Agendas for Multi-Agent Learning

Geoffrey J. Gordon

December 2006
CMU-ML-06-116

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Shoham et al. [1] identify several important agendas which can help direct research in multi-agent learning. We propose two additional agendas—called "modelling" and "design"—which cover the problems we need to consider before our agents can start learning. We then consider research goals for modelling, design, and learning, and identify the problem of finding learning algorithms that guarantee convergence to Pareto-dominant equilibria against a wide range of opponents. Finally, we conclude with an example: starting from an informally-specified multi-agent learning problem, we illustrate how one might formalize and solve it by stepping through the tasks of modelling, design, and learning. This report is an extended version of a paper which will appear in a special issue of Artificial Intelligence Journal [2]; in addition to the topics covered in that paper, this report contains several appendices providing extra details on various algorithms, definitions, and examples.

Keywords: Multi-agent learning, modelling, design, Pareto optimality, planning, no-regret learning

1 Introduction

Shoham et al. [1] propose several agendas for research into multi-agent learning, which we briefly summarize: the **computational** agenda is to design algorithms which iteratively compute properties of games such as their Nash equilibria. The **descriptive** agenda is to determine how natural agents such as humans make decisions, and predict what those decisions will be. The **normative** agenda is to determine which learning algorithms are in equilibrium with which other learning algorithms, and under what circumstances. And, the **prescriptive** agenda is to design learning algorithms which allow agents to achieve high reward in practice. Shoham et al. split the prescriptive agenda into **cooperative** and **non-cooperative** flavors, depending on whether the underlying environment gives all agents identical rewards.

To these five agendas, we propose adding two more:

Modelling Determine how best to use the formal tools of multi-agent learning to describe real environments in which we wish to act. What aspects of the environment are safe to ignore or simplify? When can we fall back on simpler tools such as probabilistic inference, convex optimization, or combinatorial optimization to describe parts of the decision-making problem without losing the essential properties of the multi-agent interaction?

Design Often we have some flexibility in setting up the problem to solve. For example, we might be able to give some of the agents the ability to send messages to other agents, or to enter into some kinds of binding contracts. How can we best use this flexibility to optimize criteria such as ease of learning, uniqueness of the final learned policies, or predicted welfare of one or more of the agents?

Modelling and design are connected to all of Shoham et al.'s agendas. For example, if we want to predict the behavior of a human playing a game, it will help if our formal model of the environment is similar to whatever internal model the human uses. In this paper, though, we will focus on the connection to the prescriptive agendas: to build a practical multi-agent system, we must model the world accurately and design away unnecessary complexity.

While there are few AI papers which specifically discuss modelling or design, any research which uses multi-agent learning tools to solve a real problem must address both questions at least implicitly. For example, these problems are covered to varying degrees in research on poker [3, 4], robotic hide and seek and laser tag [5, 6], distributed planning [7], RoboCup [8, 9], the Trading Agent Competition [10–12], and economic modelling [13].

In the remainder of this article, we will start by fleshing out the modelling and design agendas in more detail. Then we will consider possible goals for research into modelling, design, and learning. Finally, we will conclude with an example: we will start from an informal specification of a learning problem and illustrate the steps one might use to arrive at a detailed description of a multi-agent learning system.

2 Modelling and Design

When writing software to help real agents decide how to act, we often have a fair amount of flexibility in how to model the problem: we can decide which parts of the problem, and at what granularity, to treat as possible sources of interesting interactions among agents. For example, suppose we are designing a team of robots to play Capture the Flag. We could discretize the physical state space finely and represent the whole problem as one large partially-observable stochastic game. In such a game, a player's observations might be raw laser rangefinder readings, and its actions might be to apply specified torques to its motors for specified amounts of time. Alternately, we could define high-level abstract behaviors like "patrol" or "chase," and have the players reason about their actions only at the level of behaviors and behavior parameters. In principle, the most accurate description of the environment is the finely-discretized low-level model. But, it is nearly impossible to work with such a detailed model using today's computers and algorithms; so, it is likely that a team based on the higher-level model would perform better in practice.

Unfortunately, there are few general techniques for partitioning the world into "decisions for which we really should pay attention to the other agents" and "decisions where it doesn't matter so much if we pretend the other agents don't exist." (And in fact the distinction is not even so black and white as this: there are decisions for which we will want to reason about a simplified model of another player, rather than a full model or no model.) So, we identify as an agenda the problem of modelling: the discovery of engineering principles which allow us to determine when it is safe to simplify or abstract away properties of the world or capabilities of other agents.

In addition to the flexibility we have in modelling, in real environments we can often influence the rules of the game. For example, in the Capture the Flag example from above, we could equip the robots with wireless ethernet cards and let them send messages to one another during the game, or we could accomplish a similar purpose with signal lights or noisemakers. We could also tell the players useful information ahead of time, such as tie-breaking rules or dispute resolution procedures. Even if we are building only a single robot which will play in pick-up games with unknown teammates, we can try to design signals that our teammates will be able to learn to understand and take advantage of. So, we identify the agenda of design: deciding how best to set up the playing field so that agents will be able to learn to interact with one another in desired ways, or to encourage agents to adopt some behaviors and avoid others.

3 Research goals

As mentioned above, we are particularly interested in modelling and design as they relate to the prescriptive agendas. So, by doing a good job of modelling and design, we hope to make it easier for our agents to learn to achieve high rewards.

Unfortunately, the current state of the art for testing whether we have done a good job is to hand our model to some learning agents and see how much reward they can earn. This sort of test isn't a good way to draw general conclusions: if the agents do poorly, we might have made bad modelling and design decisions, or we might just have used bad learning algorithms. And even if one configuration of learning algorithms does well, we can't say whether its success would generalize to another configuration.

In order to generalize better, we need to rely on performance properties of classes of learning algorithms rather than of individual algorithms. For example, a no-regret learner always achieves at least its safety value in a repeated game. So, if we are able to ensure that all agents' safety values are high, we can say that our model works well for any set of no-regret learners. (And in fact we will use no-regret learners for this reason in the example of Section 4 below.)

Unfortunately, the performance guarantees for common classes of learning algorithms are too weak to make this approach work well in many cases. So, we believe that an important research goal—which will advance at least the modelling, design, and prescriptive agendas—is to identify learning algorithms that have stronger performance guarantees against larger classes of opponents.

In particular, we believe that the most useful guarantee is one that is not very widely used: convergence to a Pareto-dominant (or nearly Pareto-dominant) subgame-perfect Nash equilibrium. We believe that this guarantee deserves further research, both because it intuitively corresponds to what we might expect from a learning algorithm and because it allows us to make specific predictions about the outcome of learning.

3.1 Performance guarantees

Various researchers have designed learning algorithms with different types of performance guarantees. For example, Shoham et al. describe rational learning, originally defined by Kalai and Lehrer [14]. When two rational learners play each other, if their priors are mutually absolutely continuous (a strong assumption), they are guaranteed to converge to a Nash equilibrium.

For another example, Shoham et al. describe no-regret algorithms for repeated matrix games. These algorithms bound how much they will wish they had played a constant action a_i when looking back on past plays. This regret bound implies, among other things, that no-regret learners achieve nearly their safety value in a repeated game no matter what strategy their opponent follows.

For a final example, no-internal-regret algorithms [15] place guarantees on how much they will want to substitute action a_i for action a_j when looking back on the history of past plays. No-internal-regret learners have all the same guarantees that plain no-regret learners have; in addition, a set of no-internal-regret learners will always converge to the play frequencies and payoffs of some correlated equilibrium in a repeated game.

Unfortunately, these and other similar guarantees are too weak for our purposes. To see why, consider the game illustrated in Figure 1, called *repeated Battle of the Sexes* or *RBoS*. RBoS is a repeated matrix game that models the

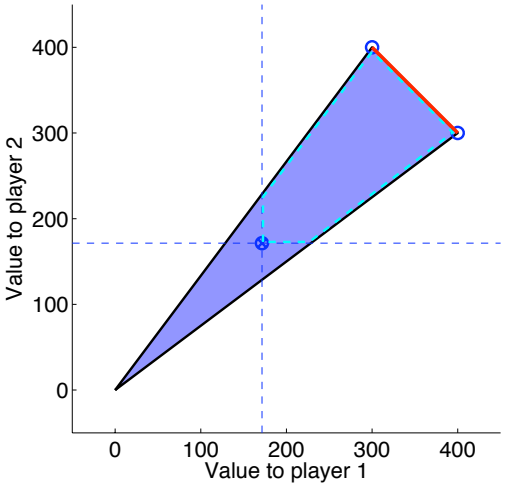

Figure 1: Illustration of feasible values, safety values, equilibria, Pareto dominance, and the Folk Theorem for RBoS.

problem facing two people who go out to an event every weekend, either the opera (O) or football (F). One person prefers opera, the other prefers football, but they both prefer to go together: the one-step reward function is

	O	F
O	$3, 4$	$0, 0$
F	$0, 0$	$4, 3$

Player p wants to maximize her expected total discounted future value V_p; we discount rewards t steps in the future by $\gamma^t = 0.99^t$. Figure 1 displays the expected value vector $(E(V_1), E(V_2))$ for a variety of situations.

The shaded triangle in Figure 1, blue where color is available, is the set of feasible expected-value vectors. Each of the points in this triangle is the expected-value vector of some joint policy (not necessarily an equilibrium).

The single-round Battle of the Sexes game has three Nash equilibria. Repeatedly playing any one of these equilibria yields an equilibrium of RBoS, and the resulting expected-value vectors are marked with circles in Figure 1. Some learning algorithms guarantee convergence of average payoffs to one of these points in self-play. For example, one such algorithm is gradient descent in the space of an agent's mixed strategies, since RBoS is a 2×2 repeated game [16].

Other algorithms, such as the no-regret learners mentioned above, guarantee that they will achieve at least the safety value of the game. The safety values for the two players are shown as horizontal and vertical thin dashed lines. So, two such algorithms playing against each other will arrive at a value vector somewhere inside the dashed pentagon (cyan where color is available).

The Folk Theorem tells us that RBoS has a Nash equilibrium for every point

inside this pentagon—that is, for every feasible and individually rational value vector.[1] So, algorithms which guarantee convergence to a Nash equilibrium of the overall game will also end up inside the pentagon. For example, two rational learners with mutually-absolutely-continuous priors yield this guarantee.

Finally, some combinations of algorithms, such as two no-internal-regret learners playing against one another, guarantee that they will converge to the per-step payoffs of a correlated equilibrium of the one-step game. In Battle of the Sexes, there is a correlated equilibrium for each point in the dashed pentagon in Figure 1.

In RBoS, every one of the above guarantees leads to the same conclusion: each player is guaranteed a total expected discounted payoff not much worse than her safety value of 166.67. And, it might be the case that no player gets much more than 166.67.

We claim that the values $(166.67, 166.67)$ cannot reasonably be called a successful outcome of learning: to achieve this level of payoff, the players can only wind up at the same event slightly less than half of the time, worse than if they picked which event to go to uniformly at random. Instead, a truly successful outcome would be something like the following: the players always go to an event together, alternating which event every weekend. This outcome is much better both in terms of social welfare (the sum of expected values, 700 for the turn-taking solution vs. 333.33 for the safety values) and individual expected value for each agent ($(350.25, 349.75)$ vs. $(166.67, 166.67)$, if the first player gets her preferred event on the first weekend).

3.2 Pareto dominance

We will say that an equilibrium, A, *Pareto dominates* another equilibrium, B, if all players achieve at least as high a value in A as they do in B, and if at least one player does strictly better in A. The *Pareto frontier* is the set of equilibria which are not Pareto dominated. The frontier for RBoS is the upper right edge of the dashed pentagon in Figure 1, marked in red where color is available.

If we can guarantee convergence to a point on or near the Pareto frontier, we can usually say much more about the agents' payoffs than we could with the weaker guarantees mentioned above. For example, in RBoS, every equilibrium on the Pareto frontier gives each agent a total discounted payoff of at least 300, almost twice as much as the guarantee from the safety values.

More importantly, the Pareto frontier captures an important part of our intuition about what it means to learn successfully: at a Pareto-dominant point, there are no "missed opportunities" which would allow one agent to achieve a higher payoff without hurting the others. Different points on the frontier might be better or worse for an agent, but we can explain these differences as results of the agent's skill or lack of skill at negotiation, rather than as failures of learning.

[1] In fact, it guarantees something stronger: every feasible and strictly individually rational value vector corresponds to a *subgame perfect* Nash equilibrium. In such an equilibrium all threats are *credible*: no agent wishes to deviate from her stated policy, even at states which can only be reached after a deviation.

For example, if the football lover negotiates well, the players might choose (F, F) often; but if they regularly chose (F, O) or (O, F) we would say that they hadn't learned very well.

There are a few examples in the literature of algorithms that guarantee they will reach a Pareto-dominant equilibrium if *all* agents use them. For example, Powers and Shoham's Metastrategy [17] guarantees that it will reach a point on the Pareto frontier in self-play in a repeated game, as does the modified no regret learning algorithm described in the Appendix. (See also [18].) Both of the above algorithms depend on the assumption that the environment is a repeated matrix game; there is no easy way to extend them to more general environments such as stochastic games.

For another example, Brafman and Tennenholtz [19] define an "efficient learning equilibrium," in which each agent specifies a learning algorithm before seeing the game, and no agent would benefit very much by switching learning algorithms after finding out the other players' choices.[2] They show the existence of a "Pareto ELE" for repeated matrix games: that is, they give an ELE in which the players always choose a nearly Pareto-dominant equilibrium. And, they extend the Pareto ELE result to stochastic games. However, their Pareto ELE can require the agents to make side payments to one another, which is not practical in some environments (and which makes the problem of agreeing on a Pareto-dominant equilibrium much easier). And, their equilibrium is not subgame perfect, which suggests that rational agents might be motivated to deviate from it.

Finally, Murray and Gordon [20] describe a family of algorithms that guarantee that they will reach nearly Pareto-dominant equilibria in stochastic games. These algorithms do not need side payments, but they do need "cheap talk," i.e., non-binding communication among agents. The resulting equilibria are subgame perfect, but in order to achieve subgame perfection the algorithms assume that suitable punishment policies are provided as input (although a newer, unpublished version of the algorithm relaxes this requirement).

3.3 Possible refinements

While the algorithms of Section 3.2 achieve better guarantees than many other multi-agent learners, these guarantees are still weaker in some ways than we might like. First, their bargaining style is inflexible: they essentially make a single take-it-or-leave-it offer at the beginning of the game and then try to stick to it until the other player capitulates. They achieve equilibrium in self-play simply because all players are designed to make the same offer. We would rather see a learning algorithm which can accept ideas from other players in a back-and-forth negotiation (while still, of course, avoiding Pareto-dominated

[2] Brafman and Tennenholtz consider a setting in which not only the behaviors of the other agents but also the payoffs of the underlying game are initially unknown and must be learned. However, this setting is equivalent to the one considered here: we can augment the game with an extra player, "Nature," whose initially-unknown but fixed strategy determines the payoffs of the other agents.

outcomes).

Second, the Pareto frontier can be a big place. In a joint strategy on the Pareto frontier, some agents might only receive their safety values. So, from an individual agent's perspective, the guarantee of Pareto dominance doesn't always translate into a better lower bound on reward than do other guarantees such as no regret. (By contrast, from a team's perspective, the Pareto dominance guarantee *is* better than other bounds: unless the set of equilibria is trivial, the social welfare of an outcome on the Pareto frontier is always better than the sum of the players' safety values.)

Finally, the algorithms of Section 3.2 only guarantee Pareto dominance when playing against themselves or other very similar learning algorithms. Since we don't necessarily control the learning algorithms used by the other players, it would be much better if we could find algorithms that needed only weak conditions on the other players to guarantee a Pareto-dominant outcome. Unfortunately it is not clear what sorts of assumptions about the other players would allow us to prove this stronger sort of guarantee.

4 An example

To illustrate the problems which arise in design and modelling, in this section we will work out an extended example of a multi-agent learning problem and discuss the tradeoffs that result from various design and modelling decisions. We will demonstrate techniques for modularization, factorization, and simplification, including:

- Engineering what the agents know about one another, to minimize the effect of partial observability while leaving communication and planning problems manageable;
- Designing markets and negotiation systems to help resolve disputes; and
- Taking advantage of no-regret techniques to provide better guarantees for the learning and planning components of an agent.

And, we will recommend different ways the agents can learn about one another.

Our example is one of supply chain management. A manufacturer has built its factory next to a set of warehouses run by its suppliers. The manufacturer can build several different types of products in its factory; each of these products is built from different parts and requires different machinery. Some of the machinery is permanently installed in the factory, while other machinery requires setup time and cost to bring online, and may incur teardown time and cost when no longer needed.

To save money, the manufacturer has no long-term storage space at its factory. Instead, it depends on the suppliers to deliver parts directly to its assembly lines as they are needed. To facilitate this just-in-time delivery, the manufacturer is willing to provide information about its operations to the suppliers. The exact nature of this information is a question of design: we should provide

enough information to allow the suppliers to choose good plans, but not so much as to be unwieldy or to violate the manufacturer's privacy.

To move parts around the complex, each supplier operates a fleet of robotic trucks. These trucks drive back and forth among the warehouses and assembly lines, picking up parts and delivering them. Each supplier owns several warehouses which stock various parts at various costs, and each part may be stocked at several warehouses. Suppliers may or may not be willing to sell parts to their competitors. The part prices may fluctuate over time, but the suppliers always know the current prices.

In this domain, learning is crucial: there are many equilibria with widely varying payoffs, so if the agents don't learn to cooperate with one another they may lose lots of money. For example, depending on how we set up the problem, there can be an equilibrium which leaves the factory and all the trucks idle.

4.1 Assumptions about the Manufacturer

If we were to try to model this problem exactly, its description would be very large: for example, the problem state would include the locations and contents of all of the trucks, the manufacturer's projected demand from its customers, the setup of the manufacturer's machinery, and the state of each partially assembled product in the factory. Worse, most players would not know every bit of this information, and so would have to worry about making forecasts and guessing what the other players' forecasts are.

Instead, we will make a series of modelling assumptions and design decisions to simplify the problem while retaining its essential features. If we make these decisions well, the problem will become much easier to solve. While we don't claim that our choices here are perfect, they illustrate one approach to making planning and learning tractable.

Our first assumption is that the manufacturer is not demand-limited: it can sell as many of its products as it can build, at prices which are enough higher than the cost of materials to cover the players' operating costs. This assumption means, first, that the manufacturer doesn't just want to shut down the factory, and second, that the manufacturer's demand forecasts are no longer important hidden state.

Our second assumption is that the process of manufacturing each product can be described by a graph like the one shown in Figure 2. To build this product, the manufacturer must start with part A. It can then continue by adding either B or C. Once both B and C are in place, it can attach parts D and E in that order. Each of the parts is relatively bulky and expensive, and takes some time to install. So, the manufacturer does not want to take delivery of a part unless it is ready to use it; for the product of Figure 2, the manufacturer would not buy part D unless it had a partial assembly containing A, B, and C already.

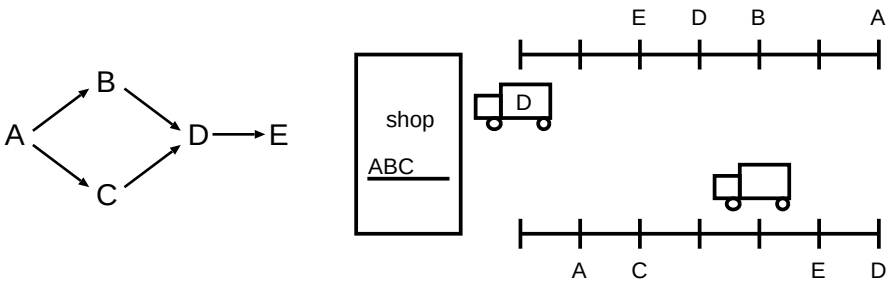

Figure 2: Left: a directed acyclic graph describing one of the manufacturer's products. Right: a simplified problem instance, with only two suppliers, each supplier's warehouses laid out along a line, only one part type per warehouse, capacity for only one part in each truck, and only one product under construction at the factory. The product currently has parts A, B, and C installed, and the first supplier is about to deliver D.

4.2 A parts market

We are now in a position to make some design decisions about how the manufacturer interacts with the suppliers. The goal of these decisions is to break the overall planning problem into separate, tractable pieces.

Our first design decision is to treat each robotic truck as a separate agent, whose goal is to make profit for its owner. We will charge a truck when it picks up a part from a warehouse, and credit it when it delivers the part to the manufacturer (or returns the part to a warehouse, should it decide it needs to switch cargoes). We will give each truck the ability to communicate with all the other trucks in a common language, so that they can coordinate their deliveries.

Treating every truck as a separate agent risks the possibility that two trucks belonging to the same manufacturer will unknowingly sabotage one another. This sort of destructive interaction might happen either because the trucks failed to learn well (which we hope to avoid by using smart learning algorithms) or because one truck might achieve an advantage for itself by hurting another truck (which we hope to avoid by designing our market well).

Our next design decision is how to let the manufacturer communicate with the suppliers so that it can buy the parts it needs. Our goal here is to design a market that avoids the planning inefficiencies that can happen when the agents lack necessary information about one another.

In order to design this market, we will make an additional modelling assumption. If the prices that the warehouses pay for parts and the prices that the manufacturer receives for finished products were constant, the agents could follow an open-loop plan for all time. If instead these prices change slowly, there will be an open-loop plan which the agents will be happy to follow for many steps into the future. So, we will assume that prices change slowly enough that the agents don't lose much welfare by committing to an open-loop plan fairly

far into the future. (The agents can always, by mutual consent, edit the plan if prices do change enough to make it worthwhile.) Then, we will say that the manufacturer and suppliers negotiate by jointly specifying an open-loop plan.

The agents can specify an open-loop plan by describing a delivery schedule for each truck along with the prices that the manufacturer will pay for each delivery. (We'll assume the manufacturer's price must at least cover the part's cost, to avoid various sorts of shady dealings.) The full plan contains much more detail than this, but from the schedule and prices each agent can plan independently and fill in how it needs to act: the manufacturer can optimize its assembly operations, and each truck can optimize its delivery route.

4.3 Negotiation

With the above design decisions, we have greatly simplified the agents' planning problems. For a fixed schedule and prices, the agents are completely decoupled: the manufacturer can optimize its assembly schedule using something similar to a job-shop scheduler, and the trucks can optimize their delivery routes using something similar to a travelling-salesman planner.

We can also greatly simplify the agents' learning problems: we will design our market so that, instead of needing to predict the other agents' behavior in every possible state of the world, an agent only needs to predict which schedules and price lists the other agents will agree to. In particular, we will design an *alternating offers bargaining game* of the sort proposed by Rubinstein [21].

In our game, the players will take turns proposing agreements, with each agreement comprising a schedule and a price list. After hearing a proposal, each player can compute a plan to achieve the proposed schedule, and decide whether she wants to accept or reject the proposal. If any player rejects the proposal, that player proposes another agreement, and the process repeats. If the players continue to reject each others' proposals, our rules will tell them to give up at some point and follow a predefined default plan called the *disagreement point*; as designers, we can pick this plan arbitrarily and tell the players what it is.

By specifying the details appropriately (see the Appendix or the similar algorithm in [20]), we can guarantee that the unique equilibrium of this game is the *Nash bargaining point*, a point on the Pareto frontier which Nash [22] suggested as a reasonable outcome of this sort of bargaining game. But even for boundedly rational players, there is never an incentive to insist on an agreement that's not on the Pareto frontier: while a computation-limited agent might not be able to come up with a proposal on the Pareto frontier, it can at least recognize when one proposal Pareto dominates another.

Another nice feature of our game for agents with limited computation is that it is comparatively easy for an agent to propose a delivery schedule that it can comply with. So, a limited agent can run its planner for however long it can afford, and make proposals based on the best schedule it finds. It can even incorporate ideas from schedules previously proposed by other agents. Furthermore, learning can whittle down the set of schedules that each agent needs to consider: for example, if one agent's cost to supply part A is much higher than

another's, that agent should be able to learn that it doesn't need to consider schedules where it supplies many copies of part A.

4.4 Optimization

To conclude our example, we will consider one last possible opportunity for learning. As mentioned above, a fixed schedule and price list allow the agents to plan independently and arbitrarily far into the future. But in reality, the agents may realize at some point that they want to change their plans: some part prices may go up or down, or some agent may discover a delivery schedule that leaves everyone better off.

We have assumed that the agents are free to renegotiate in such a circumstance. But, if this sort of renegotiation happens frequently, the agents may regret planning under the assumption of a fixed schedule: for example, a truck might be willing to pay a little bit more in order to avoid risks such as carrying expensive parts for long intervals or driving to inconvenient locations.

To learn which sorts of plans are likely to work well even in the face of changing goals and costs, we can represent an agent's planning problem as an online convex program. For example, a simplified version of a truck's planning problem is a repeated travelling salesman problem with unknown edge costs: for each delivery the truck leaves the manufacturer, picks up parts from some predetermined set of warehouses, and returns to drop them off at the manufacturer. (A more fully detailed version of the planning problem, including uncertainty about the next delivery deadline, uncertainty about the cost for each part at each warehouse, and flexibility about where to pick up parts, is described in the Appendix.)

If we represent a tour with one indicator variable e_i for each edge $i \in E$, then we can write $\mathcal{X} \subseteq \{0,1\}^{|E|}$ for the finite set of feasible tours. If $c \in \mathbb{R}^{|E|}$ is the (unknown) vector of edge costs, then the cost of a tour $x \in \mathcal{X}$ is $c \cdot x$. And, if we pick a tour at random from some distribution $P(x)$, the expected cost of our tour is $E_P(c \cdot x) = c \cdot E_P(x) = c \cdot \bar{x}$, where $\bar{x} \in \operatorname{conv} \mathcal{X}$ is the mean of the distribution P.

Since we don't know c ahead of time, we can try to learn a good distribution $P(x)$ by observing samples of c: we plan and execute a tour, observe how much it actually would have cost to traverse each edge on this round, and then plan our next tour. Since the only property of $P(x)$ that matters is its mean $\bar{x} \in \operatorname{conv} \mathcal{X}$, and since our payoffs are linear in $\bar{x}$, choosing the best $\bar{x}$ is an online convex program. (Randomization serves two purposes here: first, it assigns a meaning to interior points of $\operatorname{conv} \mathcal{X}$, and second (and more importantly), it allows us to avoid oscillations caused by feedback among multiple learners whose actions influence each others' costs.)

To solve this OCP, we can use any of a number of algorithms, such as Generalized Gradient Descent [23], GIGA [24], Lagrangian Hedging [25], or Follow the Predicted Leader [26]. In this case FPL is particularly convenient and simple to describe: we keep track of the sum of all past cost vectors $c_T =$

$\sum_{t=1}^{T-1} c_t$, and on iteration T we choose

$$\bar{x}_T = \arg\min_{x \in \bar{\mathcal{X}}} (c_T + \epsilon_T) \cdot x \tag{1}$$

where ϵ_T is a random vector from an appropriate distribution. Finding the arg min in Equation 1 is an ordinary travelling salesman problem with known costs; we can solve this problem using any of the many search algorithms that have been proposed for TSPs.

By learning which tour to take using a no-regret algorithm, we have guaranteed that our tours will cost not much more than if we had known the true average edge costs ahead of time and chosen the best fixed tour. This guarantee means that we can feel safe allowing the agents to renegotiate their plans frequently: individual agents will learn what sorts of plans are risky, and act as if those plans are more costly when negotiating.

5 Discussion

We proposed the agendas of "modelling" and "design" to complement the five agendas identified by Shoham et al. [1]. We then identified a research goal which could advance the modelling, design, and prescriptive agendas: find learning algorithms that converge to a Pareto-dominant subgame-perfect equilibrium against a wide range of opponents. Finally, we illustrated our ideas through an extended example. This example demonstrated several ways to achieve near-Pareto-dominance with current algorithms: first, we factored our model so that the agents can conduct the most important part of their negotiations up front and with negligible uncertainty. Second, we set up our negotiation problem so that an agent's best response to any reasonable opposing strategy is always on the Pareto frontier. And finally, we handled residual uncertainty using no-regret learning.

Acknowledgements

We thank Shoham et al. for starting what promises to be a very interesting discussion on multi-agent learning, and we thank the editors of the special issue for the opportunity to contribute. This research was supported in part by a grant from DARPA's CS2P program.

References

[1] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 2007. Special issue edited by M. Wellman and R. Vohra, to appear.

[2] Geoffrey J. Gordon. Agendas for multi-agent learning. *Artificial Intelligence*, 2007. Special issue edited by M. Wellman and R. Vohra, to appear.

[3] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2003.

[4] Jiefu Shi and Michael L. Littman. Abstraction methods for game theoretic poker. In *Computers and Games*, volume 2063 of *Lecture Notes in Computer Science*, pages 333–345. Springer, Berlin, 2001.

[5] Rosemary Emery-Montemerlo, Geoff Gordon, Jeff Schneider, and Sebastian Thrun. Game theoretic control for robot teams. In *Proc. Int'l Conf. on Robotics and Automation (ICRA)*, 2005.

[6] Curt Bererton. *Multi-robot coordination and competition using mixed integer and linear programs*. PhD thesis, Carnegie Mellon Robotics Institute, 2004. Available as tech report CMU-RI-TR-04-65.

[7] Carlos Guestrin and Geoffrey Gordon. Distributed planning in hierarchical factored MDPs. In A. Darwiche and N. Friedman, editors, *Uncertainty in Artificial Intelligence (UAI)*, volume 18, 2002.

[8] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for RoboCup-soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[9] Michael Bowling and Manuela Veloso. Simultaneous adversarial multi-robot learning. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2003.

[10] S.-F. Cheng, E. Leung, K. M. Lochner, K. O'Malley, D. M. Reeves, L. J. Schvartzman, and M. P. Wellman. Walverine: A Walrasian trading agent. *Decision Support Systems*, 39:169–184, 2005.

[11] David Pardoe and Peter Stone. TacTex-2005: A champion supply chain management agent. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, July 2006.

[12] Peter Stone, Michael L. Littman, Satinder Singh, and Michael Kearns. ATTac-2000: An adaptive autonomous bidding agent. *Journal of Artificial Intelligence Research*, 15:189–206, June 2001.

[13] David M. Kreps. *Game Theory and Economic Modelling*. Oxford University Press, New York, 1990.

[14] Ehud Kalai and Ehud Lehrer. Rational learning leads to Nash equilibrium. *Econometrica*, 61(5):1019–1045, 1993.

[15] Dean Foster and Rakesh Vohra. Regret in the on-line decision problem. *Games and Economic Behavior*, 29:7–35, 1999.

[16] Satinder P. Singh, Michael J. Kearns, and Yishay Mansour. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, pages 541–548. Morgan Kaufmann, 2000.

[17] Rob Powers and Yoav Shoham. New criteria and a new algorithm for learning in multi-agent systems. In *Advances in Neural Information Processing Systems*, volume 17, 2005.

[18] Geoffrey J. Gordon. Learning for multi-agent decision problems. Invited talk at AAAI workshop on Multiagent Learning, 2005. Slides available from `http://www.cs.cmu.edu/~ggordon/ggordon.2005-07-10.aaai-mal.pdf`.

[19] Ronen I. Brafman and Moshe Tennenholtz. Efficient learning equilibrium. *Artificial Intelligence*, 159(1–2), 2004.

[20] Chris Murray and Geoffrey J. Gordon. Multi-robot negotiation: approximating the set of subgame perfect equilibria in general-sum stochastic games. In *Advances in Neural Information Processing Systems*, volume 19, 2007.

[21] Ariel Rubinstein. Perfect equilibrium in a bargaining model. *Econometrica*, 50(1):97–109, 1982.

[22] John F. Nash, Jr. The bargaining problem. *Econometrica*, 18(2):155–162, 1950.

[23] Geoffrey J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, 1999.

[24] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *Proceedings of the Twentieth International Conference on Machine Learning*. AAAI Press, 2003.

[25] Geoffrey J. Gordon. No-regret algorithms for online convex programs. In *Advances in Neural Information Processing Systems*, volume 19, 2007.

[26] Adam Kalai and Santosh Vempala. Geometric algorithms for online optimization. Technical Report MIT-LCS-TR-861, Massachusetts Institute of Technology, 2002.

[27] Sergiu Hart and Andreu Mas-Colell. A general class of adaptive strategies. *Journal of Economic Theory*, 98(1):26–54, 2001.

[28] Geoffrey J. Gordon. No-regret algorithms for structured prediction problems. Technical Report CMU-CALD-05-112, Carnegie Mellon University, 2005.

[29] V. Krishna and R. Serrano. Multilateral bargaining. *Review of Economic Studies*, 1996.

[30] Robert J. Aumann and Sergiu Hart. Long cheap talk. *Econometrica*, 71(3):1619–1660, 2003.

[31] I. Barany. Fair distribution protocols or how the players replace fortune. *Mathematics of Operations Research*, 17(2):327–340, 1992.

[32] Yevgeniy Dodis, Shai Halevi, and Tal Rabin. A cryptographic solution to a game theoretic problem. In *Lecture Notes in Computer Science*, volume 1880, page 112. Springer, Berlin, 2000.

[33] Sergei Izmalkov, Silvio Micali, and Matt Lepinski. Rational secure computation and ideal mechanism design. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, 2005.

[34] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

Appendix

This appendix contains additional definitions and details which could not be included in the published version of the article [2] due to space concerns.

A Algorithms that achieve Pareto dominance

An algorithm of Powers and Shoham, called MetaStrategy [17], guarantees that it will reach a point on the Pareto frontier in self-play in a repeated game. It also guarantees to learn a near-best response against a stationary opponent, and never to do much worse than its safety value against an arbitrary opponent. MetaStrategy, as one might expect from its name, achieves its guarantees by selecting from several base strategies: first it attempts to "teach" the other player to play a particular joint strategy on the Pareto frontier by repeatedly playing its part of that strategy. If the other player doesn't seem to be learning after a while, it tries to play either a best response to what it has seen so far, or a safe strategy which limits its loss.

A similar idea, developed independently and somewhat later [18], is to use the extra degrees of freedom provided by some no-regret algorithms to try to teach the other player a favorable strategy. The advantage of this approach is that it results in an algorithm with an extremely simple proof of correctness.

To demonstrate this approach we can use a common no-regret algorithm called external regret matching [27]. ERM works as follows: if there are d actions, initialize the vector $S \in \mathbb{R}^d$ arbitrarily. On each turn, play an action according to the distribution

$$P(a) = \frac{\max(S_a, 0)}{\sum_{a'} \max(S_{a'}, 0)} \tag{2}$$

Then, if we select action a and the other player selects b, update

$$S_{a'} \leftarrow S_{a'} + R(a', b) - R(a, b)$$

for each action a'. If $S_a \leq 0$ for all a, the denominator in Equation 2 is zero, and ERM is free to play arbitrarily.

It is traditional to initialize S to 0 in ERM. But, if we initialize S so that its components are all negative, ERM can choose its initial plays arbitrarily. The penalty for this nontraditional initialization is only a delay in the time at which ERM is guaranteed to fall below a given per-play regret target.

For example, in RBoS, suppose we initialize S_a to -40 for all a and always play O when we have a choice. We will play O for at least the first 10 rounds, since each S_a can increase by at most 4 per round. If in this interval the other player learns to play O all the time, the players will continue playing (O, O) forever; if not, the algorithm will eventually start to randomize its plays to avoid suffering high regret.

This algorithm, call it "delayed ERM," achieves guarantees similar to Meta-Strategy: in self-play it will always reach a joint action on the Pareto frontier

(namely (O, O)); against a stationary opponent its play frequency will converge to a best response; and against an arbitrary opponent it will never do much worse than its safety value. (The first statement is obvious, while the second two are known properties of no-regret algorithms.)

More generally, while no regret algorithms clearly do not solve the whole problem of multi-agent learning, we believe that they are an excellent building block for more complex algorithms. Known techniques to measure and control regret, such as those based on potential functions [28], seem sufficiently flexible that we believe they can be combined with other, as yet unknown techniques to achieve tighter guarantees. In this way, an algorithm could provide a "safety net" to keep a player's loss from going much below her safety value, and at the same time attempt to reach a more favorable outcome if the other players are willing to cooperate.

B Rubinstein's Game and the Nash Bargaining Point

In a famous paper, Nash [22] studied an idealized bargaining problem. This problem, illustrated in Figure 3, is defined by a compact convex set of feasible utility vectors $S \subset \mathbb{R}^2$ and a disagreement point $d = (d_1, d_2) \in S$. S represents the possible agreements that the players could make: an agreement $u = (u_1, u_2) \in S$ means that player 1 gets utility u_1 and player 2 gets utility u_2. If the players fail to come to an agreement, then player 1 gets d_1 and player 2 gets d_2.

For example, one way that such a bargaining problem might arise is from an underlying matrix game. If we let S contain all convex combinations of expected-value vectors for equilibria of the game, then the bargaining problem is to select a distribution over equilibria for the players to implement. In this case d is usually thought of as representing the status quo, the way that players "customarily" play this game or have "typically" played it in the past. (This information might not be available, in which case it can be difficult to justify any one choice of d over another. This difficulty can be a significant problem for practical application of the Nash bargaining model.)

Nash was interested in the outcome of this bargaining problem; he wanted to predict which agreement will be selected by rational bargainers. To this end he suggested a set of four axioms that the outcome $u^* = (u_1^*, u_2^*)$ should satisfy:

Optimality The outcome should be efficient. That is, u^* should be on the Pareto frontier.

Scale invariance Scaling and shifting S and d should scale and shift u^* proportionally. That is, for $a_1, a_2 > 0$ and $b_1, b_2 \in \mathbb{R}$, if we define

$$\begin{aligned} S_{\text{scaled}} &= \{\, (a_1 u_1 + b_1,\, a_2 u_2 + b_2) \mid (u_1, u_2) \in S \,\} \\ d_{\text{scaled}} &= (a_1 d_1 + b_1,\, a_2 d_2 + b_2) \end{aligned}$$

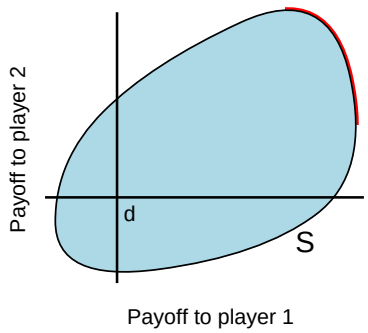

Figure 3: The Nash bargaining problem. Players must agree on a payoff vector in S. If they fail to agree their payoffs are given by the disagreement point d, in this case the origin. The Pareto frontier of S is marked in red where color is available.

then $u^*_{\text{scaled}} = (a_1 u_1^* + b_1,\, a_2 u_2^* + b_2)$ should be the outcome of the bargaining problem $(S_{\text{scaled}}, d_{\text{scaled}})$. (Since the players aren't allowed to make side payments to one another in this bargaining model, their utilities are only defined up to an arbitrary shift and positive scaling. So, it would be unnatural for this sort of transformation to change the outcome of bargaining.)

Symmetry If S and d are symmetric about the line $u_1 = u_2$, then $u_1^* = u_2^*$.

Independence of irrelevant alternatives If $S' \subset S$ but $d \in S'$ and $u^* \in S'$, then u^* is the outcome for the problem (S', d) as well. This last axiom is the most controversial of the set, and many researchers have investigated alternatives for it.

These four axioms are enough to identify a unique outcome u^* for each (S, d) pair: it is the point which maximizes the product of the amounts of utility that the players gain by cooperating,

$$u^* = \arg\max_{u \in S} (u_1 - d_1)(u_2 - d_2)$$

Nash did not model the process of negotiation by which the players might agree on a bargaining solution. To address this question, Rubinstein [21] considered a slightly simpler situation: suppose two players want to divide a slice of pie. If they can agree on a division x, y (with $x, y \geq 0$ and $x + y \leq 1$) then they can implement it, while if they cannot agree, neither player gets any pie. Player p's utility for receiving a fraction x is $U_p(x)$. U_p is assumed to be nonnegative, concave, and increasing, with $U_p(0) = 0$.

The shaded set in Figure 4 shows, for one instance of this game, the utility vectors corresponding to feasible divisions of the pie. That is, it shows the set

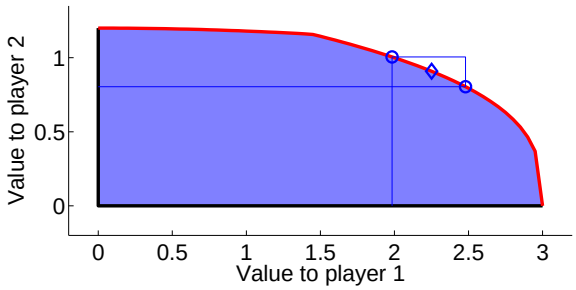

Figure 4: Equilibria of a Rubinstein game with $\gamma = 0.8$. Shaded area shows feasible value vectors $(U_1(x), U_2(y))$ where x, y is a feasible division of the pie. Circles show equilibria: right-hand circle corresponds to equilibrium when player 1 moves first, left-hand circle to equilibrium when player 2 moves first. Nash bargaining point is indicated by $\diamond$. To find the equilibria geometrically, draw two similar rectangles which touch each other as shown, with a size ratio of $\gamma : (1 - \gamma)$, and with the outer rectangle's corners on the Pareto frontier.

S of vectors $(U_1(x), U_2(y))$ for $x, y \geq 0$ and $x + y \leq 1$. The Pareto-dominant outcomes (the upper-right surface of the feasible set, shown in red where color is available) are the divisions which leave no pie on the table: $x + y = 1$. The disagreement point is $d = (0, 0)$: no pie for anyone.

Rubinstein defined an *alternating offers* bargaining game for dividing the pie. In this game, the first player offers a division x, y to the second; the second player either accepts the division, or refuses and offers her own division x, y. The process repeats until some player accepts an offer or until either player gives up. Rubinstein considered several models for how players' utilities decrease with time; perhaps the most interesting is when p's utility for fraction x at time t is

$$U_p(x, t) = \gamma^t U_p(x)$$

for a discount factor $0 \leq \gamma < 1$.

Rubinstein showed that, with this utility function, rational players will immediately agree on a division near the Nash bargaining point. That is, there is a unique subgame-perfect Nash equilibrium of the Rubinstein game; in this equilibrium, the first player immediately proposes a division near the Nash bargaining point and the second player accepts. How near depends on the discount factor γ: as $\gamma \uparrow 1$, the equilibrium will approach the Nash point more and more closely. See Figure 4 for an illustration.

For three or more players, we can extend the definition of the Nash bargaining point in an obvious way by maximizing the product of all players' excess utilities. It is not as obvious how to extend the Rubinstein game: how should the players take turns making offers, what format should these offers take, and what if some players want to accept an offer while other players do not? But, it turns out that there is an extension of the Rubinstein game which preserves most of its properties.

In the multi-player Rubinstein game [20, 29], agents take turns proposing multi-way divisions of the pie: for example, Alice might propose that she and Bob each get 40% of the pie, while Charles gets 20%. After each proposal, all agents other than the proposer decide independently whether to accept or reject. If all agents accept, the proposal is implemented. Otherwise, any agents who accepted have their shares fixed at the proposed level and are removed from further play; the next remaining agent then proposes a division of the remaining pie. In the above example, if Bob accepts Alice's proposal but Charles does not, then Bob gets 40% of the pie, and Charles proposes a division of the remaining 60% between himself and Alice. As in the two-player game, there is a unique subgame-perfect equilibrium which approaches the Nash point as $\gamma \uparrow 1$.

Dividing a pie is a special case of Nash's bargaining problem, since the shape of S is restricted: it is always feasible to give a player her disagreement value no matter what we give the other players. We will call sets like this *downward closed*. That is, S is downward closed with respect to d if, for all u and u' with $d \leq u' \leq u$, $u \in S \Rightarrow u' \in S$.

The two-player version of Rubinstein's game works for arbitrary S, but the multi-player version only works if S is downward closed: without this property, if we assign a share of the pie to one player, we might guarantee another player more than her disagreement value. That guarantee would give the second player extra leverage, allowing her to demand a larger share of the pie. We will say more below about how to ensure that S is downward closed.

C Tools

In this section we will describe and comment on a number of the tools that are available to help with the problems of modelling and design.

Abstractions In the behavior-based system mentioned in Section 2, a domain expert decides on some useful policy fragments and implements them by hand. In addition to such hand-implemented abstractions, there are many domains where researchers have picked out subproblems that can be solved automatically with non-learning optimization methods, or with learning methods that are not safe for multiple agents. Useful methods include combinatorial optimization (e.g., facility location), tree or graph search (e.g., path planning), and linear and convex optimization (e.g., regression and support vector machines).

Cheap talk We can allow the players to send each other non-binding messages during the game. This ability is called "cheap talk," to distinguish it from binding contracts. It is usual to assume that cheap talk happens between rounds of action selection, without delaying the underlying game; the players have no trouble understanding one another; and they do not have to pay to send or receive a message. We can use cheap talk for a number of purposes. For example, it can add useful equilibria to a game with incomplete information [30], and it can aid in simulating a moderator (see below).

Moderator To implement a correlated equilibrium, the players need to select joint actions from a known distribution and reveal to each player only her part of the joint actions. This joint randomization can be difficult for the players to achieve by themselves: there may be no communication between players, and even if there is, there may be no one player who can be trusted to randomize honestly. So, we can often give the players additional capability by adding a trusted moderator who can observe the game and send messages to each player.

The moderator is an important enough construct that researchers have investigated in some detail how to build one. If there are at least four players, Barany [31] shows how to simulate a moderator. Dodis et al. [32] demonstrate how to simulate a moderator using only two communicating players, but only if some cryptographic hardness assumptions hold. And Izmalkov et al. [33] suggest a way that players can use physical devices like nested envelopes and a "ballot box" (which randomizes the order of a set of envelopes) to simulate a moderator. All of these constructions may require modifying the game in some way, by adding cheap talk or extra players or by providing envelopes and ballot boxes.

Binding contracts A binding contract is a way for a player to agree to have her own utility lowered under certain circumstances. By entering into such contracts, players can rule out undesireable outcomes.

Transferable utility If we provide a common currency, then one player can pay another to take a favorable course of action. If the players can be trusted to make promised payments (for example, because of a trusted outside party who holds the money, or because a player who fails to pay can later be punished within the rules of the game), then transferable utility simplifies bargaining: for a set of feasible utility vectors S, let $u_{\text{sw}} \in S$ be a point which maximizes social welfare. That is, let

$$u_{\text{sw}} = \arg\max_{u \in S} \mathbf{1} \cdot u$$

where $\mathbf{1} = (1, 1, \ldots, 1)^{\text{T}}$. Similarly, let $u_{\overline{\text{sw}}} \in S$ be a point which minimizes social welfare. Adding transferable utility effectively replaces S with the set

$$\{u \mid u_{\overline{\text{sw}}} \cdot \mathbf{1} \leq u \cdot \mathbf{1} \leq u_{\text{sw}} \cdot \mathbf{1}\}$$

So, the Pareto frontier becomes a plane with normal $\mathbf{1}$ which passes through u_{sw}, and the Nash bargaining point will be $d + k\mathbf{1}$ for some k. Bargaining therefore reduces to picking a course of action that maximizes social utility and deciding how to split the excess utility among players; the Nash point splits excess utility evenly.

Transferable utility can simplify planning as well: the players all want to maximize social welfare in order to make the pot of money they divide as large as possible. To do so, they can use a cooperative planning algorithm instead of worrying about incentives. (Or at least, they can do so if the domain is simple enough that cooperative planning is possible.)

Markets If we can create markets where agents can buy and sell constrained resources, the agents may be able to achieve a higher social welfare. Traditional resources include coal, pork bellies, or electricity; less traditional resources might include the right to pass over a congested bridge at a specified time, or the right to require another agent to come to a specified place and help with a specified task within a specified interval.

Altering payoffs Perhaps we can, via some choice during the setup of the game, affect the costs of some courses of action. Doing so can change the available equilibria. For example, by altering payoffs we can provide:

Punishments We can let one player unilaterally reduce another player's utility by some defined amount; or, we can let a group of players acting together do so. We can even let players punish themselves, if we can guarantee that they are motivated to do so. This ability can provide a valuable enforcement tool: players can take actions which might otherwise let a cheater take advantage of them, secure in the knowledge that the cheater will be scared of future punishment.

D Example

In this section we provide additional details about the issues that arise in our extended example, as well as more information about how we might address them.

D.1 Consequences of assigning one agent per truck

We decided above to assign a separate agent to each truck, rather than one per supplier. This decision could hurt us in two ways: first, some asymmetry of the problem might give one truck bargaining leverage over the other, forcing the outcome of bargaining away from the solution which maximizes the sum of their payoffs. Second, making the trucks independent might rule out some desireable equilibria: for example, one truck might have an incentive to underbid another truck from the same supplier.

The first possibility (asymmetry) is unlikely to be much of a problem in our example: the only asymmetry between trucks from the same supplier is their current state (position and cargo), which should change rapidly enough that any bargaining advantages are fleeting. The second problem (bad incentives) is examined in more detail in the following sections.

D.2 Examining Incentives

Once we have fixed the prices and the delivery schedule, we need to worry whether any agent has an incentive to deviate. A supplier can deviate by failing to deliver a promised part (perhaps because this particular delivery doesn't

yield enough profit) or by delivering a part when it's not supposed to (say, in an attempt to undercut another supplier). The manufacturer can deviate by refusing to pay the promised price for some delivery, or by giving an extra payment to some supplier.

To test whether there is an incentive to deviate, we must specify punishments for each deviation and show that every punishment deters its crime. We will say that the manufacturer punishes a deviating supplier by refusing to deal with it in the future. Similarly, the suppliers as a group can punish the manufacturer by refusing to deal with it.

These punishments are expensive for their recipients: even if there are outside buyers it can deal with, an ostracized supplier will not be able to sell very many parts, and will still need to pay fixed costs on its trucks and warehouses. Similarly, the manufacturer will not be able to build products nearly as quickly if it has to ship in parts from far-away alternate suppliers, and must also pay fixed costs on its factory and equipment. So, we will assume that the expected discounted value for an agent which is being punished is large and negative.

On the other hand, we have assumed that good delivery schedules result in enough production to cover everyone's fixed costs. So, any such schedule (combined with appropriate prices) will correspond to a Nash equilibrium: agents will not want to deviate, since the large cost of a punishment will more than offset the short-term gain.

D.3 Subgame Perfection

We have shown that a good delivery schedule yields a Nash equilibrium. We want something stronger than a Nash equilibrium, though: if our punishments aren't credible, they will not deter agents from deviating. Normally we would rule out incredible threats by showing that our equilibrium is subgame perfect, that is, that the punishment policies are themselves equilibria (perhaps containing their own punishments which are also equilibria, and so forth). However, subgame perfection is not very satisfying here: in each punishment policy, no transactions are supposed to take place among some subset of the agents. To improve the payoffs, we would need deviations by two agents, a buyer and a seller. So, no single deviation is worthwhile, and we get subgame perfection for free.

With a couple of additional assumptions, we can prove a more satisfying result: any single deviating transaction (composed of simultaneous deviations by a buyer and a seller) will strictly lower the payoff to at least one of the deviating parties. To make this statement precise, we need to specify punishments within the punishment policies. We will say that, within the punish-manufacturer policy, if supplier i sells to the manufacturer, the other agents will all switch to the punish-supplier-i policy. Similarly, within punish-supplier-i, if the manufacturer deviates and buys from i, the suppliers will all switch to the punish-manufacturer policy; and if supplier j deviates, everyone will switch to punish-supplier-j. In other words, the punishments form a state machine: once anyone deviates, everyone always punishes someone who participated in the most recent deviation.

Now the manufacturer knows that dealing with an ostracized supplier will result in an inability to buy parts in the future. So, as long as the manufacturer can still run its factory without the ostracized supplier, the punish-manufacturer policy will be a strict deterrent. Therefore, we will assume that no crucial part has just a single feasible supplier.

To rule out deviations from the punish-manufacturer policy, we will assume that the punish-supplier-i policy yields significantly lower payoff for supplier i than punish-manufacturer does (perhaps because it reduces opportunities for i to sell to outside manufacturers as well). Then no supplier will want to break the embargo: it will gain by selling one part, but it will condemn itself to being ostracized thereafter. (For concreteness, we will split a supplier's fixed costs and its revenues from outside sales evenly among its trucks. So, no single truck from supplier i will want to sell a part and condemn itself to suffer the punish-supplier-i policy thereafter.)

With these assumptions, we have guaranteed that any desireable delivery schedule can be supported in an equilibrium, and that all threats used to sustain this equilibrium are credible. This guarantee justifies our decision to use the schedule and prices to specify an agreement among the agents. It also justifies our decision to treat each truck as a separate agent: since any sufficiently profitable schedule corresponds to an equilibrium, we haven't lost any useful equilibria by doing so.

D.4 Bargaining

As suggested in Section 4.3, we imagine enumerating every possible delivery and payment schedule. (This set is infinite, but we can approximate it arbitrarily well with finite representations.) We then pick some easily-implemented joint strategy—say, the punish-manufacturer strategy from above—as a disagreement point.

Having done so, our market becomes a Nash bargaining problem like the one in Figure 3. For a given proposed agreement, we can calculate the resulting total discounted payoff to every agent (once all of the agents fill in the details with their separate planners). The feasible set S is the set of these payoff vectors, while the disagreement point d is the payoff vector for punish-manufacturer. Given the bargaining problem (S, d), we can set up a negotiation system like the multi-player Rubinstein game to encourage the players to agree on a mutually beneficial schedule; the details of this game are presented in Section D.5.

There are couple of questions we need to answer before proceeding, though. First, we need to say why the agents should pick one disagreement point over another: if the agents can't agree on the effect of disagreement, the Nash bargaining problem is undefined. We will answer this question by saying that, as designers, we can inform the agents what the disagreement strategy is. Such a statement is self-fulfilling: because of it, each agent will expect that the others will follow the prescribed disagreement strategy if negotiations break down, and so it will follow the prescribed strategy as well.

Second, we need to deal with the possibility that the set S is not downward

```
sched ← ∅
repeat
  done ← true
  for each agent i
    choose(i)
      i says "pass"
      i adds a schedule or fragment to sched; done ← false
    end choose
  end for
  With probability ε, done ← true
until done
```

Figure 5: Phase I of the negotiation protocol.

closed; as noted in Section B, the multi-player Rubinstein game only makes sense for downward-closed utility sets. To get around this difficulty, we will again exercise our license as designers and give each agent the ability to voluntarily punish itself. In such a punishment, the agent gives up any desired amount of utility without affecting any other agent. This modification ensures that S is downward closed. (Self-punishment was put to a similar use in [20].)

We would not expect self-punishment to be needed in practice, since any strategy that includes it is Pareto-dominated. However, the mere possibility of self-punishment lets us implement the multi-player Rubinstein game. At the end of negotiation, if agent i has not accepted any proposal, the final plan can let i earn more than her disagreement value but require her to give up all or almost all of the excess. (This requirement is enforced by the threat that the other agents will switch to the punish-i policy.) The possibility of getting only her disagreement value is what motivates i to accept reasonable offers during negotiation.

D.5 Market protocol

Above, we left unspecified the exact protocol for making and accepting offers of delivery and payment schedules. In this section we suggest a more detailed protocol based on the one described in [20]. This protocol is intended to make it easier to analyze how agents should negotiate with one another: after the first phase of the protocol, even computation-limited agents can describe all of the possible equilibria.

Our protocol proceeds in two phases. In the first phase, any agent can broadcast schedules or fragments of schedules to the others. The others can then use these schedules or fragments as starting points for their own further planning. In the second phase, the agents pick from the complete schedules proposed in the first phase in a Rubinstein-like game.

```
for each agent i
  utility[i] ← d_i
  accepted[i] ← false
end for
repeat
  for each agent i
    if accepted[i] then continue
    i proposes a distribution s over complete schedules from sched
    done ← true
    for each agent j ≠ i
      if accepted[j] then continue
      u ← utility of s to j
      choose(j)
        j says "accept"; utility[j] ← u; accepted[j] ← true
        j says "reject"; done ← false
      end choose
    end for
    if done then
      utility[i] ← utility of s to i
      return
    end if
  end for
  With probability ε, done ← true
until done
```

Figure 6: Phase II of the negotiation protocol.

Pseudocode for the two phases appears in Figures 5 and 6. The **choose**(i) statement marks a place in the protocol where agent i gets to choose one of several alternatives: i picks which of the lines inside the **choose**/**end** pair will execute. The parameter ϵ is an arbitrary small positive number which determines whether we force a phase to end early; it should be small enough that there is little risk of the protocol ending before the agents want it to, but large enough that the agents feel pressure to arrive at an agreement rather than stalling forever. At the end of Phase I, the set sched contains the schedules which the agents will bargain over in Phase II. At the end of Phase II, the utility[i] array contains the agreed utilities for each agent. We can choose any schedule or distribution over schedules which attains these utilities, so long as we tell the agents ahead of time how to break ties. In general we may need to use self-punishment to achieve the desired utilities.

Just like the multi-player Rubinstein game, Phase II has a unique subgame-perfect equilibrium, near the Nash bargaining point of the set of utilities of the complete schedules that are in sched at the end of Phase I. Finding the Nash point is a convex optimization problem, and can therefore be solved efficiently;[3] so, even computation-limited agents can figure out how they should play in Phase II.

Because of the unique, easy-to-compute equilibrium in Phase II, we can give a coarse description of how the agents should play in Phase I: they should nominate schedules that give themselves high payoffs, to try to steer the outcome in their favor; but, they will also want to nominate schedules that give high payoffs to other agents, because such schedules are more likely to eventually be incorporated into the plan accepted by the group as a whole.

D.6 Optimization

As noted above, our design decisions have given each agent a number of smaller planning problems which it must solve repeatedly. For example, the manufacturer must decide how to set up its factory and how to use each part as it is delivered. The trucks must plan tours of the warehouses, with each tour spanning the interval from one dropoff at the factory to the next. And, within each tour, a truck must repeatedly plan a path from one warehouse to the next. (For even more examples of similar planning problems, see the descriptions of the players from recent Trading Agent Competitions [11].)

Also as noted above, there may be uncertainty in the problem data for these planning problems. For example, there may be traffic congestion on the path from one warehouse to another, the price of a part may change between the time a truck starts its tour and the time it reaches the warehouse, or the agents may agree on a change to the delivery schedule which forces changes in their individual plans.

[3] It is easy to see that the feasible region is convex, since it is the convex hull of the utility vectors specified in Phase I. The objective is to maximize the product of excess utilities; while this product is not generally a convex function, it is equivalent to maximize the sum of the logs of the excess utilities, which is convex.

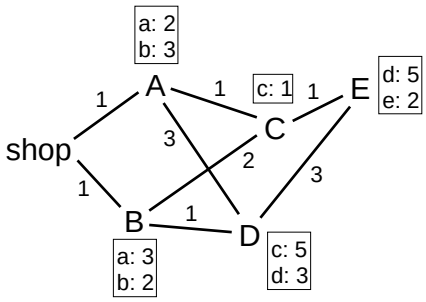

Figure 7: An instance of one of the planning problems which a truck needs to solve. The truck starts at the shop, and must deliver parts b and e by visiting a subset of the warehouses and returning to the shop. For ease of display, this instance has perfectly-known parameters.

In principle the agents could look for opportunities for cooperation arising from such variation: for example, a truck could agree to change its route slightly in order to reduce the amount of congestion on a road which some other truck needs. Our factorization of the problem means that we have given up on trying to find such opportunities for cooperation. We are willing to do so because we believe such opportunities do not greatly affect the overall payoff.[4]

Even though we have given up on cooperation in these areas, we don't want to ignore the behavior of the other agents altogether: if we assume that we know the problem data for each optimization exactly, we risk making plans which are brittle to changes in the data. This brittleness could result in poor overall performance.

To illustrate this problem, in this section we will describe a more detailed version of the planning problem facing an individual truck. The less detailed version, described in Section 4.4, is a travelling salesman problem with uncertain costs.

Our planning problem is illustrated in Figure 7. Starting at the factory, the truck must drive to some of the warehouses, pick up a specified set of parts, and return to the factory to deliver them before a specified deadline. A plan is therefore a tour connecting the shop with a subset of the warehouses, augmented with labels for each visited warehouse saying which parts the truck will buy there. For example, to buy parts b and e with deadline $t = 7$, the truck could take the tour $BCECA$, buying part b at warehouse B and part e at warehouse E. The length of this tour is 7, so the truck arrives exactly at the deadline; the cost is 4, since each part costs 2.

To specify an instance of this problem, we need the cost for each part at

[4]To expose different opportunities for cooperation, we can change our design decisions. For example, if we think road congestion is important, we can institute a market where the trucks can buy and sell tickets that allow them to pass through congested spots. Such a market, if it is well designed and if the trucks can learn how to participate effectively in it, will tend to shift the ownership of the tickets to those who can use them most profitably.

each warehouse, the length of each edge, the list of required parts, the deadline, and the cost for failure. All of these parameters are potentially uncertain, and if we do not take this uncertainty into account, our plans may be brittle. In the example given above, the tour $BCECA$ is optimal when the parameters are known. But if the deadline or the edge lengths are uncertain, planning to arrive exactly at the deadline is risky. In this case the tour $ACECA$ may be better: while its cost for parts is 5 instead of 4 (3 for b at A and 2 for e at E), its path length is 6 instead of 7, so it has a better chance of avoiding the failure penalty.

Above we recommended using no-regret learning algorithms to handle this sort of uncertainty. In particular, we recommended using a no-regret online convex programming algorithm such as follow-the-perturbed-leader [26, 34].

To solve the problem using FPL (or any other online convex programming algorithm), we can specify our tour using the variables

e_{ij}	How many times do we use the edge between warehouses i and j?
p_{ik}	How many copies of part k do we buy at warehouse i?

for all warehouses i and j and parts k. Our total parts cost and total runtime are then linear functions of e_{ij} and p_{ij}:

$$\begin{aligned} C &= \sum_{ik} p_{ik} c_{ik} \\ T &= \sum_{ij} e_{ij} t_{ij} \end{aligned}$$

where c_{ik} is the cost of part k at warehouse i and t_{ij} is the observed length of edge ij.

Our loss for a given tour is a nonconvex function of C and T: if the actual deadline is T_0 and the failure penalty is F, the total loss is

$$L = C + F\, s(T - T_0)$$

where $s(\cdot)$ is a unit step function. For learning purposes we will replace L by a convex bound

$$\overline{L} = C + F\, h(T - T_0)$$

where $h(\cdot)$ is a convex hinge loss function which upper bounds $s(\cdot)$,

$$h(t) = \max\{0, 1 + t/\epsilon\}$$

Here ϵ is a target margin: $h(\cdot)$ will be 0 if $T \leq T_0 - \epsilon$, while it will be at least 1 if $T \geq T_0$.

So, if we define X to be the convex hull of the set of feasible tours (where each tour is represented using the vector of variables described above), we want to minimize

$$\overline{L} = \sum_{ik} p_{ik} c_{ik} + F\, h\left(\sum_{ij} e_{ij} t_{ij} - T_0\right)$$

over the feasible region $(e_{ij}, p_{ij}) \in X$.[5] This is a convex problem, so we can run FPL to find good settings for e_{ij} and p_{ij}.

There are two slight difficulties with applying FPL. First, FPL is designed for a problem with a linear objective. To fix this problem we can replace $\overline{L}$ for each instance with its tangent at the current value of e_{ij} and p_{ij} [23, p. 54]. That is, if we observe $T \leq T_0 - \epsilon$ for the current example, we replace $\overline{L}$ with $\sum_{ik} p_{ik} c_{ik}$, while if we observe $T \geq T_0 - \epsilon$, we replace $\overline{L}$ with

$$\sum_{ik} p_{ik} c_{ik} + (F/\epsilon) \sum_{ij} e_{ij} t_{ij} - F T_0 / \epsilon$$

Second, each iteration of FPL requires solving a linear program whose feasible region is X. We cannot represent X explicitly, since there will generally be exponentially many feasible tours. But, solving an LP over X is equivalent to finding an optimal tour with known c_{ik} and t_{ij}, and we can find the best tour using a combinatorial optimizer. (This may take exponential time in the worst case, but there are many practically-efficient algorithms for finding good tours in similar problems. In addition, because there are a limited number of warehouses, the problem instances are small.)

FPL now provides the following guarantee: our average cost L for buying b and e will not be much more than the best convex-cost $\overline{L}$ that we could have achieved on average if we had known the uncertain parameters in advance. In particular, if it was possible to buy b and e for an average cost C while beating each deadline by at least ϵ, then our average cost will be not much more than C per trial.

[5] For clarity, we emphasize that X is not an LP relaxation of the region of feasible tours; X is the exact feasible region for an NP-complete optimization problem. Interior points of X represent distributions over tours.