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

Sue Ann Hong

CMU-CS-13-103

February 2013

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Geoffrey J. Gordon, Chair
M. Bernardine Dias
Carlos Guestrin
Jeff Schneider
Amy Greenwald (Brown University)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was funded in part by The Defense Advanced Research Projects Agency under grant number HR00110710026; the US Army Research Office under grant number W911NF0810301; the Office of Naval Research under grant number N000140911052; SRI International under subcontract 03-000211; and a fellowship from the Intel Science and Technology Center (Embedded Computing).

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
FEB 2013		**00-00-2013 to 00-00-2013**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Distributed Market-Based Algorithms for Multi-Agent Planning with Shared Resources	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Carnegie Mellon University,School of Computer Science,Pittsburgh,PA,15213**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We propose a new family of market-based distributed planning algorithms for collaborative multi-agent systems with complex shared constraints. Such constraints tightly couple the agents together, and appear in problems ranging from task or resource allocation to collision avoidance. While it is not immediately obvious, a wide variety of constraints can in fact be reduced to generalized resource allocation problems; the translation is straightforward for task or resource allocation problems, and for general problems any shared constraint between agents can be considered a resource. For example, satisfying collision-avoidance constraints is equivalent to assigning to a limited number of agents the right to claim a particular space at a particular time. Market-based algorithms have become popular in collaborative multi-agent planning particularly for task allocation, due to their intuitive and simple distributed paradigm as well as their success in domains such as robotics and software agent systems. However, they suffer from several main drawbacks: 1. it is somewhat of an art to create a reasonable pricing in each domain, requiring a human designer and parameter tuning, 2. they rarely guarantee optimality 3. they do not often have a natural way to incorporate uncertainty in planning, and 4. most existing algorithms require a central trusted auctioneer. This thesis presents a solution to address the drawbacks by providing mechanisms that automatically and optimally price the resources as well as simple, optimal bidding strategies for the agents. We formulate multi-agent planning as factored mathematical programs that are optimized in a distributed fashion. Like other market-based planning algorithms, our methods allow for the optimization of each agent?s local planning problem at the agent with limited communication; however, they are able to deal with more complex constraints than those usually used in market-based planning settings. We consider three different settings and give algorithms for each that compute resource prices automatically, and do so to guarantee (near-)optimality. First, we formalize factored mixed integer linear programs (MILPs), and give a novel distributed optimization algorithm by combining Dantzig-Wolfe decomposition a distributed method for optimizing linear programs, with a cutting-plane algorithm. Second we relax the framework with Lagrangian relaxation, for more efficient, convex optimization. Lagrangian relaxation yields an approximation to the original MILP, but we give a simple yet effective randomized rounding algorithm whose chance of failure can be bounded, and the result is a near-optimal approximation algorithm for problems with a very large number of agents contending for the resources. Finally, we study planning under uncertainty in

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**80**	

Keywords: Distributed multi-agent planning, planning under uncertainty, distributed optimization, Lagrangian relaxation, stochastic programming, traffic management, supply-chain management

Abstract

We propose a new family of market-based distributed planning algorithms for collaborative multi-agent systems with complex shared constraints. Such constraints tightly couple the agents together, and appear in problems ranging from task or resource allocation to collision avoidance. While it is not immediately obvious, a wide variety of constraints can in fact be reduced to generalized resource allocation problems; the translation is straightforward for task or resource allocation problems, and for general problems any shared constraint between agents can be considered a resource. For example, satisfying collision-avoidance constraints is equivalent to assigning to a limited number of agents the right to claim a particular space at a particular time.

Market-based algorithms have become popular in collaborative multi-agent planning, particularly for task allocation, due to their intuitive and simple distributed paradigm as well as their success in domains such as robotics and software agent systems. However, they suffer from several main drawbacks: 1. it is somewhat of an art to create a reasonable pricing in each domain, requiring a human designer and parameter tuning, 2. they rarely guarantee optimality, 3. they do not often have a natural way to incorporate uncertainty in planning, and 4. most existing algorithms require a central trusted auctioneer. This thesis presents a solution to address the drawbacks by providing mechanisms that *automatically* and *optimally* price the resources as well as simple, optimal bidding strategies for the agents.

We formulate multi-agent planning as factored mathematical programs that are optimized in a distributed fashion. Like other market-based planning algorithms, our methods allow for the optimization of each agent's local planning problem *at* the agent with limited communication; however, they are able to deal with more complex constraints than those usually used in market-based planning settings. We consider three different settings and give algorithms for each that compute resource prices automatically, and do so to guarantee (near-)optimality. First, we formalize factored mixed integer linear programs (MILPs), and give a novel distributed optimization algorithm by combining Dantzig-Wolfe decomposition, a distributed method for optimizing linear programs, with a cutting-plane algorithm. Second, we relax the framework with Lagrangian relaxation, for more efficient, convex optimization. Lagrangian relaxation yields an approximation to the original MILP, but we give a simple yet effective randomized rounding algorithm whose chance of failure can be bounded, and the result is a near-optimal approximation algorithm for problems with a very large number of agents contending for the resources. Finally, we study planning under uncertainty in the Lagrangian relaxation framework via stochastic programming, and give efficient algorithms for representing and optimizing uncertainty in factored Markov decision processes.

We evaluate our algorithms on domains ranging from task allocation and path planning to supply chain management, and demonstrate the power of wielding complex shared constraints in distributed planning, which we hope will continue to be studied in a wide range of domains.

Acknowledgments

When it comes to this thesis, I must thank first and foremost my advisor, Geoff Gordon, for picking me up in the middle of my rather long PhD career and guiding me through the finish line. I have learned not only a lot of knowledge about everything from optimization and game theory to spectral learning from him, but also how to really conduct research and organize ideas effectively and precisely in communication.

I'd also like to thank my thesis committee, Amy Greenwald, Bernardine Dias, Jeff Schneider, and Carlos Guestrin. A special thanks goes to Jeff Schneider for making my actual thesis defense possible, and to Carlos for always trying to come up with something funny to say. Sometimes it's funny, other times it's at least interesting, and all the time it's fun.

In the distant past, I worked with Tom Mitchell, a man of many visions, on reading the web. Even though I didn't end up writing my thesis on Read The Web, I had many great stimulating conversations with Tom and Andy on all topics machine learning, for which I will always be grateful.

I would also like to thank my friends and colleagues at CMU who made the gloomy Pittsburgh days much more bearable. Khalid, Lucia, Michelle, Wooyoung, Mary, and Gaurav, thanks for commiserating with me. I would like to thank Yisong for being one of my few collaborators, and for always providing entertainment along with Jing. You guys are our Canadian family. Yucheng, thanks for always entertaining Joey and just in general being jolly. I miss everyone and the SELECT Lab. And my long-lost CSD friends who have moved on to real-world lives earlier than I am, why are you all so far away? Also, Miro, I'm sorry I never finished the tech report. I'll try to work on it later. And Alice, thanks for always making me feel like there's a place for me in Pittsburgh.

I owe everything on my path to and through my PhD career to my family, especially my mom who, besides being a amazing mom, spent two crucial weeks supervising my thesis writing sessions in person. I have also gained a loving and caring family midway through grad school; my new parents, Jean Barchet and Edgar Gonzalez, your encouraging words always makes me feel better and loved.

And Joey, what would I do without you? Thank you for being there for me throughout my difficult journey. I feel truly lucky to have you in my life. Even though I had to force you to read parts of my thesis and you just said "it's all fine".

Contents

List of Figures

List of Tables

Chapter 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 or tasks, using its planner both to decide which resources are worth acquiring and to decide how to use resources if acquired. Then an auctioneer or a distributed mechanism is used to assign said resources to the agents. 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 [21, 27, 95].

However, market-based planners 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 including a strategy for pricing commodities, tuning both to balance planning effort against suboptimality. Second, most market-based planners cannot offer any guarantees on the final overall plan quality. Third, most algorithms degrade quickly in the presence of additional constraints that couple the agents' solutions, such as task ordering or collision avoidance constraints, adding to the design effort. (See [22] for a further discussion.) Finally, planning under uncertainty presents a challenge to most market-based planning frameworks, as they are often not built with uncertainty in mind, nor built upon mechanisms that easily allow incoporation of uncertainty.

This thesis addresses these problems by designing a set of principled algorithms which *automatically* and *optimally* price resources in any domain. We term such algorithms Automatically-pricing Market-based Distributed Multi-agent Algorithms (AMDMA). We frame multi-agent planning as mathematical programs, yielding combinatorial and convex optimization problems. Our al-

gorithms naturally decompose a program into *subproblems*, which correspond to agents' local planning problems and can be solved using fast domain-specific single-agent planning methods, and a *master program* that manages resource prices and communicates with the subproblems for optimal resource allocation.

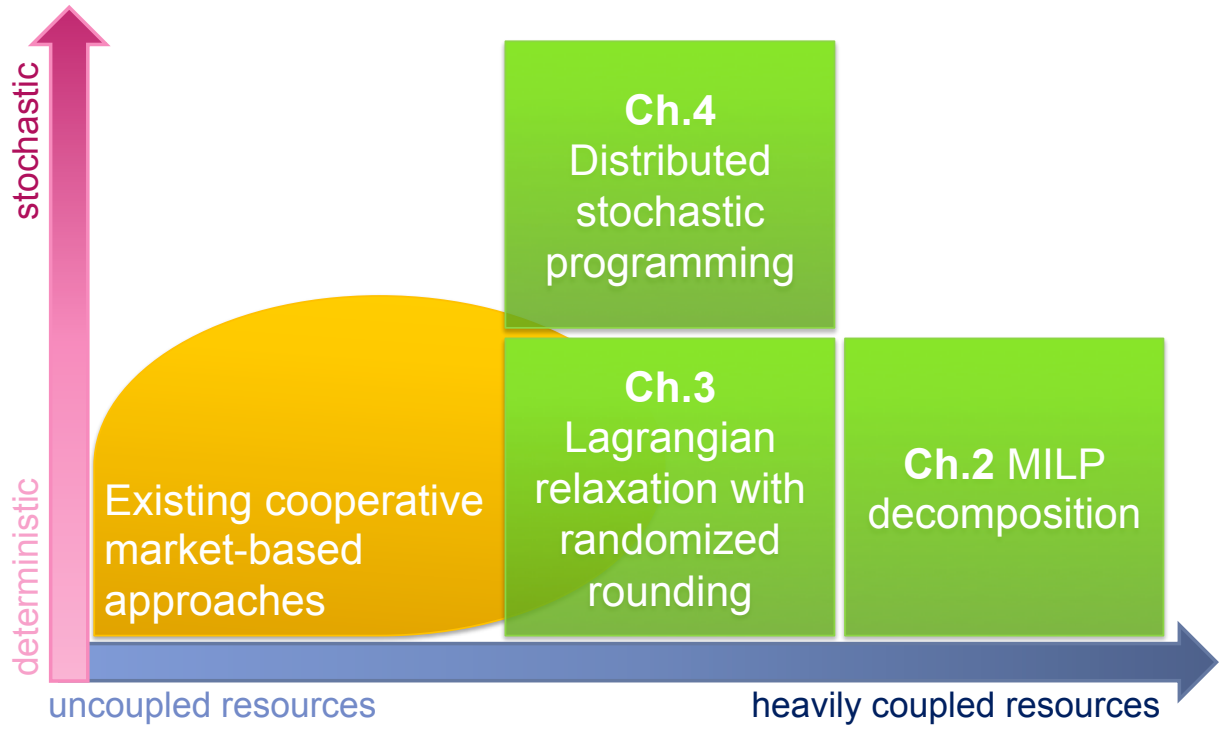


Figure 1.1: A broad view of topics covered in this thesis. Here we characterize the space of cooperative market-based planning problems using the two dimensions we focus on: the level of coupling between resources, and planning under uncertainty. A more detailed comparison of our work to research in various fields ranging from mechanism design to operations research can be found in Chapter 5.

We present three variants of AMDMA and give distributed optimization algorithms for each setting. Figure 1.1 gives a broad view of the settings this thesis covers. In more detail:

- Chapter 2 introduces the formalization of our problem setting using mixed integer linear programs (MILP). We give a novel distributed MILP optimization algorithm based on Dantzig-Wolfe decomposition and a cutting plane algorithm, with convergence and optimality guarantees. Using distributed SAT problems as an abstraction for planning problems, as well as an unmanned aerial vehicle task allocation and path planning task, we show that both the sequential and the distributed executions of the algorithm result in improvement over CPLEX, the industry-standard MILP solver.
- In Chapter 3, we relax the MILP formulation to obtain Lagrangian relaxations which can be optimized much more efficiently using convex optimization methods. We describe how to obtain feasible solutions using a randomized rounding scheme, and give conditions under

which we can guarantee near-optimal solutions. We present two gradient-based convex optimization algorithms for solving the Lagrangian relaxation, subgradient descent and an accelerated gradient algorithm. Finally, we apply the Lagrangian relaxation framework to factored MDPs in the traffic management domain. We give an augmentation method for factored MDPs that allows the use of accelerated optimization methods while maintaining fast subproblem computation.

- Chapter 4 examines planning under uncertainty in our multi-agent planning framework. In particular, we extend the Lagrangian relaxation method introduced in Chapter 3 via stochastic programming, and give an efficient construction of factored MDPs under uncertainty based on sampling. We look at the supply chain management domain as an example domain of heavy dependencies between agents, and demonstrate how our stochastic planning algorithm can ameliorate the bullwhip effect, a well-known economic phenomenon of inefficiency due to suboptimal handling of uncertainty.

The rest of the thesis provides relevant related work for each problem setting and concludes with further discussion and future work.

Chapter 2

Distributed Market-Based Multi-Agent Planning

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.

Throughout this thesis, we will address these problems under different settings using algorithms that *automatically* and *optimally* prices resources in any domain. In this chapter, we formalize our problem setting, and lay out the foundations for our solution framework in the form of a general distributed solver that prices resources as well as designs new resources, correcting pricing imbalances by adding "derivative securities" based on the original resources to the market.

We formulate planning problems as mixed integer linear programs (MILPs); MILPs are a popular representation for planning problems not only in artificial intelligence [85], but also in operations research [66], cooperative control [2, 69], and game theory [72]. 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, 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. Finally, MILPs are commonly used to represent problems of planning under uncertainty (e.g., [66, 72]). 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. In Chapter 4, we extend our framework to efficiently incorporate handling of uncertainty.

Our algorithm is based on Dantzig-Wolfe (D-W) decomposition [11, 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 individual

agents. 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 [30], a general cutting-plane algorithm, which allows us to retain optimality and finite-time convergence guarantees of the D-W decomposition algorithm for LPs. 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. 2.4 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 distributed computing issues like CPU failure or communication complexity in this thesis, 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's 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 to a server or to each other. Hence our algorithm is well-suited for situations where it is difficult to communicate large sets of constraints and objectives from individual problems to a server, or where centralized computation is not desirable for robustness.

In this chapter, 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 and path planning problems. The experiments demonstrate that (1) when applicable, market-based planning can lead to large efficiency gains, outperforming an industrial-strength MILP solver (CPLEX [39]) 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.

2.1 Problem Formulation

We formulate multi-agent planning problems as MILPs in standard form, factored over n agents:

$$\min \sum_{i=1:n} c_i(x_i) \tag{2.1.1}$$

$$\text{s.t.} \sum_{i=1:n} D_i x_i = b \tag{2.1.2}$$

$$x_i \in C_i, \qquad i = 1, \ldots, n \tag{2.1.3}$$

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 $c_i(x_i)$, not necessarily convex, its cost function. Each x_i is a mixed integer vector (its elements may be integers or reals); for convenience we assume that C_i is bounded. (2.1.2) 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 [11, 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.

2.2 Algorithm

2.2.1 Dantzig-Wolfe Decomposition

Consider a problem of the form (2.1.1-2.1.3) where each C_i is convex and polyhedral (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 $x_i^1 \dots x_i^{n_i} \in C_i$ 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:

$$\min \sum_{i=1:n} \sum_{j=1:n_i} c_i(x_i^j) w_i^j \tag{2.2.1}$$

$$\text{s.t.} \sum_{i=1:n} \sum_{j=1:n_i} w_i^j D_i x_i^j = b, \qquad w_i^j \in [0,1] \tag{2.2.2}$$

$$\sum_{j=1:n_i} w_i^j = 1, \qquad i = 1, \dots, n. \tag{2.2.3}$$

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 (2.2.1–2.2.3) by solving the restricted master, which is identical to (2.2.1–2.2.3) except that some w_i^j are forced to be zero. As in the simplex method for LPs (see, e.g., [11, 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

$$c_i(x_i^j) - q^{\mathrm{T}} D_i x_i^j - \mu_i, \tag{2.2.4}$$

where q contains the values of dual variables for shared constraints (2.2.2) at the current basic solution, and μ_i is the value of the dual variable for the sum-to-1 constraint (2.2.3) 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:

$$\min\ c_i(x_i) - q^{\mathrm{T}} D_i x_i \tag{2.2.5}$$

$$\text{s.t. } x_i \in C_i. \tag{2.2.6}$$

Note that we have altered the subproblem *only* in its objective and only by a linear term, 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:

$$\min\ \hat{c}^{\mathrm{T}} w \ \text{ s.t. } \hat{D} w = \hat{b}, \tag{2.2.7}$$

where we have renumbered the variables to be $w_1 \dots w_k$, and collected together the constraints $\hat{D}w = \hat{b}$ and objective $\hat{c}^{\mathrm{T}}w$. A subproblem solution x_i^j corresponds to a column $A_i x_i^j$ in $\hat{D}$ and an entry $c_i(x_i^j)$ in $\hat{c}$. For convenience, we have incorporated the sum-to-1 constraints (2.2.3) into $\hat{D}$, 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 $\hat{D}_{ij}$ as the usage level of resource i by plan j, as can be seen from the subproblem objective (2.2.5). If a constraint is saturated by plans currently in the restricted master, the corresponding resource will have a non-zero 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 $c_i(x_i) - q^{\mathrm{T}} D_i x_i \leq \mu_i$, 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 [19, 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.

2.2.2 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 (2.2.2) and the restricted master (2.2.7). (If C_i is convex, these constraints are unnecessary. However, for MILPs, 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 D_i has integer entries, so that all resource usages are integral[2]

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. 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, and here we use the Gomory method for generating cuts, although other cutting plane recipes can be applied as well. 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 decomposition (Alg. 1) to the MILP 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. 2.3.

[2]If not, we can always divide the entries of $D_1, \ldots, D_n$ by a common denominator, given that each D_i has rational number entries.

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 (2.2.5); 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.

2.2.3 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 (2.2.7), 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:

$$\sum_j \lfloor (B^{-1})_k \hat{D}_{*j} \rfloor w_j \leq \lfloor (B^{-1})_k \hat{b} \rfloor, \tag{2.2.8}$$

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

The cut is a *valid inequality*: any integral point that satisfies all original constraints in (2.2.2) satisfies the new inequality. Also, the current LP optimal solution violates the new constraint, ensuring progress (for details, see [11, 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] 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.

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 $\hat{D}$. The coefficient for a new column j is

$$\hat{D}_{ij} = \lfloor (B^{-1})_k \hat{D}_{(1:i-1)j} \rfloor, \tag{2.2.9}$$

where $\hat{D}_{(1:i-1)j}$ denotes the first $i-1$ rows of the j-th column of $\hat{D}$. 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 $\hat{D}$ 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 $\hat{D}$, since each cut may depend on previous ones. This fact causes two problems: we can't parallelize coefficient generation, and the coefficients $\hat{D}_{ij}$ may grow and cause numerical problems as we layer more cuts on top of each other. The first problem is not a big one since coefficient computation not a major part of the overall computational cost. The second problem can be difficult to resolve, but we leave it as a subject for future work.

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 $\hat{D}_{kj}$ of the master program's constraint matrix. Accordingly, as we saw in the subproblem objective (2.2.5), original resource usage y_i by a plan x_i is simply $y_i = D_i x_i$. 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 D corresponding to the derivative resource k, and let r_k refer to the cut recipe as defined in (2.2.9). Then, for column j, rewriting (2.2.9) in terms of y_i and z_i gives:

$$z_{ik} = \lfloor r_k u \rfloor, \tag{2.2.10}$$

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 sum-to-1 constraints in the master program. Incorporating the new variables, the subproblem objective now becomes

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

and we add the expressions above for y_i and z_{ik} as additional constraints. Now, we can encode the non-linear constraints (2.2.10) as integer linear constraints, which allows us to use a general MILP 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.

2.3 Optimality and Termination

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

Theorem 2.3.1 (Optimality)**.** *For mixed integer programs of the form (2.1.1)-(2.1.3), 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 2.3.2 (Termination for IP)**.** *For integer programs of the form (2.1.1)-(2.1.3) 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 2.2.2, 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 2.3.3 (Termination for MILP)**.** *For mixed integer programs of the form (2.1.1)-(2.1.3) 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.

2.4 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 2.1. 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.

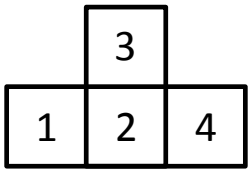


Figure 2.1: A grid world with four positions

In a typical run, our algorithm creates five cuts before 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.

2.5 Experiments

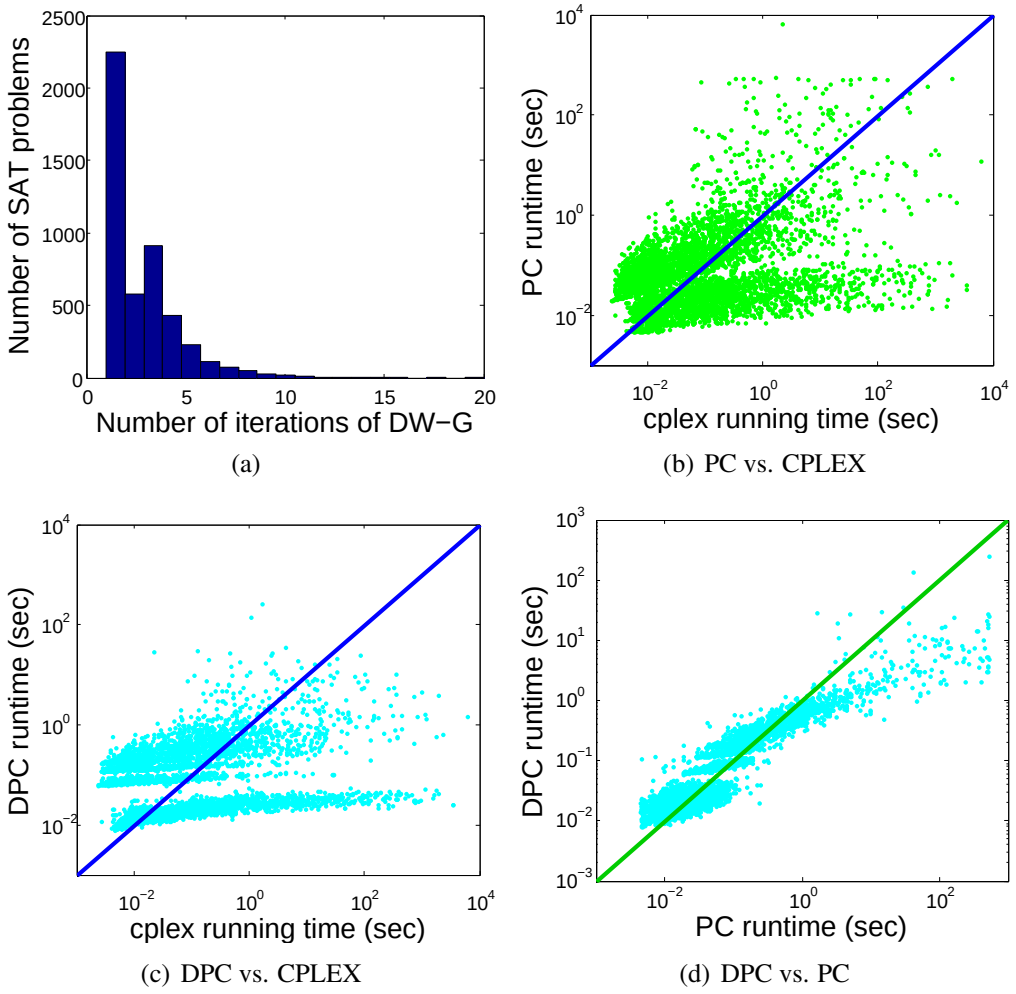
2.5.1 Timing Experiments

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 [44] 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 two-fold: (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 2.2.3.) 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 [29]; 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 MILP solver [39], a widely-used commercial optimization package, for the integer program subproblems, and the CPLEX simplex

LP solver for the restricted master LP relaxation.[4] We use the schedule from Theorem 2.3.2 to guarantee finite termination.



(a)

(b) PC vs. CPLEX

(c) DPC vs. CPLEX

(d) DPC vs. PC

Figure 2.2: (a) number of PC iterations performed, (b)–(d) runtime comparisons

We compare the runtimes of the centralized and distributed versions of PC to those of CPLEX's MILP 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, where the subproblem solvers communicated with the master over sockets. One process was dedicated to solving the restricted master LP and making cuts.

[4]We also tested the MiniSat+ solver [23], 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.

Fig. 2.2(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. 2.2(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. 2.2(c) is an analogous plot for the distributed version of PC (DPC), and exhibits similar trends, if not more pronounced. Finally, Fig. 2.2(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. 2.2(b) and 2.2(c) contains almost exclusively infeasible instances; as expected, PC is very efficient at detecting infeasibilities when a subproblem is infeasible. In Fig. 2.2(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).

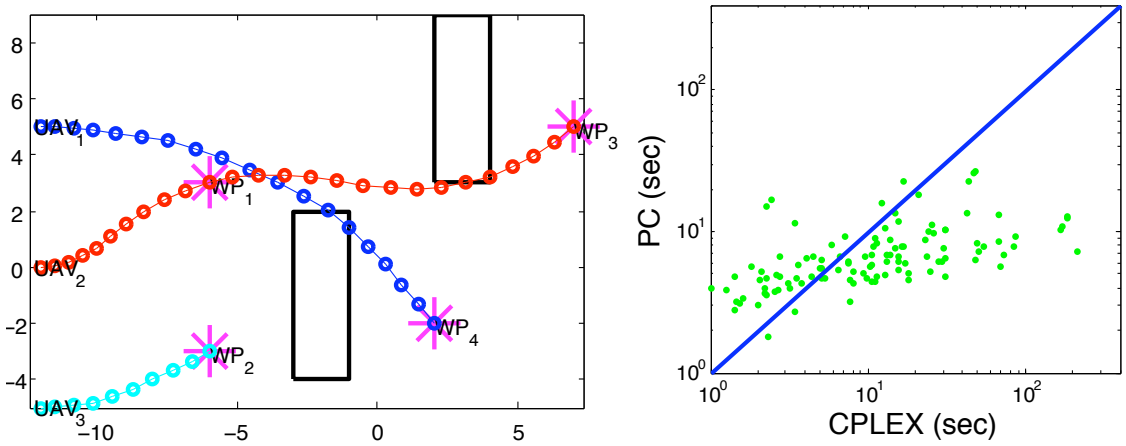


Figure 2.3: (a) an instance of the UAV planning problem (b) runtime comparisons, PC vs. CPLEX

2.5.2 UAV Task Allocation and Path Planning

We also performed experiments on UAV task allocation and path planning problems, using the MILP formulation presented in [69]. We again compared the runtimes of PC and CPLEX, in the setting shown in Fig. 2.3(a), which was studied in [69]. In this particular configuration, CPLEX

took $166s$, which is comparable to numbers reported in [69], 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 [69], 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 $624.99s$.

2.6 Conclusion

In this chapter, we laid out the foundations of our framework, upon which we will build in the next chapters. We presented a principled distributed market-based algorithm for combinatorial optimization and multi-agent planning that automatically and optimally prices shared resources. We provided optimality and convergence conditions for a general class of MILPs that includes our multi-agent formulation. Our experiments show general applicability of price-and-cut, in both centralized and distributed settings. In the following, we will examine how to obtain a more efficient, albeit less generally applicable, algorithm by relaxing the MILP framework.

Chapter 3

Scaling Up with Lagrangian Relaxation

Mixed integer linear programs (MILPs) are a versatile language for formulating and solving various multi-agent problems with shared constraints, and in Chapter 2 we explored a distributed method to solve them via decomposition. Still yet, MILPs are often quite time-consuming to solve, and thus are not ideal for domains that require real-time or near-real-time planning. One such domain is that of automated driving, which one can frame as a distributed planning problem, where each car plans its own path while communicating with other cars or a master program to satisfy global constraints that prevent collisions. In this chapter, we will study a scenario involving a heavily congested crossroad, where automated vehicles plan and resolve conflicts by communicating via a master program, with the goal of reaching their destinations safely.

A popular approach to solving a difficult MILP is to employ linear program (LP) relaxation, where integral constraints on the variables are relaxed such that the variables become continuous. LP relaxation is a special case of *Lagrangian relaxation* [25], which relaxes a subset of the constraints of a MILP to obtain what is hopefully an easier problem to solve. However, no guarantees exist in general for the quality of a Lagrangian relaxation solution. In this chapter, we describe a Lagrangian relaxation of the shared constraints that, in conjunction with randomized rounding and "cushioning" the shared constraints, can be shown to yield a near-optimal solution to the original MILP [31]. In particular, the guarantees hold for problems with a large number of agents, where many agents contend for each resource. This form of Lagrangian relaxation also allows us to keep the subproblems integral, and use the same subproblem solvers as we would for solving the MILP via price-and-cut (Chapter 2). Our experiments demonstrate that the number of violated constraints can be far reduced in the Lagrangian relaxation solutions compared to non-coordinated solutions, even in planning problems that are small enough that the theoretical guarantees are not very tight.

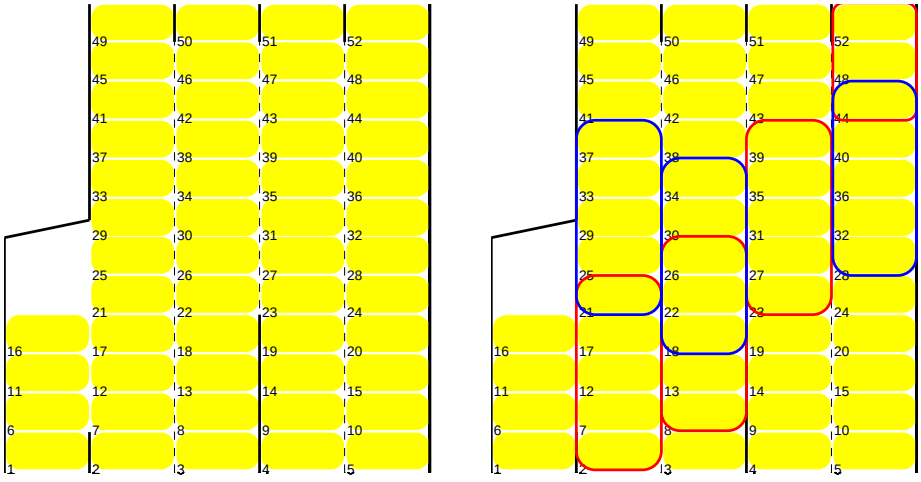
Analogously to employing Dantzig-Wolfe decomposition (Chapter 2) to a LP relaxation to obtain a distributed iterative algorithm, we can optimize the Lagrangian relaxation by using simple gradient-based optimization methods on the dual. As in Dantzig-Wolfe decomposition, dual variable values being optimized act as prices for the resources so that the subproblems can produce feasible plans accordingly. Conversely, computing the subgradient of the dual function amounts

to planning in each agent's sub-problem independently, using the current resource prices. In this chapter, we give two gradient-based optimization methods for Lagrangian relaxation with different convergence rates, sugradient descent and an accelerated gradient method, FISTA.

In particular, we discuss how to augment factored MDPs for optimization with FISTA, which requires a partly "smooth" objective function with a Lipschitz continuous gradient. We consider two penalization functions for factored MDPs: the quadratic loss penalty and the maximum causal entropy penalty [98]. In particular, the maximum causal entropy penalty yields an efficient subproblem solver, softmax value iteration, allowing us to maintain fast subproblem planning for individual agents and thus an efficient computation of the gradient. Unfortunately, augmentation can lead to errors in solution with respect to the original objective. However, we show that 1. we can bound the errors introduced by augmentation, and 2. despite such errors, empirically we can achieve convergence to an optimal solution, at a dramatically faster rate than that of exact optimization with subgradient method.

We begin by introducing our motivating domain of automated path planning in traffic.

3.1 Focus Domain: Traffic Management



(a) States (b) Resources

Figure 3.1: An instance of the planning domain of X-shaped bridges, formulated as a factored MDP for the deliberative planner. (a) shows the states, each a yellow shaded numbered rectangle. Bold lines indicate boundaries that cannot be crossed by vehicles. In (b), rounded rectangles are the resources, which are overlapping patches of the road (different colors were used only to distinguish between overlapping rectangles).

We use X-shaped bridges and interchanges as an example of a difficult traffic coordination problem, as vehicles are forced to change lanes and contend for shared road space. Such interchanges are a common culprit in heavy traffic, since agents must often cross many lanes in a short section of

a road. Many examples of such interchanges can be found in Pittsburgh, the most infamous of which is the Fort Pitt Bridge.

To study this domain, we built a simulator which computes the positions and the movements of the vehicles at a high frequency to simulate continuous-time dynamics and interactions. The final paths of the agents are based on two planners: a *deliberative planner*, which is used before execution to map out a coordinated plan and is the focus of this chapter, and a *reactive planner*, which is run at execution time on each agent to ensure collision avoidance while following the more coarse-grained plans output by the deliberative planner. The reactive planner also helps agents replan under unexpected events such as lane closures or accidents in this domain (for some of which we can plan during the deliberative stage using an algorithm such as that given in Chapter 4).

For the deliberative planner, we formulate each agent's path planning problem as a deterministic Markov decision process (MDP). The global problem then becomes a factored MDP where agents interact with one another through constraints on shared resources. In Figure 3.1(a) we depict an example problem instance, where states (physical locations of vehicles) are represented by yellow rectangles labeled with state IDs. Each agent starts at one of the first five states (1–5, i.e., entering the bridge) and must reach its goal state (randomly picked as one of the last states, here 49–52, i.e., exiting the bridge). In each state, agents are allowed to move in four directions, corresponding to staying in the current state or moving in each forward direction (moving one state to the left-forward, straight, or right-forward), but cannot cross the solid lines. The constraints in this domain encode collision avoidance by defining resources as patches of the road as shown in Figure 3.1(b) and only allowing a limited number of agents to occupy a patch at a time. In the following sections , we describe the Lagrangian relaxation algorithm for solving general MILPs, as well as its specialization to deterministic factored MDPs.

The reactive planners (and thus the simulations) are run at a high frequency with much smaller time steps than that used by the MDP, and use a potential-field method for avoiding collisions: we apply a repulsive force to the control targets of any pairs of cars which approach too closely. In particular, the reactive planners are proportional-derivative (PD) controllers based on the differentially-flat representation of a differential-drive robot (a "unicycle"). We note that while this representation is simpler than an Ackermann-steered car, it is adequate for our purposes, since the cars are always moving forward with lower-bounded velocity.

3.2 Lagrangian Relaxation

First we define our mathematical program, which is a specialization of our mixed integer programming (MILP) framework. We focus on linear objective functions for simplicity of discussion; however, the objective may in fact be composed of arbitrary convex functions with respect to each agent's local variables. We allow arbitrary subproblem constraints, only requiring the subproblem *solutions* to be bounded (so that the subgradient is bounded). We assume that the shared

constraints are linear[1].

Following a similar notation to that for MILP in Equations (2.1.1)–(2.1.3), we can write our program as:

$$\max \sum_{i=1:n} c_i^{\mathrm{T}} x_i \tag{3.2.1}$$

$$\text{s.t.} \sum_{i=1:n} D_i x_i \leq b \tag{3.2.2}$$

$$x_i \in C_i, \qquad i = 1, \ldots, n, \tag{3.2.3}$$

where (3.2.2) corresponds to shared constraints, and (3.2.3) to an individual agent's constraints. In comparison to (2.1.2), we use inequality constraints for convenience, though without loss of generality.

We now introduce an approximation that will enable us to employ more computationally efficient forms of the shared constraints.

3.2.1 Approximating Shared Constraints

The size of the domain (i.e., the bounds on the variables) in an optimization problem inherently affects the runtime of its solver. For example, consider a convex function f with variables $x \in X \subseteq \mathbb{R}^n$, and define the radius of the domain as $R = \sup_{x \in X} \|x - x^*\|$ with respect to a optimal solution x^*. In the worst case, a first-order optimization method such as subgradient method incurs error $\epsilon = f(x^{(k)}) - f(x^*)$ at $\epsilon = o(R^2/k^2)$ at any iteration $k < n$ (Theorem 3.2.1, [58]); intuitively, the algorithm must reach the optimal point from the starting point, which if we are unlucky, could be as far as traversing the entire space.

As we will see in the following section, our optimization variables are the *Lagrange multipliers*, and using hard constraints results an unbounded domain, which can break the standard convergence proofs that depend on the radius R above for finite-time convergence, for algorithms like subgradient method. Hence, here we limit the domain of the Lagrange multipliers by approximating the hard constraints with *piece-wise linear functions* of the form

$$f(y) = \max\{\alpha y + \beta \mid (\alpha, \beta) \in F\},$$

where F is a finite set of slope-intercept pairs (α, β) (following Gordon et al. [31]). Note that the approximation is in fact quite flexible: with a large enough slope a piece-wise linear function can closely approximate hard constraints[2], and with a large enough number of segments, it can approximate smooth functions to any approximation accuracy required.

[1] Or convex, since we can approximate any convex constraint with several linear constraints.

[2] In fact, for any given problem there exists a sufficiently large slope for the hinge loss (defined shortly) that there is no difference between hinge and hard constraint.

In more detail, for each resource j, we can replace the shared constraint $\sum_{i=1:n} A_{ij}x_i \leq b_j$ by adding to the objective a function f_j of resource usages, rendering the overall program to be:

$$\max \quad \sum_{i=1:n} c_i^{\mathrm{T}} x_i - \sum_{j=1:m} f_j(\sum_{i=1:n} D_{ij}x_i - b_j) \tag{3.2.4}$$

$$\text{s.t.} \quad x_i \in C_i, i = 1, \ldots, n. \tag{3.2.5}$$

A useful instance of f_j is the *hinge loss*

$$f_j^{\alpha_j}(y) = \max(0, \alpha_j y),$$

which softens the hard constraints but still penalizes over-usage of resources, with α_j controlling the strength of the soft constraint. Hinge loss is convex and continuous, making it suitable for efficient optimization, and can approximate hard constraints with a sufficiently large α_j. It is also flexible; for example, the "hinge point" can be moved to "cushion" the resource usage, to further encourage the resource constraints to be met. For these advantages, hinge loss will be our f_j of choice for the traffic management domain.

3.2.2 Lagrangian Relaxation

We are now ready to define the Lagrangian relaxation for our problem. We obtain the Lagrangian relaxation of (3.2.4)–(3.2.5) by dualizing the shared constraints in (3.2.4). The problem then becomes finding the optimal dual value

$$V^* = \inf_{\lambda} V(\lambda) \tag{3.2.6}$$

$$= \inf_{\lambda} \max_{x} \sum_{i=1:n} c_i^{\mathrm{T}} x_i + \sum_{j=1:m} [f_j^*(\lambda_j) - \lambda_j(\sum_{i=1:n} D_{ij}x_i - b_j)] \text{ s.t. } \forall i, x_i \in C_i, \tag{3.2.7}$$

where λ_j are the dual variables, and $f_j^*(\lambda_j) = \sup_z[\lambda_j z - f_j(z)]$ is the Lagrangian dual of f_j. V^*, the optimal dual value, provides an upper bound on the value of (3.2.4)–(3.2.5).

Rearranging (3.2.7), we get

$$V^* = \inf_{\lambda}[\sum_{j=1:m} [f_j^*(\lambda_j) + \lambda_j b_j] + \max_{x}[\sum_{i=1:n} (c_i^{\mathrm{T}} - \sum_{j=1:m} \lambda_j D_{ij})x_i]] \text{ s.t. } \forall i, C_i, \tag{3.2.8}$$

which reveals two terms: the first sum discourages the dual values λ from growing too large, and the latter optimizes the sum of agents' utilities, with penalty for each one's resource usage *priced by* λ.

3.2.3 Optimization

We can solve Equation (3.2.7) by iteratively updating λ using a first-order method such as subgradient projection. In Section 3.3, we discuss different gradient-based methods and their

applicability to different types of objective functions, as well as how to obtain a feasible solution to the original MILP given a near-optimal solution to the Lagrangian relaxation. Here we discuss how to compute the subgradient (needed for all first-order methods).

It is straight-forward to compute the subgradient of $V(\lambda)$, the dual value at a specific value of λ. First we rewrite $V(\lambda)$ using the definition of f_j^*, the Lagrangian dual of f_j:

$$V(\lambda) = \sum_{j=1:m} [\max_z [\lambda_j z - f_j(z)] + \lambda_j b_j] + \max_x [\sum_{i=1:n} (c_i^{\mathrm{T}} - \sum_{j=1:m} \lambda_j D_{ij}) x_i] \text{ s.t. } \forall i, C_i.$$

Now, let $z_j^* = \arg\max_z [\lambda_j z - f_j(z)]$ and $x_i^* = \arg\max_{x_i \text{ s.t. } C_i} [(c_i^{\mathrm{T}} - \sum_{j=1:m} \lambda_j D_{ij}) x_i]$. Then the subgradient of $V(\lambda)$ with respect to λ_j is

$$\partial[V(\lambda_j)] = z_j^* + b_j - \sum_i D_{ij} x_i^*.$$

In the case of the hinge loss $f_j^{\alpha_j}(z)$, for $0 \leq \lambda_j \leq \alpha_j$, we can set $z_j^* = 0$ to obtain a subgradient

$$\partial[V(\lambda_j)] = b_j - \sum_i D_{ij} x_i^*, \tag{3.2.9}$$

which corresponds exactly to the excess usage of resource j, giving rise once again to the price interpretation of the dual variables.

The form of the subgradient shows that the computation of the subgradient is easily distributed, as the subproblems

$$\max_{x_i} [(c_i^{\mathrm{T}} - \sum_{j=1:m} \lambda_j D_{ij}) x_i] \text{ s.t. } C_i$$

are independent given λ and thus x_i can be found in parallel. Hence Lagrangian relaxation leads to a naturally distributed algorithm where each agent's planning may be performed in parallel and communication is achieved solely through λ and the resource usages.

Note that since f_j is piece-wise linear, the dual function f_j^* imposes constraints on λ_j: $\alpha_j^{\min} \leq \lambda_j \leq \alpha_j^{\max}$, where $\alpha_j^{\min}$ and $\alpha_j^{\max}$ are the smallest and largest slopes of the linear segments respectively. At this point we have enough information to set up the simplest Lagrangian relaxation algorithm: we simply use projected subgradient descent to optimize λ over the box constraints $\alpha^{\min} \leq \lambda \leq \alpha^{\max}$. The projection step is

$$\lambda_j \leftarrow \max(\alpha_j^{\min}, \min(\alpha_j^{\max}, \lambda_j)),$$

and the subgradient is given in (3.2.9).

Note, however, that the projected subgradient method gives us directly only a value for the dual price vector λ. We discuss below how to go back to a primal solution; the main issue is that, due to the duality gap, standard methods for obtaining a primal solution are only guaranteed to satisfy the original hard shared constraints *in expectation*. Hence, additional steps are required to guarantee a feasible solution to the original MILP; we describe a simple randomized rounding scheme in Section 3.3.3 for obtaining a near-optimal solution to the MILP.

3.2.4 Lagrangian Relaxation for Factored MDPs

We now describe how to apply Lagrangian relaxation to deterministic factored MDPs, which we use to formulate the traffic management domain.

A general multi-agent MDP can be written as an MDP whose size is exponential in the number of agents: its state and action spaces are the cross-product of all agents' state and action spaces. However, as we have observed, a multi-agent planning problem is mostly factored, where each agent's local problem can be isolated from the global problem. Hence we can define a *factored MDP* in which the state and action probabilities of each agent only depends on other agents through a set of shared constraints.

Let random variables A_{it} and S_{it} represent the action and state of agent i at time t respectively. For agent i, given the distribution over the initial states $P(S_{i1})$, the transition probabilities $P(S_{i,t+1}|A_{it}, S_{it})$, and the rewards $r_{A_{it},S_{it}}$, the goal of MDP planning is to find a policy, $P(A_{it}|S_{it})$, that optimizes:

$$\max_{P(A_{it}|S_{it}),P(A_{it},S_{it})} \sum_{t=1}^{T} \sum_{A_{it},S_{it}} r_{A_{it},S_{it}} P(A_{it}, S_{it}) \tag{3.2.10}$$

s.t.

$$\begin{aligned}
P(A_{i1}, S_{i1}) &= P(A_{i1}|S_{i1})P(S_{i1}), \quad \forall i, A_{i1}, S_{i1} && (3.2.11)\\
P(A_{it}, S_{it}) &= P(A_{it}|S_{it}) \sum_{S_{i,t-1}} \sum_{A_{i,t-1}} P(S_{it}|A_{i,t-1}, S_{i,t-1})P(A_{i,t-1}|S_{i,t-1}), && (3.2.12)\\
&\quad \forall i, t > 1, A_{it}, S_{it} \\
&\sum_{A_{it},S_{it}} P(A_{it}, S_{it}) = 1, \quad \forall i, t && (3.2.13)\\
&\sum_{A_{it}} P(A_{it}|S_{it}) = 1, \quad \forall i, t, S_{it} && (3.2.14)\\
&P(A_{it}|S_{it}) \in \{0, 1\}, \quad \forall i, t, A_{it}, S_{it}. && (3.2.15)
\end{aligned}$$

For this chapter we will focus on *deterministic* MDPs, where the initial state distribution and the transition probabilities are assumed to be deterministic, i.e.,

$$P(S_{i1}) \in \{0, 1\}, \quad P(S_{it}|A_{i,t-1}, S_{i,t-1}) \in \{0, 1\},$$

and the agents' (deterministic) policies completely determine the joint state at all times. Since a deterministic optimal policy always exists for a MDP, constraints (3.2.15) ensure that such a policy is picked over one that may have non-integral policy probabilities due to ties in the value function. In turn, the joint state-action probabilities $P(A_{it}, S_{it})$, which are directly computable from the policy, transition probabilities, and the initial state distribution, are also deterministic. Hence, (3.2.11)–(3.2.15) define a MILP, yielding shared constraints on *indivisible* resources:

$$f_j(\sum_i D_{ij} P(A_{it}, S_{it}) - b_j).$$

Note that the program defined by (3.2.10)–(3.2.15) is not a MILP, since constraints (3.2.12) are not linear. However, since $P(S_{it}|A_{i,t-1}, S_{i,t-1})$ and $P(S_{i1})$ are binary, we can obtain an equivalent MILP by writing each of the constraints as an equivalent set of linear constraints. Here we leave them in the more intuitive form for readability. For optimization purposes, the exact expression we use here for the constraints has no effect, since our Lagrangian relaxation algorithm allows us to solve each agent's MDP using value iteration.

Using hinge loss $f_j^{\alpha_j}$ as shared constraint penalty, the global multi-agent optimization problem becomes

$$\max_{P(A_i|S_i),P(A_i,S_i)} \sum_{i=1}^{n} r_i^{\mathrm{T}} P(A_i, S_i) - \sum_{j=1}^{m} f_j^{\alpha_j}(\sum_i D_{ij} P(A_i, S_i) - b_j) \text{ s.t. } \forall i, C_i \qquad (3.2.16)$$

where C_i denotes the constraint set defined by equations (3.2.11)–(3.2.14) for each agent i, and we have vectorized rewards r_i and the state-action probabilities $P(A_i, S_i) = [P(A_{i1}, S_{i1}) \dots P(A_{iT}, S_{iT})]$. As usual, we denote matrices $D_1, ..., D_n$ such that D_i defines the linear resource usage of agent i; for factored MDPs, resource usage is a function of the state-action probabilities, $D_{ij}P(A_i, S_i)$, for agent i and resource j. For example, for the traffic management problem, the resource usage for resource j, a patch of road, is of the form

$$\sum_i \sum_t I(\text{position of agent } i \text{ indicated by } S_{it} \text{ is within road patch } j) P(S_{it})$$

where I is an indicator function, as we aim to limit the number (or more precisely, the sum of the probabilities) of agents in a particular region of road at the same time. (Note that $P(S_{it}) = \sum_{A_{it}} P(A_{it}, S_{it})$, hence the resource usages are linear in $P(A_{it}, S_{it})$.)

Now, letting vectors $x_i = P(A_i, S_i)$ for notational simplicity, the Lagrangian relaxation of (3.2.16) gives us the optimization problem

$$\inf_{\lambda} V(\lambda) = \inf_{\lambda} \max_{x} \sum_{i=1}^{n} r_i^{\mathrm{T}} x_i + \sum_{j=1}^{m} [f_j^{\alpha_j *}(\lambda_j) - \lambda_j (\sum_{i=1:n} D_{ij} x_i - b_j)] \text{ s.t. } \forall i, C_i. \qquad (3.2.17)$$

Expanding the dual function of hinge loss

$$f_j^{\alpha_j *}(\lambda_j) = \sup_x [\lambda_j x - f_j^{\alpha_j}(x)],$$

we obtain

$$V(\lambda) = \max_x \sum_{i=1}^{n} r_i^{\mathrm{T}} x_i + \sum_{j=1}^{m} [I(0 \le \lambda_j \le \alpha_j) + \lambda_j (b_j - \sum_i D_{ij} x_i)] \text{ s.t. } \forall i, C_i,$$

where again α_j is the slope of hinge loss $f_j^{\alpha_j}$, and the indicator function $I(Z)$ equals ∞ if condition Z is not satisfied, and equals 0 otherwise.

Note that $V(\lambda)$ is subdifferentiable for λ s.t. $\lambda_j \in [0, \alpha_j], \forall j$, with subgradient shown in Equation (3.2.9). To compute the gradient, x_i^* can be calculated by each agent i in a distributed fashion given the current λ. In particular, for factored MDPs, we can use value iteration to efficiently solve each subproblem to find x_i^*.

3.3 Optimization

Here we discuss two gradient-based methods, the subgradient method and an accelerated gradient descent method, FISTA, for finding a solution to the Lagrangian relaxation, and how they can be applied to the Lagrangian relaxation of factored MDPs. We then show how to obtain a near-optimal feasible solution to the original MILP once we have found a solution to the Lagrangian relaxation.

3.3.1 Subgradient Projection

Subgradient projection is a general and robust gradient-based algorithm, requiring only a subgradient at each point in the domain. Applying subgradient projection to the Lagrangian relaxation problem (3.2.7) using the subradient and the projection operator described in Section 3.2.3 leads to the subgradient Lagrangian relaxation (SLR) algorithm introduced by Gordon et al. [31], which we show here in Algorithm 3.

Algorithm 3 The subgradient Lagrangian relaxation algorithm

Inputs: η, the step size constant. T, number of iterations.
Outputs: $\overline{x}_i$, average policies. $\overline{\lambda}_j$, average dual values.
$\lambda_{j0} \leftarrow 0$
for $t \leftarrow 1, 2, ..., T$

- $\forall i, x^*_{it} \leftarrow \arg\max_{x_i \text{ s.t. } C_i} (c_i - \lambda_{*t} D_i)^{\mathrm{T}} x_i$
- $\forall j, z^*_j \leftarrow \arg\sup_z [\lambda_{jt} z - f_j(z)]$
- $\forall j, \lambda_{jt} \leftarrow \lambda_{j,t-1} - \frac{\eta}{\sqrt{t}} (z^*_j + b_j - \sum_i D_{ij} x^*_{it})$
- $\forall j, \lambda_{jt} \leftarrow \max(\alpha^{\min}_j, \min(\alpha^{\max}_j, \lambda_{jt}))$

$\forall i, \overline{x}_i = \frac{1}{t} \sum_{k=1}^{t} x^*_{ik}$
$\forall j, \overline{\lambda}_j = \frac{1}{t} \sum_{k=1}^{t} \lambda_{jk}$

Algorithm 3 encodes one of the standard methods for recovering a primal solution from the dual subgradient method: one of the outputs of the algorithm is the *average* policy $\overline{x}_i$, which can be used to find a feasible solution to the original MILP by rounding as described in Section 3.3.3. The solution x^*_{it} at some iteration t is not guaranteed to be a convergent solution, since the objective function may not necessarily be smooth and thus the solution may oscillate indefinitely[3].

Nevertheless, the average solution $\overline{x}_i$ converges the set of optimal solutions and the cost of $\overline{x}_i$ converges to the optimal cost. SLR, as per standard results about subgradient method, has

[3] For example, consider a hinge loss function for a single resource with a large number of agents. Let every agent be identical, and let each agent's reward be -1 if it did not use the resource, and 1 if it did. Then there are two policies for each agent: "use the resource" and "do not use the resource". Now, since all agents are equivalent, at each iteration every agent will compute the same policy given the resource price, and the resource price, i.e., the dual variable, will oscillate: if all agents choose not to use the resource, the resource costs will be reduced, hence in the next iteration all agents will choose to use it, which raises the resource price, and the cycle repeats.

a convergence rate of $O(\frac{1}{\sqrt{t}})$ on general convex objective functions, hence to acquire an ϵ-approximate solution requires $O(\frac{1}{\epsilon^2})$ iterations.

3.3.2 Accelerated Gradient Methods

While subgradient method is a very robust optimization algorithm, it suffers from a slow convergence rate. Quasi-Newton methods such as L-BFGS are a popular alternative to the subgradient method, and have seen much success in practice [54]. However, they assume a smooth (twice-differentiable) objective function in general, and hence are not, at least theoretically, suitable for most versions of the constraint penalty function f[4]. Here we focus on an accelerated gradient method, FISTA [7], which can handle objective functions with non-smooth parts. We describe how FISTA can be used with Lagrangian relaxation. Then in Section 3.4, we give an augmented objective for the factored MDP domain for optimizing with FISTA and demonstrate the benefits of the accelerated gradient method on efficient optimization of the Lagrangian relaxation on the traffic management problem.

FISTA is guaranteed to converge to an ϵ-approximate solution in $O(1/\sqrt{\epsilon})$ iterations given an objective function with a smooth component[5]. More precisely, FISTA works on an objective function $F = f + g$ comprised of two functions f and g, where $f : \mathbb{R}^n \to \mathbb{R}$ is a continuous convex but potentially non-smooth function, and $g : \mathbb{R}^n \to \mathbb{R}$ is a smooth function with a Lipschitz continuous gradient, i.e., for all $x, y \in \mathbb{R}^n$,

$$\|\nabla g(x) - \nabla g(y)\| \leq L\|x - y\|,$$

where $L > 0$ is called a *Lipschitz constant* for function ∇g. Intuitively, the existence of a Lipschitz constant for the gradient of a function implies that the gradient cannot change too quickly, and thus the function is smooth, which is generally nicer to optimize.

If our objective functions $c_i(x)$ are strongly convex, then the Lagrangian dual will have a Lipschitz continuous gradient with respect to λ, and we can directly apply FISTA. However, if $c_i(x)$ are not strongly convex, such as in the case of linear objective functions, we can augment the objective with a strongly convex term such as a quadratic loss:

$$g(x) = \sum_i x_i^2$$

whose strength can be controlled by a parameter which will also affect the Lipschitz constant L. If the original problem is linear, adding a quadratic term turns the subproblems into quadratic programs, which can be solved using a variety of existing quadratic program solvers.

The solution to this augmented program may not correspond to the optimum to the original program. However, it is possible to bound the error based on the penalty term and the subproblem

[4]However, L-BFGS can be fast and robust in practice when used with augmentation, as we show in Chapter 4.

[5]Subgradient method achieves $O(1/\epsilon)$ convergence for the same class of functions, hence we can use the augmentation methods in this section to accelerate convergence using subgradient method, albeit to a slower rate than using FISTA.

solver, as we show in Section 3.4 for factored MDPs. We can then account for the smoothing error as part of the approximation error for Lagrangian relaxation in achieving an ϵ-approximation solution. That is, we trade the error introduced by the penalty term against the error introduced by incomplete convergence of the optimizer given a fixed iteration budget. In Section 3.4.2 we demonstrate experimentally that the solution to the augmented problem gives us near-optimal solutions, while achieving dramatically faster convergence than subgradient projection.

We note that unlike the subgradient method, FISTA will give convergent primal solutions. However, since due to Lagrangian relaxation the primal solutions obtained will only satisfy the shared constraints in expectation, and furthermore here they are solutions to the augmented problem, we must still find a feasible solution to the original MILP. We describe one method to do so in the following section.

3.3.3 Randomized Rounding

Given an ϵ-approximate (feasible) solution to the Langrangian relaxation dual obtained via methods above, we must find a feasible primal solution, i.e., plans for agents. Here we give a simple randomized rounding scheme that gives near-optimal primal solutions when a large number of agents contend for each resource.

Given $\overline{x}$, a *distribution* over deterministic plans, which may be an average solution from the subgradient method or a solution from FISTA on the augmented program, each agent can *independently* sample a deterministic individual plan (where only one action is taken given a state). The sampled set of plans may violate the shared constraints, which we resolve via three methods.

First, for problems with a large number of agents and restrictions on each agent's contribution to overall resource usage, we can increase the likelihood of the sampled solution being globally feasible by "padding" the constraints. We can define a small "cushion" $\epsilon_j \geq 0$ on each resource usage penalty, which modifies the resource penalty function f_j to

$$f_j(\epsilon_j + \sum_i D_{ij} x_i - b_j). \tag{3.3.1}$$

The intuition for the cushions will be that since we have effectively decreased the allowed resource usage, solutions with a low penalty value should be more likely to yield solutions with acceptable levels of resource usage.

Also, following Gordon et al. [31], we can define *limited influence* of an agent on a resource in two parts. First, we assume that every agent has a similar amount of influence on the overall usage of each resource as well as on the value of the optimal plan, i.e., that there exists a constant $U > 0$ such that

$$-\frac{U}{n}|V^*| \leq c_i^{\mathrm{T}} x_i \leq \frac{U}{n}|V^*| \tag{3.3.2}$$

$$-\frac{U}{n}|u_j^*| \leq D_{ij}^{\mathrm{T}} x_i \leq \frac{U}{n}|u_j^*| \tag{3.3.3}$$

for all i, j, and $x_i \in C_i$, where $u_j^* = \sum_i D_{ij} x_i^*$ is the usage of resource j in some optimal solution. Second, we assume that a small change in the overall resource usage leads to only a small change in solution quality, i.e., that there are no sudden jumps in the solution with respect to the resource usage. In particular, let V_ϵ^* be the optimal dual value obtained using f_j of the form (3.3.1). Then we assume that there exists a bound $\epsilon_{\max} > 0$, condition number $\kappa > 0$, and a discretization level $\triangle > 0$ such that

$$V_\epsilon^* \geq V^* - \kappa \sum_j \epsilon_j - \triangle/n,$$

so long as $0 \leq \epsilon_j \leq \epsilon_{\max}$ for all j. While the second restriction limits us from having hard constraints, it allows for the use of piece-wise linear penalty functions.

Gordon et al. (Theorem 1, [31]) show that if we make the assumptions above, then the result x of our randomized rounding algorithm, sampled from a solution to the Lagrangian relaxation using $\epsilon_j = \bar{\epsilon}$ for all j where $\bar{\epsilon} \leq \epsilon_{\max}$, satisfies

$$V(x) \geq V^* - \triangle/n - (\kappa m + 1)\bar{\epsilon} - \gamma$$

with probability at leaset $1 - \delta$, where $V(x)$ is the primal value of the sampled solution x, m is the number of resources, γ is the tolerance to which the optimization algorithm was run, and

$$\delta = e^{-n\bar{\epsilon}^2/2U^2|V^*|^2} + \sum_j e^{-n\bar{\epsilon}^2/2U^2|y_j^*|^2}.$$

Hence, to limit the probability of failure to obtain a feasible solution from rounding to δ, we can set the cushion to be $\bar{\epsilon} = \Omega(\frac{\ln 1/\delta}{\sqrt{n}})$.

Second, we can repeat the randomized rounding process until we obtain a feasible solution, since given a bounded probability of failure of getting a feasible solution, the expected number of trials before succeeding is constant.

Last, we employ a reactive planner to adjust the individual plans (either centrally or by coordinating the agents) to satisfy the shared constraints in order to ultimately guarantee feasibility in practice. This reactive planner can greatly reduce the number of resampling steps in practice, since we only have to discover a plan that is close enough to feasible that the reactive planner can correct it.

3.4 An Accelerated Gradient Method for Factored MDPs

The trick of optimizing an objective augmented with a strongly convex penalty discussed in Section 3.3.2 is often used in combination with accelerated gradient methods, and has been successful in other optimization problems (e.g. [16]). However, in the case of factored MDPs, we face an additional difficulty: if we add a strongly convex penalty such as the quadratic loss to the MDP objective, we can no longer use fast MDP planning algorithms such as value iteration. If we were forced to back off to general convex optimization methods such as quadratic programming

when solving the MDP subproblems, we would lose much of the benefit of the accelerated gradient algorithm: our iteration count would be lower, but each iteration would be much more expensive. To address this issue, we propose to use a maximum causal entropy penalty [98], a common regularizer for MDPs in reinforcement learning. With this penalty, we can still use a slight modification of value iteration to solve each individual MDP. While causal entropy may not be strongly convex at every point in the domain, we demonstrate that in practice it can still give us much faster overall convergence.

3.4.1 The Augmented Program

Causal entropy is defined as

$$\sum_t H(A_{it}||S_{it}),$$

where

$$H(A_{it}||S_{it}) = -\sum_{A_{it},S_{it}} P(A_{it},S_{it}) \log P(A_{it}|S_{it}).$$

Augmented with causal entropy, our objective (3.2.16) becomes

$$\max_{P(A_{it}|S_{it}),P(A_{it},S_{it})} \sum_{i=1}^{n} \frac{1}{\beta} \sum_{t'} H(A_{it'}||S_{it'}) + r_i^{\mathrm{T}} P(A_{it},S_{it}) - \sum_{j=1}^{r} f_j^{\alpha_j}(\sum_i D_{ij} P(A_{it},S_{it}) - b_j)$$
$$\text{s.t. } \forall i, C_i,$$

where $\frac{1}{\beta}$ controls the contribution of the causal entropy term in the objective. Then, applying the Lagrangian relaxation, our goal becomes to find

$$V^* = \inf_{\lambda} \max_{x,y} \sum_{i=1}^{n} \frac{1}{\beta} H(x_i,y_i) + r_i^{\mathrm{T}} x_i + \sum_{j=1}^{m} [I(0 \le \lambda_j \le \alpha_j) + \lambda_j (b_j - \sum_i D_{ij} y_i)] \text{ s.t. } \forall i, C_i,$$

where, as before we define vectors $x_i = P(A_{it},S_{it})$ and $y_i = P(A_{it}|S_{it})$, and let $H(x_i,y_i) = \sum_{t'} H(A_{it'}||S_{it'})$.

To compute the subgradient, again we must solve the subproblems, whose objectives now contain the causal entropy term $H(x_i,y_i)$. Such subproblems can be solved using a variant of value iteration called *softmax value iteration* [97], shown in Algorithm 4. The modification from value iteration happens in the last line, where the maximum function is replaced by the softmax function

$$\text{softmax}_x^{\beta}(f(x)) = \log \sum_x e^{f(x)/\beta}.$$

Then, the final policy output by the algorithm is a stochastic policy:

$$P(A_{it}|S_{it}) = \frac{e^{\frac{1}{\beta}Q(S_{it},A_{it})}}{\sum_a e^{\frac{1}{\beta}Q(S_{it},a)}},$$

in comparison to the deterministic policies returned by value iteration:

$$P(A_{it}|S_{it}) = \begin{cases} 1, & \text{if } A_{it} = A^*_{it} \\ 0, & \text{otherwise} \end{cases}$$

where A^*_{it} is a randomly chosen element of the set $\{\arg\max_a Q(S_{it}, a)\}$.

Algorithm 4 The softmax value iteration algorithm

Input: β, the smoothing parameter.
Output: V, the value function.
$\forall s, V(s) = 0$
for $t \leftarrow 1, \ldots, T$

- $\forall s, a, Q(s, a) = R(s, a) + \sum'_s V(s')P(s'|s, a)$
- $\forall s, V(s) = \text{softmax}^{\beta}_a(Q(s', a))$

Finally, Algorithm 5 shows the resulting FISTA algorithm for factored MDPs with hinge loss penalties (AGLR).

Algorithm 5 The accelerated gradient Lagrangian relaxation algorithm for factored MDPs

Input: L, a Lipschitz constant of $\nabla V_\beta(\lambda)$.
Output: x^*_i, policies. λ_j, dual values.
$\lambda_0 \leftarrow 0$
$\nu_1 \leftarrow \lambda_0$
$d_0 \leftarrow 1$
for $t \leftarrow 1, 2, ...T$

- $\forall i, x^*_i, y^*_i \leftarrow \arg\max_{x_i, y_i} \frac{1}{\beta} H(x_i, y_i) + (r_i - \lambda D_i)^{\mathrm{T}} y_i$ // solved by softmax value iteration
- $\forall j, \lambda_{tj} \leftarrow \nu_{tj} - \frac{1}{L}(b_j - \sum_i D_{ij} y^*_i)$
- $\forall j, \lambda_{tj} \leftarrow \max(0, \min(\alpha_j, \lambda_{tj}))$
- $d_{t+1} \leftarrow \frac{1+\sqrt{1+4d_t^2}}{2}$
- $\nu_{t+1} \leftarrow \lambda_t + (\frac{d_t - 1}{d_{t+1}})(\lambda_t - \lambda_{t-1})$

Errors in Solution

Augmenting a factored MDP with causal entropy to use with FISTA introduces two kinds of errors in the computed value of V^*.

First stems from the fact that, unfortunately, causal entropy may not be strongly convex in $P(A_{it}, S_{it})$ everywhere in the domain. While subgradient method is robust to the fact, convergence guarantees for FISTA require a strongly convex component in the objective. Nevertheless,

in practice we can detect convergence by examining the primal-dual gap, and in our experiments FISTA often achieves a small primal-dual gap.

The second source of error comes simply from the fact that we are no longer optimizing the original objective due to augmentation by causal entropy. Naturally, the error depends on the strength of the penalty term controlled by β, and we can show that the suboptimality of the solution obtained by maximizing the causal entropy objective is bounded by $(T_H \log |A|)/\beta$, where $|A|$ denotes the size of the set of actions in the MDP, and T_H the time horizon.

3.4.2 Experiments

We study the convergence rate and quality of SLR and AGLR on a path planning domain of X-shaped bridges and interchanges described in Section 3.1.

Empirical Quality of Solution

First we examine the effectness of Lagrangian relaxation with approximated shared constraints. In particular, we look at how plans sampled from a solution to the Lagrangian relaxation fare with respect to the original hard shared constraints.

Figure 3.2 reveals the reduction the number of collisions in plans sampled from solutions to the Lagrangian relaxation with shared constraints compared with those from solutions to the baseline of not using shared constraints (which is equivalent to running one iteration of the Lagrangian relaxation algorithm with resource prices set to zero). For both methods, we optimized the causal entropy augmented programs via AGLR. Each bar represents a problem setting, corresponding to a specific value of two parameters that contribute to the problem's difficulty: the rate of entry onto the bridge and how likely a vehicle must change lanes to reach its goal. For each bar, the number of collisions is averaged over 4 problem instances of each setting, where for each problem instance we picked the best plan out of 100 iterations of the randomized rounding method. The percentages were calculated using the best setting of the smoothing parameter β for each method and each problem.

Bars above the zero line represent problems where incorporating shared constraints into the problem was beneficial. While in most cases shared constraints reduce the number of collisions, it can also increase the number of collisions. We posit that such increases are due to randomization playing a greater role when shared constraints must be satisfied, as through our Lagrangian relaxation the original hard shared constraints must be satisified in expectation. Randomization can cause suboptimality; for example, when a problem requires little coordination and a nearly-deterministic solution from not using shared constraints is close to optimum and deviation from such causes suboptimality.

We note that even when the number of violated constraints is reduced, we may not always find a violation-free solution by solving the Lagrangian relaxation. Two main sources of error exist. First and the most probable cause is that our problem domain may not be large enough for the

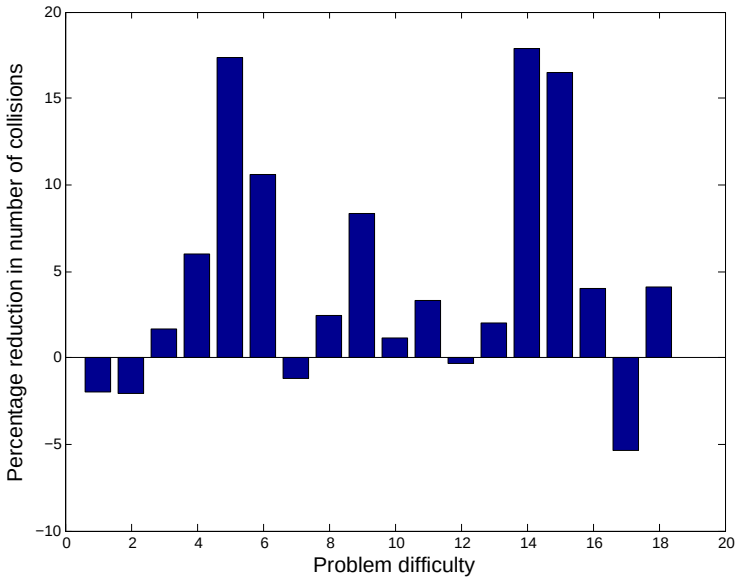


Figure 3.2: A comparison of the number of collisions in sampled plans for 20-agent X-shaped bridge traffic management problem instances. The bars shown are percentage reductions in collision counts when employing shared constraints.

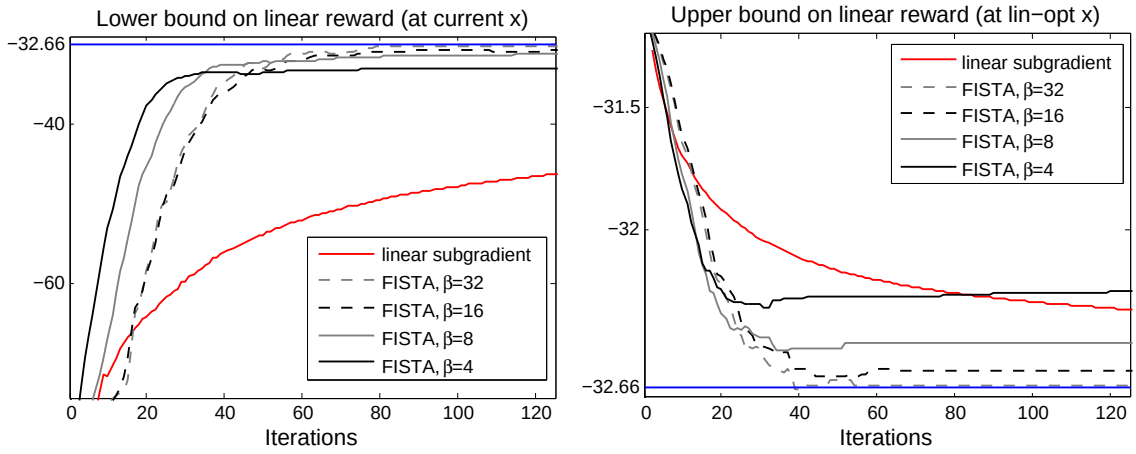
approximation guarantee to be effective, yielding a high failure rate for rounding. Second, the MDP representation models the states as non-trivially large blocks of space when in reality the space is continuous, which can handicap the agents. In particular, since agents are restricted to the discretized positions and the small number of actions to move between them, collisions may be more likely than if they were given the full freedom of the continuous space. This discretization problem helps explain why it is possible to correct for the collisions using local controllers, even when the deliberative plan contains collisions.

Hence, in the presence of such violations, we can employ a local, reactive planner to correct the "nearly-good" solutions returned by optimizing the Lagrangian relaxation. For the traffic management problem, we implemented a reactive planner (as described in Section 3.1) at each vehicle so that the vehicles try to keep a small distance from each other while following the plans returned by our algorithm.

Convergence Experiments

The convergence experiment results show how the choice of the smoothing parameter β can balance the trade off between convergence and optimality to achieve accelerated convergence while maintaining solution quality comparable to directly optimizing the linear reward function (despite the bias introduced by the causal entropy term).

Figure 3.3(a) shows the convergence of AGLR with different values of smoothing parameter β, in terms of the linear primal values (without the causal entropy term), computed at the solution to the augmented objective at each iteration. Hence these values correspond to the solution



(a) Convergence of lower bound (primal)

(b) Convergence of upper bound (dual)

Figure 3.3: Convergence comparisons. The blue lines indicate an approximate optimal solution, as it upper bounds the lower bound curves in (a) and lower bounds the upper bound curves in (b).

available to be used at each iteration, and also serve as lower bounds to the optimal solution. The lower bounds demonstrate that in all cases optimizing the augmented objective leads to improved convergence albeit at the expense of optimality with respect to the unaugmented objective. A lower value of β means the objective has a greater weight on causal entropy, and unsurprisingly, runs with lower β values converge more quickly, but to worse linear objective values. However, the subgradient Lagrangian relaxation directly optimizing the linear objective function (the solid red line), while eventually reaching the optimum solution (whereas running FISTA with a large β may not), converges extremely slowly; at iteration 1000 it achieves a lower bound of -38.543, which FISTA with $\beta = 4$ reaches at iteration 20.

In Figure 3.3(b) we plot upper bounds of the optimal primal value of the same AGLR runs; at each iteration, we plot $V_\infty(\lambda)$, i.e., the dual value of the *linear* objective at the current λ. The plot shows the trade-off between convergence speed and quality very similar to that in the lower bound plot. Moreover, it shows that with a sufficiently high value of β, we can approach the optimum very quickly; the blue horizontal line in both Figures 3.3(a)-3.3(b) represent the same value that is bounded and closely approached by the $\beta = 32$ curves in both plots. This shows that with a high enough value of β, we may recover the optimal solution with substantially fewer iterations than subgradient method.

In this example setting, $\beta = 32$ appears sufficient to achieve a good solution and very fast convergence. However, it would be interesting to see if any gain would come from adaptively tuning β over the gradient descent iterations to achieve even faster convergence while maintaining good solution quality.

Finally, the experiments demonstrate the robustness of Lagrangian relaxation; even though it was optimized using a potentially non-strongly-convex objective, we were able to find a solution to the Lagrangian relaxation with a very small duality-gap with respect to the non-augmented objective, which corresponds to finding a near-optimal solution.

3.5 Conclusion

In this chapter, we expanded upon the mathematical programming framework for distributed multi-agent planning introduced in Chapter 2 via Lagrangian relaxation for more efficient planning in time-constrained domains. We argued that for a large-scale problem where each agent is only a small part of the problem, we can find near-optimal solutions to the original unrelaxed problem via Lagrangian relaxation. We described how to optimize the Lagrangian relaxation with different optimization methods, the subgradient method and an accelerated gradient method, FISTA. In particular, we gave an augmented accelerated gradient method for factored MDPs using causal entropy regularization. Despite the potential errors caused by altering the objective function, our convergence results show 1. that the augmented program can achieve near-optimality in practice with little work in tuning the smoothing parameter, and 2. that it does so at a superior convergence rate in comparison to the standard subgradient algorithm on the original objective.

While we have established the groundwork for efficiently optimizing via Lagrangian relaxation on an instance of the traffic management problem, we believe that many domains can benefit from the relaxation and the accelerated gradient method. It would be interesting to study the algorithm on larger and more complex problems, and to extend the algorithm to adaptively control the weight of the causal entropy smoothing term to further speed up convergence.

Chapter 4

Planning under Uncertainty

Planning under uncertainty is often a non-trivial challenge in auction-based distributed planning frameworks. An advantage of the MILP representation for distributed multi-agent planning is its natural extension for planning under uncertainty, *stochastic programming* (e.g. [12]), a widely used mechanism, particularly in operations research, for planning in stages, as information is revealed sequentially. The idea is to plan for different *scenarios*, which are possible futures sampled according to a distribution of future random events such as obstruction of a road in path planning or the locations of future tasks in UAV task allocation (learned, assumed, or known otherwise)[1]. Stochastic programming provides us with a principled planning framework under uncertainty; hence, given a deterministic problem written as a MILP, we can easily incorporate uncertainty without having to devise a domain-specific recipe for handling uncertainty. Furthermore, the resulting stochastic program can be reduced to an ordinary MILP. While this MILP may be larger than a similar deterministic program, it can be solved using methods discussed in Chapters 2 and 3 for MILPs and their Lagrangian relaxations.

In this chapter, we extend the Lagrangian Relaxation framework introduced in Chapter 3 to incorporate uncertainty via stochastic programming. In particular, we focus on factored MDPs, where uncertainty, in the form of sampled *scenarios*, can be naturally encoded in the individual MDPs and solved in a distributed fashion. The number of resources (i.e., constraints) in the resulting program is linear in the number of scenarios, as we must satisfy the constraints in each scenario considered.

To better understand the role of incorporating uncertainty in planning, we study supply chain management as our motivating application. Supply chain management is an important problem in production and delivery of goods, where many producers must cooperate to maximize efficiency, and therefore profit. In particular, we examine the phenomenon of *the bullwhip effect*, where changes in the economic environment, e.g., the consumer demand, create a whiplash effect

[1] In particular, the future events considered for sampling are *random* from the perspective of the agents, and are assumed to be independent of their actions. The future as a whole is affected both by these random events and the agents' actions, and so the future is (of course) highly dependent on the agents' actions. However, our methods deal with the influence of agents' actions on the future and thus on each other by having the agents jointly plan, with enough knowledge about other agents' plans.

through the supply chain, resulting in large fluctuations in production along the chain, which are exaggerated further as we go away from the consumer demand. The bullwhip effect is a result of agents planning without consideration for a possibly changing economy, in particular failing to plan for others' reactions to the changes, and shows that such a supply chain is slow to respond and stabilize to the demand, leading to economic inefficiencies. We show in our experiments that using stochastic programming we can lessen the bullwhip effect, as agents can prepare for changing economies.

In the following, we first introduce our main domain of supply chain management and give an example of the bullwhip effect. We then provide an overview of stochastic programming on the Lagrangian relaxation, and show our novel stochastic programming formulation of factored MDPs, which can be efficiently constructed from the original factored MDPs. Finally we give experimental results demonstrating that stochastic programming can help ameliorate the bullwhip effect, yielding more efficient plans than reactively trying to fix plans that were generated based on the assumption of a deterministic world.

4.1 Supply Chain Management and the Bullwhip Effect

Supply chain management tackles the problem of efficient production and distribution of goods, determining a variety of actions from inventory management and production to transportation and returns processing. An artificial example supply chain is shown in Figure 4.1. A supply chain may include a number of different organizations, each of which may be composed of many divisions. The goal is for each entity, whether an organization or a smaller unit within, to produce its goods efficiently, while meeting the contracts promised between it and its adjoining entities.

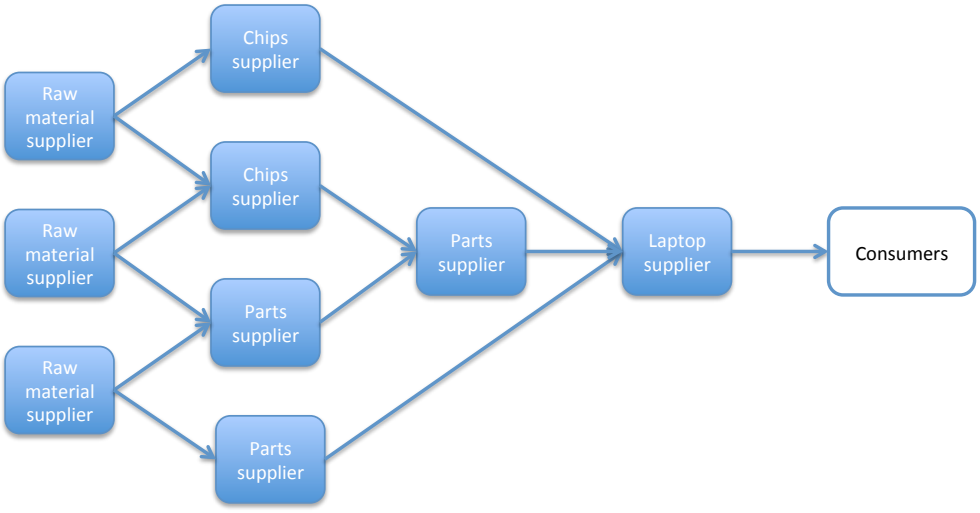


Figure 4.1: An example supply chain.

While supply chain management often includes competitive factors as businesses compete for best contracts, here we assume a supply chain comprising cooperative agents wishing to produce

and transfer goods at socially optimal costs. For example, it is common for a vertically-integrated company to need to manage different production divisions within. While such divisions will cooperate for the overall good of the company, they may be segregated (physically or managerially) enough to warrant distributed planning. The goal then is to efficiently run the supply chain to maximize profit given the current consumer demand.

For simplicity, we assume that each agent produces one kind of goods, and is locally formulated as a Markov decision process (MDP), where the state encodes its inventory levels and the actions represent the quantity of *production*, *buys*, and *sells* at each time step. Also defined for each agent is the production cost curve, and the consumer-facing agent is also aware of the consumer demand curve (example curves are shown in Figure 4.2).

The joint state is then the tuple of individual MDP states, and the joint action is the tuple of individual MDP actions. Our constraints limit the joint actions: we must match the selling and buying quantities of adjoining agents. Our planning algorithm learns prices corresponding to each of these constraints; we hope to learn prices that result in a socially-optimal joint plan.

We encode the uncertainty in the consumer demand in the form of the world economic state, which is either a *boom* or a *recession*. We assume that at each time step (a production cycle), the world may change its state with a small probability (a biased coin flip). Each economy is defined by the production cost and demand curves (e.g. see Figure 4.2). In an economic boom, the consumer demand curve is higher, leading to a higher equilibrium production point (where profit is maximized).

4.1.1 The Bullwhip Effect

If a supply chain does not prepare itself for a large change in the economy such as a change from a recession to boom, the resulting inefficiencies manifest themselves in the *bullwhip effect* [26]. Figure 4.3 demonstrates the bullwhip effect on a linear chain with four agents.

In this example, before time $t = 1$, all agents are producing for the equilibrium consumer demand level (1 unit) in recession. At $t = 1$ the economy changes to boom, and the consumer demand changes the optimal equilibrium production levels to 2 units. Agent 4, who produces the final consumer product, sells 2 units to settle into the equilibrium, at the expense of its inventory[2], then must order an extra unit of material from agent 3 and produce an extra unit to replenish its inventory. Agent 3, after selling extra units from its inventory at $t = 1$ must produce even more extra units than agent 4 in order to replenish agent 4 and its own inventories. The effect propagates up the supply chain (shown by the arrow).

In our example above, agents "overreact" to the changes in the demand at each edge in order to refill their inventories. However, the bullwhip effect can be even more exaggerated if agents, upon seeing the increased demand, anticipate an even higher demand in the future and overproduce. Hence, as

[2]In this example, a small cost is applied to low inventory (for both raw materials and produced goods) to encourage a non-zero inventory level; such a buffer is often used to cushion the effects of sudden changes in demand and to ease inventory management.

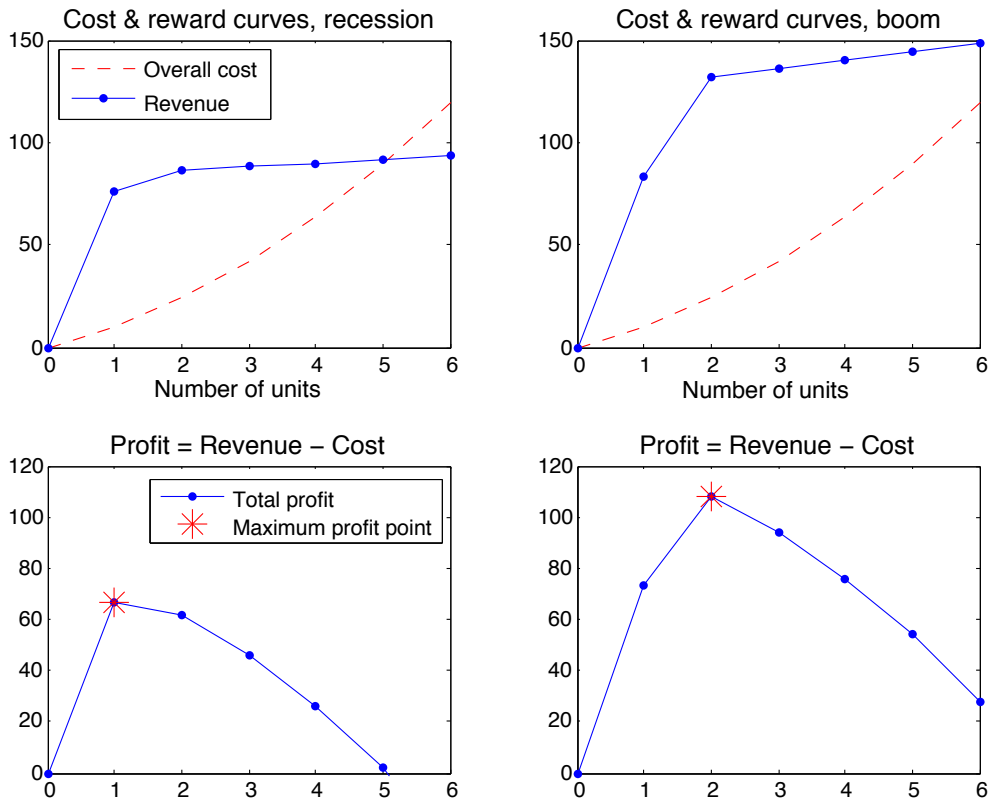


Figure 4.2: Example production cost and demand curves for each economic state. Here we have strongly convex production curves encoding overtime labor costs and capital investments required to expand production, and concave demand curves with diminishing returns.

agents up the supply chain try to meet the new demand, the deviation from the equilibrium demand is amplified[3], causing inefficient production (as the production costs are convex) and instability in inventory, and possibly run out of stock. In Section 4.3, we demonstrate how stochastic programming can ameliorate the bullwhip effect and lead to more efficient production.

4.2 Stochastic Programming

Stochastic programming incorporates uncertainty into planning by modeling it as a distribution of future random events[4]. Let v be the random variable for the sequence of future events, and $p(v)$ denote the probability distribution for v. Stochastic programming simply minimizes the *expected* cost with respect to $p(v)$.

In particular, a k-stage stochastic program with recourse models the world as k stages of informa-

[3]The amplification gives rise to the name bullwhip effect.

[4]We do not include our agents' actions as part of future events, only global events random to all agents. The uncertainty in other agents' actions from an individual agent's perspective is handled by coordinating through resource prices to obtain a joint solution.

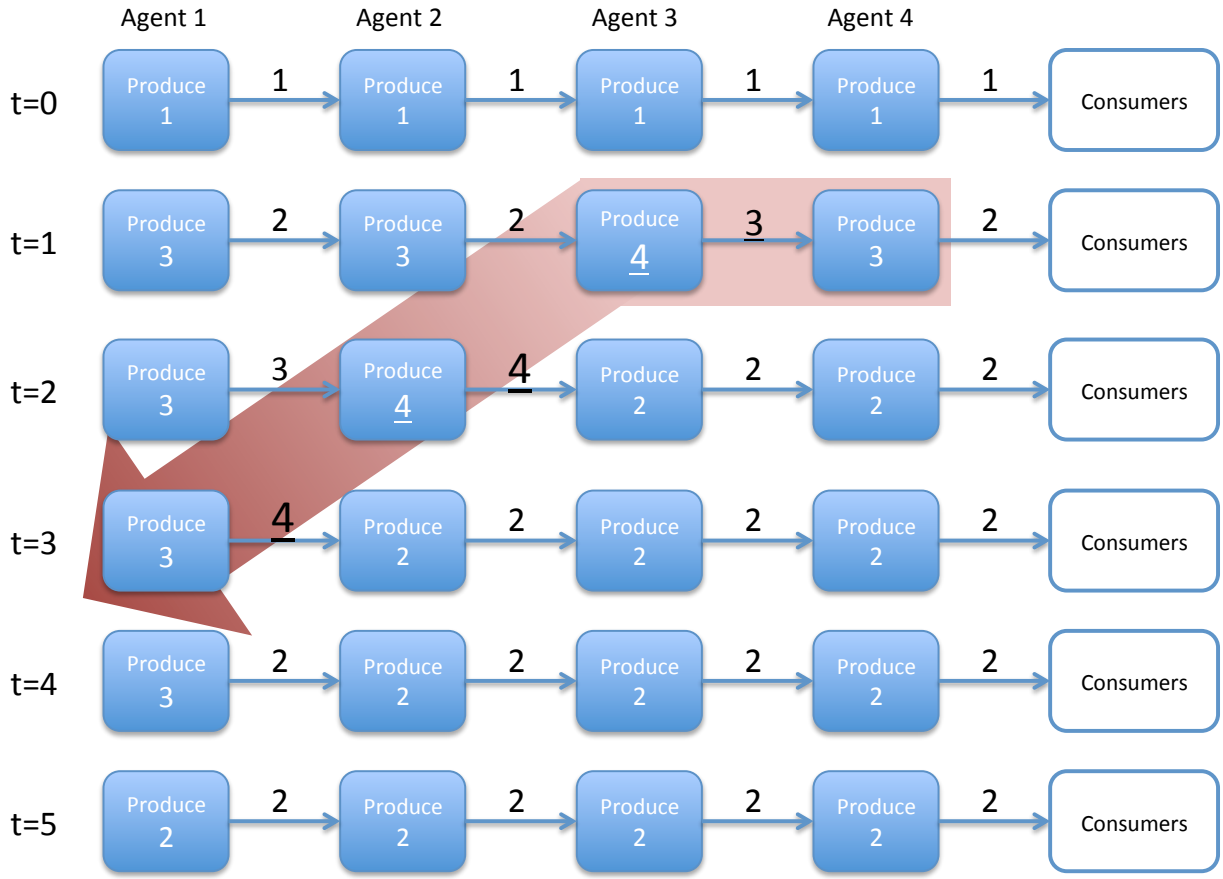


Figure 4.3: An example of the bullwhip effect on a four-agent supply chain. The red arrow shows the rise in demand propagating through the chain when the economy changes to boom from recession and the consumer demand increases.

tion revelation. At stage t, some information is revealed about the random events, and we follow the plans made specifically for the world corresponding to the revealed information, where the plans are represented by the t-th stage *recourse variables*. Here we present the 2-stage stochastic program for simplicity; however, the 2-stage representation is sufficient for factored MDPs, as we will show in Section 4.2.1 that for factored MDPs satisfying some assumptions (e.g., on the policy class), we can encode arbitrarily many stages in 2 stages.

In contrast to a MILP (3.2.1)–(3.2.3), we can write a 2-stage stochastic program as:

$$\max_{y} \sum_{i=1:n} c^{\mathrm{T}} y_i + E_{p(v)}[Q(y_i, v)] \tag{4.2.1}$$

$$\text{s.t.} \sum_{i=1:n} D_i y_i \leq b \tag{4.2.2}$$

$$y_i \in C_i, \qquad \forall i = 1, \ldots, n \tag{4.2.3}$$

where y denotes the first-stage variables and $Q(y_i, v)$ is the optimal value for agent i of the

second-stage problem

$$\max_x \sum_{i=1:n} q(v)^{\mathrm{T}} x_i \tag{4.2.4}$$

$$\text{s.t.} \sum_{i=1:n} D'_i x_i \leq b' \tag{4.2.5}$$

$$T(v)y + W(v)x = h(v), \qquad \forall i = 1, \ldots, n \tag{4.2.6}$$

$$x_i \in C'_i, \qquad \forall i = 1, \ldots, n. \tag{4.2.7}$$

Here x_i are the recourse variables for agent i under the future events represented by v, where $q(v), T(v), W(v), h(v)$ specify the second-stage problem for v. For multi-agent planning problems, which are largely factored, $T(v)$ and $W(v)$ are mostly block-diagonal, with a small number of rows for shared constraints.

Certain distributions p allow a closed-form, or a compact representation of the objective (4.2.1). For example, for the supply chain mangement problem, we assumed a sequence of Bernoulli coin flips, which seems simple enough to yield a compact representation for the stochastic program. Represented as an MDP, the individual problem size needs to be multiplied only by the number of possible world states, $|S_W|$, to be able to represent the cost function appropriately. However, the number of shared constraints (4.2.2) increases *exponentially* in the number of timesteps for which we plan, T, as the number of possible futures v is $|S_W|^{t-1}$. For factored MDPs, the fact that we must keep track of the constraints for each possible future implies that each agent's MDP state space must also be able to distinguish between the different futures, and thus the state space may be $O(|S_W|^{t-1})$ larger than the deterministic version of the MDP. Furturemore, for many distributions and cost functions, there is no closed-form expression for the expected cost objective, or computing it exactly is intractable.

To make the problem more tractable, we approximate the exponential space of possible futures by sampling from the space. Here we refer to resulting samples as *scenarios*. Given a set of scenarios Z, we can simply rewrite (4.2.1)–(4.2.3) as a *scenario-based program*:

$$\max_{y,x} \sum_{i=1:n} c^{\mathrm{T}} y_i + \sum_{\zeta \in Z} P(\zeta) q(\zeta)^{\mathrm{T}} x_{i\zeta} \tag{4.2.8}$$

$$\text{s.t.} \sum_{i=1:n} [D_i D'_i] \begin{pmatrix} y_i \\ x_{i\zeta} \end{pmatrix} = \begin{pmatrix} b \\ b' \end{pmatrix}, \qquad \forall \zeta \tag{4.2.9}$$

$$T(\zeta)y_i + W(\zeta)x_i = h(\zeta), \qquad \forall \zeta, i = 1, \ldots, n \tag{4.2.10}$$

$$y_i \in C_i, \quad x_{i\zeta} \in C'_i, \qquad \forall \zeta, i = 1, \ldots, n \tag{4.2.11}$$

where $x_{i\zeta}$ denotes the plan for agent i under scenario ζ. Here, if the set Z contains every possible scenario (with the frequency matching the distribution p), the scenario-based program is in fact equivalent to the stochastic program (4.2.1)–(4.2.3). The hope is that a tractable number of scenarios will suffice to compute a close approximation to the true optimal first-stage decision variables y and the optimal value of (4.2.1). (And in fact there are many results, both theoretical and experimental, which support this hope, e.g. [45, 46].)

4.2.1 Stochastic Programming for Factored MDPs

In Chapter 3, we introduced factored MDPs, which are represented by "small" MDPs specifying individual-agent planning problems, in addition to a set of shared constraints. In particular, we discussed *deterministic* factored MDPs where the agents' transition probabilities, initial state distributions, policies, and state-action probabilities are all deterministic, i.e., 0 or 1. Here we extend deterministic factored MDPs to incorporate uncertainty using stochastic programming.

Following the notation in Section 3.2, define A_{it}, S_{it} to be the random variables for the action and state of agent i at time t. Given the reward vector r_i, the initial state distribution $P(S_{i1})$, and the transition probabilities $P(S_{it}|A_{i,t-1}, S_{i,t-1})$, the goal of MDP planning is to find an optimal policy $P(A_i|S_i) = [P(A_{i1}|S_{i1}), \ldots, P(A_{iT}|S_{iT})]$ for a planning horizon T that maximizes the reward $r_i^{\mathrm{T}} P(A_i, S_i)$. In addition, our factored MDPs define the interaction between agents as shared constraints in the form of piece-wise linear functions f_j for each resource j.

In the stochastic program generalizing the deterministic factored MDP (3.2.16), we let the first-stage optimization variables be the randomness-dependent policies, denoted $P(A_i|S_i, V)$ for agent i and the random event V. Following the same convention for the deterministic MDP, $P(A_i|S_i, V)$ is a vector indexed by time. However, here we must specify the element at t as $P(A_{it}|S_{it}, V_t)$, corresponding to the policy at time t given V_t, the information available *up to* time t, since the policy decisions at time t should not depend on information from the future. Finally, we write $P(A|S, V) = [P(A_1|S_1, V) \ldots P(A_n|S_n, V)]$ for the vector of all agents' policies.

Now we are ready to write the stochastic program:

$$\max_{P(A|S,V)} \quad E_p[Q(P(A|S,V), v)] \tag{4.2.12}$$

$$\text{s.t.} \quad \sum_{A_{it}} P(A_{it}|S_{it}, V_t) = 1, \quad \forall S_{it}, V_t, t, i \tag{4.2.13}$$

$$P(A_{it}|S_{it}, V_t) \in \{0, 1\}, \quad \forall t, A_{it}, S_{it}, V_t, i \tag{4.2.14}$$

with the second-stage problem for a specific v:

$$\max_{P(A,S|v)} \quad \sum_{i=1:n} r_i(v)^{\mathrm{T}} P(A_i, S_i|v) - \sum_{j=1}^{m} f_j^{\alpha_j}(\sum_i D_{ij} P(A_i, S_i|v) - b_j) \tag{4.2.15}$$

$$\text{s.t.} \quad P(A_{i1}, S_{i1}|v) = P(A_{i1}|S_{i1}, v) P(S_{i1}|v), \quad \forall A_{i1}, S_{i1}, i \tag{4.2.16}$$

$$P(A_{it}, S_{it}|v) = P(A_{it}|S_{it}, v) \sum_{S_{i,t-1}} \sum_{A_{i,t-1}} P(S_{it}|A_{i,t-1}, S_{i,t-1}, v) P(A_{i,t-1}|S_{i,t-1}, v),$$

$$\forall A_{it}, S_{it}, i, t > 1 \tag{4.2.17}$$

$$\sum_{A_{it}, S_{it}} P(A_{it}, S_{it}|v) = 1, \quad \forall t, i \tag{4.2.18}$$

where the second-stage variables are the stage-action probabilities $P(A_{it}, S_{it}|v)$, and $r_i(v)$, $P(S_{i1}|v)$, and $P(S_{it}|A_{i,t-1}, S_{i,t-1}, v)$ are constants. In particular, probabilities $P(S_{it}|A_{i,t-1}, S_{i,t-1}, v)$ and $P(S_{i1}|v)$ are deterministic.

It is important to emphasize that the sub-vector of $P(A_i|S_i, V)$ for the policy at time t, $P(A_{it}|S_{it}, V_t)$, depends only on the information available up to time t, which is represented by the random variable V_t. In particular, consider writing the scenario-based program for (4.2.12)–(4.2.18). The value of V_t may not tell us exactly which scenario the world is in, since a scenario is defined by the value of random events through the planning horizon T. Instead, it can tell us which set of scenarios are compatible with the information given so far. Hence the scenarios in such a set are indistinguishable from each other given what is currently known, and the set is an *information set*. Since at each time step t we may not know the exact scenario we are in, but we do know which information set is *active*, our policies cannot be defined for individual scenarios, but for the active information set given the value of V_t.

Let I_ζ denote the vector of information sets scenario ζ belogs to over time such that $I_{\zeta t}$ is the information set ζ belongs to at time t. Then we can write the scenario form of (4.2.12)–(4.2.18) given the set of scenarios Z as

$$\max_{y,x} \quad \sum_{i=1:n} \sum_{\zeta \in Z} [P(\zeta) r(\zeta)^{\mathrm{T}} P(A_i, S_i|I_\zeta)] - \sum_{\zeta \in Z} \sum_{j=1}^{m} f_j^{\alpha_j}(\sum_i D_{ij} P(A_i, S_i|I_\zeta) - b_j) \tag{4.2.19}$$

$$\text{s.t.} \quad \sum_{A_{it}} P(A_{it}|S_{it}, I_{\zeta t}) = 1, \quad \forall S_{it}, t, \zeta, i \tag{4.2.20}$$

$$P(A_{it}|S_{it}, I_{\zeta t}) \in \{0, 1\}, \quad \forall A_{it}, S_{it}, t, \zeta, i \tag{4.2.21}$$

$$P(A_{i1}, S_{i1}|I_\zeta) = P(A_{i1}|S_{i1}, I_\zeta) P(S_{i1}|I_\zeta), \quad \forall A_{it}, S_{it}, \zeta, i \tag{4.2.22}$$

$$P(A_{it}, S_{it}|I_\zeta) = P(A_{it}|S_{it}, I_\zeta) \sum_{S_{i,t-1}} \sum_{A_{i,t-1}} P(S_{it}|A_{i,t-1}, S_{i,t-1}, I_\zeta) P(A_{i,t-1}|S_{i,t-1}, I_\zeta),$$

$$\forall A_{it}, S_{it}, t > 1, \zeta, i \tag{4.2.23}$$

$$\sum_{A_{it}, S_{it}} P(A_{it}, S_{it}|I_\zeta) = 1, \quad \forall t, \zeta, i. \tag{4.2.24}$$

Here we note that the shared constraints must be satisfied in every scenario, not in expectation over all scenarios. Also, the individual constraints (4.2.20)–(4.2.24) defined over ζ are redundant, since the same information set appears in multiple scenarios. We show below in Section 4.2.2 how to build an efficient form of the factored MDP that reinforces both the individual constraints on information sets and the shared constraints for each scenario.

Note that the scenario-based program here is not technically a MILP, since constraints (4.2.23) are not linear. However, since $P(S_{it}|A_{i,t-1}, S_{i,t-1}, v)$ and $P(S_{i1}|v)$ are binary, we can obtain an equivalent MILP by writing each of the constraints as an equivalent set of linear constraints. Here we leave them in the more intuitive form for readability. In contrast, constraints (4.2.17) in the second stage problems for the stochastic program are linear, since the first-stage variables $P(A|S, V)$ are considered fixed in the second stage problem.

Again, the number of scenarios in Z affects the quality of the scenario-based approximation. As we sample more scenarios, we expect convergence to the true distribution, and thus to the true optimal solution. In particular, the number of scenarios limits the complexity class of the policies, since too few samples can lead to overfitting the policies. While in this thesis we do not address

the convergence rate (i.e., the number of samples required to obtain a reasonable answer given a class of policies), we believe we can bound our convergence rate. One possibility would be to use a generalized VC-dimension argument similar to that given by Kearns et al. [46] for planning in single-agent MDPs by sampling. Such a theory is likely to suggest a large number of samples for small errors, however in our experiments we find that a rather small number of scenarios, 10 to 50, is sufficient (Section 4.3).

Now let $C_{i\zeta}$ denote the subset of (4.2.20)–(4.2.24) pertaining to agent i and scenario ζ, and write $x_{i\zeta} = [P(A_{i1}, S_{i1}|I_\zeta) \dots P(A_{iT}, S_{iT}|I_\zeta)]$ for state-action probabilities and $y_{i\zeta} = [P(A_{i1}|S_{i1}, I_\zeta) \dots P(A_{iT}|S_{iT}, I_\zeta)]$ for policies under scenario ζ. Then the Lagrangian relaxation of the scenario-based stochastic program for factored MDPs becomes:

$$\begin{aligned} \inf_{\lambda} \max_{x,y} \quad & \sum_{\zeta \in Z} [P(\zeta) \sum_{i=1}^{n} r_{i\zeta}^{\mathrm{T}} x_{i\zeta}] + \sum_{\zeta \in Z} [\sum_{j=1}^{m} [f_j^{\alpha_j *}(\lambda_{j\zeta}) - \lambda_{j\zeta} (\sum_{i=1:n} D_{ij} x_{i\zeta} - b_j)]] \\ \text{s.t.} \quad & \forall (i, \zeta), C_{i\zeta}, \end{aligned} \tag{4.2.25}$$

where $\lambda_{j\zeta}$ now corresponds to the price of resource j under scenario ζ.

The derivation presented here was based on the definition of factored MDPs introduced in Chapter 3, where state transitions for each agent depend only on its own previous state and action, and the interaction between agents is captured solely through the shared constraints. However, our framework for factored MDPs is also amenable to general factored MDPs (e.g. [32]), where the interaction between agents is captured by the joint transition probabilities $P(S_{i,t+1}|A_{1t}, \dots, A_{nt}, S_{1t}, \dots, S_{nt})$. Given such state transitions, we can turn them into shared constraints in our MILP formulation by translating them into logical formulas and introducing auxilary variables when needed (for example to enforce linearity in the final constraints). The joint transition probabilities will translate to the matrices $T(v)$ and $W(v)$ in (4.2.6), and hence the level of coupling between agents will manifest in the "blockiness" of the matrices.

Hence, our framework is quite flexible, encoding many different types of dependences that would not obviously be amenable to market-based planning; the question is for what class of problems our representation is *efficient*. For example, if the joint transition probabilities contain very tight coupling over many agents, representing them as shared constraints may take many auxilary variables, increasing the state space, and also create a large non-block-diagonal section in $T(v)$ and $W(v)$. It would be an interesting area of future work to define and quantify the level of coupling between agents specified in joint transition probabilities in relation to the efficiency of representing them in our representation.

4.2.2 Scenario-Based Factored MDPs

Given a set of scenarios Z, we must find a plan under each scenario, hence a naive implementation of the Lagrangian relaxation method for the scenario-based stochastic program requires $|Z|$ MDP subproblems for each agent. In addition, we must enforce further constraints to ensure consistency between the policies for the scenarios in the same information set. We can, however, *collapse* the

$|Z|$ MDPs into a single MDP using the information sets; a state in the combined MDP represents a tuple consisting of an original state and an information set, the set of scenarios indistinguishable from each other given the information so far. The resulting representation is more compact, and leads to a faster running time for value iteration, the subproblem solver, and thus faster global planning.

Figure 4.4 illustrates a transformation of two MDPs into a collapsed MDP. Scenarios are sampled from a sequence of Bernoulli coin flips, where the coin flip decides whether the world state *switches*. The example represents a simplified version of the supply chain management problem. We can think of "ws", the world state, as being in recession or boom, "as", the agent's state, as its inventory level, action at $t = 1$ as the number of products produced, action at $t = 2$ as the number of products sold. Each scenario shown has different world states over time, which changes the reward from selling products (see $r(s, a = 1)$ at $t = 2$). The combined MDP captures the information sets by grouping scenarios whose paths up to time t at the state are identical. Here we assume the scenarios are uniformly sampled in order to construct the transition probabilities in the combined MDP.

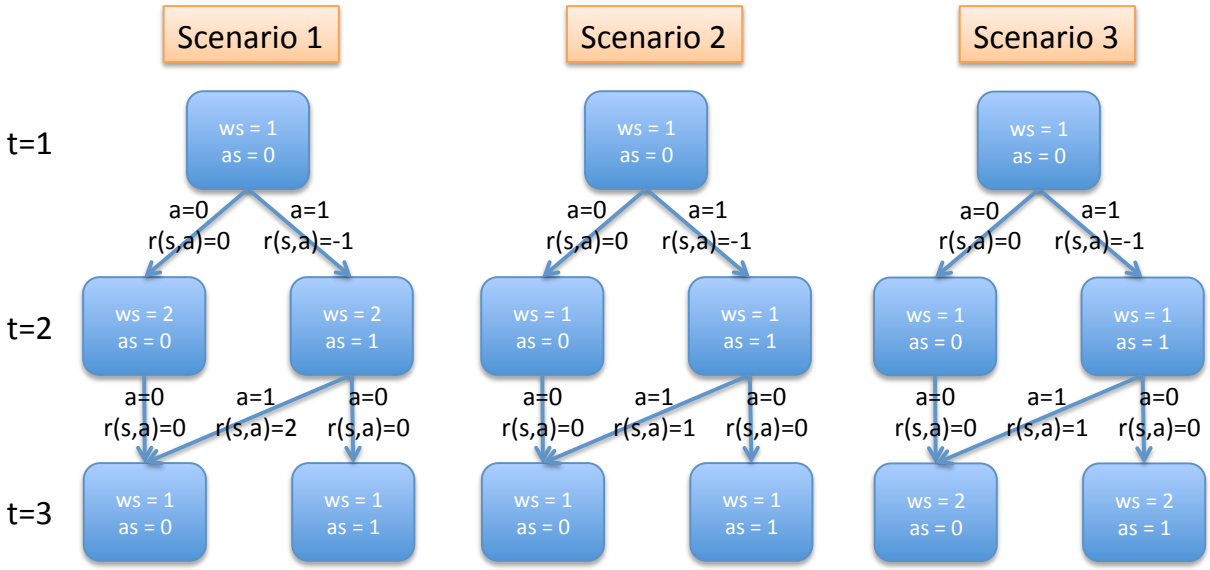
Alternatively, we can view the collapsed MDP as a cross product between the original deterministic MDP and the *scenario tree*, which is a tree MDP representing the transitions between information sets of scenarios, or *scenario groups*. Let G denote the set of scenario groups, where a scenario group g is a tuple of a set of scenarios and time $t(g)$. Also let $P(G_{t+1}|G_t)$ define the transition probability between scenario groups. In MDP notation, the scenario tree can be written as the tuple $(G, A, P(\cdot, \cdot), R(\cdot, \cdot))$, where A is a singleton set as we assume that actions do not affect the transitions in the scenario tree for notational simplicity. Also, we let the rewards $R(g, a) = 0$ since they have no role in a scenario tree.

Likewise, we can write each scenario-specific MDP as $(S_0, A_0, P_0(\cdot, \cdot), R_0(\cdot, \cdot))$, where, for simplicity we assume that all scenario-specific MDPs share the same set of states S_0, actions A_0, transition probabilities $P_0(s_{t+1}|s_t, a_t)$, and rewards $R_0(s_t, a_t)$[5]. Denoting G_t to be the random variable representing a scenario group at time t, we can write the collapsed MDP as $(S', A', P'(\cdot, \cdot), R'(\cdot, \cdot))$ where

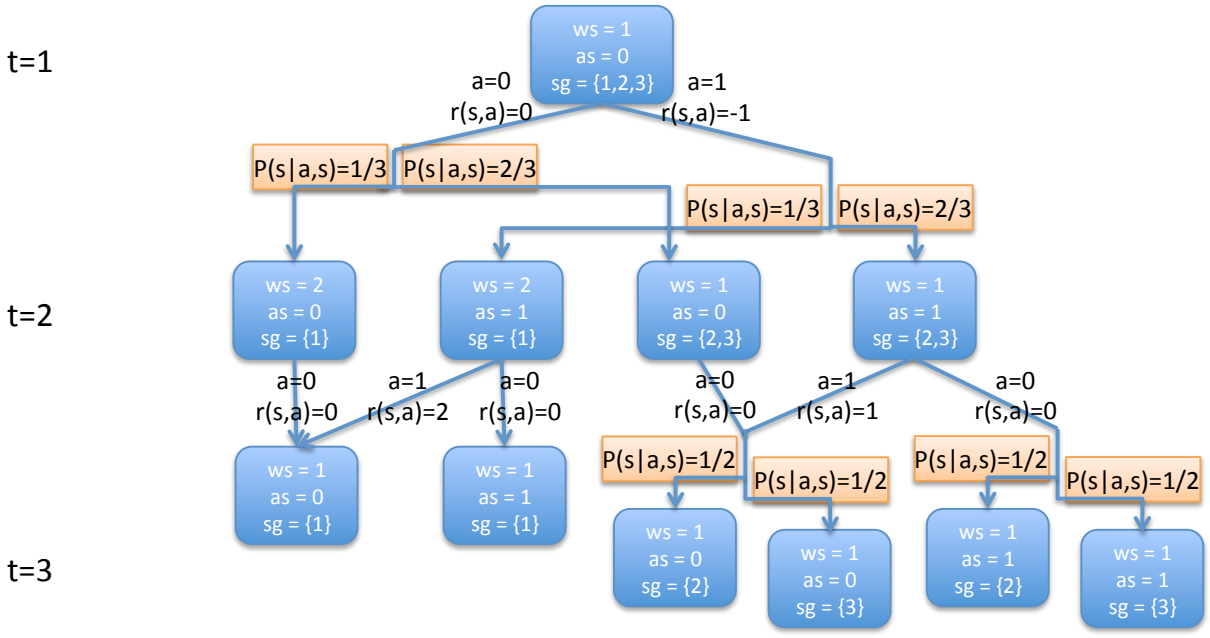
$$\begin{aligned} S' &= \{s = (g(s), s_0(s)) | g(s) \in G, s_0(s) \in S_0\}, \\ A' &= A_0, \\ P'(s_{t+1}|s_t, a_t) &= P_0(s_0(s_{t+1})|s_0(s_t), a_t) P(g(s_{t+1})|g(s_t)), \\ R'(s, a) &= R_0(s_0(s), a). \end{aligned}$$

As seen in the expression for the Lagrangian relaxation (4.2.25), constraints in the stochastic program are simply the original constraints replicated for each scenario. Hence, to compute the resource usage under each scenario, $\sum_i D_{ij} x_{i\zeta}$, we simply need to read off $x_{i\zeta} = P_\zeta(A_{it}, S_{it})$ from the collapsed MDP.

[5]For our transformation method, at least the MDPs for the scenarios in the same scenario group at time t must share the same states, actions, transition probabilities, and rewards at time t.



(a) Original MDPs



(b) Combined MDP

Figure 4.4: Transformation of (a) separate, scenario-specific MDPs into (b) one MDP. "ws" refers to the world state (determined by the coin flip at each time step), "as" the agent's state (e.g. the inventory level), and in the combined MDP "sg" denotes the set of scenarios the world could belong to given the state.

Thus the resulting problem is another, larger, factored MDP, which may be solved via the same optimization algorithms as discussed in Section 3.3.

Execution

Given a solution from optimizing stochastic factored MDP, we can execute the plan by sampling the policies as described for the deterministic program in Section 3.2.3. Additionally, the transition from a state given the sampled action is determined by the world state for the next time step. However, as the set of scenarios in the stochastic MDP may not cover the entire space of scenarios, we may run into a state with no valid transitions (i.e., we run out of scenarios). In such cases, we transition to a state matching the world and agent state but associated with another set of scenarios. Even though the new state's past (sequence of world states) may not match that of the original path, it is a good approximation, since the states are Markovian; if the sampled scenarios represented the distribution of world states exactly, over infinite time steps, every state with the same world state and agent state values would in fact be equivalent, regardless of the time step. However, while this heuristic seems to work well, it would be interesting to explore different possibilities for dealing with out-of-sample execution paths. For example, "featurizing" the scenario space so that the policies depend only on some features of the scenario could allow us to completely sample the feature space and never encounter an unplanned-for outcome, which would in fact eliminate the need for a heuristic.

4.3 Experiments

We demonstrate the improvement in efficiency achieved by stochastic programming over working with deterministic plans (such as shown in Section 4.1.1) on a set of simple supply chains. In particular, we focus on linear supply chains to study the impact of stochastic programming on the bullwhip effect, since the observation of bullwhip effect is independent of the width of the supply chain. In each case we assume two world states: recession and boom. Thus, each scenario is a sequence of world states, sampled assuming that the world state may change with a small probability at each time step (we use 0.1).

We employ causal entropy augmentation (Section 3.4.1) for faster optimization using L-BFGS[6]. As a solution to the Lagrangian relaxation, even with near-zero hinge loss value, only satisfies the original constraints of the MIP *in expectation*, we built a simple reactive planner to be used at execution time to ensure that the sell and buy amounts match at every edge in the supply chain. Given sampled plans per the execution method in Section 4.2.1, the reactive planner simply tries to satisfy the demand at each edge if possible (i.e., sell as much as possible). The same reactive planning is used at execution of plans from non-stochastic planning (such as those used to create the bullwhip example in Section 4.1.1).

[6]While the objective is not twice differentiable due to the hinge loss and thus L-BFGS may not theoretically converge on the problem, in our experiments convergence was achieved with little tweaking of the parameters (the hinge loss slopes α_j and the causal entropy weight β). Here we use $\beta = 100$ and α_j values in the range of 200–1000.

Figures 4.2–4.4 show the improvement percentage in total reward (revenue − production cost) of stochastic planners (using 10, 20, and 50 scenarios) over the deterministic planner, for supply chains of lengths ranging from 2 to 5.[7] Tables 4.2 and 4.3 give the results on the "training set" of scenarios, i.e., the scenarios that appear in the sampled set for each stochastic planner. They serve as a sanity check; we expect the stochastic planner to generally outperform the deterministic planner since the stochastic planner plans exactly for the set of scenarios given. Table 4.3 shows numbers for scenarios where the world state changes at least once. As expected, the stochastic planners outperform the deterministic planner. Table 4.2 on the other hand gives the results on the scenario where the world state stays in recession. The deterministic planner should give the optimal plan for such a case; however, since plans sometimes may not satisfy the shared constraints, it is possible for a stochastic planner to do better than the deterministic planner, once the reactive planner has corrected for the broken constraints. In general, we see that all planners perform comparably on the unchanging-economy scenario, suggesting that the stochastic planners lose little on optimalizing for the unchanging scenario (the best case for the deterministic planner) while improving performance on scenarios containing changes in world state.

Table 4.4 shows the rewards on the "test set", i.e., scenarios sampled according to the distirbution, excluding the unchanging-economy scenario. Here we evaluated each planner on 10,000 scenarios. The results again show that the stochastic planners outperform the deterministic planner on changing scenarios. It is also worth noting that the stochastic planners give little improvement upon the deterministic planner on the 2-agent chain where we expect the bullwhip effect to be least problematic, which suggests that the stochastic planners achieve their gains by ameliorating the bullwhip effect.

4.4 Conclusion

Planning under uncertainty often presents to be a non-trivial challenge to auction-based distributed planning frameworks. In this chapter, we introduced a natural extension to our distributed multi-agent planning framework using stochastic programming. In particular, we provided a method for compactly representing uncertainty in factored MDPs while maintaining the structure that allows us to employ the same set of optimization methods used for non-scenario-based factored MDPs. Using supply chain management as an example domain, we demonstrated that planning under uncertainty via our method can in fact provide better plans in a changing environment, while sacrificing little in solution quality for when the environment stays static. While our experiments cover a limited set of supply chains, we showed that an interesting phenomenon in economics, the bullwhip effect, can be ameliorated by using a relatively small set of scenarios. In the future, it would be beneficial to investigate how we can scale our method to larger problems and correspondingly larger sets of scenarios. It would also be interesting to provide statistical guarantees on the solution quality in terms of the number of samples used. Finally, we believe incorporating communication at run-time may help agents adapt more quickly and efficiently in a changing environment.

[7] Sign test is significant for improvement of the stochastic planner over the deterministic planner for all tables.

#agents	10 scenarios	20 scenarios	50 scenarios
2	20	20	1
3	20	1	1
4	20	1	1
5	20	1	1

Table 4.1: Number of scenarios trees trained and tested.

#agents	10 scenarios	20 scenarios	50 scenarios
2	0.42%	0.67%	1.38%
3	-0.18%	-0.01%	0.17%
4	1.71%	0.74%	0.67%
5	1.15%	1.13%	1.09%

Table 4.2: Improvement of the stochastic planners with different size scenario sets over the deterministic planner, on the scenario with the economy unchanging from recession. The results are averaged over the samples in "training" and "test" sets, and look very similar to those for each set.

#agents	10 scenarios	20 scenarios	50 scenarios
2	0.04%	-0.37%	3.65%
3	2.95%	6.21%	2.41%
4	4.11%	6.85%	4.78%
5	4.66%	4.97%	6.84%

Table 4.3: Improvement of the stochastic planners with different size scenario sets over the deterministic planner, on the "training set" of scenarios with changing economies.

#agents	10 scenarios	20 scenarios	50 scenarios
2	0.31%	-0.12%	10.53%
3	2.66%	6.59%	2.54%
4	7.27%	6.43%	4.22%
5	7.16%	4.89%	6.58%

Table 4.4: Improvement of the stochastic planners with different size scenario sets over the deterministic planner, on the "test set" of scenarios with changing economies.

Chapter 5

Related Work

5.1 Market-based Distributed Multi-Agent Planning

Various forms of market-based distributed planning have been studied for collaborative domains. The contract net protocol [76] gave an early general market-based framework for formulating collaborative planning problems. More specific work has subsequently followed, in particular by Dias et al. [21], Gerkey [27], and Koenig et al. [50] for task allocation. A common feature in market-based algorithms is the trusted auctioneer that assigns tasks based on agents' bids, which may be used alone (e.g. [48], [49], [81]), or in conjunction with agent-to-agent negotiation to improve the solution quality after the central initial allocation (e.g. [95], [20], [53]).

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 those by Lagoudakis et al. [52] for sequential single-task auctions, by Koenig et al. [49] for sequential single-task auctions with regret clearing, by Zheng et al. [96] for an improved version of sequential single-task auctions using hill climbing, and by Gerkey [27] for one-task-per-robot domains. Most methods 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 the allocations of some tasks to depend on others'. Combinatorial auctions [18] allow for such constraints by letting agents bid on bundles of resources, and have been studied in the collaborative setting for robot exploration [10], role assignment [38], and task allocation [57], among others. However, both the bidders' problem (what to bid) and the auctioneer's problem (assigning items to bidders) become intractable in combinatorial auctions. Hence, actual implementations often resort to heuristics (e.g., handcrafted bidding rules), and

therefore to suboptimal solutions. Therefore, for settings without such coupled constraints, fast single-item auction algorithms may be preferable, especially if real-time performance is expected.

Like combinatorial auctions, our AMDMA algorithms naturally handle constraints coupling resources. However, unlike most work using combinatorial auction in collaborative domains, our algorithms use 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 [62], which have been mainly studied for non-collaborative settings.

We note that many auction solutions (e.g. combinatorial auctions) are mainly studied in mechanism design [24], even if they have been used in collaborative settings. The goal of mechanism design is to provide incentives to potentially self-interested agents in order to achieve a global objective (e.g. social optimum). In contrast, we, like most market-based planners, assume collaborative agents, and focus on obtaining the optimal solution without explicitly trying to provide incentives for the agents to follow the solution; the market abstraction provides a convenient way to formulate distributed multi-agent planning in the presence of shared resources. Hence while incentive-providing auctions are more robust to different types of agents, market-based planners are able to achieve potentially better global solutions in collaborative settings.

We again stress that our methods automatically provide resource prices without domain knowledge, and potentially to optimality, whereas most studies under either auction paradigm engineer markets and resources specifically for each problem domain, and often without theoretical guarantees. AMDMAs may be combined with problem-specific approaches to yield additional resources that are more specialized but principled and easier to design. In Price-and-Cut, 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. [74], who use 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 [40], 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 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.

5.2 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 that of Guestrin and Gordon [33] for hierarchically factored Markov decision processes (MDPs), of Bererton et al. [9] for a class of loosely coupled MDPs, and of Calliess et al. [14] 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).

General frameworks for Dantzig-Wolfe decomposition for mixed integer programming have been explored, particularly in the operations research community (e.g., [82, 84]). 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 [6, 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 [83] 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 Price-and-Cut 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 [64].

Thus our main contribution with respect to distributed MIP optimization 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 [56, 92]. 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

[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.

problems (DisCSP) [94] allow hard constraints but do not optimize a cost function. Hence both DCOP and DisCSP solve simpler problems than general MILPs.

5.3 Lagrangian Relaxation for Multi-Agent Planning

Lagrangian relaxation is used heavily in large-scale optimization problems, and much work has been focused on improving the quality of subgradient descent for faster convergence [35, 43]. In terms of applying Lagrangian relaxation to multi-agent planning, our work on decomposing MILPs (Chapter 2) can be considered a form of Lagrangian relaxation. The idea of leveraging bounded influence of each agent on resources to obtain guarantees for Lagrangian relaxation in multi-agent planning was introduced by Gordon et al. [31]. They give an approximation guarantee for the subgradient Lagrangian relaxation algorithm, upon which we base our near-optimality guarantees for the optimization methods we use (Section 3.3.3).

Guestrin and Gordon [33] give a distributed planning algorithm for factored MDPs [13], a closely related formulation to our MDP framework in Chapters 3 & 4. Whereas we apply Lagrangian relaxation to a scenario-based approximation of the joint MDP to obtain a factored problem, they employ an approximation to the Bellman LP of a factored MDP using a basis and subsequently apply Lagrangian relaxation to obtain a similarly factored problem. Since their framework is based on MDPs, it is possible to plan under uncertainty about the world by incorporating a scenario tree into the transition probabilities as we do in Chapter 4. However, their work assumes infinitely divisible resources and thus is strictly subsumed by our formulation.

The trick of augmenting the objective function with a smooth penalty is often used with dual decomposition, and has been studied at least since Hestenes [36, 37] and Powell [65], and subsequently in a variety of convex optimization settings [59, 60, 70, 71]. Such methods have seen a recent surge of interest in machine learning and optimization communities [16, 80, 86], and in some cases, optimizing the strength of the strongly convex penalty has shown to be an effective extension (e.g. [16]). Though not always, augmentation is often used to allow optimization with accelerated gradient methods [16, 67, 68], as in our case (Chapter 3).

5.4 Multi-Agent Planning Under Uncertainty

Planning under uncertainty in market-based distributed multi-agent planning frameworks is usually handled via tight coordination between agents at execution (e.g., [42]), where little planning is done with respect to future events[2]. Such run-time coordination could be beneficial in our framework, which currently only deals with coordination and communication during the deliberate planning stage. Combining the two (pre and during) communication approaches would be an interesting area of future work.

[2] We note, however, that there is a larger body of work on planning under uncertainty in *competitive* domains. Parkes et al. [63] give a good overview of recent work on incorporating uncertainty in mechanism design.

Stochastic programming and scenario-based optimization [12] have been applied to many planning problems including supply chain management [5, 73, 79]. While supply chain management includes a wide range of problems from facility location to reverse (return) network design, many consider problems we address such as finding optimal contracts between agents and maintaining inventory levels under demand uncertainty (see [55, 77] for reviews). Our work differs from existing work in two main areas. First, while we get the contract prices "for free" from the resource prices, in existing work they are explicitly optimized as variables in the problem, adding to the design and optimization effort (see e.g. [4]). Second, stochastic programs in most domains are often solved using Benders decomposition (e.g. [75]), which decomposes the problem into subproblems; however, each subproblem corresponds to a scenario, in comparison to our work where subproblems correspond to agents[3]. Hence, while both decomposition methods are amenable to distributed optimization, ours is more natural in a multi-agent setting, as the computation is assigned to the entity who "owns" the subproblem, which results in far less communication.

There has also been interest in supply chain management from the multi-agent community [88], which puts a greater focus on distributed coordination. Most work focuses on creating incentives for agents to coordinate, from optimizing contracts by coordinating at local levels [1] to studying combinatorial auctions for negotiating contracts in competitive settings [90]. However, similarly to general market-based planning frameworks, uncertainty is rarely a main concern, and thus is usually handled by coordination at execution, without deliberate planning.

Scenario-based planning under uncertainty has also been studied in machine learning, in particular in the context of reinforcement learning. One notable example is the PEGASUS algorithm proposed by Ng et al. [93], which uses sampling (i.e., scenarios) to efficiently plan in partially observable Markov decision processes (POMDPs). A particularly relevant line of research stems from work by Kearns et al. on planning using scenario trees for single-agent MDPs [45] and POMDPs [46]. They provide efficient planning algorithms for (PO)MDPs that simulate execution paths from the (PO)MDP, i.e., create scenario trees, and optimize on those samples instead of the entire (PO)MDP. Our work uses a similar sampling method for handling uncertainty in MDPs, while explicitly dealing with coordination in multiagent settings; while such problems can be formulated as a centralized MDP, our use of factorization and decomposition gives us a naturally distributed algorithm with smaller problems that are thus more efficient to optimize.

A general framework for multiagent planning under uncertainty is decentralized POMDPs, which extends POMDPs to cooperative multiagent settings. Many variants have been proposed to increase efficiency by approximation or by imposing further assumptions [8, 28, 34, 61, 91]. For example, Spanan et al. [78] propose a method that plans under uncertainty by explicitly modeling communication as part of the optimization problem, where communication is used to decrease both uncertainty in other agents' actions as well as their observations which may be unknown to each other. Our work automatically incorporates communication of each agent's intent through resource prices, and focuses on optimally planning under uncertainty in observations available globally to all agents, which allows us to use scenario trees for efficient planning. Whereas Spannan et al. sequentially optimize each agent with other agents' plans fixed, which can often

[3]Our methods are closely related to Dantzig-Wolfe decomposition, which is the dual of Bender's decomposition.

lead to suboptimality, we are able to jointly optimize all agents using our automatic pricing scheme.

5.5 Supply Chain Management

Many aspects of supply chain management, ranging from network design to pricing and production optimization, have been studied in operations research [55, 79]. The most relevant line of work casts the pricing and production problem as a MILP or a stochastic program [4, 5, 73, 77]; however, most provide a centralized solution, and the use of decomposition methods are limited to the usual decomposition of stochastic programs by scenarios (e.g. [75]), which is not amenable to distributed multi-agent planning.

Supply chain management has also been studied from a multi-agent perspective. There exists a long line of work in using multi-agent architectures to define the interactions in a supply chain, whether with automated or human agents (e.g. [51]). Much effort has been made in designing different types of contracts between agents to obtain an efficient supply chain, with varying levels of focus on the optimization problem at each agent. Of particular interest are initiatives to study supply chain management using ideas from mechanism design and auctions. Notably, Walsh and Wellman argued that the multi-agent community is in a particularly good position to solve supply chain management problems with automated agents [87]. Much of the follow up work has involved the Supply Chain Trading Agent Competition (TAC SCM), which formulates the supply chain management as a game between self-interested agents competing for customers and supplies [3, 47, 89], but similar ideas have also been explored in fields ranging from management science [15] to economics [41]. While many multi-agent based settings do not explicitly consider uncertainty, mechanism design research is an exception; an interesting case study is that of Chick et al. [17] on influenza vaccine supply chain coordination that provides a cost-sharing contract that both incentivizes the agents as well as achieves global optimization, under uncertainty in the efficacy of the vaccines.

Chapter 6

Conclusion and Future Work

In this thesis, we presented a general, principled framework for the design and optimization of coordination in cooperative market-based multi-agent planning with shared constraints. We formalized multi-agent planning as mixed integer programs, and gave decomposition-based distributed planning algorithms with optimality guarantees. The formalization allowed us to incoroporate complex shared constraints, as well as planning under uncertainty, both of which are not well-treated in market-based distributed planning literature. Our Automatically-pricing Market-based Distributed Multi-agent Algorithms (AMDMAs) price resources automatically and optimally given only the overall objective and shared constraints, removing the often time-consuming efforts used in designing a market. AMDMAs allow each agent to solve its local planning problem while only communicating its intentions regarding shared resources to other agents, and the communication protocol is defined automatically along with resource pricing.

We studied three instances of AMDMA, addressing planning by exactly solving MILPs, faster planning via relaxation, and planning under uncertainty:

First, in Chapter 2 we gave a novel distributed MILP optimization algorithm, price-and-cut, based on Dantzig-Wolfe decomposition and Gomory cuts, with convergence and optimality guarantees. Price-and-cut can be seen as running Dantzig-Wolfe decomposition on the LP relaxation of the MILP, while adding cuts generated by the Gomory cutting-plane method to further and further constrain the LP until we find a (mixed) integral solution. The cuts correspond to *derivative resources* that represent bundles of existing resources. We showed price-and-cut's effectiveness in distributed planning using two problem domains: 1. distributed SAT problems as an abstraction for planning problems, and 2. an unmanned aerial vehicle task allocation and path planning task. In particular, we demonstrated that price-and-cut can improve the running time substantially for large problems, in comparison to CPLEX, an industry-standard MILP solver, not only under distributed execution, but even when it is run sequentially.

Next, in Chapter 3, we extended our framework by introducing another decomposition method, Lagrangian relaxation, in order to handle larger problems. Programs resulting from Langrangian relaxation can be optimized much more efficiently using convex optimization methods. While Lagrangian relaxation does not in general come with solution quality guarantees, for an impor-

tant set of problems, where many agents must share the resources, our Lagrangian relaxation method is shown to give near-optimal solutions. The method comprises three parts. First, we introduced Langrangian relaxation, which dualizes the shared constraints. Lagrangian relaxation decomposes the original MILP into subproblems, much like price-and-cut. However, it gives us a convex optimization problem and introduces an approximation, as it in effect relaxes the shared constraints to be satisfied in expectation. Second, we gave two methods, subgradient projection and an accelerated gradient method, FISTA, to optimize the Lagrangian relaxation, including an augmentation scheme to allow the use of FISTA on linear programs. In particular, we proposed the use of maximum causal entropy penalty for MDPs, which enables us to retain fast subproblem solvers. And finally, we gave a simple but effective rounding scheme for obtaining a feasible solution to the original MILP given a solution to the Lagrangian relaxation. For problems with a large number of agents where each agent's influence on resource consumption is limited, our algorithms combined with the rounding scheme are shown to given near-optimal solutions. On a difficult traffic management problem, we showed that the use of the accelerated gradient method dramatically improves the convergence of optimization, at little loss to the quality of the solution, even though the use of augmentation leads to optimizing a perturbed objective function.

Finally, in Chapter 4, we examined distributed planning under uncertainty by further extending our Lagrangian relaxation framework using multi-stage stochastic programming. In particular, we gave an efficient construction of factored MDPs under uncertainty based on sampling, which is in fact a generalization of the construction of the factored MDPs in the deterministic setting (Chapter 3). We studied the supply chain management problem as an example domain of heavy dependencies between agents and non-trivial uncertainty, and demonstrated that our stochastic planning algorithm can ameliorate the bullwhip effect, a well-known economic phenomenon of inefficiency under lack of planning under uncertainty.

6.1 Future Work

While we have laid the foundation for general, principled distributed multi-agent planning with shared constraints, many interesting avenues of future work remain.

While AMDMAs are distributed algorithms, the master program is a centralized sequential computation, whether in implementation it is performed on one server or on multiple agents simultaneously. Distributing the master program among the agents would add further robustness to failure and help speed up computation. Our experiments have demonstrated the promise of the proposed algorithms, but work remains to make them more scalable for larger, more complex problems and to evaluate them in additional domains.

Our work has focused on deliberate planning before execution. However, exploring run-time coordination via communication would be an interesting area of future work. In principle, run-time communication merely requires giving the agents communication actions and optimizing their policies, but in practice, many complications are likely to arise—e.g., how expressive a communication language can we give the agents while still arriving at a tractable planning problem

in the end? In addition, it would be interesting to investigate whether we could optimize both the deliberate planner and the reactive (execution-time) planner jointly, especially knowing what the deficiencies of the deliberate planner are, such as its relaxed nature (e.g. Lagrangian relaxation) or its limited coverage of uncertainty from using a small number of samples.

With respect to planning under uncertainty, we believe an immediately impactful work would be to more precisely define the relationship between different representations for planning uncer uncertainty. For example, we gave a formulation of factored MDPs with shared constraints as a stochastic program in Section 4.2.1. Of particular relevance is the precise relationship between factored MDPs with shared transition functions (e.g. [32]) and factored MDPs with shared constraints (Section 3.2.4). We believe the former can be translated to the latter in order to take advantage of our distributed algorithm. However, understanding the precise relationship as well as each formulation's representational power will allow us to analyze their efficiency and prescribe a "formula" for under what settings each formulation is more practical. Such a characterization will help researchers and practitioners from different fields including operations research, multi-agent planning, and controls, to better formulate, construct and solve multi-agent problems.

Furthermore, it would be very interesting to study how applying our uncertainty handling trick to price-and-cut, our exact MILP solver, would interact with derivative resources, which we did not study in Chapter 4. Also, it may be beneficial to better understand the results of scenario-based handling. For example, uncovering any potential patterns in the resulting plans with respect to their corresponding scenarios may help us plan even more efficiently.

Finally, it would be very interesting to extend our work to the *non-collaborative* setting where agents may possess interests conflicting with the global objective of the system or the socially optimal joint plans. Such situations arise in many domains; for example, in supply chain management, it is extremely natural for each agent to try to maximize its profit without regard to others' profits. In fact, an agent may be directly competing with another agent for sales. Also, even when agents are cooperative, they may not want to reveal everything about their intentions.

Planning for collaborative agents can be seen as a global optimization problem, hence the solutions we have described so far are general-purpose distributed solvers for MILPs. However, when planning for self-interested 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.

A possible approach would be to exploit duality and no-regret learning in an analogous fashion to [14], who addressed the simpler problem of divisible resources, leading to a convex optimization problem. As in [14], one could 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. The hope would be to establish similar guarantees to [14], which gives asymptotic guarantees on the cost of the average plan (Theorem 3.4, [14]) and its convergence to a Nash equilibrium (Theorem 4.1, [14]).

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, one must ensure that agents are given an incentive to only generate valid and useful cuts. Also, for completeness, theoretically one needs 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, it may be possible 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.

Bibliography

[1] HyungJun Ahn and SungJoo Park. Modeling of a multi-agent system for coordination of supply chains with complexity and uncertainty. In Jaeho Lee and Mike Barley, editors, *Intelligent Agents and Multi-Agent Systems*, volume 2891 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-20460-2. doi: 10.1007/978-3-540-39896-7_2. URL http://dx.doi.org/10.1007/978-3-540-39896-7_2. 5.4

[2] 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

[3] J. Andrews, M. Benisch, A. Sardinha, and N. Sadeh. Using information gain to analyze and fine tune the performance of supply chain trading agents. In *Agent-Mediated Electronic Commerce: Designing Trading Agents and Mechanisms, LNBIP series*, volume 13, pages 182–199. Springer, 2008. 5.5

[4] J. Ashayeri, A.J. Westerhof, and P.H.E.L. van Alst. Application of mixed integer programming to a large-scale logistics problem. *International Journal of Production Economics*, 36(2):133 – 152, 1994. ISSN 0925-5273. doi: 10.1016/0925-5273(94)90020-5. URL http://www.sciencedirect.com/science/article/pii/0925527394900205. 5.4, 5.5

[5] A. Azaron, K.N. Brown, S.A. Tarim, and M. Modarres. A multi-objective stochastic programming approach for supply chain design considering risk. *International Journal of Production Economics*, 116(1):129 – 138, 2008. ISSN 0925-5273. doi: 10.1016/j.ijpe.2008.08.002. URL http://www.sciencedirect.com/science/article/pii/S0925527308002375. 5.4, 5.5

[6] 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. 5.2

[7] Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. on Imaging Sciences*, 2(1):183–202, 2009. doi: DOI:10.1137/080716542. URL http://dx.doi.org/10.1137/080716542. 3.3.2

[8] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman. Solving transition independent decentralized markov decision processes. *Journal of Artificial Intelligence Research*, 22:423–455, 2004. 5.4

[9] C. Bererton, G. Gordon, S. Thrun, and P. Khosla. Auction mechanism design for multi-robot coordination. In *Advances in Neural Information Processing Systems*, 2003. 5.2

[10] 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. 5.1

[11] Dimitris Bertsimas and John N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, Belmont, MA, 1997. 2, 2.1, 2.2.1, 2.2.3

[12] John R. Birge and Franois V. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, New York, NY, 1997. 4, 5.4

[13] C. Boutilier, T. Dean, and S. Hanks. Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999. 5.3

[14] 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. 5.2, 6.1

[15] R. R. Chen, R. O. Roundy, R. Q. Zhang, and G. Janakiraman. Efcient auction mechanisms for supply chain procurement. *Management Science*, 51(3):467, 2005. 5.5

[16] Xi Chen, Qihang Lin, Seyoung Kim, Jaime Carbonell, and Eric P. Xing. Smoothing proximal gradient method for general structured sparse learning. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI)*, 2011. 3.4, 5.3

[17] Stephen E. Chick, Hamed Mamani, and David Simchi-Levi. Supply chain coordination and influenza vaccination. *Operations Research*, 56(6):1493–1506, 2008. 5.5

[18] Peter Cramton, Yoav Shoham, and Richard Steinberg, editors. *Combinatorial Auctions*. MIT Press, Cambridge, MA, 2005. 5.1

[19] George B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963. 2.2.1

[20] M. Dias and A. Stentz. Opportunistic optimization for market-based multirobot control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, 2002. 5.1

[21] M. Bernardine Dias. *TraderBots: A New Paradigm for Robust and Efficient Multirobot Coordination in Dynamic Environments*. PhD thesis, CMU, 2004. 1, 5.1

[22] 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. 1

[23] Niklas Een and Niklas Sorensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation 2*, 2006. 4

[24] E. Ephrati and J. S. Rosenschein. The clarke tax as a consensus mechanism among automated agents. In *In Proceedings of the 9th National Conference on Artificial Intelligence (AAAI-91)*, pages 173–178. AAAI Press, 1991. 5.1

[25] M. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981. 3

[26] Jay Wright Forrester. *Industrial Dynamics*. MIT Press, Cambridge, MA, 1961. 4.1.1

[27] Brian Gerkey. *On Multi-Robot Task Allocation*. PhD thesis, University of Southern California, 2003. 1, 5.1

[28] C. Goldman and S. Zilberstein. Optimizing information exchange in cooperative multi-agent systems. In *Proceedings of the Second International Conference on Autonomous Agents and Multi Agent Systems (AAMAS-03)*, 2003. 5.4

[29] Carla Gomes, Heny Kautz, Ashish Sabharwal, and Bart Selman. Satisfiability solvers. In *Handbook of Knowledge Representation*, chapter 2. Elsevier B.V., 2007. 2.5.1

[30] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc.*, 64(5):275–278, 1958. 2

[31] G. J. Gordon, P. Varakantham, W. Yeoh, H. C. Lau, A. S. Aravamudhan, and S. Cheng. Lagrangian relaxation for large-scale multi-agent planning. In *IAT-12*, 2012. 3, 3.2.1, 3.3.1, 3.3.3, 3.3.3, 5.3

[32] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Articial Intelligence Research*, 19, 2003. 4.2.1, 6.1

[33] Carlos Guestrin and Geoffrey J. Gordon. Distributed planning in hierarchical factorered mdps. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, 2002. 5.2, 5.3

[34] A. Guo and V. Lesser. Planning for weakly-coupled partially observable stochastic games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005. 5.4

[35] M. Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6(1):62–88, 1974. 5.3

[36] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:302–320, 1969. 5.3

[37] M. R. Hestenes. Multiplier and gradient methods. In L. A. Zadeh, L. W. Neustadt, and A. V. Balakrishnan, editors, *Computing Methods in Optimization Problems*. Academic Press, 1969. 5.3

[38] 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. 5.1

[39] ILOG. Ibm ilog cplex optimizer, 2010. `http://www.ilog.com/products/cplex/`. 2, 2.5.1

[40] 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. 5.1

[41] Toshiya Kaihara. Multi-agent based supply chain modelling with dynamic environment. *International Journal of Production Economics*, 85(2):263 – 269, 2003. ISSN 0925-5273. doi: 10.1016/S0925-5273(03)00114-2. URL `http://www.sciencedirect.com/`

science/article/pii/S0925527303001142. ¡ce:title¿Supply Chain Management¡/ce:title¿. 5.5

[42] N. Kalra, D. Ferguson, and A. Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 1170 – 1177, april 2005. doi: 10.1109/ROBOT.2005.1570274. 5.4

[43] C. Kaskavelis and M. Caramanis. Efficient lagrangian relaxation algorithms for industry size job-shop scheduling problems. *IIE Transactions*, 30(11):1085–1097, 1998. 5.3

[44] 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. 2.5.1

[45] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999. 4.2, 5.4

[46] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. Approximate planning in large pomdps via reusable trajectories. In *Neural Information Processing Systems (NIPS)*, 2000. 4.2, 4.2.1, 5.4

[47] Philipp W. Keller, Felix-Olivier Duguay, and Doina Precup. Redagent-2003: An autonomous market-based supply-chain management agent. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 3*, AAMAS '04, pages 1182–1189, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 1-58113-864-4. doi: 10.1109/AAMAS.2004.225. URL http://dx.doi.org/10.1109/AAMAS.2004.225. 5.5

[48] 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. 5.1

[49] 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. 5.1

[50] S. Koenig, P. Keskinocak, and C. Tovey. Progress on agent coordination with cooperative auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2010. 5.1

[51] Ohbyung Kwon, Ghi Paul Im, and Kun Chang Lee. Mace-scm: A multi-agent and case-based reasoning collaboration mechanism for supply chain management under supply and demand uncertainties. *Expert Systems with Applications*, 33(3):690 – 705, 2007. ISSN 0957-4174. doi: 10.1016/j.eswa.2006.06.015. URL http://www.sciencedirect.com/science/article/pii/S0957417406001941. 5.5

[52] 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. 5.1

[53] 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. 5.1

[54] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989. 3.3.2

[55] M.T. Melo, S. Nickel, and F. Saldanha da Gama. Facility location and supply chain management a review. *European Journal of Operational Research*, 196(2):401 – 412, 2009. ISSN 0377-2217. doi: 10.1016/j.ejor.2008.05.007. URL `http://www.sciencedirect.com/science/article/pii/S0377221708004104`. 5.4, 5.5

[56] Pragnesh Jay Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. Adopt: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal (AIJ)*, 2005. 5.2

[57] 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. 5.1

[58] Y. Nesterov. *Introductory Lectures on Convech Optimization: A Basic Course*. Springer Verlag, New York, NY, 2003. 3.2.1

[59] Yurii Nesterov. Smooth minimization of non-smooth functions. *Math. Program.*, 103(1): 127–152, 2005. 5.3

[60] Yurii Nesterov. Smoothing technique and its applications in semidefinite optimization. *Mathematical Programming*, 110(2):245–259, 2007. 5.3

[61] S. Paquet, L. Tobin, and B. Chaib-draa. An online pomdp algorithm for complex multiagent environments. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, 2005. 5.4

[62] David C. Parkes. *Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency*. PhD thesis, University of Pennsylvania, 2001. 5.1

[63] David C. Parkes, Ruggiero Cavallo, Florin Constantin, and Satinder Singh. Dynamic incentive mechanisms, 2010. 2

[64] 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. 5.2

[65] M. J. D. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*. Academic Press, 1969. 5.3

[66] 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. 2

[67] Z. Qin and D. Goldfarb. Structured sparsity via alternating direction methods. *Journal of Machine Learning Research*, 13:1373–1406, 2012. 5.3

[68] S. Ramani and J.A. Fessler. Parallel mr image reconstruction using augmented lagrangian

methods. *Medical Imaging, IEEE Transactions on*, 30(3):694 –706, march 2011. ISSN 0278-0062. doi: 10.1109/TMI.2010.2093536. 5.3

[69] 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. 2, 2.5.2

[70] R. T. Rockafellar. Augmented lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research*, 1(2), 1976. 5.3

[71] Andrzej Ruszczynski. On convergence of an augmented lagrangian decomposition method for sparse convex optimization. *Mathematics of Operations Research*, 20(3), 1995. 5.3

[72] Tuomas Sandholm, Andrew Gilpin, and Vincent Conitzer. Mixed-integer programming methods for finding Nash equilibria. In *AAAI-05*, 2005. 2

[73] R.C. Schaub and Stanford University. *Stochastic Programming Solutions to Supply Chain Management*. Stanford University, 2009. ISBN 9781109077438. URL `http://books.google.com/books?id=iOZFB3QMnV4C`. 5.4, 5.5

[74] 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. 5.1

[75] P. Schutz, A. Tomasgard, and S. Ahmed. Supply chain design under uncertainty using sample average approximation and dual decomposition. *European Journal of Operational Research*, 199:409–419, 2009. 5.4, 5.5

[76] R.G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *Computers, IEEE Transactions on*, C-29(12):1104 –1113, dec. 1980. ISSN 0018-9340. doi: 10.1109/TC.1980.1675516. 5.1

[77] ManMohan S. Sodhi and Christopher S. Tang. Modeling supply-chain planning under demand uncertainty using stochastic programming: A survey motivated by assetliability management. *International Journal of Production Economics*, 121(2):728 – 738, 2009. ISSN 0925-5273. doi: 10.1016/j.ijpe.2009.02.009. URL `http://www.sciencedirect.com/science/article/pii/S092552730900070X`. 5.4, 5.5

[78] Matthijs T.J. Spaan, Geoffrey J. Gordon, and Nikos Vlassis. Decentralized planning under uncertainty for teams of communicating agents. In *Proceedings of the Fifth International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 06)*, 2006. 5.4

[79] Douglas J. Thomas and Paul M. Griffin. Coordinated supply chain management. *European Journal of Operational Research*, 94(1):1 – 15, 1996. ISSN 0377-2217. doi: 10.1016/0377-2217(96)00098-7. URL `http://www.sciencedirect.com/science/article/pii/0377221796000987`. 5.4, 5.5

[80] Ryota Tomioka, Taiji Suzuki, Masashi Sugiyama, and Hisashi Kashima. A fast augmented lagrangian algorithm for learning low-rank matrices. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010. 5.3

[81] 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. 5.1

[82] 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. 5.2

[83] François Vanderbeck. Implementing mixed integer column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 331–358. Springer, 2005. 5.2

[84] 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. 5.2

[85] T. Vossen, M. Ball, and R. H. Smith. On the use of integer programming models in AI planning. In *IJCAI-99*, 1999. 2

[86] B. Wahlberg, S. Boyd, M. Annergren, and Y. Wang. An admm algorithm for a class of total variation regularized estimation problems. In *16th IFAC Symposium on System Identification*, July 2012. 5.3

[87] William E. Walsh and Michael P. Wellman. Modeling supply chain formation in multiagent systems. In *IJCAI Workshop on Agent Mediated Electronic Commerce*, 1999. 5.5

[88] William E. Walsh and Michael P. Wellman. Modeling supply chain formation in multiagent systems. In Alexandros Moukas, Fredrik Ygge, and Carles Sierra, editors, *Agent Mediated Electronic Commerce II*, volume 1788 of *Lecture Notes in Computer Science*, pages 94–101. Springer Berlin Heidelberg, 2000. ISBN 978-3-540-67773-4. doi: 10.1007/10720026_5. URL `http://dx.doi.org/10.1007/10720026_5`. 5.4

[89] William E. Walsh, Michael P. Wellman, and Fredrik Ygge. Combinatorial auctions for supply chain formation. In *Proceedings of the 2nd ACM conference on Electronic commerce*, EC '00, pages 260–269, New York, NY, USA, 2000. ACM. ISBN 1-58113-272-7. doi: 10.1145/352871.352900. URL `http://doi.acm.org/10.1145/352871.352900`. 5.5

[90] William E. Walsh, Michael P. Wellman, and Fredrik Ygge. Combinatorial auctions for supply chain formation. In *Proceedings of the 2nd ACM conference on Electronic commerce*, EC '00, pages 260–269, New York, NY, USA, 2000. ACM. ISBN 1-58113-272-7. doi: 10.1145/352871.352900. URL `http://doi.acm.org/10.1145/352871.352900`. 5.4

[91] P. Xuan, V. Lesser, and S. Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001. 5.4

[92] 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. 5.2

[93] Andrew Y. Yg and Michael Jordan. Pegasus: A policy search method for large mdps and

pomdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2000. 5.4

[94] 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. 5.2

[95] 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. 1, 5.1

[96] X. Zheng and S. Koenig. Sequential incremental-value auctions. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2010. 5.1

[97] Brian Ziebart, Nathan Ratliff, Garratt Gallagher, Kevin Peterson, J. Andrew Bagnell, Martial Hebert, Anind K. Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. Technical Report Paper 331, Robotics Institute, Carnegie Mellon University, 2009. 3.4.1

[98] Brian D. Ziebart, J. Andrew Bagnell, and Anind K. Dey. Modeling interaction via the principle of maximum causal entropy. In *Proc. of the International Conference on Machine Learning*, pages 1255–1262, 2010. 3, 3.4