc-MIP. We also tried the heuristic LP approach described in Section 6.2. Instead of using FRAC-MIP to solve for the coalitional fractions, we solve the system of linear equations (Equations 5.10 and 5.11) iteratively with an LP. We will label this approach Acc-LP. In both of these

	A	B	AB	C	AC	BC	ABC
Buyer 1	10	3	18	2	18	10	20
Buyer 2	4	9	15	3	12	18	20
Buyer 3	1	3	11	9	16	17	25
Buyer 4	7	7	16	7	16	16	20

Table 7.1: Wurman's Example. By looking across each row, we can determine agent's values for the given bundles. For instance, $v_1(A) = 10, v_3(B) = 3, v_1(AB) = 18$.

implementations, we use the constraint generation algorithms to dynamically generate T-interesting coalitions and to check for the correct duration (FRAC-CG and DUR-CG). We will label the direct implementation by Pure Proxy. Note that when we refer to rounds these will have different meanings for the accelerated implementations and Pure Proxy. For the accelerated implementations, a round refers to an entire stage, while for Pure Proxy, a round refers to a single iteration in the direct implementation.

7.2 Previous Examples in Literature

We first begin with previous small examples that can be found in the literature. Wurman et. al. offer a single small example, and Hoffman et. al. offer a set of six small test cases.

7.2.1 Wurman's Example

We begin with the example from Wurman et. al.[22]. We can represented the submitted XOR bids as a chart in Table 7.1.

Of note in Table 7.2 is that the result of Acc-LP and Acc-MIP is equivalent to the result of Pure Proxy. Though it looks like the final allocation is different, if we look closer, we see that this is just a case of tie-breaking. The result of Pure Proxy and the result of Acc-MIP and Acc-LP generate the same revenue (25.00). We now check to make sure all agents receive equal payoffs. Agent 1 receives the same payoff since she receives the same bundle for the same price in both outcomes. Agent 2 receives payoff $v_2(B) - 8 = 1$ in Pure Proxy, and she receives payoff $v_2(BC) - 17 = 1$ in Acc-MIP and Acc-LP. Agent 3 receives payoff $v_3(C) - 9 = 0$ in Pure Proxy, and she

Method	Rounds	Allocation {Agent-Package}	Prices ($)	Revenue ($)	Value ($)
Pure Proxy	3234	{1-A, 2-B, 3-C}	{8.01,8.01,9.00}	25.02	28.00
Safe Start	51	{1-A, 2-BC}	{7.51, 17.51}	25.02	28.00
Incremental Scaling	39	{1-A, 2-BC}	{7.51, 17.51}	25.02	28.00
Incremental Scaling w/ Safe Start	11	{1-A, 2-BC}	{7.51, 7.51}	25.02	28.00
Wurman et. al.	11	{1-A, 2-BC}	{8.00, 17.00}	25.00	28.00
Vickrey Payments	-	{1-A, 2-BC}	{7.00, 17.00}	24.00	28.00
Day et. al.	5	{1-A, 2-BC}	{7.50, 17.50}	25.00	28.00
Acc-MIP	19 (23, 21)*	{1-A, 2-BC }	{8.00, 17.00}	25.00	28.00
Acc-LP	19 (28, 23, 21)*	{1-A, 2-BC}	{8.00, 17.00}	25.00	28.00

Table 7.2: Results on Wurman's Example. For Acc-MIP, we have (i, c), where i is the number of times we run FRAC-CG, and c is the number of times we run DUR-CG. For Acc-LP, we have (l, i, c) where i and c are the same and l denotes the number of linear programs we solved (number of solutions to NSYS).

receives no bundles in Acc-MIP and Acc-LP so she also receives payoff 0. Therefore, we see that the result does actually mimic Pure Proxy APA.

We also observe that the Pure Proxy results given assume a given bid increment. It is possible that this bid increment is very small, and this is why Pure Proxy takes so many rounds. Indeed, in our later experiments, we will try to take as a large as bid increment as possible as long the result is still efficient.

Comparisons In comparing our algorithm with the others, we should really look at the number of winner determination type problems being solved since the meaning of rounds is different for each algorithm. Acc-MIP solves 23 + 21 = 44 problems that are as hard as the winner determination problem along with 19 MIPs which are harder than the winner determination problem. Acc-LP also solves 44 problems that are as hard as winner determination, but solves 28 pure LPs instead of 19 MIPs.

In comparing with Safe Start and Incremental Scaling, we see that we solve around the same number of winner determination problems. Safe Start with Incremental Scaling appears to do better than our algorithm. Wurman only takes eleven rounds, but within each round, Wurman solves a MIP that is harder than the FRAC-MIP in Acc-MIP. Additionally, we hypothesize that Wurman's algorithm would need to employ the constraint generation techniques from Chapter 6 in order to efficiently generate interesting coalitions and find the acceptable duration. This would result in

the solving for more winner determination type problems for Wurman's algorithm. Therefore, Wurman's algorithm requires comparable if not more computation than Acc-LP and Acc-MIP. Day's method only requires five problems as hard as winner determination, which is fewer than both our algorithms.

We also observe from this example the fact that Acc-MIP and Acc-LP directly replicate the APA outcome. Hoffman's and Day's methods both arrive at different outcomes even in this small example. Wurman's algorithm arrives at the same outcome, but this is a special case where iBundle(2) and iBundle(3) happen to arrive at the same result. Because our result replicates the Pure Proxy result, we expect that our algorithm will take longer than the others which only approximately replicate the result or focus on replicating properties of the result.

We have provided similar charts based on the example set provided by Hoffman et. al. in the Chapter A. In general, we find that we exactly replicate the results of Pure Proxy APA in far fewer rounds. In these examples, auctions that took thousands of rounds take fewer than ten rounds. In comparing with the other algorithms, we do better than Safe Start or Incremental Scaling alone, and do slightly worse than Safe Start with Incremental Scaling. We do not have results of the Wurman and Day algorithms on these other small examples. As stated above, in making these comparisons, we should keep in mind that Acc-LP and Acc-MIP exactly replicate the APA outcomes whereas Hoffman and Day's approaches offer an approximation.

7.3 Direct Comparison

7.3.1 Setup

Aside from running our algorithm on the small examples found in the literature, we have run extensive tests on problems from the Combinatorial Auction Test Suite (CATS) developed by Leyton-Brown et. al.[13]. CATS provides a number of distributions which are representative of classes of problems that are related to combinatorial auctions. The distributions include *paths, matching, scheduling, regions, arbitrary* and various legacy distributions indexed by Li where i is an integer. The named distributions are described in [13] while the legacy distributions were taken

from previous work on tests for combinatorial auctions. The paths distribution represents settings where the goods in question are paths in space. Trucking routes is a good example of this distribution. The matching distribution represents goods which have complementarities in time, such as airport slots. The scheduling distribution represents a formulation of the job-scheduling problem in terms of auctions. The regions distribution describes goods that have a spatial relationship such as rights to oil fields. Finally, the arbitrary distribution represents goods which may have weak relationships that cannot be categorized into the others [13].

For our experiments, we compared Pure Proxy APA with our accelerated implementations. The performance of Pure Proxy was highly dependent on the bid increment chosen, so we show results for multiple bid increments. For each set of results, we chose the bid increment to be a percentage of the average submitted bid for the given problem. We decided to use a percentage instead of a fixed value since the effect of the bid increment depends on the exact magnitude of the bids submitted.

The points in our graphs represent the average runtime over ten random inputs from a given distribution. We define runtime to be the sum of the CPU time used for each algorithm. By using CPU time, we avoid dealing with variations due to system conditions. We annotate each of our graphs with the efficiency of the outcome, which is defined as the value of the outcome divided by the maximal value. For each of our charts, we give an average efficiency for the algorithms. These averages efficiencies were within 0.01 of the true efficiency for all data points for a given algorithm. For algorithms whose efficiencies vary, we annotate the graphs with the efficiencies for each data point.

In choosing bid increments for Pure Proxy, we tried to find the largest possible bid increments that still yielded the desired efficiencies. This offers a fair comparison with the accelerated algorithms. Instead of choosing arbitrarily small bid increments for Pure Proxy (which would make Pure Proxy run for a very long time), we experimentally calibrated to find the largest bid increments that still yield the desired efficiency levels.

7.3.2 Bids

One set of experiments we performed was on the effect of varying the number of bids in our test cases. We use these experiments to shed light on two comparisons. The first comparison is between Acc-MIP and Acc-LP, and the second comparison is between the accelerated implementations and Pure Proxy.

Acc-MIP vs. Acc-LP When we performed experiments on Acc-MIP and Acc-LP, we found that Acc-MIP took significantly more time. For many of the distributions, Acc-MIP took so much longer that graphing its runtime on the same plot would make it difficult to discern the runtime of Acc-LP and the Pure Proxy algorithms. As an example, we consider experiments on the matching distribution.

Table 7.3 gives the runtime and various counts to help describe the amount and kinds of computations taking place. Notice that we do not include the number of stages in the chart because the number of stages are equal for Acc-LP and Acc-MIP (the work required within a stage differs, but the number of stages are equal).

As seen in Table 7.3, the runtime of Acc-MIP is significantly larger than the runtime of Acc-LP. In addition to the runtime comparison, the number of calls to fractional constraint generation (FRAC-CG) recorded in Table 7.3 are very interesting. We see that Acc-LP actually calls FRAC-CG fewer times than Acc-MIP. This implies that by using the heuristic of removing the coalitions which are assigned negative fractions, we actually converge to the correct set of fractions faster than when using the theoretically correct MIP formulation. We see that this is the case with other distributions as well (Appendix B.1). Therefore, it appears that the LP approach is able to zero in on the competitive coalitions faster than the MIP. This accounts for the signficantly worse performance of Acc-MIP. Each time constraint generation is called for the coalitional fractions, we need to resolve the MIP. Therefore, the number of MIPs solved is equal to Frac-CG for Acc-MIP. For the matching distribution, the MIP is actually called more times than the corresponding LP in Acc-LP because of the faster convergence of Acc-LP. Since MIPs are much slower than LPs, based on the counts, there is no way that the runtime of Acc-MIP can compete with that of Acc-LP.

	Runtime(s)		Frac-CG		LPs Solved
Bids	Acc-MIP	Acc-LP	Acc-MIP	Acc-LP	Acc-LP
100	1.4	0.7	38.0	31.2	47.6
200	9.9	2.8	114.3	86.1	125.8
300	120.5	15.1	229.2	155.4	222.8
400	185.1	14.0	346.8	225.2	314.0
500	650.6	18.5	458.6	272.8	370.8

Table 7.3: Frac-CG denotes the number of times constraint generation was called to check the coalitional fractions. LPs solved only applies to Acc-LP and refers to the number of times we solve the system of equations (Equations 5.10 and 5.11).

Acc-LP vs. Pure Proxy Having compared Acc-LP against Acc-MIP when varying the number of bids, we now compare Acc-LP with Pure Proxy, mindful of the fact that Acc-MIP is much worse than Acc-LP. When comparing these results, we make two observations:

- Pure Proxy's performance relative to Acc-LP is dependent on the bid increment. For bid increments that always yield efficiency 1, Pure Proxy is worse. For bid increments that yield average efficiency .99, Pure Proxy does better.
- Acc-LP appears to have a greater dependency on the number of bids than Pure Proxy.

As seen in Figure 7.1, Acc-LP performs better than Pure Proxy with bid increment of one percent, while Acc-LP performs worse than Pure Proxy with a bid increment of ten percent. This shows that Acc-LP arrives at a fully efficient outcome faster than Pure Proxy, since Pure Proxy is only faster when we relax the efficiency to 0.99. Also, for the L6 distribution, Acc-MIP does not perform nearly as bad as for the matching distribution, though it still appears to be worse than Acc-LP. To examine the effect of increasing the number of bids, we refer to Table 7.4 which gives a fuller description of the number of winner determination type problems solved for each algorithm. Notice that both constraint generation phases for Acc-LP require the solution of a winner determination type problem (actually slightly harder because of added constraints).

At first, if we only look at the number of rounds in Table 7.4, it looks like we are saving a lot in Acc-LP. However, each round in Acc-LP requires the solving of multiple winner determination type problems along with a number of LPs. When we refine

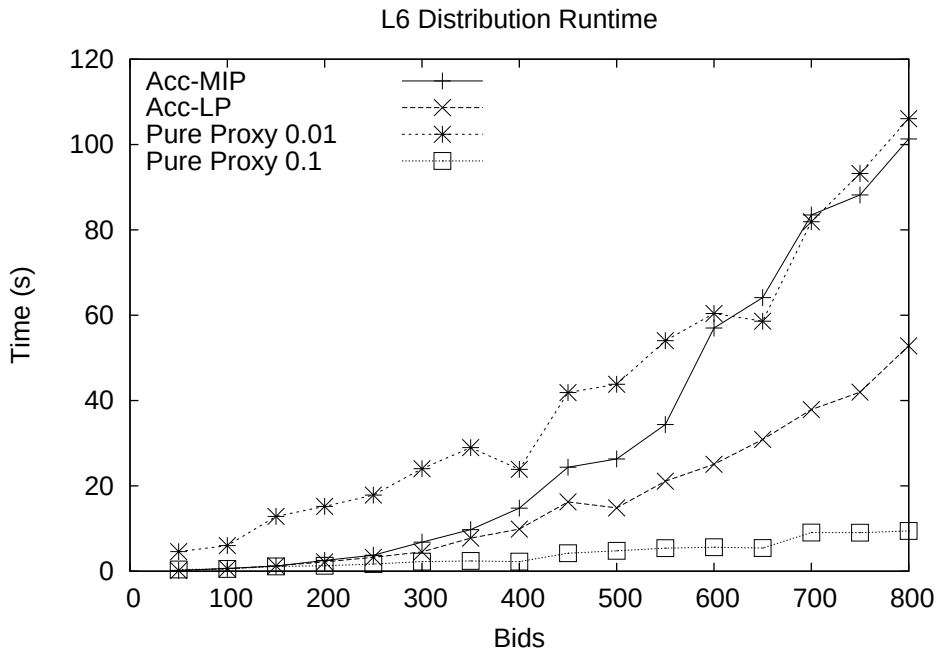

Figure 7.1: L6 Distribution varying Number of Bids. Goods = 50. Bids Per Agent = 10. Efficiency of Acc-MIP = 1. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.10) = 0.99.

	Acc-LP						Pure(0.01)	Pure(0.1)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
100	22.7	15.4	9.6	25.0	0.8	8.8	541.0	50.8
200	54.8	35.6	31.2	66.8	6.8	24.4	770.2	75.0
300	81.7	53.1	49.5	102.6	10.9	38.6	721.8	69.6
400	118.2	75.8	81.7	157.5	24.0	57.7	648.2	65.0
500	135.9	88.9	89.5	178.4	23.1	66.4	907.6	90.6
600	171.7	110.3	119.6	229.9	32.9	86.7	855.6	84.4
700	195.1	124.9	131.5	256.4	33.8	97.7	964.6	93.6
800	219.0	140.0	149.7	289.7	37.7	112.0	853.6	84.6

Table 7.4: L6 Distribution, Goods = 50, Bids Per Agent = 10. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

our observations, we see that as expected, the number of winner determination type problems solved by Acc-LP falls in between Pure Proxy(0.01) and Pure Proxy(0.1).

We also notice from Table 7.4 that the number of rounds for Pure Proxy grows slower with the number of bids than the number of stages for Acc-LP. For 100 bids, Acc-LP solves 25 winner determination problems while for 700 bids, Acc-LP solves around 250, a ten-fold increase. Alternatively, for 100 bids, Pure Proxy(0.01) solves 541.0 winner determination problems while for 700 bids, it solves 964.6 winner determination problems, a 2-fold increase. The same is true of Pure Proxy(0.1). It appears that workload of Acc-LP is much more dependent on the number of bids.

This makes sense when we think about the dependencies of each algorithm. For Pure Proxy, as long as the magnitude of the bids and the bid increment stay the same, the number of rounds should not grow significantly as the number of bids increase. An increase in bids may cause more complex competition, but each round still sees bidders increase their bids by the bid increment. If the bid magnitude does not change (as with this test case), the number of rounds should not increase linearly in the number of bids.

However, for the accelerated implementation, the number of rounds is proportional to the number of bids submitted. This is due to the fact that a new stage begins whenever a change in an agent's demand set occurs. Every new bid that does not yield maximal value for its agent will manifest itself as a demand set change. This is because agents will only bid on the maximal value bundles at the start of the auction. A change in the demand sets will occur when agents actually start bidding on these bundles. Since the number of stages in the accelerated implementation is proportional to the changes in agent's demand set, the number of stages in the accelerated implementation is proportional and constrained by the number of bids. On the other hand, the rounds in Pure Proxy are less dependent on the number of bids. As a result, as we increase the number of bids, we see that the accelerated implementation's work load increases more.

It is also worth noting here that while increasing the bid increment may not affect efficiency, it does affect the final price vector of the outcome. Therefore, even though we may arrive at an equally efficient outcome, there are still benefits to using smaller bid increments and hence, the accelerated implementations (which has infinitesimally small bid increments).

Frequency of Catching Up in Acc-LP One interesting observation about Acc-LP that can be seen in Table 7.4 and tables in Appendix B.2 is the number of actual coalitions that catch up in comparison to the number of rounds. In general, we see that the number of stage changes caused by catching-up is very small. In Table 7.4, about one-fourth of the stage changes are caused by catching-up. On the extreme end, if we look at the chart for the scheduling distribution (Appendix Table B.8), we see that there are virtually zero stage changes caused by catching-up. This implies that when we go through constraint generation to check if we have the correct duration, we will find that there are no violations. However, we have absorbed the cost of running one MIP to obtain this information. Since the number of stage changes caused by collisions is small, it may be beneficial to try to first decide if there is a violation, and only afterwards find the violating coalition. We tried this preliminarily with an LP-relaxation, but this did not seem to help much. Investigating efficient algorithms which simply tell us whether there is a violation could be an area of future research.

7.3.3 Bids Per Agent

In addition to charting the variation of runtime with increasing number of bids, we also tested the effect of increasing the number of bids per agent for the legacy distributions. A bids per agent value is necessary for the legacy distributions because those distributions simply generate bids assuming each agent is single-minded and only submits a single bid.

When we varied the number of bids, we saw that Acc-LP performed better than always efficient Pure Proxy for bids per agent values greater than six, but performed significantly worse for very small bids per agents values. As with the dependencies on the number of bids, this suggests differing dependencies on the number of bids per agent.

Similar to the results we saw with varying the number of bids, we found that varying the number of bids per agent has more of an effect on the accelerated implementation than on Pure Proxy. As seen in Figure 7.2, a small number of bids per agent (meaning a large number of agents) caused a larger slow down in Acc-LP than in Pure Proxy. Even though always efficient Pure Proxy is slower than Acc-LP for larger bids per agent values, Acc-LP is slower for small bids per agent values. This is also reflected in Table 7.5 which details the number of stages and winner

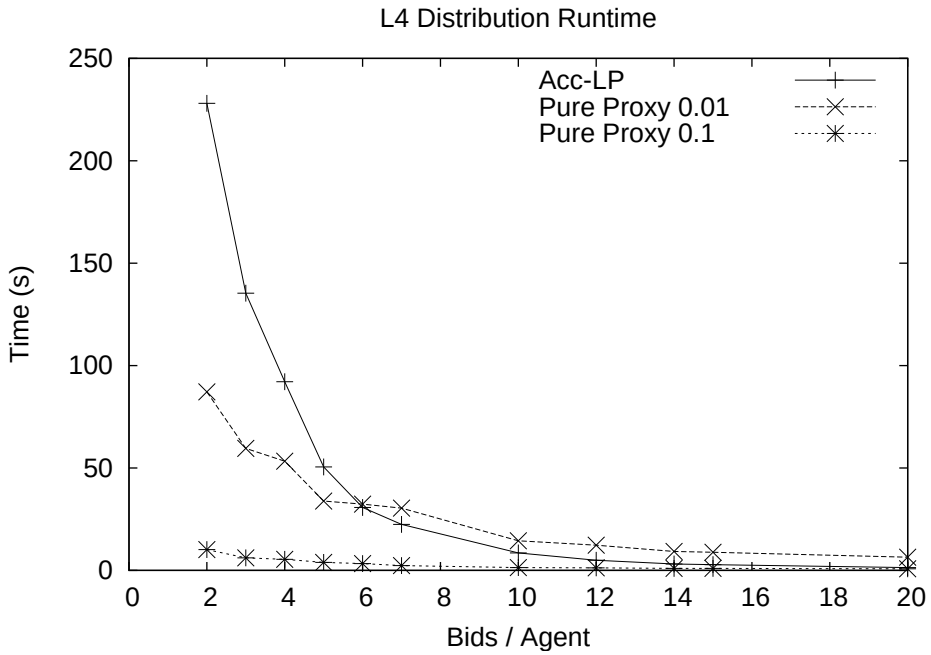

Figure 7.2: L4 Distribution varying Bids per Agent. Goods = 50, Bids = 420. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.1) = 0.99.

	Acc-LP			Pure(0.01)	Pure(0.05)
Bids / Agent	Rounds	LPs	Total-CG	Rounds	Rounds
2	373.8	1294.0	1440.8	789.8	156.6
4	200.3	523.6	661.6	753.8	149.6
6	115.5	296.3	367.1	689.8	136.6
10	60.3	145.7	178.5	578.2	114.8
12	46.2	116.6	140.4	644.8	127.2
14	34.3	88.4	102.9	540.4	106.2
20	21.9	55.5	65.3	560.4	110.0

Table 7.5: L4 Distribution varying Bids Per Agent. Goods = 50, Bids = 420. LPs = number of times we solve for coalitional fractions. Total-CG = total number of times we call constraint generation (to check fractions and durations).

determination-like problems solved in each stage. Increasing the bids per agent drastically decreases the number of stages in Acc-LP from between 200 and 300 to less than 30. Alternatively, for both Pure Proxy runs, increasing bids per agent has a much less significant impact on the number of rounds.

This again can be attributed to the different dependencies of Acc-LP and Pure Proxy. The number of rounds in Acc-LP is dependent on the number of bids submitted and the coalitional structure. A large number of coalitions means there will be more collisions and more violations when we perform constraint generation to check the duration. Since the duration of each stage depends on the number of collisions, a more complicated coalitional structure means that the duration for each stage in Acc-LP is likely to be smaller. Therefore, it makes sense that a small number of bids per agent (a large number of agents) will cause a significant increase in runtime. On the other hand, Pure Proxy is much less dependent on the coalitional structure. Pure Proxy is not as dependent on the coalitional structure because at each round, some coalition wins and all non-winning agents increase their bids. The algorithm terminates when there are no more bids. Regardless of the coalitional structure, from round to round, most bids in Pure Proxy are increasing by the bid increment. Of course the coalitional structure affects the hardness of the winner determination problems solved and the frequency of winning for coalitions, but this effect appears to be much less significant than the impact on Acc-LP.

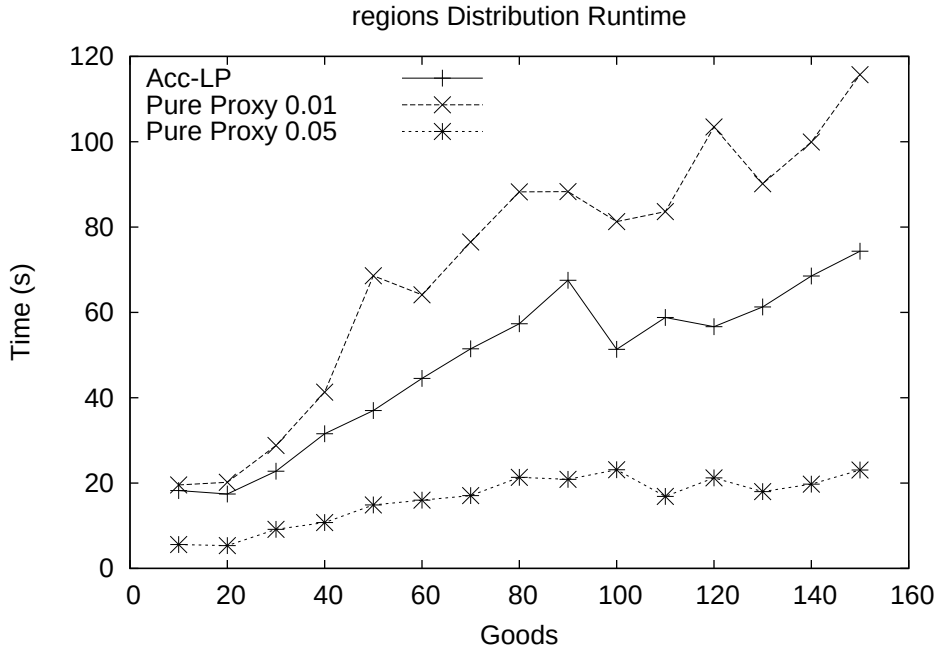

Figure 7.3: regions Distribution varying Goods. Bids = 300. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.05) = 0.98.

7.3.4 Goods

We also performed experiments when varying the number of goods in the test cases. As was the case with varying the number of bids and varying the number of bids per agent, the runtime for Acc-LP falls in between always efficient Pure Proxy and 0.99 efficiency Pure Proxy (see Figure 7.3). However, differing from the previous two cases where Acc-LP was affected more by the changes in variables, it appears that Pure Proxy is affected more when we vary the number of goods.

When we look at the winner determination counts as we did before, we see that Acc-LP is less affected by an increase in the number of goods than Pure Proxy. As seen in Table 7.6, the number of rounds for Pure Proxy doubles when the goods move from 20 to 120 whereas the total winnder determination problems solved by Acc-LP increases by about less than a factor of 1.5. Though this difference is not as drastic as the differences when we varied the number of bids and bids per agent, the number of goods does appear to have a more substantial effect on the performance of Pure

	Acc-LP			Pure(0.01)	Pure(0.05)
Goods	Rounds	LPs	Total-CG	Rounds	Rounds
20	90.1	177.4	231.2	398.6	78.8
40	99.1	209.8	276.8	509.0	99.4
60	103.3	232.5	296.7	606.4	120.8
80	120.1	284.1	365.25	712.8	141.0
100	110.4	253.8	330.0	663.4	132.2
120	113	259.8	339.3	691.2	136.2

Table 7.6: regions Distribution varying Goods. Bids = 300. LPs = number of times we solve for coalitional fractions. Total-CG = total number of times we call constraint generation (to check fractions and durations).

Proxy. Along the same lines, the number of rounds for Acc-LP does not appear to be significantly increasing with the number of goods.

Again, we can think of these findings with respect to the computations required by each mechanism. When the number of goods increases, it is more likely that bidders are bidding on different goods. Hence, the coalitions that are formed will be larger. This would make Pure Proxy take more rounds since larger coalitions means more agents are winning in each round. This means it will take longer for agents to reach the point when they bid their entire valuations since they are winning more often. For Acc-LP, the effect of increasing the number of goods is not as strong. Acc-LP depends more on the coalitional structure and the number of bids. It does not matter as much to Acc-LP the size of the coalitions since this will affect the specific magnitude of the fractions generated but not the duration or the number of stages.

We have seen with the direct comparisons that Acc-LP performs better than always efficient Pure Proxy, but when Pure Proxy is given some leeway for efficiency, Pure Proxy performs better. Also, we have identified the different dependencies of the algorithms. Acc-LP is dependent on the number of bids and the number of agents while Pure Proxy is more dependent on the number of goods and the magnitude of bids.

7.4 Approximations

As discussed in Section 6.4.1, there are natural ways that we can introduce error into constraint generation. The three different epsilons that we discussed were ϵFRAC, ϵDUR, and per-ϵFRAC. We present results for some experimentation with each of these error introducing methods. We were unfortunately unable to test out the STAGE-LIMIT technique also mentioned in Section 6.4.1. All of our tests are performed on Acc-LP. For each ϵ type, we fixed ten random test cases on which we ran Acc-LP for different ϵ values. We first present results of trials on the arbitrary distribution.

7.4.1 ϵFRAC

The first ϵ that we discussed was ϵFRAC. This was the constant ϵ introduced into the constraint generation that checked the coalitional fractions. As seen in Figure 7.4, the ϵ values we try will be proportional to the sizes of the competitive coalitions. In general, the reasoning behind this is that the price increase of a coalition will be proportional to the number of agents. Therefore, since FRAC-CG checks for violation of price increase, the magnitude of ϵ will be similar to coalitional sizes.

In Figure 7.4, it appears that increasing ϵ does not bring about a significant savings in runtime. However, when we investigate further, we see that the ϵ is indeed having an effect. From Table 7.7, we can see that as we increase ϵ, the number of times we call constraint generation decreases monotonically. The ϵ is indeed lessening the workload of Acc-LP. However, we see that this ϵ can only help so much. A lower bound on the number of times we call FRAC-CG is the number of stages. As seen in the chart, as we increase ϵ, the number of calls to FRAC-CG approaches this lower bound. This embodies the fact that even if we tolerate a large error, we still have to run the costly IP once to find out that we are not in violation. As a result, we do not see a large savings in runtime because we still check in each stage to make sure we are not in violation. The number of stages does not decrease significantly since this ϵ does not have an impact on either the duration of the stage nor the entrance of new bundles in agent's demand sets. Since these are the events that define new stages, this ϵ will not have a marked effect on the number of stages.

We also notice from Figure 7.4 that the efficiency does not seem to be affected

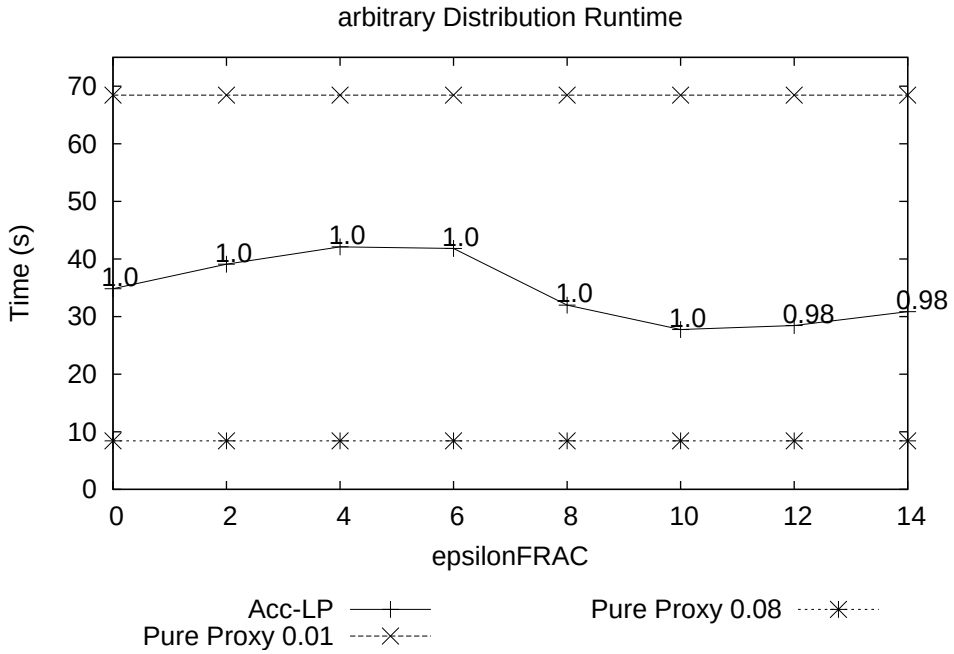

Figure 7.4: arbitrary Distribution varying ϵ-FRAC. Goods = 50, Bids = 300. Efficiency of Acc-LP is labeled. Efficiency of Pure Proxy(0.01) = 0.99. Efficiency of Pure Proxy(0.1) = 0.98.

Epsilon	Frac-CG	Dur-CG	Total-CG	Rounds
0	97.2	96.1	193.3	77.5
2	82.0	100.6	182.6	79.3
4	79.8	97.2	177.0	77.8
6	75.8	90.1	165.9	74.5
8	75.6	89.3	164.9	74.5
10	72.3	83.5	155.8	71.8
12	71.9	82.7	154.6	71.7
14	71.1	81.7	152.8	71.1

Table 7.7: arbitrary Distribution varying ϵFRAC. Goods = 50, Bids = 300. Frac-CG = total number of times we call constraint generation to check fractions. Dur-CG = total number of times we call constraint generation to check durations.

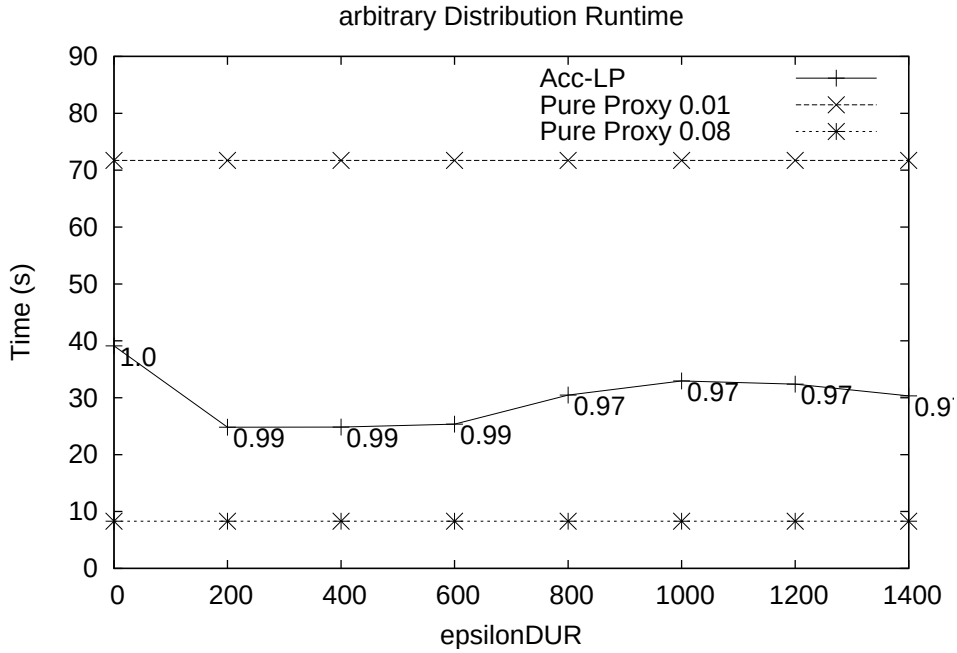

Figure 7.5: arbitrary Distribution varying ϵDUR. Goods = 50, Bids = 300. Efficiency of Acc-LP is labeled. Efficiency of Pure Proxy(0.01) = 0.99. Efficiency of Pure Proxy(0.08) = 0.98.

much by an increase in ϵ. This is probably due in part to the small effect of the ϵ, but it also points to the robustness of the accelerated implementation. Even if we do not have the exactly correct coalitional fractions, it appears that the outcome will still be reasonably efficient.

7.4.2 ϵDUR

Another error introduction we discussed was inserting an ϵ into the constraint generation process for checking duration (DUR-CG). If the coalition with maximal revenue after the proposed duration is within ϵ of the maximal revenue competitive coalition, we assume that this is not a violation. Notice that these ϵ values will have magnitude proportional to the magnitude of bids submitted since the unit of concern here is revenue. As seen in Figure 7.5, increasing this ϵ value does indeed improve the runtime at the cost of lower efficiency. We see a runtime savings of around ten seconds, or twenty-five percent when moving from an ϵ of 0 to an ϵ of 200. This drop is reflected

Epsilon	Frac-CG	Dur-CG	Total-CG	Rounds
0	106.5	110.9	217.4	83.9
200	99.0	61.2	160.2	60.9
400	98.7	60.7	159.4	60.6
600	98.7	60.7	159.4	60.6
800	98.6	60.5	159.1	60.5
1000	98.6	60.5	159.1	60.5

Table 7.8: arbitrary Distribution varying ϵDUR. Goods = 50, Bids = 300. Frac-CG = total number of times we call constraint generation to check fractions. Dur-CG = total number of times we call constraint generation to check durations.

in Table 7.8 which shows the significant decrease in the number of rounds and calls to constraint generation as we move from a 0 ϵ to an ϵ of 200.

In this case, we see a significant decrease in the number of stages due to introducing ϵ. This makes sense since $\epsilon - DUR$ deals with DUR-CG, and therefore, adding more leeway here allows for longer durations. However, once again, we see that increasing ϵ further does not improve the runtime since we run into the same problem of reaching the lower bound. For large ϵ in this case, we see that the number of times we call constraint generation for the duration is exactly the number of stages. Again, even though there are no violations, we still have to call constraint generation for the duration once in each stage.

In terms of efficiency, we again see that larger ϵ values decrease the efficiency, but the efficiencies for extremely large values are still reasonably good (0.97). One explanation for this would be that the impact of the ϵ is buffered by the lower bound on computation described above. Alternatively, it may be that the accelerated implementation has built-in robustness to these errors we have introduced.

7.4.3 per-ϵFRAC

We also discussed introducing error into the checking of coalitional fractions by using a fractional ϵ instead of a constant. The amount of error we will tolerate now becomes a fraction of the current price increase of the competitive coalitions. As a result, this ϵ_1 will have units that represent what percent of the current price increase we are willing to tolerate. In Section 6.4.1, we describe a second ϵ_2 to be used in combination

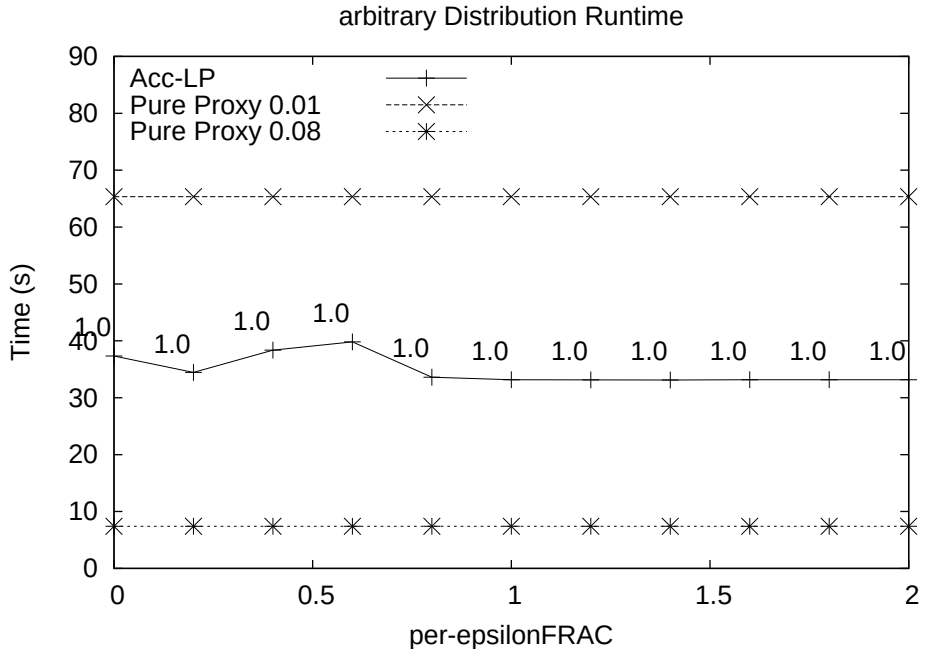

Figure 7.6: arbitrary Distribution varying per-ϵFRAC. Goods = 50, Bids = 300. Efficiency of Acc-LP is labeled. Efficiency of Pure Proxy(0.01) = 0.99. Efficiency of Pure Proxy(0.08) = 0.98.

with the percentage epsilon. For our experiments, we choose a minimal ϵ_2 to isolate the effects of varying ϵ_1. A value of 0.5 for ϵ_1 means that we are willing to tolerate price increase that are 1.5 times the current competitive price increase.

From Figure 7.6, we see once again that increasing ϵ does not have a significant impact on the runtime. This follows from our analysis for ϵFRAC since this ϵ also affects only FRAC-CG and does not impact the duration of the stage or the number of stages.

When we examine Table 7.9, we see that increasing ϵ does decrease the amount of work performed, though not significantly enough to make a big difference in runtime. We also notice from Table 7.9 that we seem to hit a wall once we raise ϵ beyond 1. This makes sense since it is not very likely that we will find in FRAC-CG coalitions that have price increase that is twice as large as the current competitive price increase. As was the case with ϵFRAC and ϵDUR, per-ϵFRAC also does not appear to have a significant impact on efficiency.

Epsilon	Frac-CG	Dur-CG	Total-CG	Rounds
0.0	107.6	109.0	216.6	84.5
0.2	90.6	113.6	204.2	85.7
0.4	88.4	112.3	200.7	85.3
0.6	86.4	108.4	194.8	83.7
0.8	85.8	107.7	193.5	83.4
1.0	83.9	104.1	188.0	81.9
1.2	83.9	104.1	188.0	81.9
1.4	83.9	104.1	188.0	81.9
1.6	83.9	104.1	188.0	81.9
1.8	83.9	104.1	188.0	81.9
2.0	83.9	104.1	188.0	81.9

Table 7.9: arbitrary Distribution varying per-ϵDUR. Goods = 50, Bids = 300. Frac-CG = total number of times we call constraint generation to check fractions. Dur-CG = total number of times we call constraint generation to check durations.

7.4.4 Analysis of ϵ Introduction

From our tests, we see that the ϵ errors we introduced did not have significant effects on efficiency. We did see some effects on runtime, especially with ϵDUR which impacted the duration of the stages.

By further investigating the amount of work performed, we saw that our error introductions were indeed decreasing the number of IPs solved. However, we saw that we reached a lower bound which prevented our methods from further increasing runtime. Namely, constraint generation requires a single call even when there are no violations. Therefore, for every stage, we need to solve two IPs (one in each constraint generation phase) even if there are no violations. The errors we introduced did not affect the number of stages, so we did not see drastic improvements in runtime. If we look at Table 7.10 copied from the Appendix, we see that starting out, we do not find many violations in constraint generation when using the arbitrary distribution. For our test case of 300 goods, we see that the average number of stages was 83.9, and there were 105.5 calls to constraint generation for coalitional fractions and 110.4 calls to constraint generation for the duration. This means that more often than not constraint generation tells us there are no violations. Hence, there is not much room for improvement by trying to increase our tolerance for violation since most of the them are not violating. Indeed, by looking at the charts in Appendix B.2, we see that many distributions show this behavior. This may explain why the errors we

	Acc-LP						Pure(0.01)	Pure(0.08)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
100	54.6	36.3	33.1	69.4	6.2	26.9	520.0	62.6
200	107.7	71.7	70.3	142.0	14.2	56.1	648.2	77.6
300	157.9	105.5	110.4	215.9	26.5	83.9	578.6	70.6
400	200.2	132.4	138.8	271.2	31.7	107.1	596.4	72.6
500	270.0	174.5	189.8	364.3	47.5	142.3	645.8	80.0

Table 7.10: arbitrary Distribution varying Bids. Goods = 50. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

introduced did not have significant impacts on the runtime.

This suggests three areas that may be researched further. The first is to find problems where constraint generation finds more violations. This would allow the ϵ introductions to play a larger role since we would be able to decrease the number of violations. The second potential area is to develop efficient algorithms that can simply tell us "yes" or "no" as to whether there are any violations. Our IP in FRAC-CG and DUR-CG tell us "yes" or "no" in addition to giving us the actual coalition that violates. This extra effort is costly when there are no violating coalitions most of the time. If we had a more efficient algorithm to determine if there are violations, we would only have to incur the cost of the FRAC-CG and DUR-CG IP after knowing that there are violating coalitions. The third direction is to try to find some way to decrease the number of stages. This may potentially be done by placing a lower limit on the duration of a stage or introducing an ϵ into agent's demand sets.

Chapter 8

Concluding Remarks

Combinatorial auctions remain a promising area which may have widespread practical applications. By allowing the bidder to specify more information than simply her valuation on single goods, combinatorial auctions can allow for better outcomes and give bidders more power to express their preferences. With these benefits however comes the cost of computing the outcomes in combinatorial auctions. Because the number of possible bundles is exponential in the number of goods, the computations associated with combinatorial auctions are non-trivial.

In this thesis, we focused on one type of combinatorial auction, the ascending proxy auction or iBundle(3) with straightforward bidding. This auction exemplified the tradeoffs between the quality of our outcome and computational efficiency. The ascending proxy auction always terminates with a *core* outcome (no set of agents can dispute the outcome), yet the direct implementation requires solving numerous NP-complete problems to arrive at this outcome. This thesis provided an alternative to the direct implementation, and we have shown that for given problem sizes and types, our accelerated algorithm performs better than the direct implementation. We hope that addressing the computational issues of combinatorial auctions will help make them practical choices for resource allocation.

8.1 Brief Review

In Chapter 1 we introduced some settings where combinatorial auctions may have potential applications as motivation. We saw that combinatorial auctions may be used in resource allocation problems such as the auctioning of take-off and landing slots at airports and the planning of trucking routes. In Chapter 2, we defined the combinatorial auction problem that we address in this thesis. We provided a formal framework in which to abstractly model the combinatorial auction problem. Having defined the problem, we also developed the notion of *core* outcomes, and showed why this notion may be a good indicator of the quality of an outcome. Finally, we presented the NP-complete winner determination problem, which is at the center of the computational issues related to combinatorial auctions. The winner determination problem embodies the inherent computational difficulties of this setting.

In Chapter 3, we introduced one of the first and most famous combinatorial auction mechanisms, the VCG mechanism. We drew an analogy to the single good setting Vickrey auction or second-price auction, and showed that the VCG mechanism was the natural extension to the combinatorial setting. We developed the notions of *allocative efficiency* and *strategyproofness*, and saw that VCG outcomes were always efficient and the mechanism was strategyproof. However, we also cited how the VCG mechanism could produce counter-intuitive outcomes. Some of these included zero seller revenue (the agents receive goods for free) and the susceptibility to collusion by multiple bidders. We noticed that the reason for these deficiencies was that the VCG outcome is not always a core outcome. Additionally, we introduced the notion of preference elicitation and saw that the VCG mechanism suffered from costly preference elicitation.

Having seen the various issues that arise in designing combinatorial auctions through the VCG mechanism, we moved to a newer set of combinatorial auctions in Chapter 4. We first introduced iterative combinatorial auctions which address the problem of costly preference elicitation. We then developed the ascending proxy auction as a particular iterative combinatorial auction coupled with a straightforward bidding strategy. We proceeded to a formal definition of the ascending proxy auction, and discussed its various properties. Most importantly, we saw that APA outcomes are always in the core.

Having developed APA theoretically, we then proceeded to discuss its actual implementation. We looked at the straightforward direct implementation of APA (Pure Proxy). We saw that it was dependent on the chosen bid increment and required iteratively solving a number of winner determination problems. Because of the intractability of winner determination problems, we motivated the development of accelerated algorithms that avoided these large amounts of computation. We discussed related work that approximately replicated the ascending proxy outcome. Finally, we turned to the work of Wurman and Parkes in accelerated exact replication of APA.

As this thesis grew out of Parkes's ideas about exact replicatio, this naturally led to a description of our algorithm in Chapter 5. The key notions that we developed early in this chapter were the notions of interesting coalitions and stages. Interesting coalitions allowed us to focus on only those coalitions that contributed to the dynamics of the auction. The notion of the stage served as the foundation for our accelerated algorithm. By splitting the auction into stages, we could simulate a large number of rounds at once by performing some pre-computation. This led to a description of FRAC, which was our algorithm for determining dynamics within a stage. Once we had the output of FRAC, we needed to determine when the stage would end. To address this issue, we introduced Algorithm DUR. At the end of this chapter we provided some illustrative examples that show some of the more complicated aspects of our algorithm.

Having described our algorithm on a theoretical level, Chapter 6 addressed some of the computational issues that arise from our algorithm. We showed how we could compute the interesting coalitions dynamically as needed using constraint generation (FRAC-CG). We also introduced a heuristic method for computing the coalitional fractions which replaced a the costly mixed integer linear program (FRAC-MIP) with an efficient a linear program. We then also applied constraint generation to computing the durations of each stage (DUR-CG). Finally, in order to mirror the ϵ bid increment in Pure Proxy, we hypothesized about some possible ways to systematically introduce error into our algorithm. Chapter 7 contained experimental results which are summarized below.

8.2 Summary of Results

We first compared the heuristic LP based approach with the theoretically correct MIP approach. We found that the LP based approach significantly outperformed the MIP approach. Not only were we solving easier problems (LPs instead of MIPs), there were instances in which we were solving fewer LPs. While the MIP approach provides a good theoretical foundation for the accelerated algorithm, the LP approach fares much better in practice.

We also compared our LP based approach with the direct Pure Proxy approach. Our findings in this area were very interesting as the two algorithms which produce the same result have very different dependencies. The accelerated approach was highly dependent on the number of bids and the number of agents while the Pure Proxy approach was highly dependent on the number of goods and the magnitude of bids. The accelerated approach was more dependent on the number of coalitions whereas the Pure Proxy was more dependent on the size the coalitions. These different dependencies suggest settings where the accelerated algorithm would be preferred over Pure Proxy and vice versa.

8.2.1 Bottlenecks

In experimenting with the accelerated algorithm and especially with error introduction in Section 7.4, we have also been able to get a feel for the bottlenecks in the algorithm. As seen in the experimentation with error introduction, one of the bottlenecks we found was that constraint generation could be wasteful when we do not have any violations. The constraint generation paradigm requires that we run an IP at least once even if there are no violations. In practice, the catching-up phenomena does not occur that often. If we had an efficient way to check whether a violation exists, this would greatly improve the runtime.

Additionally, another bottleneck of the accelerated algorithm is a small stage duration. We put a lot of work into computing the coalitional fractions, but it may be that these coalitional fractions only apply for a short duration. Then we need to recalculate the coalitional fractions which is expensive. The accelerated algorithm works best when there are few stages and each stage lasts for a long duration. Since

computing coalitional fractions is more expensive than solving the winner determination problem, if our durations are smaller than the bid increment in Pure Proxy, we would be better off just using Pure Proxy.

A final bottleneck was the number of stages that we were required to go through. As discussed, the number of stages grows linearly in the number of bids since each bid that is not of maximal value will cause a demand set change will cause a new stage in the accelerated algorithm.

8.3 Open Questions and Future Research

Our work has created many open questions and potential areas for future research. They include:

- Exploration of whether the coalitional fractions that equalize price increase are unique.
- Exploration of the convergence of the LP based heuristic approach. Theoretical bounds on the expected convergence time.
- The meaning of a negative coalitional fraction. How can this information help us improve the accelerated algorithm?
- Attempt to develop a "yes" or "no" algorithm that tells us whether or not a violation occurs. This would allow for the avoidance of expensive constraint generation even when there are no violations.
- Experiment with more ways to introduce error into the algorithm. Specifically, we should focus on the bottlenecks of short stage duration and large number of stages. How can we systematically enforce long stage durations without completely changing the dynamics of the algorithm? Related to this question is whether there is some way to systematically decrease the number of stages.
- Improvements to the exact accelerated algorithm. Figure out how we can make stage durations longer. Perhaps some changes in agent's demand sets are not as significant as others. Maybe some changes do not affect the competitive

dynamics at all. In general, how can we relax the stage definition so that stages can last longer yet still exactly replicate the ascending proxy auction.

8.4 Conclusion

The accelerated algorithm offers an exciting new set of open questions since its dependencies and characteristics are very different than Pure Proxy. At this point, we have seen that the accelerated implementation is useful if we want to be guaranteed to have fully efficient outcomes. Additionally, the accelerated implementation will perform well in situations with a large number of goods and relatively sparse bids. Moreover, the accelerated implementation removes the need to calibrate since it is independent of epsilon. To run Pure Proxy, it is necessary to run preliminary calibration trials to determine what bid increments to use to attain a desired efficiency level. Of course we could always use incredibly small bid increments, but this drastically increases the runtime of Pure Proxy. In cases where calibration may be expensive, the accelerated implementation may prove to be very useful. While it may not be a completely dominant algorithm, our new accelerated algorithm offers an alternative to direct Pure Proxy and may be more appropriate for certain problem instances and settings.

Appendix A

Hoffman's Test Cases

In this section, we give a complete record of running Acc-LP and Acc-MIP on the examples from Hoffman et. al. [11]. For each of the test cases, the efficient allocation is indicated by an asterisk next to the package. Also, for Acc-LP and Acc-MIP we include extra counts in parenthesis next to the round counts. For Acc-LP, we include (l, f, c) where l stands for the number of LPs solved, f is the number of times constraint generation is called for checking coalitional fractions, and c is the number of times constraint generation is called for checking for collisions. For Acc-MIP, we include (f, c) since we we do not use an LP in that formulation. Unlike Acc-LP, for Acc-MIP, the number of times we solve the MIP is equal to the number of times constraint generation is called for checking coalitional fractions. For Acc-LP, it might be the case that we solve more than one LP for each call to constraint generation since we need to resolve after deleting all negative fractions.

Table A.1: Case 1: AAS satisfied, BSM satisfied

Agent	1	2		3	4	
Package	AB*	AB	C*	AB	AB	C
Value	15	14	5	9	10	4

Optimal Allocation {1-AB, 2-C}; Optimal Value = 20, Optimal Revenue = 17.02

Method	Rounds	Revenue	Prices paid by winning agents	
			Agent 1, {AB}	Agent 2, {C}
Pure Proxy	2450	17.02	13.01	4.01
Safe Start	1	17.01	13.00	4.01
Incremental Scaling	31	17.02	13.01	4.01
Incremental Scaling w/ Safe Start	7	17.02	13.01	4.01
Vickrey Payments	-	17	13.00	4.00
Acc-MIP	4(11,4)	17	13.00	4.00
Acc-LP	4(16,11,4)	17	13.00	4.00

Table A.2: Case 2: AAS satisfied, BSM not satisfied

Agent	1	2	3	4	5
Package	AB	BC	C	C*	AB*
Value	21	35	14	20	22

Optimal Allocation {4-C, 5-AB}; Optimal Value = 42, Optimal Revenue = 35.02

Method	Rounds	Revenue	Prices paid by winning agents	
			Agent 4, {C}	Agent 5, {AB}
Pure Proxy	4025	36.76	15.75	21.01
Safe Start	1	35.02	14.01	21.01
Incremental Scaling	38	35.01	14.00	21.01
Incremental Scaling w/ Safe Start	6	35.02	14.01	21.01
VCG Payments	-	35	14.00	21.00
Acc-MIP	4(8,4)	35	15.75	21.00
Acc-LP	4(12,8,4)	35	15.75	21.00

Table A.3: Case 3: AAS satisfied, BSM not satisfied

Agent	1	2	3	4	5
Package	AB*	CD	CD*	BD	AC
Value	10	20	25	10	10

Optimal Allocation {1-AB, 3-CD}; Optimal Value = 35, Optimal Revenue = 20.02

Method	Rounds	Revenue	Prices paid by winning agents	
			Agent 1, {AB}	Agent 3, {CD}
Pure Proxy	3250	27.52	7.51	20.01
Safe Start	1	20.02	0.01	20.01
Incremental Scaling	18	20.02	0.01	20.01
Incremental Scaling w/ Safe Start	7	20.02	0.01	20.01
Vickrey Payments	-	20.00	0.00	20.00
Acc-MIP	3(6,3)	27.50	7.50	20.00
Acc-LP	3(9,6,3)	27.50	7.50	20.00

Table A.4: Case 4: AAS not satisfied

Agent	1		2		3
Package	A*	B	A	B*	AB
Value	16	16	8	8	10

Optimal Allocation {1-A, 2-B}; Optimal Value = 24, Optimal Revenue = 10.02

Method	Rounds	Revenue	Prices paid by winning agents	
			Agent 1, {A}	Agent 2, {B}
Pure Proxy	1500	10.02	5.01	5.01
Safe Start	401	10.02	6.01	4.01
Incremental Scaling	19	10.02	5.01	5.01
Incremental Scaling w/ Safe Start	9	10.02	6.01	4.01
Vickrey Payments	-	2	2.00	0.00
Acc-MIP	2(4,2)	10	5.00	5.00
Acc-LP	2(6,4,2)	10	5.00	5.00

Table A.5: Case 5: AAS not satisfied

Agent	1		2		3		4	5
Package	AB*	C	BC	B	AC	C	AB	C*
Value	15	5	15	5	12	3	12	6

Optimal Allocation {1-AB, 5-C}; Optimal Value = 21, Optimal Revenue = 17.02

Method	Rounds	Revenue	Prices paid by winning agents	
			Agent 1, {AB}	Agent 2, {C}
Pure Proxy	1890	17.02	12.01	5.01
Safe Start	101	17.00	13.00	4.00
Incremental Scaling	23	17.01	12.00	5.01
Incremental Scaling w/ Safe Start	6	17.02	13.01	4.01
Vickrey Payments	-	15	12.00	3.00
Acc-MIP	7(12,7)	17.11	12.00	5.10
Acc-LP	7(17,12,7)	17.11	12.00	5.10

Table A.6: Case 6: AAS not satisfied

Agent	1	2	3	4
Package	AB	BC*	AC	A*
Value	20	26	24	16

Optimal Allocation {2-BC, 4-A}; Optimal Value = 42, Optimal Revenue = 24.02

Method	Rounds	Revenue	Prices paid by winning agents	
			Agent 2, {BC}	Agent 4, {A}
Pure Proxy	3100	24.02	12.01	12.01
Safe Start	801	24.02	16.01	8.01
Incremental Scaling	20	24.02	17.01	7.01
Incremental Scaling w/ Safe Start	15	24.02	16.01	8.01
Vickrey Payments	-	8.00	8.00	0.00
Acc-MIP	3(5,3)	24.00	12.00	12.00
Acc-LP	2(7,5,3)	24.00	12.00	12.00

Appendix B

Varying Bids

Here we provide the results of varying bids on distributions other than L6 distribution which was contained the main text. For the most part, we see the same trends in all of these graphs and charts. For the graphs where including the runtime of Acc-MIP would cause distortion (Acc-MIP's runtime was signficantly larger than all others) we do not include Acc-MIP. We see that:

- Acc-MIP has worse asymptotics than Acc-LP as expected.
- Acc-LP falls in between Pure Proxy that is always efficient and Pure Proxy that has average efficiency .99.

B.1 Acc-LP vs. Acc-MIP

In this section, we include charts similar to the one included in the main text for distributions other than matching. Notice that in all cases, Acc-MIP takes more time than Acc-LP. Also, it looks like Acc-MIP goes through more constraint generation phases to check fractions than Acc-LP. This again indicates that Acc-LP converges faster than the Acc-MIP by using the heuristic of throwing out non-competitive coalitions.

	Runtime(s)		Frac-CG		LPs Solved
Bids	Acc-MIP	Acc-LP	Acc-MIP	Acc-LP	Acc-LP
100	0.261	0.213	11.2	11.1	16.4
200	1.587	1.438	35.4	33.6	50.5
300	5.893	3.668	60.1	56.4	90.3
400	42.551	9.464	110.0	104.6	170.0
500	45.564	15.515	128.1	107.6	173.2

Table B.1: L4 Distribution varying Bids. Goods = 50, Bids per Agent = 10. Frac-CG denotes the number of times constraint generation was called to check the coalitional fractions. LPs solved only applies to Acc-LP and refers to the number of times we solve the system of equations (Equations 5.10 and 5.11).

	Runtime(s)		Frac-CG		LPs Solved
Bids	Acc-MIP	Acc-LP	Acc-MIP	Acc-LP	Acc-LP
200	2.527	2.193	37.4	35.6	54.8
400	14.773	9.881	77.4	75.8	118.2
600	57.002	25.075	120.6	110.3	171.7
800	101.297	52.824	149.4	140.0	219.0

Table B.2: L6 Distribution varying Bids. Goods = 50, Bids per Agent = 10. Frac-CG denotes the number of times constraint generation was called to check the coalitional fractions. LPs solved only applies to Acc-LP and refers to the number of times we solve the system of equations (Equations 5.10 and 5.11).

	Runtime(s)		Frac-CG		LPs Solved
Bids	Acc-MIP	Acc-LP	Acc-MIP	Acc-LP	Acc-LP
200	0.322	0.215	10.8	10.8	15.6
400	0.988	0.473	24.5	20.1	30.2
600	2.158	0.878	62.8	26.4	41.4
800	3.365	1.339	88.8	34.0	54.5
1000	4.473	1.705	109.9	36.7	58.3

Table B.3: scheduling Distribution varying Bids. Goods = 50. Frac-CG denotes the number of times constraint generation was called to check the coalitional fractions. LPs solved only applies to Acc-LP and refers to the number of times we solve the system of equations (Equations 5.10 and 5.11).

	Runtime(s)		Frac-CG		LPs Solved
Bids	Acc-MIP	Acc-LP	Acc-MIP	Acc-LP	Acc-LP
100	5.603	4.42	45.0	42.1	63.1
200	24.083	18.94	86.1	88.3	137.3
300	77.312	39.114	142.7	131.1	206.8
400	132.797	61.639	204.6	176.7	280.4
500	515.186	110.221	266.0	240.3	381.0

Table B.4: regions Distribution varying Bids. Goods = 50. Frac-CG denotes the number of times constraint generation was called to check the coalitional fractions. LPs solved only applies to Acc-LP and refers to the number of times we solve the system of equations (Equations 5.10 and 5.11).

	Runtime(s)		Frac-CG		LPs Solved
Bids	Acc-MIP	Acc-LP	Acc-MIP	Acc-LP	Acc-LP
100	4.824	3.344	37.6	36.3	54.6
200	24.261	14.614	76.3	71.7	107.7
300	52.147	35.022	112.5	105.5	157.9
400	134.963	65.108	142.8	132.4	200.2
500	310.433	120.075	191.5	174.5	270.0

Table B.5: arbitrary Distribution varying Bids. Goods = 50. Frac-CG denotes the number of times constraint generation was called to check the coalitional fractions. LPs solved only applies to Acc-LP and refers to the number of times we solve the system of equations (Equations 5.10 and 5.11).

	Runtime(s)		Frac-CG		LPs Solved
Bids	Acc-MIP	Acc-LP	Acc-MIP	Acc-LP	Acc-LP
20	0.373	0.223	13.4	12.6	18.9
40	2.633	1.051	59.5	44.8	68.0
60	17.701	2.233	105.7	74.4	112.7
80	107.646	4.629	201.3	137.4	202.3
100	263.481	7.11	225.3	151.8	220.5

Table B.6: paths Distribution varying Bids. Goods = 10. Frac-CG denotes the number of times constraint generation was called to check the coalitional fractions. LPs solved only applies to Acc-LP and refers to the number of times we solve the system of equations (Equations 5.10 and 5.11).

B.2 Acc-LP vs. Pure Proxy

In this section, we provide graphs and charts similar to the comparisons we made using the L6 distribution in the main text. Notice here that Acc-LP usually falls in between Pure Proxy with different bid increments. There are certain exceptions to this such as the paths and scheduling distributions. Also, when we look at the number of coalitions that actually catch up in the charts, we see that very few of the stages end because of catching up. The charts also show that Acc-LP is much more dependent on the number of bids than Pure Proxy.

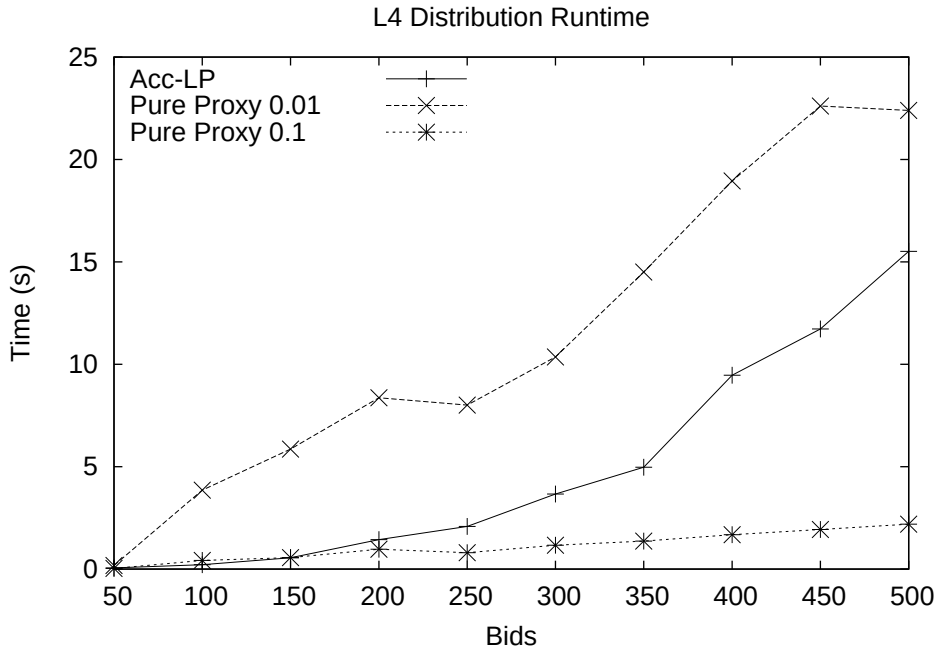

Figure B.1: L4 Distribution varying Number of Bids. Goods = 50. Bids Per Agent = 10. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.10) = 0.99.

	Acc-LP						Pure(0.01)	Pure(0.1)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
100	16.4	11.1	4.5	15.6	0.0	4.5	627.2	59.8
200	50.5	33.6	22.9	56.5	4.6	18.3	703.6	67.0
300	90.3	56.4	49.8	106.2	15.1	34.7	574.2	54.2
400	170.0	104.6	104.8	209.4	39.3	65.5	695.0	69.0
500	173.2	107.6	110.1	217.7	37.0	73.1	650.2	62.8

Table B.7: L4 Distribution, Goods = 50, Bids Per Agent = 10. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

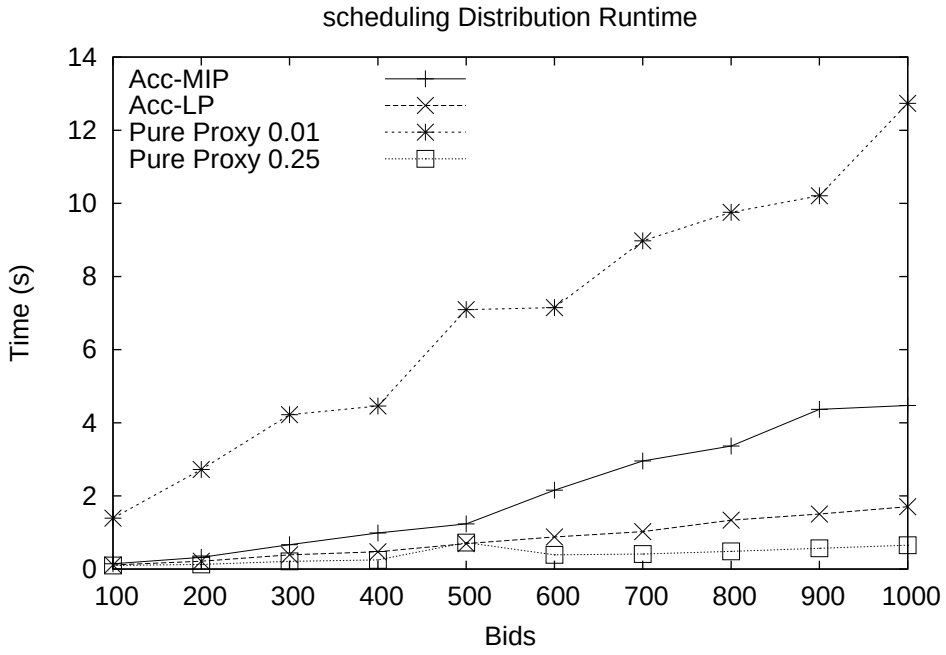

Figure B.2: scheduling Distribution varying Number of Bids. Goods = 50. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.25) = 0.98.

	Acc-LP						Pure(0.01)	Pure(0.25)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
200	15.6	10.8	5.8	16.6	0.0	5.8	438.4	18.8
400	30.2	20.1	11.1	31.2	0.0	11.1	503.2	25.8
600	41.4	26.4	17.2	43.6	0.0	17.2	543.6	26.6
800	54.5	34.0	23.8	57.8	0.1	23.7	584.4	26.2
1000	58.3	36.7	27.9	64.6	0.0	27.9	584.6	31.4

Table B.8: scheduling Distribution varying Bids. Goods = 50. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

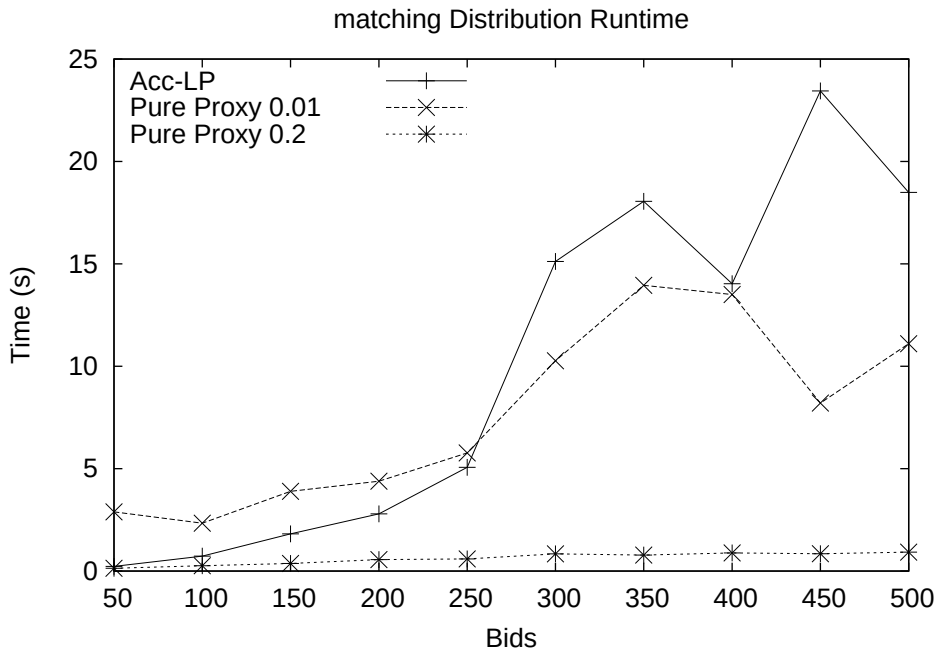

Figure B.3: matching Distribution varying Number of Bids. Goods = 40. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.2) = 0.99.

	Acc-LP						Pure(0.01)	Pure(0.2)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
100	47.6	31.2	12.7	43.9	0.7	12.0	214.0	30.2
200	125.8	86.1	26.4	112.5	1.0	25.4	360.0	36.4
300	222.8	155.4	40.7	196.1	0.1	40.6	662.0	37.0
400	314.0	225.2	56.4	281.6	0.5	55.9	566.0	38.0

Table B.9: matching Distribution varying Bids. Goods = 40. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

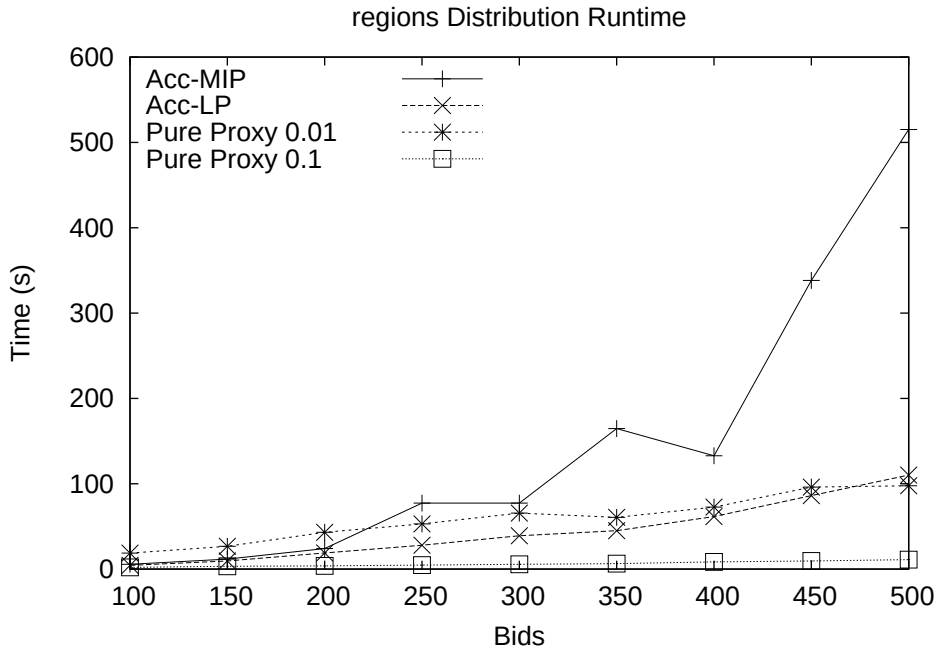

Figure B.4: regions Distribution varying Number of Bids. Goods = 50. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.1) = 0.99.

	Acc-LP						Pure(0.01)	Pure(0.1)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
100	63.1	42.1	38.7	80.8	8.8	29.9	530.0	48.8
200	137.3	88.3	85.4	173.7	22.3	63.1	602.0	57.6
300	206.8	131.1	130.7	261.8	36.8	93.9	649.8	69.6
400	280.4	176.7	185.6	362.3	59.4	126.2	602.2	66.8
500	381.0	240.3	243.7	484.0	79.2	164.5	639.4	60.2

Table B.10: regions Distribution varying Bids. Goods = 50. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

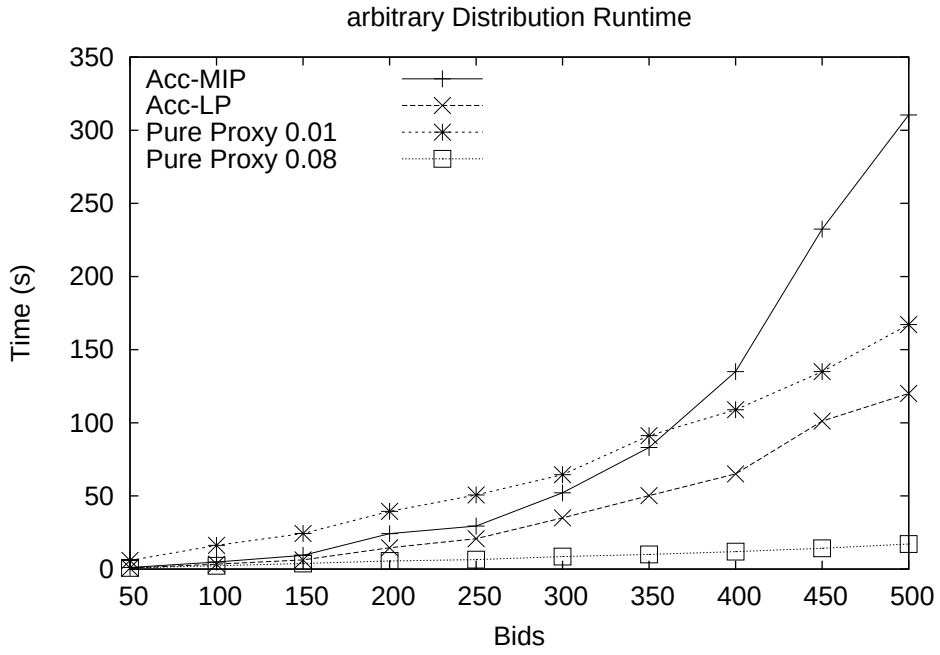

Figure B.5: arbitrary Distribution varying Number of Bids. Goods = 50. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.08) = 0.99.

	Acc-LP						Pure(0.01)	Pure(0.08)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
100	54.6	36.3	33.1	69.4	6.2	26.9	520.0	62.6
200	107.7	71.7	70.3	142.0	14.2	56.1	648.2	77.6
300	157.9	105.5	110.4	215.9	26.5	83.9	578.6	70.6
400	200.2	132.4	138.8	271.2	31.7	107.1	596.4	72.6
500	270.0	174.5	189.8	364.3	47.5	142.3	645.8	80.0

Table B.11: arbitrary Distribution varying Bids. Goods = 50. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

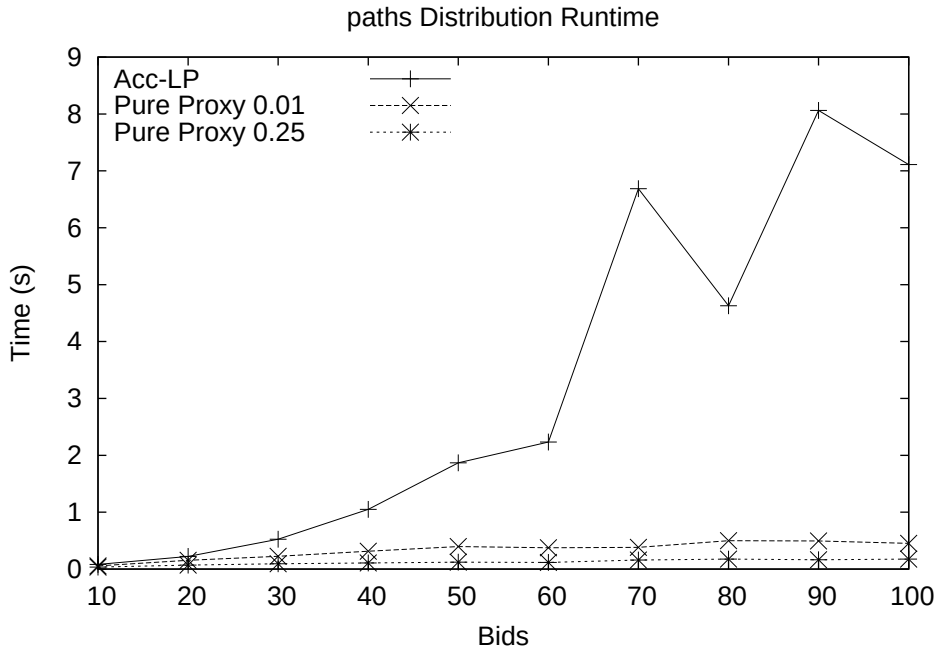

Figure B.6: paths Distribution varying Number of Bids. Goods = 10. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.25) = 0.99.

	Acc-LP						Pure(0.01)	Pure(0.25)
Bids	LPs	Frac-CG	Dur-CG	Total	Catch	Rounds	Rounds	Rounds
20	18.9	12.6	5.0	17.6	0.0	5.0	41.4	17.2
40	68.0	44.8	15.9	60.7	0.0	15.9	48.4	17.6
60	112.7	74.4	24.1	98.5	0.0	24.1	43.2	15.8
80	202.3	137.4	40.8	178.2	0.0	40.8	48.4	19.2
100	220.5	151.8	41.7	193.5	0.0	41.7	44.0	17.6

Table B.12: paths Distribution varying Bids. Goods = 10. LPs = number of times we solve for coalitional fractions. Frac-CG = number of times constraint generation called to check coalitional fractions. Dur-CG = number of times constraint generation called to check duration. Total = Frac-CG + Dur-CG. Catch = number of times a catching-up occurs.

Appendix C

Varying Goods

Here we provide the results of varying goods on distributions other than regions distribution which was contained the main text. For the most part, we notice the same trends across all distributions. Mainly, we see that Pure Proxy is more dependent on the number of goods than Acc-LP. Also, as with varying the number of bids and bids per agents, Acc-LP performs in between fully efficient Pure Proxy and 0.99 efficiency Pure Proxy. The exception these is the scheduling distribution which seems to become easier as the number of goods increases. This may be because increasing the number of goods decreases competition and agents can all be part of the winning coalition when there are more goods. For this section we do not have results from the matching, paths, and arbitrary distributions.

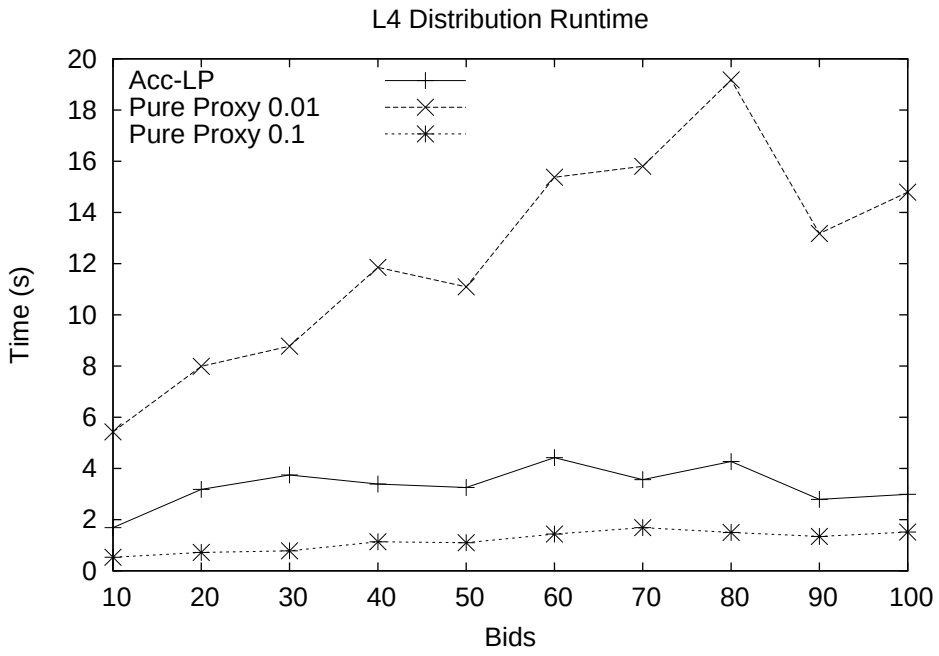

Figure C.1: L4 Distribution varying Goods. Bids = 300, Bid per Agent = 10. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.10) = 0.99.

	Acc-LP			Pure(0.01)	Pure(0.1)
Goods	LPs	Total-CG	Rounds	Rounds	Rounds
10	62.6	77.3	32.3	428.8	42.0
20	79.4	99.4	37.4	540.4	52.6
30	95.2	114.1	39.7	523.0	49.0
40	84.1	100.4	35.1	649.4	63.4
50	85.1	100.6	33.6	586.2	57.6
60	94.3	117.5	39.1	715.4	69.2
70	86.2	107.1	33.3	736.8	71.6
80	98.4	124.4	38.2	791.6	75.8
90	79.7	93.6	28.1	678.6	66.0
100	82.6	97.7	28.7	726.2	69.6

Table C.1: L4 Distribution varying Goods. Bids = 300, Bid per Agent = 10. LPs = number of times we solve for coalitional fractions. Total-CG = total number of times we call constraint generation (to check fractions and durations).

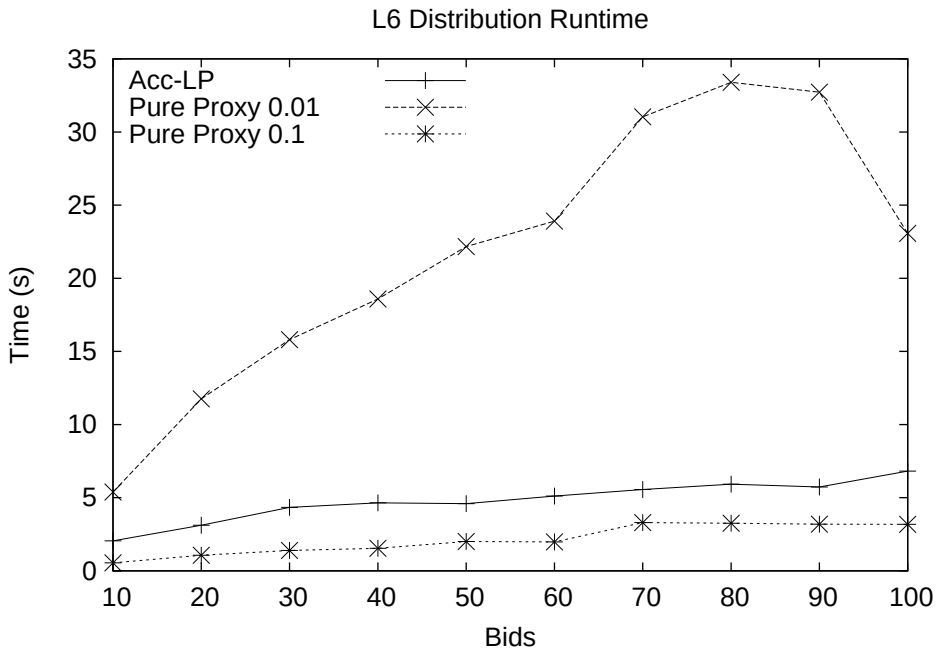

Figure C.2: L6 Distribution varying Goods. Bids = 300, Bid per Agent = 10. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.10) = 0.99.

	Acc-LP			Pure(0.01)	Pure(0.1)
Goods	LPs	Total-CG	Rounds	Rounds	Rounds
10	63.5	84.5	34.7	407.8	40.4
20	71.3	92.6	36.3	617.8	61.4
30	75.1	100.7	39.6	647.8	61.4
40	83.9	105.6	39.3	665.0	66.0
50	84.2	106.0	38.9	720.0	70.0
60	79.9	100.1	37.3	710.6	69.4
70	85.5	110.3	40.2	883.6	85.4
80	90.0	115.7	40.5	920.8	90.8
90	101.5	130.5	44.3	932.0	91.0
100	100.3	129.2	44.0	727.2	73.2

Table C.2: L6 Distribution varying Goods. Bids = 300, Bid per Agent = 10. LPs = number of times we solve for coalitional fractions. Total-CG = total number of times we call constraint generation (to check fractions and durations).

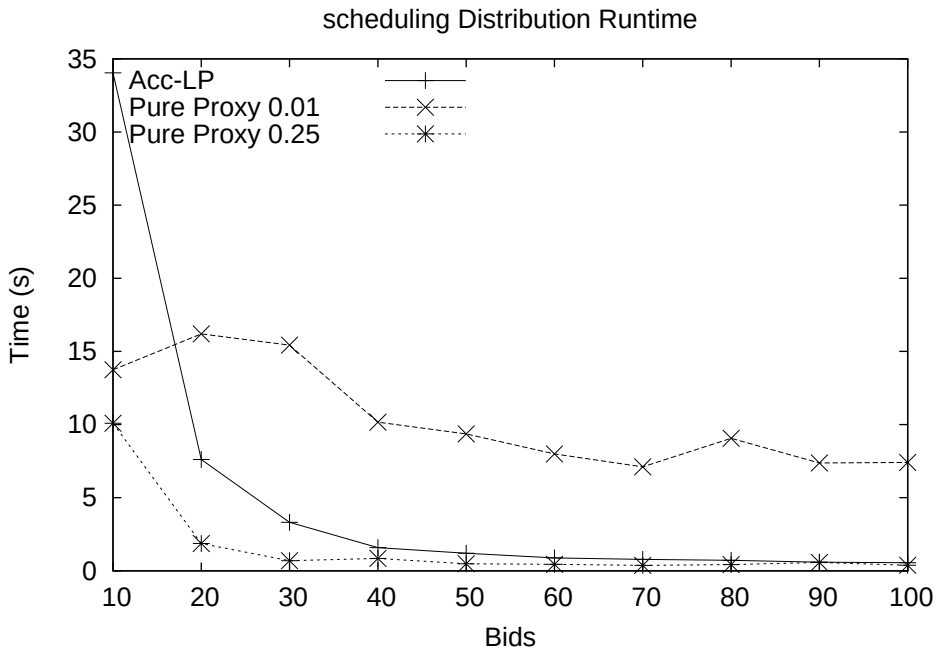

Figure C.3: scheduling Distribution varying Goods. Bids = 800. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.25) = 0.99.

	Acc-LP			Pure(0.01)	Pure(0.25)
Goods	LPs	Total-CG	Rounds	Rounds	Rounds
10	399.4	450.4	163.4	256.4	194.8
20	148.5	161.6	64.3	557.2	71.0
30	86.3	95.0	40.8	690.8	28.6
40	59.2	64.8	27.7	596.4	48.4
50	50.2	53.6	22.3	565.0	26.4
60	42.8	45.3	18.1	574.0	29.0
70	37.5	39.3	15.1	509.8	24.8
80	35.7	37.3	14.1	615.2	26.6
90	32.2	33.5	12.3	556.4	42.4
100	30.3	31.4	11.2	553.0	25.6

Table C.3: scheduling Distribution varying Goods. Bids = 300. LPs = number of times we solve for coalitional fractions. Total-CG = total number of times we call constraint generation (to check fractions and durations).

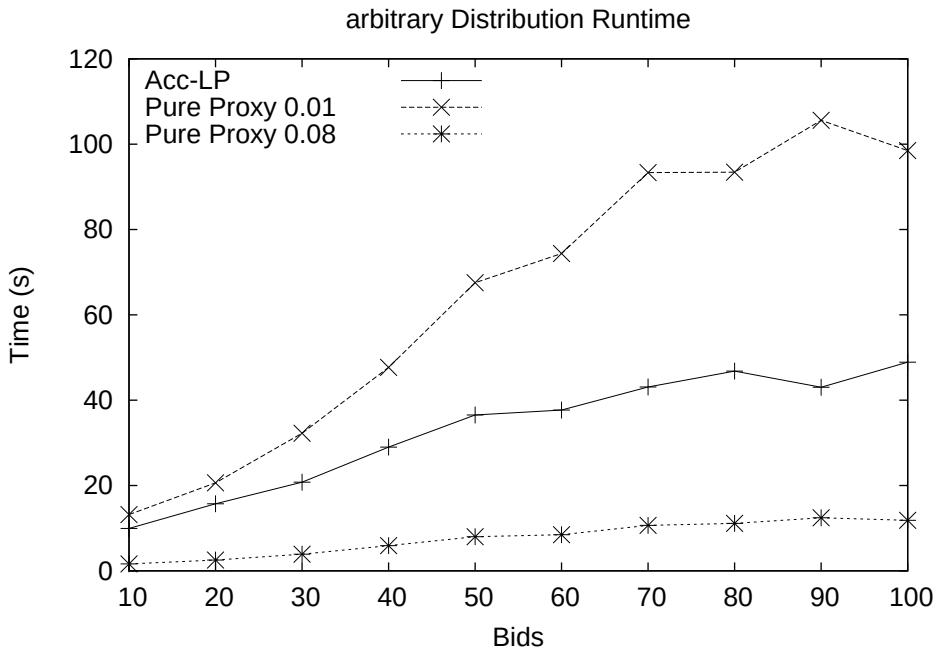

Figure C.4: arbitrary Distribution varying Goods. Bids = 300. Efficiency of Acc-LP = 1. Efficiency of Pure Proxy(0.01) = 1. Efficiency of Pure Proxy(0.08) = 0.99.

	Acc-LP			Pure(0.01)	Pure(0.08)
Goods	LPs	Total-CG	Rounds	Rounds	Rounds
10	63.5	84.5	34.7	407.8	40.4
20	71.3	92.6	36.3	617.8	61.4
30	75.1	100.7	39.6	647.8	61.4
40	83.9	105.6	39.3	665.0	66.0
50	84.2	106.0	38.9	720.0	70.0
60	79.9	100.1	37.3	710.6	69.4
70	85.5	110.3	40.2	883.6	85.4
80	90.0	115.7	40.5	920.8	90.8
90	101.5	130.5	44.3	932.0	91.0
100	100.3	129.2	44.0	727.2	73.2

Table C.4: arbitrary Distribution varying Goods. Bids = 300. LPs = number of times we solve for coalitional fractions. Total-CG = total number of times we call constraint generation (to check fractions and durations).

Bibliography

[1] Arne Andersson, Mattias Tenhune, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. pages 39–46, 2000.

[2] Lawrence M. Ausubel and Paul Milgrom. Ascending auctions with package bidding. *Frontiers of Theoretical Economics*, 1(1):Article 1, 2002.

[3] Lawrence M. Ausubel and Paul Milgrom. *Combinatorial Auctions*, chapter 1. MIT Press, 2006. forthcoming.

[4] Michael O. Ball, George L. Donohue, and Karla Hoffman. *Combinatorial Auctions*, chapter 20. MIT Press, 2006. forthcoming.

[5] Martin Bichler, Andrew Davenport, Gail Hohner, and Jayant Kalagnanam. *Combinatorial Auctions*.

[6] Estelle Cantillon and Martin Pesendorfer. *Combinatorial Auctions*, chapter 22. MIT Press, 2006. forthcoming.

[7] Chris Caplice and Yossi Sheffi. *Combinatorial Auctions*, chapter 21. MIT Press, 2006. forthcoming.

[8] Robert W. Day and S. Raghavan. Bidder-pareto-optimal core solutions and a constraint generation pricing algorithm for combinatorial auctions, 2005.

[9] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *IJCAI '99: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 548–553, 1999.

[10] Rica Gonen and Daniel Lehmann. Optimal solutions for multi-unit combinational auctions. In *ACM Conference on Electronic Commerce*, pages 13–20, 2000.

[11] Karla Hoffman, Dinesh Menon, Susara van den Heever, and Thomas Wilson. *Combinatorial Auctions*, chapter 17. MIT Press, 2006. forthcoming.

[12] Matthew O. Jackson. Mechanism theory, 2000.

[13] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce*, 2000.

[14] Paul Milgrom and Robert J. Weber. A theory of auctions and competitive bidding. *Econometrica*, 50:1089–1122, 1982.

[15] David C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, 2001.

[16] David C. Parkes. Notes on indirect and direct implementations of core outcomes, 2002. Technical report, Division of Engineering and Applied Sciences, Harvard University, 2002.

[17] David C. Parkes and Lyle H. Ungar. Iterative combinatorial auctions: Theory and practice. In *AAAI/IAAI*, pages 74–81, 2000.

[18] Michael H. Rothkopf, Aleksandar Pekec, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44:1131–1147, 1998.

[19] Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135(1-2):1–54, 2002.

[20] Tuomas Sandholm and Subhash Suri. BOB: Improved winner determination in combinatorial auctions and generalizations. *Artificial Intelligence*, 145:33–58, 2003.

[21] William Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.

[22] Peter R. Wurman, Jie Zhong, and Gangshu Cai. Computing price trajectories in combinatorial auctions with proxy bidding. *Electronic Commerce Research and Applications*, 2004. to appear.