A Kernel-Based Iterative Combinatorial Auction

Sébastien Lahaie
Yahoo! Research
New York, NY 10018
lahaies@yahoo-inc.com

Abstract

This paper describes an iterative combinatorial auction for single-minded bidders that offers modularity in the choice of price structure, drawing on ideas from kernel methods and the primal-dual paradigm of auction design. In our implementation, the auction is able to automatically detect, as the rounds progress, whether price expressiveness must be increased to clear the market. The auction also features a configurable step size which can be tuned to trade-off between monotonicity in prices and the number of bidding rounds, with no impact on efficiency. An empirical evaluation against a state of the art ascending-price auction demonstrates the performance gains that can be obtained in efficiency, revenue, and rounds to convergence through various configurations of our design.

Introduction

Combinatorial auctions have become a canonical mechanism for resource allocation in the presence of non-additive values. In these kinds of auctions, agents place bids on *packages* to express complementarities between resources; application areas include task assignment, distributed scheduling, spectrum allocation and supply chain management, among many others (Cramton et al. 2006). Iterative auctions, which proceed over rounds, are particularly attractive in many domains because they incorporate preference elicitation into the auction procedure—at each round, agents only need to evaluate the parts of their preferences relevant to bidding against the current prices (Parkes and Ungar 2000). In light of this motivation, the choice of price structure (e.g., linear) is a central element of an iterative auction design because it impacts the complexity of bidding.

In this work we describe an iterative combinatorial auction that achieves modularity in price structure. Our auction applies to the class of single-minded bidders, which is the simplest valuation class exhibiting complementarity. We develop our auction by relying on the primal-dual paradigm of auction design introduced by de Vries et al. (2007). Under this paradigm, one formulates the efficient allocation problem as an appropriate linear program, and then examines the mechanics of the primal-dual algorithm on this formulation. It turns out that the steps of the algorithm have natural interpretations as auction mechanics: bidding, allocation, price update, and termination. However, each application of the paradigm requires a domain-specific formulation of the allocation problem and therefore a separate implementation.

To develop a modular implementation of the paradigm we derive a formulation of the allocation problem that is in a sense parametrized by the price structure. We achieve this by drawing on techniques from kernel methods, building on the work of Lahaie (2009), who introduced the idea of using kernel representations for prices in market settings. By substituting in the appropriate kernel, a practitioner can tailor our auction implementation to the application at hand: there is a kernel for uniform prices, as in the auction of Ausubel (2004); a kernel for linear prices, as in the auction of Demange et al. (1986); and a kernel for bundle prices, as in the auction of de Vries et al. (2007) among others.

Besides the standard price structures just mentioned, our framework can accommodate much more exotic structures than have been considered so far—for instance, we can draw on the huge variety of kernels that already exist in the machine learning literature. We identify the property that must hold for a kernel to work within our auction, and explain how a generic kernel can be adapted so that it is satisfied. In our implementation we use the polynomial kernel, which leads to prices that are fixed-degree polynomials, interpolating between linear and bundle prices. We show how to detect when the degree is too low to clear the auction and therefore update it when needed. This strategy can in fact be applied with any family of kernels of increasing complexity.

Our auction is neither purely ascending nor descending; the price trajectory on any bundle can oscillate. While ascending prices are sometimes considered easier for bidders to cope with, flexibility in the price trajectory allows one to decrease the number of rounds to convergence without impacting efficiency. We evaluate our auction implementation against *i*Bundle (Parkes and Ungar 2000), a state of the art ascending-price combinatorial auction, and find that it compares favorably along several metrics including efficiency, revenue, and number of rounds. In the process, we also discover that the problem instances generated by CATS (Leyton-Brown et al. 2000), a standard test suite for evaluating combinatorial auctions, can be cleared to very high levels of efficiency using no more than quadratic prices. We note that we do not address incentives in this work—our focus is on the computational aspects of auction design.

Model

There is a set of buyers $N = \{1, \ldots, n\}$ and m distinct indivisible items held by a single seller. A *bundle* is a subset of the items. We associate each bundle with its indicator vector, and denote the set of bundles by $X = \{0, 1\}^m$. The bundle containing all items is denoted by $\mathbf{1}$, while the empty bundle is denoted by $\mathbf{0}$, though for clarity we will often also use $\emptyset$ for the latter. We write $x \le x'$ to denote that bundle x is contained in bundle x' (the inequality is understood component-wise).

Buyer i has a valuation function $v_i : X \to \mathbf{R}_+$ denoting how much it is willing to pay for each bundle. In this work, we assume that each buyer is *single-minded*, meaning that its valuation is characterized by a bundle-value pair (x_i, v_i) as follows:

$$v_i(x) = \begin{cases} v_i & \text{if } x \ge x_i \\ 0 & \text{otherwise} \end{cases}$$

(We are using v_i in two different roles here with a slight abuse of notation.) We require that $x_i \ne \emptyset$; therefore each valuation is normalized, with $v_i(\emptyset) = 0$. In words, buyer i would like to acquire all the items in bundle x_i, but is not interested in any others. We assume that the seller does not derive any value from retaining any bundle of items.

An *allocation* is a vector of n bundles indicating what each buyer receives. Since buyer i is only interested in bundle x_i, we can restrict our attention to allocations where i receives an element of $X_i = \{\emptyset, x_i\}$. Therefore we can equivalently represent an allocation by a subset $I \subseteq N$ denoting which buyers obtain their desired bundles. Given this subset we will write the actual allocation as x_I, which denotes a vector of bundles whose ith entry is x_i if $i \in I$, and $\emptyset$ otherwise. An allocation I is *feasible* if $\sum_{i \in I} x_i \le \mathbf{1}$. We denote the set of feasible allocations by $\mathcal{I}$. An allocation is *efficient* if it is feasible and maximizes the total value to the buyers (the seller does not enter this definition because it does not value the items). Formally, allocation x_I is efficient if $I \in \operatorname{argmax}_{I' \in \mathcal{I}} \sum_{i \in I'} v_i$.

The main purpose of an auction is to identify an efficient allocation together with prices $p : X \to \mathbf{R}$ that balance supply and demand. We assume that, like valuations, prices are normalized: $p(\emptyset) = 0$. We assume that the buyers have quasi-linear utilities, so that the utility to buyer i when obtaining bundle x at prices p is $u_i(x; p) = v_i(x) - p(x)$. Buyer i's *demand set* at prices p is defined as

$$D_i(p) = \operatorname*{argmax}_{x \in X_i} u_i(x; p),$$

namely those bundles in X_i that maximize its utility. Similarly, we define the seller's *supply set* at prices p as

$$S(p) = \left\{ x_I : I \in \operatorname*{argmax}_{I' \in \mathcal{I}} \sum_{i \in I'} p(x_i) \right\}.$$

We say that prices p *support* a given allocation x_I if it lies in the seller's supply set at those prices, $x_I \in S(p)$, and each bundle in x_I lies in the respective agent's demand-set: $i \in I$ only if $x_i \in D_i(p)$, and $i \notin I$ only if $\emptyset \in D_i(p)$. We say that prices are *competitive* if they support some allocation.

It is well-known that if an allocation is supported by some prices, then that allocation is efficient, and conversely if an allocation is efficient, then there exist competitive prices to support it (Bikhchandani and Ostroy 2002). Note that, as defined, prices can be arbitrary functions over the bundles in general. These existence results may not hold if we impose some specific structure on prices, such as linearity.[1] Our kernel-based approach will provide a modular way to introduce structure to auction prices, and adapt it as needed to support efficient allocations.

Kernels

To achieve modularity in the price structure, we draw on ideas from kernel methods in machine learning, following (Lahaie 2009). We define a new encoding for the bundles within a *feature space* $Y = \mathbf{R}^M$ via a mapping $\phi : X \to Y$, where $M \ge m$. Entry j in the vector $\phi(x)$ corresponds to the value of the jth "feature" of bundle x, for $j = 1, \ldots, M$. Prices are now taken to be linear functions over feature space, so that $p \in \mathbf{R}^M$ and the price of a bundle x is evaluated by taking the inner product $\langle p, \phi(x) \rangle$. We will sometimes continue to write $p(x)$ rather than $\langle p, \phi(x) \rangle$, keeping the mapping implicit.

To encode allocations in feature space, we simply sum up their constituent bundles, so that x_I is mapped to $\sum_{i \in I} \phi(x_i)$; with a slight abuse of notation, we use $\phi(x_I)$ to denote this induced allocation map. This is consistent with our concept of prices, because the price of a re-encoded allocation is simply its revenue:

$$\langle p, \phi(x_I) \rangle = \left\langle p, \sum_{i \in I} \phi(x_i) \right\rangle = \sum_{i \in I} \langle p, \phi(x_i) \rangle .$$

To ensure that prices are normalized, we require that the mapping be *centered*: $\phi(\emptyset) = \mathbf{0}$. If this is not the case (e.g., with the Gaussian kernel in machine learning), we can center the mapping by using $\phi'(x) = \phi(x) - \phi(\emptyset)$ instead.

Now, to ensure that competitive prices exist we may need the dimension M of the prices to be very large, even exponential in the number of items. In such cases it becomes intractable to explicitly work with feature space encodings. The key computational idea behind kernel methods is the "kernel trick": rather than work in feature space, we formulate the problem at hand (in this case, allocation and pricing) purely in terms of inner products of feature space vectors. What makes this practical is that, for many useful mappings, the inner products can be evaluated in time independent of M. The inner products are given via a *kernel function* k over bundle pairs: $k(x, x') = \langle \phi(x), \phi(x') \rangle$.

[1]As defined prices can in general be arbitrary over the bundles, but they are still *anonymous* in the sense that different agents see the same prices for the same bundles. It is known that for single-minded bidders anonymous prices suffice to support efficient allocations (Parkes and Ungar 2000), but for other valuation classes *personalized* prices may be needed (Bikhchandani and Ostroy 2002). The kernel approach allows for personalized prices if we define kernels over personalized bundles (formally, bundle-agent pairs), but we do not pursue this here.

The following kernels are of immediate relevance to auction design as the associated price structures reoccur throughout the literature, except for the last kernel which forms the basis of our implementation.

Unit Perhaps the simplest possible kernel is $k(x, x') = |x||x'|$, where $|x|$ denotes the number of items in x. This corresponds to the map $\phi(x) = |x|$. This leads to uniform prices.

Linear The linear kernel is defined by $k(x, x') = \langle x, x' \rangle$, corresponding to the identity map $\phi(x) = x$. This leads to linear prices.

Identity The identity kernel is defined by $k(x, x') = 1$ if $x = x'$ and neither bundle is empty, and 0 otherwise. In this case the feature space is of dimension $2^m - 1$, with one dimension for every non-empty bundle. The associated map ϕ sends non-empty bundle x to the canonical unit vector with a 1 in the component corresponding to x, and sends the empty bundle to the origin. The resulting prices are general: each bundle is separately priced.

Polynomial As a generalization of the linear kernel, we have the polynomial kernel defined by $k(x, x') = \langle x, x' \rangle^d$. The associated map has a dimension for every non-empty bundle x with $|x| \leq d$. The component corresponding to $x \neq \emptyset$ in $\phi(x')$ is 1 if $x \leq x'$ and 0 otherwise. The resulting prices price every combination of items up to size d.

Now, it is not the case that any kernel can be used in the context of an auction. Because the auction must balance supply and demand, it is crucial that the associated mapping retain information on how bundles aggregate into allocations. To illustrate, suppose we have two distinct items. Under the unit kernel above, the infeasible allocation $(10, 10)$ and the feasible allocation $(10, 01)$ both map to 2. Therefore, if the first allocation represented demand and the second supply, we would be mislead into thinking that supply meets demand and that the items can be feasibly allocated.

To avoid this the mapping must not lose essential information about which items a bundle contains. Recall that $\mathcal{I}$ denotes the feasible allocations, and let $\mathcal{I}^c$ denote its complement. Let $\phi(\mathcal{I})$ denote the image of $\mathcal{I}$ under the allocation map induced by ϕ, and similarly for $\mathcal{I}^c$. The following property ensures that feasible and infeasible allocations are not confused in feature space.

Definition 1 *A kernel k is* respectful *if its associated allocation map satisfies $\phi(\mathcal{I}) \cap \phi(\mathcal{I}^c) = \emptyset$.*

Given any kernel k (e.g., drawn from the machine learning literature), there is a straightforward way to extend it to make it respectful if it is not already. We can augment the underlying feature map ϕ by appending the original bundle representation, resulting in $\phi'(x) = [\phi(x), x]$. The corresponding kernel function is simply $k'(x, x') = k(x, x') + \langle x, x' \rangle$. This amounts to introducing the price structure implicit in k on top of linear prices. It is not hard to see that all the kernels introduced above are already respectful, except for the unit kernel. The unit kernel is respectful for homogeneous but not heterogeneous items.

Formulation

To apply the primal-dual paradigm of auction design we first need to formulate the efficient allocation problem as a linear program. Using a feature map ϕ to maintain flexibility in the eventual price structure, the *primal* formulation is as follows.

$$
\begin{aligned}
\max_{y \geq 0, \bar{y} \geq 0} \quad & \sum_{i \in N} v_i y_i \\
\text{s.t.} \quad & \sum_{i \in N} y_i \phi(x_i) = \sum_{I \in \mathcal{I}} \bar{y}_I \phi(x_I) && (1) \\
& \sum_{I \in \mathcal{I}} \bar{y}_I \leq 1, \quad y_i \leq 1 \ \ (i \in N) && (2)
\end{aligned}
$$

Here we have a variable $y_i \in [0, 1]$ for each buyer $i \in N$ indicating whether i obtained its desired bundle x_i. We also have a variable $y_I \in [0, 1]$ for each $I \in \mathcal{I}$ indicating which allocation the seller supplies. The equality (1) balances supply and demand.

This formulation cannot be solved using generic linear programming algorithms because it has an exponential number of seller variables, and (1) corresponds to M constraints, which could be prohibitively large. The primal-dual paradigm together with the kernel trick will allow us to address these issues. However, a prerequisite for applying the paradigm is that the primal formulation have an integer optimal solution. For now we assume that ϕ has been chosen so that this holds, and explain later how our auction (implicitly) adapts ϕ to ensure this. We could use the identity kernel to guarantee integrality, because in that case we recover the formulation of Bikhchandani and Ostroy (2002), but a much simpler price structure may suffice.

To confirm that our formulation is correct, let $I' = \{i \in N : y_i = 1\}$ be the allocation on the demand side and let I be the allocation on the supply side, for which $\bar{y}_I = 1$. Note that I is feasible by definition, and since $\phi(I') = \phi(I)$ by constraints (1), it follows that I' is also feasible if the kernel is respectful. This shows that the constraints indeed characterize feasible allocations.

The *dual* problem is that of computing competitive prices, formulated as follows.

$$
\begin{aligned}
\min_{\pi \geq 0, \bar{\pi} \geq 0, p} \quad & \sum_{i \in N} \pi_i + \bar{\pi} \\
\text{s.t.} \quad & \pi_i \geq v_i - \langle p, \phi(x_i) \rangle \quad (i \in N) && (3) \\
& \bar{\pi} \geq \langle p, \phi(x_I) \rangle \qquad (I \in \mathcal{I}) && (4)
\end{aligned}
$$

Here the variables $p \in \mathbf{R}^M$, corresponding to the constraints (1), have a natural interpretation as prices. We have a variable π_i for each buyer $i \in N$, which at an optimal solution equals the buyer's maximum possible utility at prices p, because we have $\pi_i = \max\{v_i - p(x_i), 0\}$. Similarly, $\bar{\pi}$ equals the seller's maximum revenue over all feasible allocations. From the optimal primal solution let $I = \{i : y_i = 1\}$ and from the optimal dual solution take the prices p. As de Vries et al. (2007) have shown, the complementary slackness conditions imply that p supports x_I, which confirms that the dual correctly computes competitive prices.

1. Demand Quote the current prices p and record the demand set $D_i(p)$ of each buyer $i \in N$.

2. Supply Compute a revenue-maximizing allocation $J \in S(p)$ by solving a weighted set-packing problem, taking as input the weight-set pairs $(p(x_i), x_i)$ for each $i \in N$.

3. Termination Check whether $x_i \in D_i(p)$ for each $i \in J$ and $\emptyset \in D_i(p)$ for each $i \notin J$. If so, the auction terminates and outputs allocation x_J together with prices p. Otherwise, the round proceeds.

4. Price Update Compute the price update q given allocation x_J and the demand set $D_i(p)$ of each buyer $i \in N$. Compute the step size θ and set $p \leftarrow p + \theta q$.

Figure 1: Generic auction round.

Auction

An auction can be viewed as a formal process of balancing supply and demand via prices, with prices updated at each round according to the discrepancy between the two. Figure 1 outlines a generic auction round for single-minded bidders; auctions that fall within the primal-dual paradigm follow this template (with only minor variations), including our own.

In the first round demand sets are collected—this amounts to bidding. In practice, it is common to collect ϵ-demand-sets, meaning all bundles that maximize a buyer's utility to within some additive *slack* of $\epsilon \geq 0$. This can speed up convergence significantly even for moderate ϵ relative to buyer values, as it increases the chances that supply will meet demand. In the second round supply is collected; the revenue-maximization problem here can be formulated as an integer program, and solved at large scales in practice using general or special-purpose solvers (Cramton et al. 2006). The third round checks whether supply meets demand, and if this is the case, the auction terminates. Note that, upon termination, the prices p support allocation x_J, so the latter is efficient; if ϵ-demand-sets are used, the allocation is efficient to within an additive error of $n\epsilon$ (Parkes and Ungar 2000). To complete the specification, we still need to elaborate the details of step 4.

Price Update

The price update under the primal-dual paradigm involves formulating a linear program that attempts to satisfy the complementary slackness conditions, or more intuitively, to balance supply and demand. Given the demand sets under the current prices p, let $N^+ = \{i \in N : x_i \in D_i(p)\}$ and let $N^- = \{i \in N : \emptyset \in D_i(p)\}$. Note that these two sets may intersect if some buyer demand set is composed of $\{x_i, \emptyset\}$. The *restricted primal* is formulated as follows.

$$\max_{z \geq 0, \bar{z} \geq 0} \quad \sum_{i \in N^+} z_i - \sum_{i \in N^-} z_i + \bar{z}$$

$$\text{s.t.} \quad \sum_{i \in N} z_i \phi(x_i) = \bar{z}\phi(x_J) \tag{5}$$

$$\bar{z} \leq 1, \quad z_i \leq 1 \ (i \in N) \tag{6}$$

Here we again have a variable $z_i \in [0, 1]$ for each $i \in N$, indicating whether buyer i receives a bundle from its demand set. We also have a variable $\bar{z} \in [0, 1]$ that enters the objective, indicating whether the seller's revenue-maximizing allocation is accepted. Note that the maximum possible objective value for the restricted primal is $\ell = |N^+ \backslash N^-| + 1$, which occurs when supply meets demand.

If the restricted primal does not find a solution that satisfies every agent as well as the seller, then the primal-dual paradigm dictates that we appeal to the dual of the restricted primal to obtain price updates. Specifically, the dual variables $q \in \mathbf{R}^M$ corresponding to constraints (5) in the restricted primal constitute the update (de Vries et al. 2007). We have now reached an impasse, because the large number of constraints in (5) still remains, and q cannot be explicitly represented. To overcome this, we apply the kernel trick by reformulating the restricted primal purely in terms of inner products.

The idea, introduced in (Lahaie 2009), is to use a penalty method to solve the restricted primal without explicitly working with the M-dimensional vectors in constraints (5), instead using information from the kernel function. We remove constraints (5) and replace them with a quadratic penalty term in the objective:

$$-\frac{\nu}{2} \left\| \sum_{i \in N} z_i \phi(x_i) - \bar{z}\phi(x_J) \right\|^2. \tag{7}$$

Here ν is taken large to ensure the original constraints are (very close to) satisfied at an optimal solution. In practice the method proceeds in iterations, starting with an initial setting of $\nu^0 = 1$ and updating $\nu^{t+1} \leftarrow \tau\nu^t$ at iteration t as long as the norm in (7) is not close enough to zero, where typically $\tau \in [4, 10]$ (Bertsekas 1996). Evaluating the squared norm in (7), one finds that it only involves inner products between feature space representations of bundles, and the resulting Hessian matrix is of size $(n+1) \times (n+1)$. Therefore each iteration of the method involves solving a quadratic program of polynomial size that can be formulated using the kernel function. This provides a tractable way of solving the restricted primal.

To obtain a sparse representation of the update q, we appeal to a result on penalty methods by Bertsekas (1996):

$$q = \lim_{\nu \to +\infty} \nu \sum_{i \in N} z_i \phi(x_i) - \nu \bar{z}\phi(x_J),$$

where $(z, \bar{z})$ here is now the optimal solution to the restricted primal, and q is the optimal solution to its dual. Applying this to a generic bundle x and collecting terms, we find that the price update takes the form:

$$\langle q, x \rangle = \sum_{i \in N \backslash J} \nu z_i \, k(x_i, x) + \sum_{i \in J} (\nu z_i - \nu\bar{z}) \, k(x_i, x). \tag{8}$$

In this representation we only need to keep track of n coefficients, one for each agent, and to update prices we simply need to add the update coefficients to the price coefficients; at the initial round all price coefficients are zero leading to zero prices. This completes the specification of the price update for a fixed kernel k.

Structure Update

We have so far assumed that the kernel k (equivalently, the mapping ϕ) has been chosen in such a way that our primal has an integer solution. Without keen insight into the agents' valuations, it can be difficult for the auctioneer to ensure this a priori without resorting to a highly nonlinear price structure. Here we show how to detect the need for a more complex kernel at each round of the auction. We appeal to the following simple result.[2]

Proposition 1 *The primal has an integer solution if and only if the final restricted primal has an integer solution.*

In light of this result, we can check whether the solution to the restricted primal, as solved by the penalty method, is integer in order to detect the need for a kernel update. There is complete flexibility in the kernel one can choose as a replacement. In our implementation we use the polynomial kernel, so a natural update is to increment the degree d.

We should clarify that we do not just replace the kernel in the current price representation (8). This could lead to a large break in the current prices. Instead, we keep the price representations for each separate kernel on a stack, and push a new (initially zero) price representation onto the stack each time a new kernel is introduced. Price updates only apply to the top element of the stack, and the price of a bundle is the sum of its prices under each element of the stack. Under the polynomial kernel, d is updated at most m times, so the price representation under this scheme never needs more than nm coefficients.

Step Size

In current auction designs the step size is often set to a constant, typically $\theta = \epsilon$, where ϵ is the slack we allow for the demand sets (Parkes and Ungar 2000). The choice of ϵ under this scheme can significantly affect the performance of the auction: a larger ϵ decreases the number of rounds to convergence, but the efficiency of the final allocation suffers. A large step size can cause the auction prices to overshoot the set of competitive prices leading to inefficiency, and in an ascending auction there is no way to correct this.

Because our auction is neither purely ascending nor descending, we can choose an adaptive stepsize rule that begins aggressively and diminishes as needed, to achieve both fast convergence and high efficiency. Given the slack parameter ϵ and the current round's computed update q, our step size is set to $\theta = \gamma\epsilon / \max_{i \in N} |q(x_i)|$, where $\gamma \geq 1$. Under this step size an agent sees a price change of no more than $\gamma\epsilon$ on its desired bundle. The parameter γ can be set large initially to implement aggressive updates, and diminished when needed until it reaches a minimum value of 1. We consider two policies for diminishing γ.

The first policy tries to detect whether prices have over- or under-shot their competitive levels. To determine this, we examine the variable $\bar{z}$ in the solution to the restricted primal. Note that $\bar{z} = 1$ indicates *overdemand*, because the restricted primal tries to satisfy as many agents as possible, while $\bar{z} = 0$ indicates *underdemand*, because in this case most agents demand $\emptyset$. Therefore, a switch in the value of $\bar{z}$ from round to round provides a convenient heuristic to detect whether prices have bypassed their competitive levels. When a switch occurs we diminish the stepsize by decrementing γ by 1.

The next policy simply decrements γ by 1 whenever a fixed number of rounds ρ has passed. We found that, even with the first policy in place, prices can oscillate from ascending to descending without decrementing γ, so this second policy ensures that the maximum price update can eventually reach ϵ.

Empirical Evaluation

In this section we report on experiments run to evaluate our kernel-based auction in terms of several performance metrics including efficiency, revenue, and speed of convergence. In our implementation we use the polynomial kernel beginning with degree $d = 1$, incrementing the degree when needed as described. As a benchmark we implemented *i*Bundle (Parkes and Ungar 2000), which differs from our auction only in the price update step. We used the CATS suite of distributions to generate problem instances (Leyton-Brown et al. 2000). CATS takes in as input the number of goods m and "bids" n, and generates n single-minded bidders, represented as bundle-value pairs. We specifically used the following distributions from the suite: arbitrary, paths, regions, scheduling. Throughout we set $m = 30$, and $n = 50$ unless otherwise noted.

Our kernel-based auction was implemented in Python 2.6, using Cbc to solve for the allocation in step 2 of each round (recall Figure 1), and cvxopt to solve the quadratic program that computes the price update in step 4.[3] The penalty method's update parameter was set to $\tau = 6$ throughout; we found that tuning τ within $[2, 10]$ had little impact on auction performance. We set the first step size parameter to $\gamma = 10$ unless otherwise noted; as it directly determines the magnitude of the step size, it has a significant impact on the number of rounds. The second step size parameter was set to $\rho = 5$, which ensures that the step size diminishes every five rounds; this parameter does have a moderate impact on the number of rounds, but due to space constraints we do not report on its effect here. Finally, to ensure that our auction and the benchmark are on the same footing, we used the same demand slack parameter for both, calibrating it to $\epsilon = \min_{i \in N} v_i / 2$ for each instance. This led to 99% efficiency on average over all distributions.

Results. The performance of our auction against the benchmark is summarized in Table 1. The *efficiency* metric is defined as the percent of the maximum possible value achieved by the final allocation, while *revenue* is the percent of this same value paid out by the buyers upon termination. The *correctness* metric records what percent of the final allocations were perfectly (as opposed to approximately) efficient. Finally, the *monotonicity* metric measures the percent

[2]Proof omitted due to space constraints—simply observe that an integer optimal solution to the primal translates into an integer optimal solution to the restricted primal, and vice-versa.

[3]`projects.coin-or.org/Cbc`
`abel.ee.ucla.edu/cvxopt/`

of the total price variation over rounds that is due to price increases, averaged over the buyers.

distribution	auction	rounds	eff.	rev.	corr.	monot.
arbitrary	*i*Bundle	81	99	95	90	100
	k-auction	**66**	99	**97**	90	81
paths	*i*Bundle	**45**	99	82	80	100
	k-auction	65	99	**91**	**92**	74
regions	*i*Bundle	122	99	94	83	100
	k-auction	**94**	99	**97**	**85**	82
scheduling	*i*Bundle	**46**	99	94	84	100
	k-auction	62	99	**96**	**98**	60

Table 1: Performance of the kernel-based auction against the benchmark. All metrics except rounds are in percents, averaged over 50 runs. Bold indicates the best performing auction.

We see that each auction always achieves 99% efficiency (rounded down), as expected given the calibration. Neither auction dominates in terms of the number of rounds. We see that the kernel-based auction tends to do better for the two distributions that require more rounds in general, namely arbitrary and regions. However, the number of rounds is much more consistent across distributions for our auction than for the benchmark, with standard deviations of 15 and 36, respectively. Our auction dominates the benchmark in terms of revenue; we attribute this to its ability to make more aggressive price updates. Our auction also dominates in terms of correctness, though this is not as important since high levels of efficiency are reached in any case. Finally, it is interesting to note that the price updates in our auction are mostly increasing, but still a substantial fraction of the price trajectory is decreasing.

Figure 2 provides more refined insights into the impact of the γ step size parameter. We see that as γ is varied, there is an optimum where the step size is neither too conservative nor aggressive, and the number of rounds is minimized. Note that efficiency does not vary with γ—this is due to the fact that the ϵ slack is decoupled from the step size in our auction, unlike *i*Bundle. Monotonicity decreases smoothly with γ as expected, and there is an inherent trade-off between the number of rounds and monotonicity.

It is interesting to consider the degree of the final prices obtained by our kernel-based auction. Remarkably, we found that over 50 runs on each distribution, d never exceeded 2, meaning the auction was always able to clear the market with at most quadratic prices. (Even over hundreds of other test runs, we never encountered $d > 2$.) The percent of instances that cleared with *linear* prices was as follows: 100% for arbitrary, 90% for regions, 60% for scheduling, and 10% for paths. This was an unexpected finding because the arbitrary and regions distributions are known to be the hardest of the CATS distributions, in terms of winner determination (Leyton-Brown et al. 2000), and this was borne out in our experiments. We believe this indicates that the LP relaxations of these instances under the linear kernel, while rarely integer, nevertheless contain integer solutions that are very close to optimal.

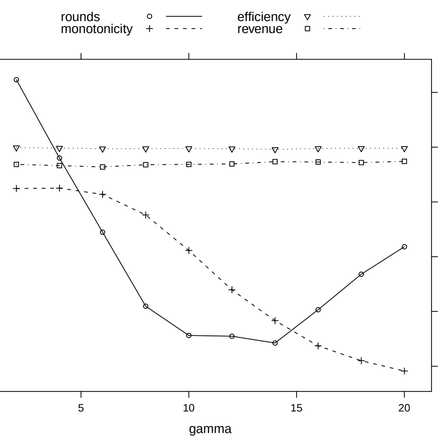

Figure 2: Effect of the γ parameter under the arbitrary distribution. All metrics except rounds are in percents. Each point is an average over 50 runs.

To conclude we briefly consider the scalability of our auction. On the arbitrary distribution (the hardest we observed), the average runtime of our auction for 50 buyers was 38s over fifty runs, compared to 19s for *i*Bundle. For 80 buyers, the average runtime was 156s, compared to 88s for *i*Bundle. We see that our auction is slower by around a factor of 2, which is expected given that the two auctions only essentially differ in their price update step, and ours is much more intensive to accommodate a wide variety of price structures. Given that the quadratic program in the price update step has a very simple form (box constraints), we believe significant runtime improvements could be gained by exploiting its special structure (Byrd et al. 1995).

References

Ausubel, L. M. 2004. An efficient ascending-bid auction for multiple objects. *American Economic Review* 94(5):1452–1475.

Bertsekas, D. P. 1996. *Constrained Optimization and Lagrange Multiplier Methods*. Athena Scientific.

Bikhchandani, S., and Ostroy, J. M. 2002. The package assignment model. *Journal of Economic Theory* 107:377–406.

Byrd, R. H.; Lu, P.; Nocedal, J.; and Zhu, C. 1995. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific and Statistical Computing* 16(5):1190–1208.

Cramton, P.; Shoham, Y.; and Steinberg, R., eds. 2006. *Combinatorial Auctions*. MIT Press.

de Vries, S.; Schummer, J.; and Vohra, R. V. 2007. On ascending Vickrey auctions for heterogeneous objects. *Journal of Economic Theory* 132(1):95–118.

Demange, G.; Gale, D.; and Sotomayor, M. 1986. Multi-item auctions. *Journal of Political Economy* 94(4):863–872.

Lahaie, S. 2009. A kernel method for market clearing. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 208–213.

Leyton-Brown, K.; Pearson, M.; and Shoham, Y. 2000. Towards a universal test suite for combinatorial auction algorithms. In *Proc. second ACM Conference on Electronic Commerce (EC)*, 66–76.

Parkes, D. C., and Ungar, L. H. 2000. Iterative combinatorial auctions: Theory and practice. In *Proc. 17th National Conference on Artificial Intelligence (AAAI)*, 74–81.