Thesis Proposal

Computer Science Department

Carnegie Mellon University

Distributed Market-Based Algorithms for Multi-Agent Planning with Shared Resources

Sue Ann Hong

Committee:

Geoffrey J. Gordon (Chair)

M. Bernardine Dias

Carlos Guestrin

Amy Greenwald (Brown University)

November 2, 2010

Abstract

We propose a new family of market-based distributed planning algorithms for multi-agent systems, in particular for the problem of allocating indivisible resources or tasks among agents. Market-based algorithms have become popular in collaborative multi-agent planning due to their intuitive and simple distributed paradigm as well as their success in planning in domains such as robotics and software agents. However, they suffer from two main drawbacks: 1. it is somewhat of an art to create a reasonable pricing in each domain, requiring a human designer and parameter tuning, and 2. they rarely guarantee optimality. Also, most existing algorithms require a central trusted auctioneer, or require the domain to be *collaborative*, where all agents are cooperating towards a common goal without conflicting interests. This thesis aims to provide the solution to both problems, and extends the existing market-based paradigm to non-collaborative settings with potentially adversarial agents, while also distributing the auctioneer's computation to the agents. Our algorithms are based on a decomposition method for mixed integer programs, and compute resource prices *automatically*, and does so to guarantee optimality at termination. The completed preliminary work shows the promise of our approach in the collaborative setting; our proposed thesis work includes extending the algorithm to the fully distributed non-collaborative setting by employing no-regret learning for each agent. We hope to show that our algorithm leads to a notion of social optimum and plan to conduct empirical evaluations in domains such as robot path planning and task allocation.

1 Introduction

Multi-agent planning is often naturally formulated as a mostly factored combinatorial optimization problem: each agent has its own local state, constraints, and objectives, while agents interact by competing for scarce, shared resources. Such a problem is usually intractable as a single centralized optimization, as the joint state space over all agents is exponential in the number of agents. Hence, given the natural factorization of the planning problem it is beneficial to seek a distributed planning solution. In the ideal setting, each agent will solve its individual planning problem mostly independently of other agents, with little complication imposed by the shared global constraints, using a fast single-agent planning algorithm such as those which already exist in many domains.

Market-based planning has become a popular approach to collaborative multi-agent distributed planning, thanks to its intuitive paradigm of agents bidding for tasks or resources through a market while solving their local problems independently. It has proven successful in a number of domains with robotic and software agents, in particular task allocation [Dias, 2004] [Gerkey, 2003] [Zheng and Koenig, 2009]. However, there are several well-known limitations. First, to set up a good market is something of an art: a human designer must choose carefully a strategy for pricing commodities for every new problem domain, balancing planning effort against suboptimality, which often involves extensive parameter tuning or guessing. Also, the task-oriented nature of these algorithms mean that they cannot handle additional constraints, for example on limited resources, in a general and efficient manner, adding to the design effort. And even with careful engineering, most market-based planners cannot offer guarantees on the final overall plan quality.

This thesis will explore the use of mixed integer linear program (MILP) formulation and a family of decomposition-based distributed algorithms for multi-agent planning in both the traditional collaborative setting as well as non-collaborative setting with self-interested agents. The decomposition machinery solves the difficulty of designing resource prices by *automatically* pricing the resources given any domain, and guarantees a globally optimal joint plan at termination. Our method is most suited for domains with largely individual planning problems coupled by a set of shared resources, such that even when the agents are self-interested, their costs are coupled with other agents'. Popular such domains include task allocation and supply-chain optimization, both of which are usually considered under fully cooperative agents, but one can easily consider a non-collaborative setting by having companies competing for tasks or production. The MILP representation lets us easily generalize the usual task-allocation setting to those with complex constraints over multiple agents, such as deadlines for tasks or limited resources.

Moreover, mixed integer programming has become a common approach in many planning domains. In operations research, a common approach to solving multi-stage planning problems under uncertainty is stochastic programming with recourse (e.g., [Powell, 1996]); or more specifically in artificial intelligence and cooperative control, MILPs have been successfully used for problems such as UAV task allocation [Richards et al., 2002] [Alighanbari and How, 2005] and for general STRIPS-style planning [Vossen et al., 1999], and under a game-theory framework for reasoning about uncertainty due to the actions of other agents in a nonzero-sum game [Sandholm et al., 2005]. Such popularity of MILPs makes our methods readily applicable since for many problems a MILP formulation already exists; and we expect an immediate impact as the distributed nature of our methods alone will enable larger problems to be solved than previously possible.

However, a common concern with using the MILP formulation is that while it lets us compute global joint plans optimally, it may not provide the level of efficiency and robustness required in domains such as field robotics. It is not yet clear that our algorithms can meet the demands of such fields; however, they do provide partial solutions to different challenges of distributed planning. First, our methods do not always guarantee a feasible plan under communication or machine failure, but they are *anytime*: feasible joint plans can, and mostly like will, be generated before the final optimal joint plan is found, and in case of failure, the agents can execute the last chosen joint plan. Second, we hope that by distributing computation among agents we can arrive at a good joint solution quickly. Last, while we do not focus on communication complexity, our algorithms incur only a small communication cost by communicating information about only relevant resources or tasks in contention. And as with most distributed planning algorithms, an individual agent's local problem and constraints never need to be communicated to other agents or to a central server.

In the following, we first describe related work, then present our preliminary work on a simplified version of the branch-and-price-and-cut algorithm [Barnhart et al., 1996] for solving MILPs, which we show to be simple and effective as a distributed algorithm. The algorithm is based on Dantzig-Wolfe decomposition for linear programs, which we extend to solve MILPs by incorporating the Gomory cutting plane algorithm, while retaining its optimality guarantees. We show experimentally that the algorithm is efficient in comparisons against CPLEX [ILOG, 2010], a highly optimized commercial MILP solver, in both centralized and distributed implementations. As we will see, the algorithm follows a standard model in market-based planning where a trusted central "auctioneer" allocates resources to the bidding agents. We propose to further extend the basic algorithm by 1. distributing the central computation of pricing resources for faster computation and more robustness, and 2. providing a mechanism under which even *non-collaborative* agents, with possibly conflicting interests, will reach a socially optimal joint solution. We hope to show that the agents converge to at least a boundedly-rational equilibrium, an equilibrium for agents not capable of computing an incentive to change their strategy due to computational limitations. We aim to show that the agents also converge to a good global solution, possibly through a statement on the price of anarchy of the mechanism. We conclude the document with a proposed time schedule for finishing the thesis.

2 Related Work

2.1 Market-Based Planning and Optimization-Based Distributed Planning for Cooperative Agents

Various forms of market-based distributed planning have been studied for collaborative domains, notably by Dias [2004], Gerkey [2003], and Koenig et al. [2010] for task allocation with robotic agents. A common feature in market-based algorithms is the central trusted auctioneer that assigns tasks based on agents' bids, which may be used alone (e.g. [Koenig et al., 2007], [Koenig et al., 2008], [Tovey et al., 2005]), or in conjunction with agent-to-agent negotiation to improve the solution quality after the central initial allocation (e.g. [Zheng and Koenig, 2009], [Dias and Stentz, 2002], [Lemaire et al., 2004]).

Otherwise, most existing market-based algorithms can be classified into two categories; those based on single-item auctions and those based on combinatorial auctions. In a single-item auction, each agent bids for each item separately, and the items are allocated independently of each other to the agents. Most work has been in this setting, including recent papers that have presented algorithms with approximation guarantees on the cost of the global solution, namely Lagoudakis et al. [2005] for sequential single-task auctions, Koenig et al. [2008] for sequential single-task auctions with regret clearing, Zheng and Koenig [2010] for an improved version of sequential single-task auctions using hill climbing, and Gerkey [2003] for one-task-per-robot domains. Most algorithms employ a greedy algorithm for the auctioneer to allocate tasks (i.e., to the highest bidder), and thus are extremely fast.

However, complex constraints often found in more realistic problems such as those for collision avoidance or task deadlines cannot be considered naturally in the single-item auction framework as the constraints will force allocation of some tasks to depend on others'. Combinatorial auctions [Cramton et al., 2005] allow for such constraints by letting agents bid on bundles of resources, and have been studied in the collaborative setting for robot exploration [Berhault et al., 2003], role assignment [Hunsberger and Grosz, 2000], and task allocation [Nair et al., 2002], among others. However, the bundles render the auctioneer's computation to be potentially exponential in the number of resources, and yet the handcrafted bidding strategies lead to suboptimal solutions. Hence, for settings without such coupling constraints, one may prefer the fast single-item auction algorithms, especially if real-time performance is expected.

Like combinatorial auctions, our algorithms naturally handle constraints coupling resources. However, unlike most work using combinatorial auction in collaborative domains, our algorithm uses iterative bidding on small sets of bundles and price feedback from the auctioneer, which allows the agents to rebid in subsequent iterations toward a globally optimal solution. Hence the agents may not need to communicate their preferences on exponentially many bundles as may be the requirement under combinatorial auctions in some settings. Our mechanism is most similar to iterative combinatorial auctions [Parkes, 2001], which have been mainly studied for non-collaborative settings; we discuss them further in Section 2.3.

We again stress that our method automatically provides resource prices without domain knowledge, and to optimality, whereas most studies under either auction paradigm engineer markets and resources specifically for each problem domain, and often without theoretical guarantees. Our method may be combined with problem-specific approaches to yield additional resources that are more specialized but principled and easier to design. In our general version, such additional resources are based on bundles of resources, created to compensate for inefficiencies in the bundles currently available to the agents.

Potentially to automate resource pricing, machine learning techniques have been applied to learn bidding strategies. The most relevant work is by Schneider et al. [2005], which uses the notion of opportunity cost, based on rewards for each task, to learn the bidding strategies for each agent. However, this method in effect still requires the human designer to price resources, or tasks, through rewards. In contrast, learning can be effective for automatic pricing in scenarios such as oversubscribed domains where not all tasks are expected to be completed but a penalty is defined for unfinished tasks [Jones et al., 2007], since the penalty represents a natural quantity to be learned and minimized. Also, learning would be useful if uncertainty existed in the global cost function; however this setting is not considered by this thesis or the learning-based market-based algorithms mentioned above. Hence, in the mentioned papers learning is used simply as a substitute for optimization, which may be sensible if efficiency is more important than the quality of the solution, as it may require many iterations until a good predictor is learned.

2.2 Decomposition-Based Optimization and Planning

Distributed planning algorithms based on decomposition methods have been explored in the machine learning and planning communities. The most relevant works include [Guestrin and Gordon, 2002] for hierarchically factored Markov decision processes (MDPs), [Bererton et al., 2003] for a class of loosely coupled MDPs, and [Calliess and Gordon, 2008] which frames the market-based interactions as a game between adversarial agents computing resource prices and learning agents that represent the agents in the original problem. However, existing work, including the aforementioned, can be seen as applying a decomposition for linear or convex programs, which limits them to *infinitely divisible* resources only (although the infinitely divisible solution can often be a reasonable approximation even in the presence of discrete resources). Note that Calliess and Gordon [2008] consider self-interested agents; we will discuss in Section 4 how we plan to extend our work to the non-collaborative setting with similar ideas.

General frameworks for Dantzig-Wolfe decomposition for mixed integer programming have been explored, particularly in the operations research community (e.g., Vanderbeck [2000], Vanderbeck and Savelsbergh [2006]). Known as *branch-and-price* or *branch-and-price-and-cut*, these frameworks typically focus on sequential or tightly-coupled parallel execution. They use branch-and-bound for solving integer programs, and at each node of the search tree, employ Dantzig-Wolfe decomposition (and in some cases cutting planes) to solve a linear program relaxation and obtain bounds. (See [Barnhart et al., 1996, Ch. 6.7] for a good overview.) If we added branching to our algorithm, it would fit nicely into this line of research; however, it is not clear how to do so efficiently and robustly in a distributed framework. Much attention has been drawn to the implementation details of branch-and-price algorithms Vanderbeck [2005] which would need to be resolved in the distributed setting; in particular, keeping track of the branch tree can be tricky, and it is an art to find good branching strategies.[1] In contrast, our algorithm is simple to implement and intuitively distributed, and we demonstrate distributed operation over simple socket-based communication links. In addition, cuts considered in branch-and-price-and-cut papers are often problem-specific, most prevalently for the vehicle routing problem with time windows Petersen et al. [2008].

Thus our main contribution in the collaborative setting is twofold: first, removing branch-and-bound from the combination leads to an algorithm that is much more naturally distributed, without much loss in efficiency as we will see in our experiments. Second, previous work has not applied such algorithms to distributed multi-agent planning; we draw the connection to market-based planning, where our algorithm provides a principled way to introduce new resources and set prices of resources (in contrast to previous heuristic methods).

While not based on a decomposition, distributed constraint optimization (DCOP) is a popular distributed planning method that frames planning problems as constraint satisfaction problems and solve them in a fully-distributed fashion by having disjoint sets of variables assigned to each agent and using a search algorithm to find an assignment on which all agents agree [Modi et al., 2005], [Yeoh et al., 2008]. No definitive definition seems to exist in the multi-agent planning literature for DCOP, but most consider DCOP to be defined for discrete variables, to assume that each constraint only involves two agents, and to not allow hard constraints. On the other hand, distributed constraint satisfaction problems (DisCSP) [Yokoo et al., 1998] allow hard constraints but do not optimize a cost function. Hence both DCOP and DisCSP solve simpler problems than general MILPs.

2.3 Distributed Planning in Non-Collaborative Domains

There have been significant work in distributed planning in non-collaborative domains with divisible resources. As discussed in the previous section, no-regret learning has been used by Calliess and Gordon [2008] for settings with a finite number of agents and divisible resources. Also, no-regret learning has been studied in particular in routing with infinitesimal agents ([Blum et al., 2006],[Roughgarden, 2007]), with results in strong performance guarantees and price of anarchy statements. Price of anarchy is a useful concept in the analysis of mechanisms without dominant strategies, revealing their (in)efficiency in terms of social welfare. Blum et al. [2008] define price of *total* anarchy to be the price of anarchy achieved at the equilibrium reached by agents playing no-regret algorithms, which is a possible solution concept for our proposed work.

Mechanism design is the study of creating systems that achieve desired global behavior (such as a socially optimal outcome) under strategic agents. Distributed mechanism design has been studied prevalently in the area of routing and for divisible resources otherwise [Feigenbaum et al.,

[1]One possibility is to extend the techniques used to maintain search trees in distributed constraint optimization, although DCOP are much simpler problems than general MILPs.

2005], [Feigenbaum et al., 2001]. Indivisible resources have been also considered, for example for task allocation, with some success in approximation algorithms [Feigenbaum and Shenker, 2002].

Combinatorial auctions are a natural formulation for planning with indivisible resources in non-collaborative domains. However, little work exists on distributed combinatorial auctions, perhaps because majority of work on combinatorial auctions is concerned with finding incentive-compatible mechanisms (see [Conitzer, 2010] for a good overview).

Iterative combinatorial auctions [Parkes, 2001] are particularly relevant as their general form is equivalent to our collaborative algorithm: agents sequentially reveal preferences, by bidding on a small set of bundles at each round and receiving feedback on the current prices of the resources from the auctioneer, which enables them to change their bids in the next rounds. Although the algorithms are not distributed and the form of bundles allowed are often limited, iterative combinatorial auctions have been studied for non-collaborative settings under various valuations and forms of bundles, with results of convergence to equilibria [Cramton et al., 2005, Ch.2]. One particularly interesting algorithm is iBundle by Parkes and Ungar [2000] for XOR bids, which uses a primal-dual algorithm for LPs to price resources and is guaranteed to achieve bounded inefficiency.

3 Preliminary Work

In this section we present two sets of completed work. The first describes the preliminary work on collaborative agents, on which we will build our proposed fully distributed planning algorithm for non-collaborative agents, and the second provides a possible representation for efficiently describing the planning problems.

3.1 Price-and-Cut: A Market-Based Planning Algorithm for Collaborative Domains

For the collaborative setting, we designed and tested *price-and-cut market-based planning* (or *price-and-cut* for short). The algorithm *automatically* prices resources and furthermore designs new resources, correcting pricing imbalances by adding "derivative securities" based on the original resources to the market. Note that the collaborative setting can be framed purely as a global optimization problem, which we represent as a MILP. Our algorithm is based on Dantzig-Wolfe (D-W) decomposition, a classical column generation technique (see e.g. [Bertsimas and Tsitsiklis, 1997, Ch. 6]), in which the problem is reformulated and decomposed into a *master program* enforcing shared constraints over individual plans and *subproblems* for individual agents' problems. Among the plans generated by the subproblems so far, the master program finds the best set of plans, then "prices" the shared resources, or constraints, based on the corresponding dual multiplier values. Subproblems then adjust their objectives based on the constraint prices, which leads to better plans with respect to the shared constraints by incurring lower cost in the dual program. Since the subproblems are solved independently of each other, all computation can be distributed: agents plan in their local problems and communicate only when they might contend for resources. Hence our algorithm is well-suited for situations where it is costly to communicate large sets of constraints and objectives from individual problems to a server, or where centralized computation is not desirable for robustness.

As the individual plans are computed independently by each agent, the decomposition lets us take advantage of fast single-agent planning algorithms. For example, if the individual agents must plan motions in a continuous space, they can use fast randomized tree-growing algorithms (e.g. [Karamana and Frazzoli, 2010]); or, if the individual problems are network flow or path planning problems, they can use fast combinatorial algorithms. In some cases the best subproblem planner will be a mixed integer linear program (MILP) solver, as there has also been a great deal of research on using general-purpose MILP solvers for planning [Vossen et al., 1999]; in these cases we still gain greatly from factorization since the individual problems are typically much smaller than the joint problem.

While D-W decomposition provides factorization and distributed computation, it is originally defined only for linear programs (LPs). We extend the formulation to MILPs using Gomory cuts [Gomory, 1958], a general cutting-plane algorithm, while retaining optimality and finite-time guarantees of the D-W decomposition algorithm for LPs. Our algorithm can be seen as a special case of the *branch-and-price-and-cut* framework [Barnhart et al., 1996], and its simpler structure makes it much more naturally distributed than the general version. Constraints created by Gomory cuts can be interpreted as additional, *derivative* resources, whose pricing can be communicated to subproblem solvers just as for the original resources, maintaining the market-based distributed nature of D-W decomposition. As the Gomory cuts are constraints on nonlinear functions of multiple original resource consumption levels, their prices can represent ideas like discounts on particular baskets of resources, or penalties for exceeding a certain level of resource consumption.

Note that in the current algorithm, the master program is not solved in a distributed fashion. Two possible implementations exist: one is to have a trusted central server that performs the master program computations, and another is for every agent to solve the master program. The former may be less robust as the server can be a single point of failure, but communication cost at each iteration is linear in the number of agents. The latter is more robust, but with an added communication cost as every agent needs to communicate with every other agent about their current resource usages. In this section we assume the former implementation and do not address machine and communication failures. It is worth noting, however, that even with a central server, the algorithm is not hopeless against system and communication failure. The algorithm is *anytime*; it may (and typically will) generate feasible but suboptimal global solutions before terminating with an optimal solution, and the agents can execute the last feasible plan picked if they lose the central server during planning.

In the following sections, we describe the price-and-cut algorithm, and illustrate how new resources are formed and how one may interpret them. We then show experimental results comparing its performance against the CPLEX MIP solver on a set of synthetic problems that show its effectiveness in both centralized and distributed settings.

3.1.1 Dantzig-Wolfe decomposition

Consider the following mixed integer program (MIP) in the standard form, factored over n agents:

$$\min \sum_{i=1:n} c(x_i) \tag{3.1}$$

$$\text{s.t.} \sum_{i=1:n} A_i x_i = b \tag{3.2}$$

$$x_i \in C_i, \qquad i = 1, \ldots, n \tag{3.3}$$

where x_i represents the plan for agent i, C_i its domain, i.e., the set of plans satisfying agent i's individual constraints, and $c(x_i)$ its cost. Each x_i is a mixed integer vector (its elements may be integers or reals); we assume that C_i is bounded[2]. (3.2) defines the shared constraints, where each A_i is a matrix with the same number of rows, i.e., the number of shared constraints. Note that even if a natural multi-agent structure is not present, any MIP may be written in this form. The distributed formulation becomes especially beneficial if the program contains structure, for example a partially block-diagonal constraint matrix.

In Dantzig-Wolfe decomposition, we reformulate the program to consist of a *master program* and n *subproblems*. The master program is defined in terms of the n_i basic feasible solutions $x_i^1 \ldots x_i^{n_i}$ to each individual subproblem i. It includes variables w_i^j which indicate whether the individual solution $x_i^j \in C_i$ is part of the optimal joint solution:

$$\min \sum_{i=1:n} \sum_{j=1:n_i} c(x_i^j) w_i^j \tag{3.4}$$

$$\text{s.t.} \sum_{i=1:n} \sum_{j=1:n_i} w_i^j A_i x_i^j = b, \qquad w_i^j \in \{0,1\} \tag{3.5}$$

$$\sum_{j=1:n_i} w_i^j = 1, \qquad i = 1, \ldots, n. \tag{3.6}$$

For LPs, C_i is convex, hence we can allow $w_i^j \in [0,1]$, rendering the master program a linear program. However, for MIPs, a convex combination of plans may not be a valid plan, thus we must impose integrality constraints $w_i^j \in \{0,1\}$, rendering the master program an integer linear program. In this section we work with the linear program (LP) relaxation of the master program; below, we discuss how to use Gomory cuts to find an integer solution to the master.

The number of constraints in the master program may be much smaller than in the original formulation, as the master does not include the individual constraints. However, the number of variables in the master program may be prohibitively large, as it is equal to the number of possible individual basic plans for all agents. We thus define and solve the *restricted master program*, whose variables include only a subset of all possible plans. Suppose we have a basic feasible solution to the restricted master program. As in the simplex method for linear programming (see e.g. [Bertsimas and Tsitsiklis, 1997, Ch.3]), to determine whether our current solution is optimal, we can observe the

[2]We expect the extension to unbounded domains to be analogous to the extention for Dantzig-Wolfe decomposition for linear programs, which employs an additional set of master variables to denote extreme rays.

reduced cost of each non-basic variable in the solution. The reduced cost of a variable w_i^j can be written as

$$c(x_i^j) - q^{\mathrm{T}} A_i x_i^j - \mu_i, \tag{3.7}$$

where q corresponds to the values of dual variables for shared constraints (3.5) at the current basic solution, and μ_i to the value of the dual variable for the convexity constraint (3.6) for agent i. To find the variable w_i^j with the least reduced cost, we can solve

$$\min c(x_i) - q^{\mathrm{T}} A_i x_i \tag{3.8}$$
$$\text{s.t. } x_i \in C_i, \tag{3.9}$$

which defines the subproblem for agent i. Note that we have altered the subproblem *only* in its objective vector, so domain-specific solution algorithms will typically still be able to solve the altered subproblem. Now we can add a new variable to the restricted master, corresponding to the solution found by the subproblem, guaranteeing progress when the restricted master is solved again.

In more detail, suppose the restricted master constains k variables. Given a new subproblem solution x_i^j, we add a corresponding variable w_{k+1} to w, the term $c(x_i^j)w_{k+1}$ to the restricted master's objective, and the column $A_i x_i^j$ as the $(k+1)$-th column in the shared constraint matrix $\hat{A}$ corresponding to the shared constraints (3.5). The restricted master thus has the following form after k columns have been generated:

$$\min c^{\mathrm{T}} w \tag{3.10}$$
$$\text{s.t. } \hat{A}w = \hat{b}, \qquad w \in \{0,1\}^k \tag{3.11}$$

Here, we have incorporated the convexity constraints (3.6) into the shared constraints for convenience: we define $\hat{b} = (1, \ldots, 1, b^{\mathrm{T}})^{\mathrm{T}}$.

Recall that in the market-based view, each shared constraint is considered a resource. The dual values q communicated to subproblems then can be interpreted as resource prices, and $\hat{A}_{ij}$ as the usage level of resource i by plan j, as can be seen from the subproblem objective (3.8). If a constraint is heavily violated by plans currently known to the restricted master, the corresponding resource should have a high price, leading to new individual plans that avoid heavy usage of the resource. Note that we assume that resource usages can be counted in "units", allowing usage levels that are integer or rational, and rational coefficients in A_i. From here on, we only consider integer resource usages for notational simplicity.

Algorithm 1 shows an outline of the generic Dantzig-Wolfe decomposition algorithm (D-W). For linear programs, the master program and all subproblems are linear programs, and steps 1 and 2 can be solved by a linear program solver; in the following sections, we describe how to incorporate Gomory cuts to handle integer linear master programs.

As presented, Algorithm 1 may not terminate in a finite number of iterations when degeneracy is present; however, anticycling rules may be employed to ensure finite-time termination, as typically done in the simplex method. We refer the reader to [Dantzig, 1963, Ch. 23-1] for a discussion of anticycling rules applicable to the Dantzig-Wolfe decomposition algorithm.

Algorithm 1 Dantzig-Wolfe decomposition algorithm (D-W)

```
0. Create the restricted master program using subproblem solutions with 0 resource prices.
Repeat:
    1. Solve the restricted master program, returning dual values $q$, $\mu$.
    2. Solve subproblems using the new resource prices $q$.
    3. For each subproblem $i$'s returned solution $x_i$,
       If the subproblem solution's objective value $c(x_i) - q^{\mathrm{T}} A_i x_i < \mu_i$,
       Generate the new column and variable in the restricted master.
    4. If no new column has been generated in this iteration, terminate.
```

3.1.2 Incorporating a cutting plane algorithm

Cutting plane algorithms are designed to solve a mixed integer program by adding cuts, or additional constraints, to its LP relaxation, thereby "cutting off" fractional solutions and eventually reaching an integral optimal solution in the LP relaxation. Here we will use a cutting plane algorithm to solve the master program to optimality; for each set of cuts, we can use D-W decomposition to solve the LP relaxation of the master program. The process is summarized as Algorithm 2. This algorithm is naturally distributed: in the original Dantzig-Wolfe decomposition method, subproblems are solved independently by each agent, while the restricted master program is solved either by a designated agent, or by all agents simultaneously using a deterministic algorithm. Then, cuts can be created by a designated agent, or by all agents simultaneously using a deterministic algorithm.

Algorithm 2 The main algorithm: price-and-cut market-based planning

```
Repeat:
    1. Perform $m$ iterations of steps 1-4 in Dantzig-Wolfe decomposition algorithm (D-W), or perform the algorithm to termination ($m = \infty$), and return its optimal solution $w$ to the restricted master LP relaxation. Report whether $w$ is optimal for the full master LP relaxation; i.e., whether any new column was generated in step 4.
    2. If $w$ is integral, and is optimal for the full master LP relaxation, terminate.
    3. If $w$ is not integral, perform Gomory cuts and add constraints to the restricted master program, until $k$ cuts have been made or no more cuts are available ($k = \infty$) for the current LP solution $w$.
```

The main algorithm, *price-and-cut*, admits two parameters, and this flexibility allows different *schedules* over iterations of D-W and Gomory cuts. Different schedules may lead to varying optimality guarantees, and in practice, to different execution times. One particularly illuminating schedule is $m = 1$ and $k = \infty$ used in our experiments. This version of price-and-cut is equivalent to the D-W decomposition algorithm (Algorithm 1) where the restricted master program in step 1 is solved to its integer optimal solution, and it is guaranteed to terminiate in finite time with a globally optimal solution for IPs. On the other extreme is $m = \infty$ and $k = 1$, in which we solve the master LP relaxation exactly and introduce a single cut at each iteration. This latter schedule guarantees optimality in a finite number of iterations, but in practice may prove less efficient than other schedules, since we must find the optimal solution to the full master program at each iteration. In general, one may choose a particular schedule depending on the domain to balance between the number of columns and the number of cuts generated.

Two issues arise in applying cutting plane algorithms to D-W decomposition. First, since columns are generated incrementally, when we generate a new cut, we do not want to compute all of its elements immediately—else we lose the benefit of a small restricted master program. Thus we need an efficient way to record our cuts and compute new columns for the cut constraints as more variables are generated. Second, the new constraints must be taken into account in the subproblems. Intuitively, new constraints become new resources that can potentially be priced in the subproblems. (We refer to these resources as *derivative resources*, to differentiate them from the resources corresponding to the original constraints.) Derivative resource usages will be a function of original resource usages, since cuts are generated by performing operations on subsets of existing constraints. However, the functions will typically be nonlinear with respect to the variables in the subproblem, unlike the original resources in expression (3.8); depending on the form of the individual problems, it may be easy or difficult to plan to optimize combined (original and derivative) resource usage.

As we will see, it is relatively straight-forward to solve both issues when using Gomory cuts, our cutting plane algorithm of choice. Our choice of Gomory cuts is based on its simplicity and generality; but, the Gomory method is only one of many cutting-plane algorithms, and we expect that other rules may be used in place of Gomory cuts, as long as the two issues above can be resolved.

3.1.3 The Gomory cutting plane algorithm

Suppose we have an optimal basic solution to the LP relaxation of the restricted master program, associated with the basis B, which is composed of the columns of $\hat{A}$ that correspond to the basic variables in the solution.[3] To make a cut on the constraints (3.11), we first choose a row of B^{-1}, say $(B^{-1})_k$, such that $(B^{-1})_k\hat{b}$, the constant term in the constraint, is fractional. For example, one may choose a row randomly based on the magnitude of the fractional component of $(B^{-1})_k\hat{b}$. The cut then has the form:

$$\sum_j \lfloor (B^{-1})_k \hat{A}_{*j} \rfloor w_j \leq \lfloor (B^{-1})_k \hat{b} \rfloor, \tag{3.12}$$

where $\hat{A}_{*j}$ denotes the j-th column of $\hat{A}$. The cut is added to the constraint matrix $\hat{A}$, using a slack variable to maintain the standard form.

The cut is a *valid inequality*: any integral point that satisfies all original constraints in (3.5) satisfies the inequality. Also, the current LP optimal solution violates the new constraint, ensuring progress (for details, see e.g. [Bertsimas and Tsitsiklis, 1997, Ch. 11]). Furthermore, when no more cuts are available, we have an integral optimal solution. These properties guarantee that the Gomory cutting plane algorithm is a finitely terminating algorithm for solving MILPs, including the integer master program (3.4-3.6).

We now discuss how to resolve the two aforementioned issues under Gomory cuts.

[3]If we use the simplex method to solve the master LP relaxation, we automatically obtain the basis needed for making Gomory cuts, which is another potential advantage of Gomory cuts.

The cut recipe To solve the first issue of efficiently generating new columns for cut constraints, we can simply store a "recipe" for each cut. Since a Gomory cut only requires a sum over the columns, we can simply generate each coefficient as its column is generated. (Other coefficients are multiplied by zero and therefore ignorable, since their corresponding variable is not in the restricted master yet.) Let $(B^{-1})_k$ denote the row of the basis used to make the cut, and i refer to the row number of the cut in $\hat{A}$. The coefficient for a new column j is

$$\hat{A}_{ij} = \lfloor (B^{-1})_k \hat{A}_{(1:i-1)j} \rfloor, \tag{3.13}$$

where $\hat{A}_{(1:i-1)j}$ denotes the first $i-1$ rows of the j-th column of $\hat{A}$. Hence, if there exist multiple constraints in $\hat{A}$ arising from cuts, the coefficients must be generated iteratively, in the order in which the cuts were added to $\hat{A}$. Note that for each cut, we need to store additionally only the row $(B^{-1})_k$ used to make the cut, which is a vector the size of the basis set, or the number of rows (i.e., constraints) in $\hat{A}$ at the time of the cut.

Formulating derivative resources in subproblems For subproblem i, define y_i to be the usage vector of original resources and z_i to be the usage vector of derivative resources. Recall that the usage for resource k for column j is equal to the element $\hat{A}_{kj}$ of the master program's constraint matrix. Accordingly, as we saw in the subproblem objective (3.8), original resource usage y_i by a plan x_i is simply $y_i = A_i x_i$. We can also write z_{ik}, the usage of derivative resource k, in terms of y_i and previous elements of z_i. Let k' denote the row number in A corresponding to the derivative resource k, and let r_k refer to the cut recipe as defined in (3.13). Then, for column j, rewriting (3.13) in terms of y_i and z_i gives:

$$z_{ik} = \lfloor r_k u \rfloor, \tag{3.14}$$

where $u = (e_i^{\mathrm{T}} \;\; y_i^{\mathrm{T}} \;\; z_{i(1:k-1)}^{\mathrm{T}})^{\mathrm{T}}$, with e_i an n-dimensional unit vector whose i-th element is 1, corresponding to the convexity constraints in the master program. Incorporating the new variables, the subproblem objective now becomes

$$\min c(x_i) - q^{\mathrm{T}} (y_i^{\mathrm{T}} \;\; z_i^{\mathrm{T}})^{\mathrm{T}}$$

and we add the expressions above for y_i and z_{ik} as additional constraints. Furthermore, we can encode the non-linear constraints (3.14) as integer linear constraints:

$$\begin{aligned} z_{ik} &\leq r_k u, \\ z_{ik} &\geq r_k u - (1 - \frac{1}{2M}), \\ z_{ik} &\in \mathbb{Z} \end{aligned}$$

where M is the least common multiple of the denominators of the coefficients in r_k, which allows us to use a general mixed integer program (MIP) solver to solve the subproblems.

Depending on the particular subproblem solver we are using, we may have to handle derivative resources in a domain-specific way. In general, adding derivative resources to a subproblem can increase the size of its state space, since the subproblem planner may now need to track resource usage where it didn't before. For example, in a path planner, with resources representing tickets which confer the right to pass through specific areas, the original subproblem planner can simply add costs to edges which enter controlled areas, and buy tickets as needed without remembering total resource usage. But, each derivative resource will correspond to a combination of tickets; so, to make action costs Markov, we must keep track of which tickets we have bought so far, in order to compute the proper charge or credit when we purchase a ticket involved in a combination.

A more complete examination of a variety of subproblem solvers is future work. However, we point out here that there is a limit to the possible increase in state space size, since at worst we need to keep track of our usages of all of the original resources: usage of any derivative resource is a deterministic function of the original resource usages. Furthermore, depending on the domain, a subproblem solver may already keep track of some or all of the original resource usages, in which case the state space increase is limited still further.

3.1.4 An Example: Derived Resources

In addition to automatically pricing existing resources, our algorithm creates *derivative* resources that guide the agents towards creating plans that will lead to a global integer solution in the master. Before we show experimental results on real-sized problems in Section 3.1.5, here we present a simple example to provide intuition for the form of derivative resources and their interpretation.

Consider a small grid world consisting of four positions and two agents, as shown in Figure 1. Agent 1 starts in position 1, and must to go to position 4; agent 2 must do the opposite. At each time step, an agent can move to a neighboring position, with constraints: 1. agents cannot occupy the same position at the same time and 2. agents cannot "swap" positions, effectively occupying the same "edge" between two positions at the same time. It is easy to see that there is a bottleneck at position 2, and one agent must wait in position 3 for the other to pass through position 2.

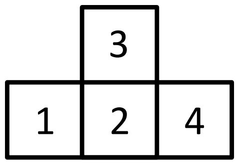


Figure 1: A grid world with four positions

In a typical run, our algorithm creates five cuts before termination, some of which we will examine here. Let our variables be binary, such that if $x^i_{jt} = 1$ then agent i is at position j at time t. The discretizing (floor) operations in Gomory cuts make it possible for us to write the cut as a combination of logical constraints between the position variables, which is quite natural to intepret. For example, we will see conjunctions (variables connected by $\wedge$), which will represent partial paths, and are examples of "baskets" of resources. For convenience, we will use the convention that *true* and *false* correspond to 1 and 0 respectively and write our cuts in mixed logic and arithmetic notations. The first cut we obtain is of the form:

$$(x^1_{22} \wedge x^1_{43}) + (x^2_{22} \vee (x^2_{42} \wedge x^2_{23})) \leq 1.$$

First we see that, as expected, the cut heavily concerns position 2, which is the bottleneck resource. Furthermore, this cut penalizes agent 1 for the partial path $(2@t2, 4@t3)$ of being at position 2 at time 2 and position 4 at time 3, and agent 2 for the partial path $(4@t2, 2@t3)$, but the penalization simply corresponds to the original constraint that the agents should not swap positions in a time step. However, it also penalizes agent 2 being in position 2 at time 2, only if agent 1 tries to be in position 2 at time 2 *and* in position 4 at time 3, quantifying the evident correlation between the occupation of position 2 at time 2 and the movement of the two agents in the vicinity.

Perhaps the more interesting is the second cut, which is derived from the first cut as well as two of the original constraints:

$$[(x^1_{12} \wedge x^1_{23}) \vee (x^1_{22} \wedge x^1_{43})] + (x^2_{22} \wedge x^2_{13})) \leq 1.$$

This new constraint penalizes agent 1 for taking the partial path $(2@t2, 4@t3)$. However, it does not penalize agent 2 for taking the partial path $(4@t2, 2@t3)$ and thus does not duplicate any of the original constraints; the derivative resource provides a way to guide the agents that had been previously unavailable. In the final solution agent 1 ends up yielding position 2 at time 2 to agent 2 partly due to this constraint; the constraint is tight at the final optimal solution, where we see the partial path $(1@t2, 2@t3)$ for agent 1.

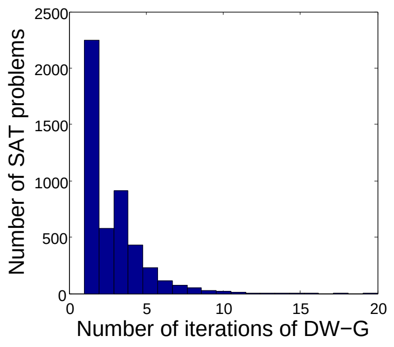
3.1.5 Experiments

Here we demonstrate the effectiveness of our algorithm in both centralized and distributed settings on randomly-generated factored zero-one integer programs. This domain is both difficult (general-purpose solvers take 10^3–10^4 seconds) and relevant: for example, the method of propositional planning [Kautz and Selman, 1996] encodes a planning problem as a similar constraint-satisfaction problem, with feasible points corresponding to feasible plans, and an objective corresponding to plan cost. The experiments demonstrate that (1) when applicable, market-based planning can lead to large efficiency gains, outperforming an industrial-strength MILP solver (CPLEX [ILOG, 2010]) for large problems, *even if* we force all computation to be sequential; and (2) the benefit can become even greater in distributed settings, where we can take advantage of each agent's computational resources without a large communication overhead.

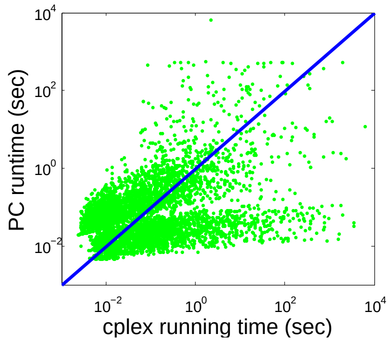
For each factored program, we partition our variables into subsets, and generate a set fraction of our constraints among variables in the same subset; for the remaining shared constraints, we select variables at random from all subsets at once. In our experiments we use random 3SAT constraints, together with the objective "set as few variables to 1 as possible." We picked SAT constraints in order to set the ratio of constraints to variables near the well-known hardness threshold of 4.26; however, the resulting problem is not a SAT instance due to the objective, and therefore SAT-specific solvers are not directly applicable. Our implementation uses the CPLEX MIP solver [ILOG, 2010], a widely-used commercial optimization package, for the integer program subproblems, and the CPLEX simplex LP solver for the restricted master LP relaxation.

We compare the runtimes of the centralized and distributed versions of price-and-cut to those of CPLEX's MIP solver on the original formulation of the problem, using 4713 randomly generated problems. The problem sizes varied between 2 to 10 subproblems, 10 to 200 variables, and 41 to

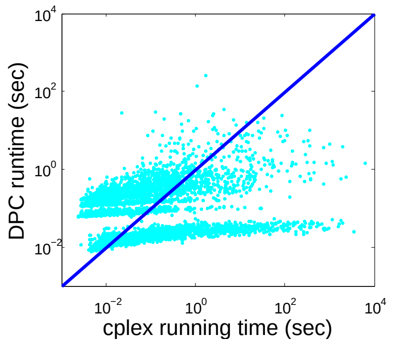
900 total clauses, of which 0.11% to 17.65% were shared. The ratio of the number of clauses to the number of variables was between 4.0 and 4.5. In the centralized version of price-and-cut (PC), subproblems were solved sequentially on one CPU (alternating with the master program), incurring no communication cost. The distributed runs (DPC) were performed on two 8-core machines, where the subproblem solvers communicated with each other over sockets. One process was dedicated to solving the restricted master LP and making cuts.



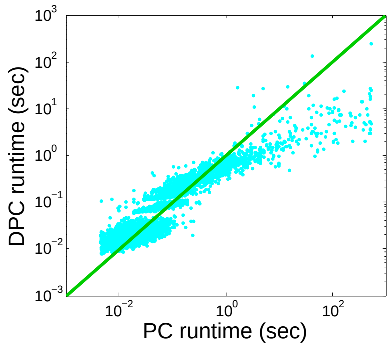
(a)



(b) PC vs. CPLEX



(c) DPC vs. CPLEX



(d) DPC vs. PC

Figure 2: (a) number of PC iterations performed, (b)–(d) runtime comparisons

Figure 2(a) shows the distribution of the number of iterations of PC (steps 1-3 in Algorithm 2) to reach optimality. Most cases required only a few iterations, and only 34 cases out of 4713 required more than 10. In Figure 2(b), each point represents a problem; on the x axis is the runtime of CPLEX, and on the y axis is the runtime of PC, both in log scale. The diagonal line is the identity: any point below the line represents a problem instance where PC outperforms CPLEX. Our observations suggest that CPLEX running time is heavily dependent on the number of total clauses in the problem. We can see here that PC outperforms CPLEX handily for larger problem sizes, as the advantage of market-based planning outweighs the overhead in our implementation: PC outperformed CPLEX on 92.38% of the instances where CPLEX took more than 1 second.

Figure 2(c) is an analogous plot for the distributed version of price-and-cut, and exhibits similar trends, if not more pronounced. Finally, Figure 2(d) gives a similar comparison between PC and DPC: DPC outperformed PC on 96.12% of the instances where PC took more than 1 second. It is interesting to note that, even for problems which take only $1s$ to solve, the benefit of parallelism outweighs communication overhead, despite our simple implementation.

3.2 First-Order Mixed Integer Linear Programming

First-order programs, introduced in Gordon et al. [2009], are a generalization of MILPs and first order logic that combine the representational and reasoning strengths of both languages. In essence, first-order programs are *lifted* MILPs, where a group of constraints may be represented as one constraint, modified by quantifiers analogous to first-order logic quantifiers *for all* and *there exists*. These quantifiers not only serves as syntactic convenience but also gives first-order programs the power to represent constraints over infinitely many objects or unknown objects. In Gordon et al. [2009] we describe the language and a complete lifted inference algorithm based on Gomory cuts that allows us to reason more efficiently when first-order constraints are present. It does so by creating cuts based on the first-order constraints and in effect treating the set of potentially infinitely many constraints as one; hence the lifting may benefit even ordinary MILPs with "lifted" constraints over a finite set of objects.

While incorporating first-order reasoning into multi-agent planning is beyond the scope of our thesis, our work on first-order programs incorporates ideas for optimizing integer programs that we believe will be useful in our proposed research. In addition, we plan to investigate whether first-order programs can serve well as the representational language for our domains.

4 Proposed Research

Price-and-cut provides an optimal distributed market-based planning algorithm for collaborative domains. However, the master program is a centralized sequential computation, whether in implementation it is performed on one server or on multiple agents simultaneously. In this proposed thesis, we will investigate distributing the master program among the agents for faster computation and further robustness to failure, and do so for *non-collaborative* settings where agents may possess interests conflicting with the global objective of the system or the socially optimal joint plans. (Whether adversarial or simply maximizing one's utility, we will refer to agents in non-collaborative settings as *self-interested* agents.)

Planning for collaborative agents can be seen as a global optimization problem, hence the solution we have described so far is a general-purpose distributed solver for MILPs. However, when planning for self-interested, possibly adversarial, agents, we desire a stronger guarantee: that the agents, given our market mechanism, will act to converge to a socially optimal joint plan. One formalization of a social optimum is a game-theoretic equilibrium: if agents know that following a certain strategy will lead to an equilibrium, agents have no incentive to change their strategy. Another, weaker formulation is a boundedly-rational equilibrium: agents can not discover an incentive to change their strategy using some limited type of computation. The latter formulation may lead to a smaller price of anarchy, and thus may be more appropriate for us, since our goal is to not only incentivize the agents with a guarantee of equilibria but also seek some notion of social optimum, or a globally good solution.

Our initial approach will be to exploit duality and no-regret learning in an analogous fashion to Calliess and Gordon [2008], who addressed the simpler problem of divisible resources, leading to a convex optimization problem. As in [Calliess and Gordon, 2008], we will extend the master LP computation in price-and-cut by assigning subsets of resources to the agents to be priced, and using no-regret learning for every agent to solve their subproblems as well as resource pricing. We hope to establish similar guarantees to [Calliess and Gordon, 2008], which gives asymptotic guarantees on the cost of the average plan (Theorem 3.4, [Calliess and Gordon, 2008]) and its convergence to a Nash equilibrium (Theorem 4.1, [Calliess and Gordon, 2008]).

Certainly, non-convexity comes with an additional set of challenges. The main challenge will be to generate cuts in a distributed manner in addition to solving the master LP in a distributed fashion. To do so, we must ensure that agents are given an incentive to only generate valid and useful cuts. Also, for completeness, theoretically we need to be able to generate derivative resources from arbitrary combinations of resources, which may require communication between every pair of resource-pricing agents. To reduce communication, we may be able to perform further decomposition if such a structure is present in the problem, or we can explore strategies for finding good cuts while minimizing communication, which could be implemented as agents discovering agents with whom to communicate.

Finally, we will evaluate our proposed family of algorithms on various multi-agent planning domains. In particular, we have begun work on applying price-and-cut to a wider variety of problems such as robot motion planning in continuous domains, where the domains may yield specialized individual planning algorithms that may be adapted as fast subproblem solvers. In the process we will be able to test its performance in terms of solution quality and speed in practice under approximate subproblem solvers, various schedules, and early stopping. Other planning domains of interest include production or task allocation, and cooperative control problems under limited resources, where a MILP formulation of the problems is often available and thus our algorithms are readily applicable. In such areas we hope that our distributed algorithm will enable us to scale the problems to sizes that were previously impossible to solve with existing solvers.

5 Proposed Schedule

- Fall 2010: Proposal. A further empirical study of price-and-cut in robot path planning in a continuous domain.
- Spring 2011: Distributed LP solving with incentives.
- Summer-Fall 2011: Distributed cut generation.
- Spring 2012: Finish writing thesis, defend.

References

M. Alighanbari and J. P. How. Cooperative task assignment of unmanned aerial vehicles in adversarial environments. In *Proc. IEEE American Control Conference (ACC)*, pages 4661–4667, 2005.

Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh, and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1996.

C. Bererton, G. Gordon, S. Thrun, and P. Khosla. Auction mechanism design for multi-robot coordination. In *Advances in Neural Information Processing Systems*, 2003.

M. Berhault, H. Huang, P. Keskinocak, S. Koenig, W. Elmaghraby, P. Griffin, and A. Kleywegt. Robot exploration with combinatorial auctions. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, MA, 1997.

A. Blum, E. Even-Dar, and K. Ligett. Routing without regret: on convergence to nash equilibria of regret-minimizing algorithms in routing games. In *PODC '06: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 45–52, 2006.

Avrim Blum, Mohammad Taghi Hajiaghayi, Katrina Ligett, , and Aaron Roth. Regret minimization and the price of total anarchy. In *STOC*, 2008.

Jan-Peter Calliess and Geoffrey J. Gordon. No-regret learning and a mechanism for distributed multi-agent planning. In *Proc. 7th Intl. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008.

Vincent Conitzer. Comparing multiagent systems research in combinatorial auctions and voting. *The Annals of Mathematics and Artificial Intelligence (AMAI)*, 2010.

Peter Cramton, Yoav Shoham, and Richard Steinberg, editors. *Combinatorial Auctions*. MIT Press, Cambridge, MA, 2005.

George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

M. Dias and A. Stentz. Opportunistic optimization for market-based multirobot control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2002.

M. Bernardine Dias. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, CMU, 2004.

J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Dial-M '02*, 2002.

J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *J. Compt. Syst. Sci.*, 63(1):21–41, 2001.

J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A bgp-based mechanism for lowest-cost routing. *Distrb. Comput.*, 18(1):61–72, 2005.

Brian Gerkey. *On Multi-Robot Task Allocation*. PhD thesis, University of Southern California, 2003.

R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc.*, 64(5):275–278, 1958.

Geoffrey J. Gordon, Sue Ann Hong, and Miroslav Dudik. First-order mixed integer linear programming. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009.

Carlos Guestrin and Geoffrey J. Gordon. Distributed planning in hierarchical factorered mdps. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002.

L. Hunsberger and B. J. Grosz. A combinatorial auction for collaborative planning. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS)*, 2000.

ILOG. Ibm ilog cplex optimizer, 2010. `http://www.ilog.com/products/cplex/`.

E. Gil Jones, M. Bernardine Dias, and Anthony Stentz. Learning-enhanced market-based task allocation for oversubscribed domains. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2007.

Sertac Karamana and Emilio Frazzoli. Incremental sampling-based optimal motion planning. In *Proceedings of the Robotics: Science and Systems*, 2010.

Henry Kautz and Bart Selman. Pushing the envelope: Planning, propositional logic and stochastic search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201, Menlo Park, August 4–8 1996. AAAI Press / MIT Press. ISBN 0-262-51091-X.

S. Koenig, C. Tovey, X. Zheng, and I. Sungur. Sequential bundle-bid single-sale auction algorithms for decentralized control. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1359–1365, 2007.

S. Koenig, X. Zheng, C. Tovey, R. Borie, P. Kilby, V. Markakis, and P. Keskinocak. Agent coordination with regret clearing. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pages 101–107, 2008.

S. Koenig, P. Keskinocak, and C. Tovey. Progress on agent coordination with cooperative auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010.

M. Lagoudakis, V. Markakis, D. Kempe, P. Keskinocak, S. Koenig, A. Kley-wegt, C. Tovey, A. Meyerson, and S. Jain. Auction-based multi-robot routing. In *Robotics: Science and Systems*, 2005.

T. Lemaire, R. Alami, , and S. Lacroix. A distributed tasks allocation scheme in multi-uav context. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal (AIJ)*, 2005.

R. Nair, T. Ito, M. Tambe, and S. Marsella. Task allocation in the rescue simulation domain: A short note. In *In RoboCup-2001: The Fifth Robot World Cup Games and Conferences*, 2002.

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

David C. Parkes and Lyle H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proc. 17th National Conference on Articial Intelligence (AAAI-00)*, pages 74–81, 2000.

Bjorn Petersen, David Pisinger, and Simon Spoorendonk. Chvatal-gomory rank-1 cuts used in a dantzig-wolfe decomposition of the vehicle routing problem with time windows. *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 397–419, 2008.

W. B. Powell. A stochastic formulation of the dynamic assignment problem, with an application to truckload motor carriers. *Transportation Sci.*, 30(3):195–219, 1996.

Arthur Richards, John Bellingham, Michael Tillerson, and Jonathan How. Coordination and control of multiple uavs. In *Proceedings of American Institute of Aeronautics and Astronautics (AIAA) Guidance, Navigation, and Control Conference and Exhibit*, 2002.

Tim Roughgarden. Selfish routing and the price of anarchy (survey). *OPTIMA*, 2007.

Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *AAAI-05*, 2005.

J. Schneider, D. Apfelbaum, D. Bagnell, and R. Simmons. Learning opportunity costs in multi-robot market based planners. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

C. Tovey, M. Lagoudakis, S. Jain, and S. Koenig. The generation of bidding rules for auction-based robot coordination. *Multi-Robot Systems: From Swarms to Intelligent Automata*, pages 3–14, 2005.

Francois Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48:111–128, 2000.

François Vanderbeck. Implementing mixed integer column generation. pages 331–358. Springer, 2005.

François Vanderbeck and Martin W. P. Savelsbergh. A generic view of dantzig-wolfe decomposition in mixed integer programming. *Operations Research Letters*, 34:296–306, 2006.

T. Vossen, M. Ball, and R. H. Smith. On the use of integer programming models in AI planning. In *IJCAI-99*, 1999.

W. Yeoh, A. Felner, and S. Koenig. Bnb-adopt: An asynchronous branch-and-bound dcop algorithm. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 591–598, 2008.

Makoto Yokoo, Edmund H. Durfee, Toru Ishida, and Kazuhiro Kuwabara. The distributed constraint satisfaction problem: Formalization and algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10:673–685, 1998.

X. Zheng and S. Koenig. K-swaps: Cooperative negotiation for solving task-allocation problems. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 373–379, 2009.

X. Zheng and S. Koenig. Sequential incremental-value auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2010.