Decomposition-Based Optimal Market-Based Planning for Multi-Agent Systems with Shared Resources

Sue Ann Hong
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
sahong@cs.cmu.edu

Geoffrey J. Gordon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
ggordon@cs.cmu.edu

Abstract

Market-based algorithms have become popular in collaborative multi-agent planning due to their simplicity, distributedness, low communication requirements, and proven success in domains such as task allocation and robotic exploration. Most existing market-based algorithms, however, suffer from two main drawbacks: resource prices must be carefully handcrafted for each problem domain, and there is no guarantee on final solution quality. We present an optimal market-based algorithm, derived from a mixed integer program formulation of planning problems. Our method is based on two well-known techniques for optimization: Dantzig-Wolfe decomposition and Gomory cuts. The former prices resources optimally for a relaxed version of the problem, while the latter introduces new derivative resources to correct pricing imbalances that arise from the relaxation. Our algorithm is applicable to a wide variety of multi-agent planning domains. We provide optimality guarantees and demonstrate the effectiveness of our algorithm in both centralized and distributed settings on synthetic planning problems.

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 centralized optimization problem, as the joint state over all agents is exponential in the number of agents. Hence, given the natural factorization over agents, it is beneficial to seek a distributed solution, where each agent solves its individual local planning problem with a fast single-agent planning algorithm.

Market-based planners provide a natural and intuitive framework to leverage such multi-agent structure to yield an efficient algorithm for distributed planning in collaborative multi-agent systems: each agent bids for resources, using its planner both to decide which resources are worth acquiring and to decide how to use resources if acquired. Market-based planners impose low communication costs, are simple to implement, and have been shown to work well, for example in the task allocation domain [7, 10, 27]. However, they suffer from several well-known limitations. First, to set up a good market is something of an art: a human designer must choose carefully, for every new problem domain, a set of commodities and a market structure, tuning both to balance planning effort against suboptimality. Second, most market-based planners cannot offer any guarantees on the final overall plan quality. Finally, most algorithms degrade quickly in the presence of additional constraints that couple the agents' solutions, such task ordering constraints or collision avoidance. (See [8] for a further discussion.)

To address these problems, we design a principled algorithm which *automatically* and *optimally* prices resources in any domain. We formulate planning problems as mixed integer linear programs (MILPs); MILPs are a popular representation not only in artificial intelligence [26], but also in operations research [20], cooperative control [21, 1], and game theory [22]. The MILP representation lets us easily generalize the usual task allocation setting to handle complex constraints over multiple agents. Also, the popularity of MILPs makes our method readily applicable,

Appearing in Proceedings of the 14^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2011, Fort Lauderdale, FL, USA. Volume 15 of JMLR: W&CP 15.

since for many problems a MILP formulation already exists. Finally, MILPs are commonly used to represent problems of planning under uncertainty (e.g., [20]). Most market-based planning algorithms do not account for uncertainty, e.g. in future resource availability or future tasks, and therefore can return suboptimal plans in such domains; our method therefore holds the promise of extending optimal market-based planning to uncertain domains. (Our experimental results so far are only in deterministic domains, so realizing this promise remains future work; however, we provide an example in the Appendix to illustrate the idea).

Our algorithm is based on Dantzig-Wolfe (D-W) decomposition [3, Ch. 6], a classical column generation technique for linear programs (LPs), in which the problem is reformulated into a *master program* enforcing shared resource constraints, along with *subproblems* for each agent. The master program corresponds to the auctioneer in market-based planners, choosing the resource allocation and giving agents (subproblems) the current prices of the resources. Based on the current prices, the agents iteratively change their demands towards a globally optimal solution.

As D-W decomposition is defined only for LPs, we extend the formulation to MILPs using Gomory cuts [12], which allows us to retain optimality and finite-time convergence guarantees. The cutting plane algorithm generates new constraints to "cut off" the optimal solution of the LP relaxation. Each new constraint can be interpreted as a *derivative* resource representing the discretized consumption level of a combination of the original resources. Hence its price can represent ideas like discounts or penalties on particular baskets of resources (see Sec. 4 for examples).

Since subproblems are independent of one another, all subproblem computation can be distributed for speed and robustness. While we do not specifically address machine failure or communication complexity in this paper, our algorithm is *anytime*: feasible joint plans may be generated before reaching the optimal solution, and if the situation requires, can be executed. Also, since we consider collaborative domains, the auctioneer computation may be replicated on multiple machines (or agents) for redundancy. Finally, our algorithm has low communication requirements, since agents communicate only when they might contend for resources, and they never need to communicate their individual problems or complete plans.

We conduct experiments on randomly-generated factored integer programs, which are both difficult to solve and a versatile representation for planning problems, as well as on simulated unmanned aerial vehicle (UAV) task allocation problems. The experiments demonstrate that (1) when applicable, market-based planning can lead to large efficiency gains, outperforming an industrial-strength MILP solver (CPLEX [15]) 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.

2 RELATED WORK

Collaborative market-based planning has been popular in particular in robotic task allocation domains [7, 10, 18]. Most such algorithms are based on sequential single-item (SSI) auctions, and while older studies have been mostly empirical, recent work has shown that SSI auctions give a constant approximation factor in some types of problems [28]. However, SSI auctions cannot capture constraints that couple multiple resources. Combinatorial auctions allow resource coupling by having agents bid on *bundles* of resources. They have been studied extensively in competitive settings [5], and iterative combinatorial auctions (ICAs) [19] in particular parallel our algorithm: the auctioneer gives agents the current price of the resources at each round and allows them to rebid, to hopefully achieve a better solution, and to avoid eliciting preferences on all possible bundles of resources. One can view ICAs as having a predetermined set of possible derivative resources that is the set of allowed bundles, while our algorithm computes on the fly which bundles are needed.

Distributed planning algorithms based on decomposition methods have been explored in the machine learning and planning communities; the most relevant research includes work on hierarchically factored Markov decision processes (MDPs) [13], loosely coupled MDPs [2], and the approach by Calliess and Gordon which frames the market-based interactions as a game between adversarial agents and learning agents [4]. However, existing works (including [13] and [4]) employ 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).

General frameworks for Dantzig-Wolfe decomposition for mixed integer programming have been explored, particularly in the operations research community [23, 25]. Known as *branch-and-price (B&P)*, these frameworks typically focus on sequential execution. They use branch-and-bound and sometimes cutting planes, and at each node of the search tree employ D-W decomposition to solve a linear program relaxation and obtain bounds. If we added branching to our algorithm, it would fit nicely into this line of re-

search; 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 B&P algorithms [24] 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. In contrast, our algorithm is simple to implement and intuitively distributed.

Very limited work exists on applying a decomposition algorithm for MILPs to distributed planning. Karaman et al. [16] claim to apply B&P to UAV task allocation, but only solve the master as a LP without branching, and cannot guarantee their solution will satisfy integral constraints. Holmgren et al. [14] apply B&P to supply-chain management and guarantee feasible solutions, but forgo optimality as deep branches make the procedure impractical. Also, they do not evaluate the algorithm in any distributed manner. To our knowledge, our combination of D-W decomposition and cuts, which we found to be a practical and easily distributed algorithm, has not been explored for distributed planning.

Thus our main contribution is twofold: first, from the optimization viewpoint, removing branch-and-bound from the combination leads to an algorithm that is much more naturally distributed, without much loss in efficiency (as seen in our experiments). Second, from the market-based planning viewpoint, previous work has not drawn the connection between multi-agent planning and distributed algorithms for mathematical programming and pricing resources; by doing so, our algorithm provides a principled way to introduce new resources and set their prices, a problem which has proven difficult in previous research and which has typically been solved with heuristic methods.

3 ALGORITHM

3.1 Problem Formulation

We formulate multi-agent planning problems as MIPs 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); for convenience we assume that C_i is bounded. (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.

In the following section, we introduce Dantzig-Wolfe (D-W) decomposition [3, Ch. 6], which is defined for LPs. We then describe how D-W decomposition can be combined with a cutting plane algorithm to solve MILPs.

3.2 Dantzig-Wolfe Decomposition

Consider a problem of the form (3.1-3.3) where each C_i is convex (relaxed from the presentation above). In D-W 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 $x_i^1 \ldots x_i^{n_i} \in C_i$ to each individual subproblem i; note n_i may be very large. Its variables are w_i^j, indicators of whether the individual solution x_i^j is part of the optimal joint solution:

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

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

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

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 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, selected by variable generation. In more detail, we can find a basic feasible solution to (3.4–3.6) by solving the restricted master, which is identical to (3.4–3.6) except that some w_i^j are forced to be zero. As in the simplex method for LPs (see, e.g., [3, Ch. 3]), to determine whether our current solution is optimal, we can observe the *reduced cost* of each non-basic variable in the solution. The reduced cost of a variable w_i^j is

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

where q contains the values of dual variables for shared constraints (3.5) at the current basic solution, and μ_i is the value of the dual variable for the sum-to-1 constraint (3.6) for agent i. To find the variable w_i^j with the least reduced cost (so we can add it to our basis), we can solve a modified subproblem for agent i:

$$\min c(x_i) - q^{\mathrm{T}} A_i x_i \tag{3.8}$$

$$\text{s.t. } x_i \in C_i. \tag{3.9}$$

Note that we have altered the subproblem *only* in its objective, so domain-specific algorithms will typically still be able to solve the altered subproblem. If w_i^j was

not already in the restricted master, we can now add it, guaranteeing progress when we solve the restricted master again.

For conciseness, we write the restricted master as:

$$\min \hat{c}^{\mathrm{T}} w \text{ s.t. } \hat{A} w = \hat{b}, \tag{3.10}$$

where we have renumbered the variables to be $w_1 \dots w_k$, and collected together the constraints $\hat{A}w = \hat{b}$ and objective $\hat{c}^{\mathrm{T}}w$. A subproblem solution x_i^j corresponds to a column $A_i x_i^j$ in $\hat{A}$ and an entry $c(x_i^j)$ in $\hat{c}$. For convenience, we have incorporated the sum-to-1 constraints (3.6) into $\hat{A}$, so $\hat{b} = (1, \dots, 1, b^{\mathrm{T}})^{\mathrm{T}}$.

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 saturated by plans currently in the restricted master, the corresponding resource will have a positive price, leading to new individual plans that avoid heavy usage of the resource.

Algorithm 1 shows an outline of the Dantzig-Wolfe decomposition algorithm. For linear programs, the master program and all subproblems are linear programs, and steps 1 and 2 can be solved by a LP solver.

Algorithm 1 Dantzig-Wolfe decomposition algorithm

0. Solve subproblems with resource prices = 0; use solutions to initialize the restricted master.[1]

Repeat:

1. Solve the restricted master; get dual values q, μ.
2. Solve subproblems using new resource prices q.
3. For each subproblem i's returned solution x_i, if the objective value satisfies $c(x_i) - q^{\mathrm{T}} A_i x_i \leq \mu_i$, generate a new column and variable in the restricted master.
4. If no new column has been generated in this iteration, terminate.

As presented, Alg. 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. See [6, Ch. 23-1] for a discussion on anticycling rules applicable to the D-W decomposition algorithm.

[1]To handle infeasibility in the restricted master, we include, for each agent, a fictitious plan that has feasible resource usages and a very large cost. Picking fictitious plans will lead to high prices for over-subscribed resources, guiding the subproblems to return plans that better meet the resource constraints.

3.3 Incorporating a Cutting Plane Algorithm

To represent and solve a mixed integer program using D-W decomposition, we must add the integrality constraints $w \in \{0, 1\}^k$ to the master (3.5) and the restricted master (3.10). (If C_i is convex, these constraints are unnecessary. However, for MIPs, a convex combination of plans may not be a valid plan, so we need the integrality constraints.) To do so, we will employ a cutting plane algorithm. For notational simplicity we will assume that each A_i has integer entries, so that all resource usages are integral.

Cutting plane algorithms 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. (In particular, we use the Gomory method for generating cuts.) To use a cutting plane algorithm to solve the master program, for each set of cuts we can use D-W decomposition to solve the LP relaxation of the master program (i.e., without the integrality constraints). The process is summarized as Alg. 2. This algorithm is naturally distributed: as in the original D-W decomposition method, subproblems are solved independently by each agent, while the restricted master program is either solved by a designated agent, or replicated on several agents simultaneously using a deterministic algorithm. Then, cuts can be created by a designated agent, or by several agents simultaneously using a deterministic algorithm.

Algorithm 2 Price-and-cut market-based planning

Repeat:

1. Perform m iterations of steps 1-4 in Dantzig-Wolfe decomposition algorithm, 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, which 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 $k = \infty$; this version of price-and-cut is equivalent to applying D-W decompo-

sition (Alg. 1) to the MIP and solving the restricted master in step 1 to its integer optimal solution. 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 at each iteration. We give optimality results for general schedules in Sec. 3.5.

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. Thus we need an efficient way to record our cuts and compute new columns for the cut constraints incrementally. Second, the new constraints must be taken into account in the subproblems. Intuitively, new constraints become new resources; 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 straightforward to solve both issues when using Gomory cuts, which is why we choose Gomory cuts here. 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.

3.4 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.[2] To make a cut on the constraints (3.10), 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:

$$\sum_j \lfloor (B^{-1})_k \hat{A}_{*j} \rfloor w_j \leq \lfloor (B^{-1})_k \hat{b} \rfloor, \tag{3.11}$$

[2]Using the simplex method to solve the master LP relaxation automatically gives us the basis needed for making Gomory cuts, which is another advantage of Gomory cuts.

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 new inequality. Also, the current LP optimal solution violates the new constraint, ensuring progress (for details, see [3, 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.

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

3.4.1 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

$$\hat{A}_{ij} = \lfloor (B^{-1})_k \hat{A}_{(1:i-1)j} \rfloor, \tag{3.12}$$

where $\hat{A}_{(1:i-1)j}$ denotes the first $i-1$ rows of the j-th column of $\hat{A}$. For each cut, we need to store only the row $r_k = (B^{-1})_k$ used to make the cut, which is a vector the size of the basis set (the number of rows (i.e., constraints) in $\hat{A}$ at the time of the cut). Note that, for multiple cuts, we need to generate coefficients serially, in the order in which the cuts were added to $\hat{A}$, since each cut may depend on previous ones.

3.4.2 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 (3.12). Then, for column j, rewriting (3.12) in terms of y_i and z_i gives:

$$z_{ik} = \lfloor r_k u \rfloor, \tag{3.13}$$

where $u = (e_i^{\mathrm{T}} \quad y_i^{\mathrm{T}} \quad z_{i(1:k-1)}^{\mathrm{T}})^{\mathrm{T}}$, with e_i an n-dimensional unit vector whose i-th element is 1, corresponding to the sum-to-1 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. Now, we can encode the nonlinear constraints (3.13) as integer linear constraints, which allows us to use a general MIP solver to solve the subproblems:

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

where M is the least common multiple of the denominators of the coefficients in r_k.

Depending on the particular subproblem solver used, 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. However, note 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.

3.5 Optimality and Termination

We now present conditions for optimality and termination of price-and-cut.

Theorem 3.1 (Optimality)**.** *For mixed integer programs of the form (3.1)-(3.3), the solution returned by price-and-cut using optimal subproblem solvers is an optimal solution to the master program.*

Proof. At termination, the solution is optimal to the full master program LP relaxation, and is integral, which implies that it is also an optimal solution to the master integer program.

Theorem 3.2 (Termination for IP)**.** *For integer programs of the form (3.1)-(3.3) with bounded variables, price-and-cut under the schedule with $m = 1$ and $k = \infty$ will terminate within a finite number of iterations of both price-and-cut and D-W if the restricted master programs are nondegenerate or an anticycling rule is used.*

Proof sketch. As mentioned in Section 3.3, price-and-cut with the said schedule is equivalent to the Dantzig-Wolfe decomposition algorithm where the restricted master is solved to its integer optimal solution every iteration. The integer optimal solution is guaranteed to be found in finite time due to the finite-time guarantee for Gomory cuts. Also, there can only be a finite number of iterations inside D-W, since Gomory cuts do not affect the number of integer solutions, and thus only finitely many columns can be added. Anticycling prevents the subproblem solvers from returning the same column infinitely often.

Theorem 3.3 (Termination for MIP)**.** *For mixed integer programs of the form (3.1)-(3.3) with bounded variables, price-and-cut using optimal subproblem solvers will terminate within a finite number of iterations of both price-and-cut and D-W, if the employed schedule allows only a finite number of iterations of price-and-cut between applications of Gomory cuts to a basic optimal solution of the **full** master's LP relaxation, and the restricted master programs are nondegenerate or an anticycling rule is used.*

Proof. The finite-time termination guarantee of Gomory cuts ensures that the number of iterations of price-and-cut spent on cutting basic LP optimal solutions to the full master is finite, and there are only a finite number of iterations between such iterations. Each call to D-W is guaranteed to terminate in a finite number of iterations (with anticycling) if $m = \infty$ in the schedule, or it will terminate in m iterations.

While finite-time guarantees give us little assurance for the actual execution time of price-and-cut (and we would not expect otherwise, since general integer programming is NP-hard), as we will see, our experiments suggest that only a small number of iterations of price-and-cut may be required in practice.

4 ILLUSTRATING DERIVATIVES

Before we show experimental results on larger sized problems, we present a simple example to provide intuition for the form and interpretation of derivative resources created by Gomory cuts.

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 that agents cannot occupy the same position simutaneously, and also may not swap positions, i.e., occupy the same edge between two positions simultaneously. It is easy to see the bottleneck at position 2: one agent must wait in position 3 for the other to pass through position 2.

In a typical run, our algorithm creates five cuts be-

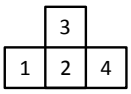


Figure 1: A grid world with four positions

fore termination, which are all tight at termination; we will examine the first two cuts here. Our variables are binary: $x^i_{jt} = 1$ means that 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.

5 EXPERIMENTS

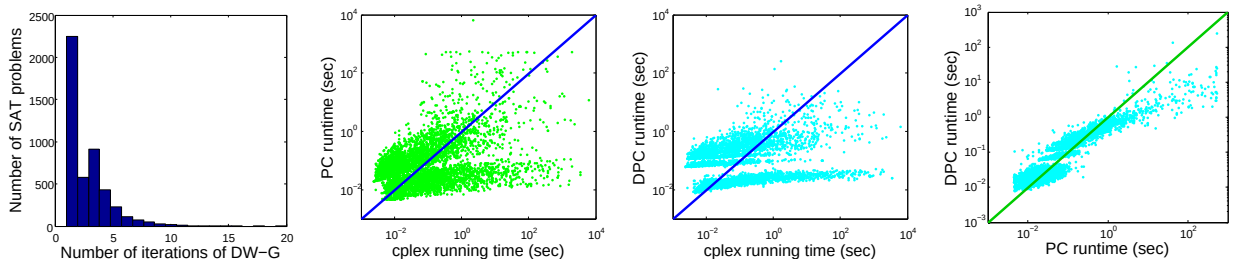
Timing Experiments We demonstrate the effectiveness of price-and-cut planning (PC) and its distribted version (DPC) on randomly-generated factored zero-one integer programs. This domain is both difficult (general-purpose solvers take 10^3–10^4 seconds) and relevant: for example, the method of propositional planning [17] encodes a planning problem as a similar constraint-satisfaction problem, with feasible points corresponding to feasible plans, and an objective corresponding to plan cost. Our goal is twofold: (1) to show that PC is an efficient solver for factored integer programs, and (2) to investigate the effects of communication overhead required by PC in distributed settings.

To generate a random instance, we pick a number of variables and constraints, and for each constraint we select a sparse set of variables and integer coefficients at random. (Hence the constraint matrices A_i and the bounds b in each instance start out integral; integrality of the original A_i and x_i imply integral usages for the original resources, which allows us to use the integer program subproblem formulation from Section 3.4.2.) To make sure the instance is factored, 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 so that we can set the ratio of constraints to variables near the well-known empirical hardness threshold of 4.26 [11]; 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 [15], a widely-used commercial optimization package, for the integer program subproblems, and the CPLEX simplex LP solver for the restricted master LP relaxation.[3] We use the schedule from Theorem 3.2 to guarantee finite termination.

We compare the runtimes of the centralized and distributed versions of PC to those of CPLEX's MIP solver, using 4713 randomly generated problems. We 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 set to between 4.0 and 4.5 to keep the problems difficult. In the centralized version of PC, subproblems were solved sequentially on one CPU, alternating with the master program, incurring no communication cost. The distributed runs were performed on two 8-core machines,

[3]We also tested the MiniSat+ solver [9], a specialized zero-one IP solver based on a SAT solver, MiniSat. Depending on problem characteristics, it was sometimes faster than CPLEX and sometimes slower. We report results using CPLEX since it is more general.



(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

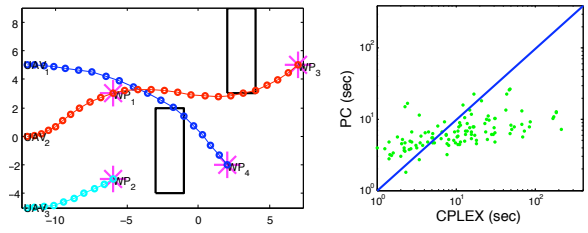
where the subproblem solvers communicated with the master over sockets. One process was dedicated to solving the restricted master LP and making cuts.

Fig. 2(a) shows the distribution of the number of iterations of PC to reach optimality. Most cases required only a few iterations, and only 34 cases out of 4713 required more than 10. In Fig. 2(b), each point represents a problem; on the x axis is the runtime of CPLEX, and on the y axis is that of centralized 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.

Fig. 2(c) is an analogous plot for the distributed version of PC (DPC), and exhibits similar trends, if not more pronounced. Finally, Fig. 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.

The bottom horizontal "tier" of points in Figs. 2(b) and 2(c) contains almost exclusively infeasible instances; as expected, PC is very efficient at detecting infeasibilities when a subproblem is infeasible. In Fig. 2(c), additional tiers represent the effects of parallelism: different tiers roughly correspond to different percentages of shared constraints in the problem, with smaller percentages corresponding to lower tiers, where more execution time is spent in subproblems (which can be parallelized) instead of the master (which is not parallelized in our implementation).

Multi-UAV Planning We also performed experiments on UAV task allocation and path planning problems, using the MILP formulation presented in [21]. We again compared the runtimes of PC and CPLEX, in the setting shown in Fig. 3(a), which was studied in [21]. In this particular configuration, CPLEX took 166s, which is comparable to numbers reported in [21], almost a decade ago, whereas PC found the answer in 31s. We studied 150 instances based on this map, with randomly generated targets. We used as the objective the sum of the agents' finish times; there is little computational advantage to using our decomposition under the maximum of finish times objective used in [21], as it creates a star graph between the agents. We observed trends similar to the 3SAT timing experiments: PC outperforms CPLEX in 96.9% of the instances where CPLEX took more than 10s. The mean runtimes were 7.24s and 30.7s for PC and CPLEX respectively, and max times 26.85s and 624.99s.



(a) (b)

Figure 3: (a) an instance of the UAV planning problem (b) runtime comparisons, PC vs. CPLEX

6 CONCLUSION

We presented a principled distributed market-based algorithm for combinatorial optimization and multi-agent planning, and provided optimality conditions. Our experiments show promise for general applicability of price-and-cut, in both centralized and distributed settings. Our future work includes investigation of quality guarantees under approximate subproblem solvers, as well as application of the method to multi-agent planning in continuous spaces through the use of specialized subproblem solvers.

Acknowledgments
Sue Ann Hong and Geoffrey J. Gordon were supported by ONR MURI grant number N00014-09-1-1052.

References

[1] 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.

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

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

[4] 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.

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

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

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

[8] M. Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stenz. Market-based multirobot coordination: A survey and analysis. Technical Report CMU-RI-TR-05-13, Robotics Institute, Carnegie Mellon University, April 2005.

[9] Niklas Een and Niklas Sorensson. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation 2*, 2006.

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

[11] Carla Gomes, Heny Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, chapter 2. Elsevier B.V., 2007.

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

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

[14] Johan Holmgren, Jan Persson, and Paul Davidsson. Agent-based Dantzig-Wolfe decomposition. In *Agent and Multi-Agent Systems: Technologies and Applications*, volume 5559 of *Lecture Notes in Computer Science*, pages 754–763. Springer Berlin / Heidelberg, 2009.

[15] ILOG. IBM ILOG CPLEX optimizer, 2010. `http://www.ilog.com/products/cplex/`.

[16] Sertac Karaman and Gokhan Inalhan. Large-scale task/target assignment for UAV fleets using a distributed branch and price optimization scheme. In *Proc. 17th World Congress, The International Federation of Automatic Control*, 2008.

[17] 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.

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

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

[20] 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.

[21] 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.

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

[23] 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.

[24] François Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 331–358. Springer, 2005.

[25] 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.

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

[27] 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.

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