



Bargaining with limited computation: Deliberation equilibrium[☆]

Kate Larson^{*}, Tuomas Sandholm

Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Received 6 November 2000; received in revised form 11 April 2001

Abstract

We develop a normative theory of interaction—negotiation in particular—among self-interested computationally limited agents where computational actions are game theoretically treated as part of an agent's strategy. We focus on a 2-agent setting where each agent has an intractable individual problem, and there is a potential gain from pooling the problems, giving rise to an intractable joint problem. At any time, an agent can compute to improve its solution to its own problem, its opponent's problem, or the joint problem. At a deadline the agents then decide whether to implement the joint solution, and if so, how to divide its value (or cost). We present a fully normative model for controlling anytime algorithms where each agent has statistical performance profiles which are optimally conditioned on the problem instance as well as on the path of results of the algorithm run so far. Using this model, we introduce a solution concept, which we call *deliberation equilibrium*. It is the perfect Bayesian equilibrium of the game where deliberation actions are part of each agent's strategy. The equilibria differ based on whether the performance profiles are deterministic or stochastic, whether the deadline is known or not, and whether the proposer is known in advance or not. We present algorithms for finding the equilibria. Finally, we show that there exist instances of the deliberation–bargaining problem where no pure strategy equilibria exist and also instances where the unique equilibrium outcome is not Pareto efficient. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Automated negotiation; Multiagent systems; Game theory; Resource-bounded reasoning; Bounded rationality; Bargaining; Anytime algorithm

[☆] A short early version of this paper appeared in the proceedings of the National Conference on Artificial Intelligence (AAAI), Austin, TX, August 2000.

^{*} Corresponding author.

E-mail addresses: klarson@cs.cmu.edu (K. Larson), sandholm@cs.cmu.edu (T. Sandholm).

1. Introduction

Systems, especially on the Internet, are increasingly being used by multiple parties—or software agents that represent them—with their own preferences. This invalidates the traditional assumption that a central designer controls the behavior of all system components. The system designer can only control the *mechanism* (rules of the game), while each agent chooses its own *strategy*. The economic efficiency that a system yields depends on the agents' strategies. So, to develop a system that leads to desirable outcomes, the designer has to make sure that each agent is motivated to behave in the desired way. This can be achieved by analyzing the game using the *Nash equilibrium* solution concept from game theory (or its refinements): no agent is motivated to deviate from its strategy given that the others do not deviate [19,20].

However, the equilibrium for rational agents does not generally remain an equilibrium for computationally limited agents.¹ This leaves a potentially hazardous gap in game theory as well as automated negotiation (see, for example, [15,24,31]) because computationally limited agents are not motivated to behave in the desired way. This paper presents a framework and first steps toward filling that gap.

In this paper we begin to develop a theory of interaction—negotiation in particular—where computation actions are treated as part of an agent's strategy. We study a 2-agent bargaining setting where at any time, the agent can compute to improve its solution to its own problem, its solution to the opponent's problem, or its solution to the joint problem where the tasks and resources of the two agents are pooled. The bargaining occurs over whether or not to use a solution to the joint problem, and how to divide the associated value or cost.

Early on, it was recognized that humans have bounded rationality, for example, due to cognitive limitations, so they do not act rationally as economic theory would predict [7, 35]. Since then, considerable work has focused on developing *normative* models that prescribe how a computationally limited agent *should* behave (see, for example, [5,9, 26]). This is a highly nontrivial undertaking, encompassing numerous fundamental and technical difficulties. As a result most of those methods resort to simplifying assumptions such as myopic deliberation control [3,28,29], conditioning the deliberation control on hand-picked features [28,29], assuming that an algorithm's future performance can be deterministically predicted using a performance profile [10,11], assuming that an anytime algorithm's future performance does not depend on the run on that instance so far [4,9,38, 39] or that performance is conditioned on quality so far but not the path [8], or resorting to asymptotic notions of bounded optimality [27].

While such simplifications can be acceptable in single-agent settings as long as the agent performs reasonably well, any deviation from full normativity can be catastrophic in multiagent settings. If the designer cannot guarantee that the strategy (including deliberation actions) is the best strategy that an agent can use, there is a risk that an

¹ In the relatively rare settings where the incentives can be designed so that each agent is motivated to use the desired strategy *independent of what others do* (dominant strategy equilibrium), a rational agent is best off maintaining its strategy even if some other agents do not act rationally, for example, due to computational limitations.

agent is motivated to use some other strategy. Even if that strategy happens to be “close” to the desired one, the social outcome may be far from desirable. Therefore, a fully normative deliberation control method is required as a basis for analyzing each agent’s best strategy. This paper introduces such a fully normative deliberation control method. It takes into account that each agent may use all the information it has available to control its computation, including conditioning on the problem instance and the path of solutions found on the run so far. This paper will discuss how this deliberation control method can be used as a basis for deciding on an agent’s best-response strategy: what deliberation actions and negotiation actions (offers, acceptances, and rejections) the agent should execute at any point in the game.

Game theorists have also realized the significance of computational limitations (see, for example, [25]), but the models that address this issue have mostly analyzed how complex it is to compute the rational strategies [14] (rather than the computation impacting the strategies), memory limitations in keeping track of history in repeated games via deterministic finite automata or Turing machines (see, for example, [1,6,22]), limited uniform-depth lookahead capability in repeated games [12], or showing that allowing the choice between taking one computation action or not undoes the dominant strategy property in a Vickrey auction [32]. On the other hand, in this paper, the limited rationality stems from the complexity of each agent’s optimization problem (each agent has a computer of finite speed, some anytime algorithm which might not be perfect, and finite time), a setting which is ubiquitous in practice.²

In this paper we investigate an ultimatum game where agents must first use their limited computational resources in order to determine whether any bargaining should occur, and if so, what the agents are actually bargaining over, that is, what is the value they are trying to agree to split. While bargaining has been well studied in the economics and game theory literature (see, for example, [21]), most models of bounded-rational agents assume that the agents know what they are bargaining over *a priori* but must learn which strategies work well [30]. Instead, in our model, agents have to use their limited resources in order to determine exactly what they are bargaining over.

2. An example application

To make the presentation more concrete, we now discuss an example domain where our methods are needed. Consider a distributed vehicle routing problem [33] with two geographically dispersed dispatch centers that are self-interested companies (Fig. 1). Each center is responsible for certain tasks (deliveries) and has a certain set of resources (vehicles) to take care of them. So each agent—representing a dispatch center—has its own vehicles and delivery tasks.

Each agent’s *individual problem* is to minimize transportation costs (driven mileage) while still making all of its deliveries while honoring the following constraints [33]:

² The same *source* of complexity has been addressed [33], but that paper only studied outcomes, not the process or the agents’ strategies. It was also assumed that the algorithm’s performance is deterministically known in advance. Finally, the agents had costly but unlimited computation, while in this paper the agents have free but limited computation.

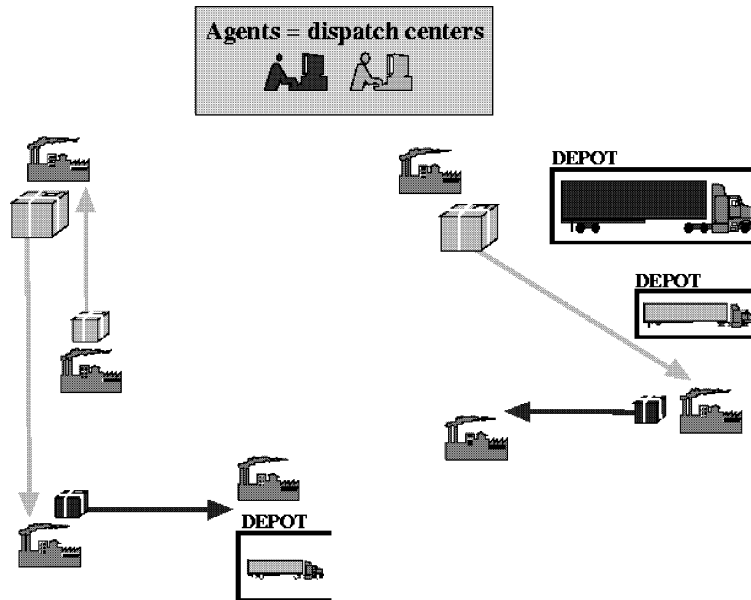


Fig. 1. Small example problem instance of the distributed vehicle routing problem. This instance has two dispatch centers represented in the figure by computer operators. They receive the delivery orders and route the vehicles. The light dispatch center has light tasks and trucks while the dark dispatch center has darker tasks and trucks. The dispatch centers receive all of their delivery orders at once, and then have some time to compute a routing solution before the trucks need to be dispatched. For example, in some practical settings, the delivery tasks are known by Friday evening and the route plan for the next week has to be ready by Monday morning when the trucks need to be dispatched [33].

- Each vehicle has to begin and end its tour at the depot of its center (but neither the pickup nor the drop-off locations of the orders need to be at the depot).
- Each vehicle has a maximum load weight constraint. These may differ among vehicles.
- Each vehicle has a maximum load volume constraint. These may differ among vehicles.
- Each vehicle has a maximum route length (prescribed by law).
- Each delivery has to be included in the route of some vehicle.

An agent's individual problem is NP-hard since ΔTSP^3 can be trivially reduced to it. The problem is in NP because the cost and feasibility of a solution can be checked in polynomial time. Therefore, the problem is NP-complete.

The geographical operation areas of the centers overlap. This creates the potential for either center to handle a delivery. There is a potential for savings in driven mileage by pooling the agents' tasks and resources since one agent may be able to handle some of the other's tasks with less driving than the other due to adjacency. The objective in this *joint problem* is to again minimize driven mileage. This problem is again NP-complete.

³ The ΔTSP is a Traveling Salesman Problem where the distances between cities satisfy the triangle inequality.

Whether the agents actually decide to coordinate their deliveries is determined by the costs associated with the different solutions. The agents must negotiate, or bargain, about whether to independently deliver their own packages, or whether to share their delivery tasks and resources in order to reduce costs. They must also negotiate as to how they will split the costs and benefits of the joint solution, if they agree to carry out a joint solution. However, before agents can decide whether to carry out a joint solution or the two individual solutions, they must have solutions (possible routes) for the three problems.

3. The general setting

The distributed vehicle routing problem is only one example problem where the methods of this paper are needed. In general, they are needed in any setting with two self-interested agents where each agent has an intractable *individual problem*, and there is a potential savings from pooling the problems, giving rise to an intractable *joint problem*. We also assume that the value of any solution to an agent's individual problem is not affected by what solution the other agent uses for its individual problem.

Applications with these characteristics are ubiquitous, including transportation as discussed above, manufacturing (where two companies that potentially subcontract with each other need to construct their manufacturing plans and schedules), electric power negotiation between a custom provider and an industrial consumer (where the participants need to construct their production and consumption schedules), classroom scheduling, scheduling of scientific equipment among multiple users, and bandwidth allocation and routing in multi-provider multi-consumer computer networks to name just a few.

In order to determine the gain generated by pooling instead of each agent operating individually, agents need to compute solutions to both agents' individual problems as well as to the joint problem. We assume that the agents have anytime algorithms that can be used to solve problems so that some feasible solution is available whenever the algorithm is terminated, and the solution improves as more computation time is allocated to the algorithm.

By computing on the joint problem, an agent reduces the amount of time it has for computing on its individual problem. This may increase the joint value to the agents (reduce the sum of the agents' costs), but makes this agent's fallback position worse when it comes to bargaining over how the joint value should be divided between the two agents. Also, if one agent is computing on the joint problem, would it not be better for the other agent to compute on something different so as not to waste computation? In this paper we present a model where each agent strategically decides on how to use its limited computation in order to maximize its own expected payoff in such settings.

4. The model

In this section we introduce the computational bargaining model. There are two distinct parts to the model: the deliberation control part and the bargaining part. However, the actions that an agent takes in one part affect the actions that the agent should take in the

other part, as well as the actions that the other agents will take in both parts. So, although the deliberation precedes bargaining, these two stages are deeply interrelated as we will show.

Let there be two agents, α and β , each with its own *individual problem*. They also have the possibility to pool, giving rise to a *joint problem*. We assume that time is discretized into time units and each computational step takes one time unit. However, time is limited, so agents can deliberate for at most T time steps. Agents must decide how and when to compute on the three problems (the two individual problems and the joint problem), and which offers to make and which to accept. Each agent makes these decisions online based on the results of its computations up to that point (which can change the agent's payoff expectations and its expected gain from different future computations).

In this section we introduce a normative deliberation control method that captures the possibilities that agents have in controlling their computation and how it affects the bargaining process.

4.1. Normative control of deliberation

Each agent has an anytime algorithm that has a feasible solution available whenever it is terminated, and improves the solution as more computation time is allocated to the problem. Let $v_\alpha^\alpha(t)$ be the value of the solution to agent α 's individual problem after computing on it for t time steps. Similarly, $v_\alpha^\beta(t)$ is the value of the solution to agent β 's individual problem after agent α has computed on it for t time steps. Finally, $v_\alpha^{\text{joint}}(t)$ is the value of the solution to the joint problem after computing on it for t time steps.

The agents have statistical performance profiles that describe how their anytime algorithms improve the solutions as a function of the allocated computation time. As will be discussed later, each agent uses this information to decide how to allocate its computation at every step of the game, optimally striking a tradeoff between computation time and solution quality.

A common representation of performance profiles is a table of discrete values [8,39]. This approach requires discretizing time into a finite number of time steps and solution quality into a finite number of solution levels. For each time step and each level of solution quality, the table contains the probability that the solution will be of that quality. The resolution of the discretization determines a tradeoff between accuracy of the performance profile and the amount of data needed (and the space needed to store it) to populate the space of performance profile.

Instead of using a table of discrete values to represent the performance profile, we propose storing the values in a tree structure. While this does not change the tradeoff between accuracy and required data, using a tree structure allows for optimal conditioning on results of execution so far which the earlier methods do not support.

We index the problem (agent α 's, agent β 's, and the joint) by z , $z \in \{\alpha, \beta, \text{joint}\}$. For each z there is a performance profile tree, \mathcal{T}_i^z , representing the fact that an agent i can condition its algorithm's performance profile on the problem instance. Fig. 2 exemplifies one such tree. Each depth of the tree corresponds to the time t of the run that the algorithm has executed on that problem instance. Each node at depth t of the tree represents a possible solution quality (value), v_i^z , that is obtained by running the algorithm for t time steps on

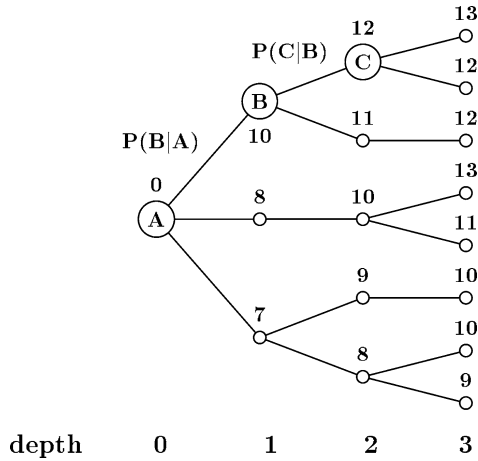


Fig. 2. A performance profile tree.

that problem. There may be several nodes at a depth since the algorithm may reach different solution qualities for a given amount of computation depending on the problem instance. At any depth there may be more than one node with a given value as the path taken to reach a certain value may be different depending on the problem instance. We assume that the solution quality in the performance profile tree, \mathcal{T}_i^α , of agent α 's individual problem is discretized into a finite number of levels. Similarly, the solution quality in \mathcal{T}_i^β is discretized into a finite number of levels, as is the solution quality in $\mathcal{T}_i^{\text{joint}}$.

Each edge in the tree is associated with the probability that the child is reached in the next computation step given that the parent has been reached. This allows one to compute the probability of reaching any particular future node in the tree given the node that has been reached so far. This is accomplished by multiplying the probabilities on the path between these two nodes. If there is no path, the probability is 0.

The tree is constructed by collecting statistical data from previous runs of the algorithm on different problem instances. The more finely solution quality and time are discretized, the more accurate deliberation control is possible. However, with more refined discretization, the number of possible runs increases (it is $O(m^D)$ where m is the number of levels of solution quality and D is depth of the tree), so more runs need to be seen to populate the space. A tighter bound can be obtained once the observation is made that the values of the solutions are always increasing and can be represented as step functions. The bound is $O(Nd)$ where N is the number of leaves in the tree and d is the average depth [3]. Furthermore, the space should be populated densely to get good probability estimates on the edges of the performance profile trees.⁴ Each run is represented as a path in the tree. As a run proceeds along a path in the tree, the frequency of each edge of that path is incremented, and the frequencies at the nodes on the path are normalized to obtain probabilities. If the run leads to a value for which there is no node in the tree, the node is generated and an edge is inserted from the previous node to it.

⁴ If the algorithm is stochastic, variability can occur even across multiple runs on the same instance.

We denote by $\text{time}(n)$ the depth of node n in the performance profile tree. In other words, $\text{time}(n)$ is the number of computation steps used to reach node n . We denote by $V(n)$ the value of node n .

Definition 1. The *state of deliberation* of agent α at time step t is

$$\theta_\alpha(t) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle,$$

where n_α^α , n_α^β , and n_α^{joint} are the nodes where agent α is currently in each of the three performance profile trees and $\text{time}(n_\alpha^\alpha) + \text{time}(n_\alpha^\beta) + \text{time}(n_\alpha^{\text{joint}}) = t$. The state of deliberation for agent β is defined analogously.

In practice it is unlikely that an agent knows the solution quality for every time allocation without actually doing the computation. Rather, there is uncertainty about how the solution value improves over time. Our performance profile tree allows us to capture this uncertainty. For example, with a depth t search in the tree one can determine $P(v^z | t)$, denoting the probability that running the algorithm for t time steps produces a solution of value v^z .

The *deliberation set* is the set of deliberation states that an agent can reach in exactly t deliberation actions.

Definition 2. The *deliberation set* of agent α at time t is

$$\Theta_\alpha(t) = \{ \theta_\alpha(t) \mid \text{time}(n_\alpha^\alpha) + \text{time}(n_\alpha^\beta) + \text{time}(n_\alpha^{\text{joint}}) = t \}.$$

The deliberation set for agent β is defined analogously.

We shall sometimes use the notation $\Theta_\alpha(t_\alpha^\alpha, t_\alpha^\beta, t_\alpha^{\text{joint}})$ to represent the restricted deliberation set

$$\Theta_\alpha(t_\alpha^\alpha, t_\alpha^\beta, t_\alpha^{\text{joint}}) = \{ \theta_\alpha(t_\alpha^\alpha + t_\alpha^\beta + t_\alpha^{\text{joint}}) \mid \text{time}(n_\alpha^\alpha) = t_\alpha^\alpha, \text{time}(n_\alpha^\beta) = t_\alpha^\beta, \text{time}(n_\alpha^{\text{joint}}) = t_\alpha^{\text{joint}} \}.$$

Unlike previous methods for performance profile based deliberation control, our performance profile tree directly supports conditioning on the *path* of solution quality so far.⁵ The performance profile tree that applies given a path of computation so far is simply the subtree rooted at the current node n . We denote this subtree by $\mathcal{T}^z(n)$. If an agent is at a node n with value v , then when estimating how much additional deliberation would increase the solution value, the agent need only consider paths that emanate from node n . The probability, $P_n(n')$, of reaching a particular future node n' in $\mathcal{T}^z(n)$ given that the current node is n is simply the product of the probabilities on the path from n to n' .

⁵ Our results apply directly to the case where the conditioning on the path is based on other solution features in addition to solution quality. For example, in a scheduling problem, the distribution of slack can significantly predict how well an iterative refinement algorithm can further improve the solution.

Similarly, given that the current node is n , the expected solution quality after allocating t more time steps to this problem is

$$\sum_{\{n'|n' \text{ is a node in } \mathcal{T}^z(n) \text{ with depth } t\}} P_n(n') \cdot V(n').$$

This can be easily computed using depth-first search with a depth limit t in $\mathcal{T}^z(n)$.

Computation plays several strategic roles in the game. First, it improves the solution that is available—for any one of the three problems. Second, it resolves some of the uncertainty about what future computation steps will yield. Third, it gives information about what solution qualities the opponent has encountered and can expect. This helps in estimating what solution quality the other agent has available on any of the three problems. It also helps in estimating what computations the other agent might have done and might do. Therefore, in equilibrium, an agent may want to allocate computation on its individual problem, the joint problem, and even on the opponent's problem. Agents may not share algorithms for the problems, and so may obtain different results for the solution of the joint problem. However, if one agent has computed a high value for the joint, then it is likely that the other agent will also be able to obtain a high value. The agents know this, and, where applicable, can use this knowledge to speculate as to how the other agent is using its deliberation resources. We will show how agents use the performance profile trees to optimally handle these considerations.

4.1.1. Special case: Deterministic performance profiles

In a deterministic performance profile, the algorithm's performance can be projected with certainty. In this setting, the tree that represents the performance profile has only one path. Before using any computation, an agent can determine what the value will be after any number of computation steps devoted to any problem z , that is, $v^z(t) \in \mathbb{R}$ is known for all t . So, computation does not provide any information about the expected results of future computations. Also, computation does not provide any added information about the performance profiles, which could be used to estimate the other agent's computational actions.

As will be presented later, in some of our settings where the performance profiles are not deterministic, we assume that the agents have the same performance profile trees \mathcal{T}^α , \mathcal{T}^β , and $\mathcal{T}^{\text{joint}}$. In some of our settings we additionally assume that \mathcal{T}^α , \mathcal{T}^β , and $\mathcal{T}^{\text{joint}}$ are common knowledge. One scenario where the agents have the same performance profile trees is where the agents use the same algorithm and have seen the same training instances. This is arguably roughly the case in practice if the parties have been solving the same type of instances over time, and the algorithms have evolved through experimentation and publication. In settings where the performance profiles are deterministic, all of our results go through even if the agents have different performance profile trees $\mathcal{T}_\alpha^\alpha$, \mathcal{T}_α^β , $\mathcal{T}_\alpha^{\text{joint}}$, \mathcal{T}_β^α , \mathcal{T}_β^β , and $\mathcal{T}_\beta^{\text{joint}}$. Yet in some of these settings we assume that $\mathcal{T}_\alpha^\alpha$, \mathcal{T}_α^β , $\mathcal{T}_\alpha^{\text{joint}}$, \mathcal{T}_β^α , \mathcal{T}_β^β , and $\mathcal{T}_\beta^{\text{joint}}$ are common knowledge.

4.2. Bargaining

The term *bargaining* is used to refer to a situation in which:

- (1) Agents have the possibility of concluding a mutually beneficial agreement.
- (2) There is a conflict of interests about which agreement to conclude.
- (3) No agreement may be imposed on any individual without its approval.

In our setting agents bargain over how to divide the surplus (or cost) associated with implementing the joint solution. Each agent prefers to receive more rather than less and each has the possibility to opt out of the bargaining procedure and to implement its individual solution with the associated value v .

At some point in time, T , there is a deadline at which time both agents must stop deliberating and decide how they will execute the solutions based on the outcome of a bargaining round. The agents perform their computational actions in parallel with no communication between them until the deadline is reached. Call the value of the solution computed by the deadline by agent $i \in \{\alpha, \beta\}$ to agent α 's problem v_i^α , to agent β 's problem v_i^β , and to the joint problem v_i^{joint} . Through bargaining, the agents decide whether to pool or not, and in the former case they also decide how to divide the value of the solution to the joint problem. If the value of the solution to the joint problem is higher than the sum of the values of the solutions to the individual problems, then there is a potential gain from agreeing to implement the joint solution.

We restrict the bargaining so that only one agent is allowed to make an offer, while the other agent has the ability to either accept or reject the offer made (that is, the agents are involved in an ultimatum game). If a proposal is accepted, the joint solution is implemented and the surplus is divided as determined by the agreed upon proposal. If no agreement is reached then the agents implement their individual solutions with no further interaction.

The values that have been computed by the agents affect the bargaining process and outcome. For example, if both agents decide to devote no computation on the solution for the joint problem, then it is unlikely that any agreement will be reached in the bargaining process on whether to execute the joint solution. Instead, both agents would likely act independently, implementing their own individual solutions. If, on the other extreme, both agents had computed only on the solution for the joint problem, then it is more likely that agreement will be reached. The bargaining strategies of the agents are determined by the values they have computed on all problems. The offer that an agent makes is determined by the value of the joint solution that it has computed as well as the value it has obtained for its individual solution and on the value it believes the other agent has computed for its own individual solution. Similarly, the offer that an agent will accept is determined by the value that it has computed for its individual problem, since that is its fallback value (that is, the agent is guaranteed to receive at least that amount if the agreement is not reached).

Say that agent α is the proposer. It makes a take-it-or-leave-it offer, x_α^o , to the other agent, β , about how much agent β 's payoff will be if they pool.⁶ Agent β can then accept or reject. If agent β accepts the offer, the agents pool and use agent α 's solution to the joint problem. Agent β 's payoff is x_α^o as proposed and agent α gets the rest of the value

⁶ We allow an agent to make a negative “unacceptable” offer which signals that it does not want to coordinate or implement the joint solution.

Table 1

If agent α makes an offer, x_α^o , to agent β , then agent β has the choice of either accepting or rejecting it. The payoffs for the agents in either situation are listed in the table above

Agent	Payoff if the offer is accepted	Payoff if the offer is rejected
α	$v_\alpha^{\text{joint}} - x_\alpha^o$	v_α^α
β	x_α^o	v_β^β

of the solution: $v_\alpha^{\text{joint}} - x_\alpha^o$. If agent β rejects, both agents implement their own computed solutions to their own individual problems, in which case agent α 's payoff is v_α^α and agent β 's payoff is v_β^β . The payoffs are presented in Table 1.

Before the deadline, the agents may or may not know which one of them is the proposer. In any case, if the agents agree to implement the joint solution, the joint solution computed by the proposer is used. In our model the probability that agent α will be the proposer is P_{prop} , and this is common knowledge. When agents reach the bargaining stage, each agent's strategy is captured by an *offer-accept vector*. An offer-accept vector for agent α is $OA_\alpha = (x_\alpha^o, x_\alpha^a) \in \mathbb{R}^2$, where x_α^o is the amount that agent α would offer if it were the proposer, and x_α^a is the minimum value it would accept if agent β made the proposal. The offer-accept vector for agent β is defined similarly.

4.3. Definition of strategies

The agents' strategies incorporate actions from both the deliberation part and the bargaining part of the game. For the deliberation part of the game, an agent's strategy is a mapping from the state of deliberation to the next deliberation action (that is, selecting which solution z , $z \in \{\alpha, \beta, \text{joint}\}$ to compute another time step on—in words, whether to compute on the agent's own problem, the other agent's problem, or the joint problem).

Definition 3. A *deliberation strategy* for agent α with deadline T is

$$S_\alpha^D = (S_\alpha^{D,t})_{t=0}^{T-1},$$

where

$$S_\alpha^{D,t} : \Theta_\alpha(t) \rightarrow \{a^\alpha, a^\beta, a^{\text{joint}}\}$$

is a mapping from a deliberation state at time t , $\theta_\alpha(t) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle$, to a deliberation action a^z where a^z is the action of computing one time step on the solution for problem $z \in \{\alpha, \beta, \text{joint}\}$. The deliberation strategy of agent β , S_β^D , is defined analogously.

In a deterministic setting, taking a deliberation action causes the agent to move into a specific state of deliberation. However, in the stochastic setting, if an agent is in a certain state of deliberation at time t , then the action of computing on a problem will cause the agent to be in any one of several states of deliberation at time $t + 1$. The probability with which an agent will enter into a specific state of deliberation is determined by the performance profiles.

At the deadline, T , each agent has to decide on its offer-accept vector. Therefore, the strategy at time T is a mapping from the state of deliberation at time T to an offer-accept vector.

Definition 4. A *bargaining strategy* for agent α with deadline T ,

$$S_\alpha^B : \Theta_\alpha(T) \rightarrow \mathbb{R}^2$$

is a mapping from a state of deliberation at time T , to an offer-accept vector, (x_α^o, x_α^a) . The bargaining strategy of agent β , S_β^B , is defined analogously.

An agent's strategy consists of a deliberation strategy and a bargaining strategy.

Definition 5. A *strategy* for agent α with deadline T is

$$S_\alpha = (S_\alpha^D, S_\alpha^B).$$

A strategy for agent β , S_β is defined analogously.

Our analysis will also allow *mixed strategies*. A mixed strategy for agent α is $\tilde{S}_\alpha = (\tilde{S}_\alpha^D, \tilde{S}_\alpha^B)$ where \tilde{S}_α^D is a mapping from a deliberation state $\theta_\alpha(t)$ to a probability distribution over the set of deliberation actions $\{a^\alpha, a^\beta, a^{\text{joint}}\}$. We let p^α be the probability that an agent takes action a^α , p^β be the probability that an agent takes action a^β , and therefore, $1 - p^\alpha - p^\beta$ is the probability that an agent takes action a^{joint} . The mixed bargaining strategy, \tilde{S}_α^B , is a mapping from a deliberation state $\theta_\alpha(T)$ to a probability distribution over offer-accept vectors.

5. Equilibria and algorithms

We want to make sure that the strategy that we propose for each agent—and according to which we study the outcome—is indeed the best strategy that the agent has from its self-interested perspective. This makes the system behave in the desired way even though every agent is designed by and represents a different self-interested real-world party. One approach would be to just require that the analysis shows that no agent is motivated to deviate to another strategy given that the other agent does not deviate. This would be the *Nash equilibrium* solution concept from noncooperative game theory [20]. We actually place a stronger requirement on our method. We require that at any point in the game, an agent's strategy prescribes optimal actions from that point on, given the other agent's strategy and the agent's beliefs about what has happened so far in the game. We also require that the agent's beliefs are consistent with the strategies. This type of equilibrium is called a *perfect Bayesian equilibrium (PBE)* [19]. We introduce a new equilibrium for computationally limited agents:

Definition 6. A (Nash, perfect Bayesian) *deliberation equilibrium* for computationally limited agents is a (Nash, perfect Bayesian) equilibrium where the agents' deliberation strategies form a (Nash, perfect Bayesian) equilibrium and the agents' bargaining strategies are in (Nash, perfect Bayesian) equilibrium.

An agent's offer-accept vector is affected by the solutions that it computes and also what it believes the other agent has computed for solutions. The *fallback* value of an agent is the value it obtained for the solution to its own problem. Clearly, an agent will not accept any offer less than its fallback.

In making a proposal, agent α must try to estimate agent β 's fallback value and then decide whether, by making an acceptable proposal to agent β , agent α 's payoff would be greater than or less than its own fallback.⁷

The games differ significantly based on whether the proposer is known in advance or not, as will be discussed in the next sections.

6. Known proposer

For an agent that is never going to make an offer, we can prescribe a dominant strategy independent of the statistical performance profiles:

Proposition 1. *If an agent, β , knows that it cannot make a proposal at the deadline T , then it has a dominant strategy of computing only on its own problem, and accepting any offer x_α^o such that $x_\alpha^o \geq V(n)$ where n is the node in the performance profile leT^β that agent β has reached at time T . If the performance profile does not flatten before the deadline ($V(n') < V(n)$ for every node n' on the path to n), then this is the unique dominant strategy.*

Proof. In the event that an agreement is not reached, agent β could not have achieved higher payoff than by computing on its individual problem (even if it knows that further computation will not improve its solution). In the event that an agreement is reached, agent β would have been best off by computing so as to maximize the minimal offer it will accept, $V(n_\beta^\beta)$. Since solution quality is nondecreasing in computation time, if agent β deviates and computes t steps on a different problem, then the value of its fallback is $V(n_{\beta'}^\beta) \leq V(n_\beta^\beta)$ where $\text{time}(n_{\beta'}^\beta) = \text{time}(n_\beta^\beta) + t$. If $V(n') < V(n)$ for every node n' on the path to n , then this inequality is strict. \square

Corollary 1. *In the games where the proposer is known, there exists a pure strategy PBE.*

Proof. By Proposition 1, the receiver of the offer has a dominant strategy. Say the proposer were to use a mixed strategy. In general, every pure strategy that has nonzero probability in a best-response mixed strategy has equal expected payoff [19]. Since mixing by the proposer will not affect the receiver's strategy, the proposer might as well use one of the pure strategies in its mix. \square

The equilibrium differs based on whether or not the deadline is known, as discussed in the next subsections.

⁷ Since solution values are discretized, the best-response offer-accept vectors will also be from a discrete space.

6.1. Known proposer, known deadline

In the simplest setting, both the deadline and proposer are common knowledge. Without loss of generality we assume that agent α is the proposer and the deadline is at time T . Therefore, from Proposition 1, agent β has a dominant strategy, S_β , which is to compute on the solution for its own problem and accept any offer that is greater than (or equal to) the value of its computed solution. Knowing this, agent α can determine a strategy which is a best-response.

Assume that by following a deliberation strategy S_α^D , the proposing agent α reaches deliberation state $\theta_\alpha(T) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle$ at time T . It is possible to compute agent α 's expected utility of following a bargaining strategy S_α^B where it makes an offer x_α^o to agent β . The expected utility is

$$E[\pi_\alpha((S_\alpha^D, S_\alpha^B), S_\beta)] = P_a(x_\alpha^o)[V(n_\alpha^{\text{joint}}) - x_\alpha^o] + (1 - P_a(x_\alpha^o))V(n_\alpha^\alpha), \quad (1)$$

where $P_a(x_\alpha^o)$ is the probability that agent β will accept an offer x_α^o . These probabilities are determined by agent α 's beliefs about what value agent β has computed for its own individual problem. In a setting where agent β has a dominant strategy (that is, it computes only on the solution for its own problem), agent α can compute its beliefs that agent β will accept an offer of x with probability $P_a(x)$, simply by noting the values of the nodes that can be reached at time T in the performance profile for agent β 's individual problem, and computing the probability of reaching each node.⁸

In a stochastic setting, there is uncertainty as to whether following a certain deliberation strategy will result in being in a specific deliberation state. We can determine the proposer's expected utility from following a particular strategy as follows. Assume agent α is executing strategy $S_\alpha = (S_\alpha^D, S_\alpha^B)$. At time T , when the agent must make a proposal, it is in some deliberation state

$$\theta_\alpha(T) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle,$$

where $\text{time}(n_\alpha^\alpha) = t_\alpha^\alpha$, $\text{time}(n_\alpha^\beta) = t_\alpha^\beta$, and $\text{time}(n_\alpha^{\text{joint}}) = t_\alpha^{\text{joint}}$. If its bargaining strategy, S_α^B dictates that it make an offer of x_α^o , then agent α 's expected utility from following S_α is

$$E[\pi_\alpha(S_\alpha, S_\beta)] = \sum_{\theta_\alpha(T) \in \Theta_\alpha(t_\alpha^\alpha, t_\alpha^\beta, t_\alpha^{\text{joint}})} p(\theta_\alpha(T) | S_\alpha^D) (P_a(x_\alpha^o)[V(n_\alpha^{\text{joint}}) - x_\alpha^o] + (1 - P_a(x_\alpha^o))V(n_\alpha^\alpha)), \quad (2)$$

where $p(\theta_\alpha(T) | S_\alpha^D)$ is the probability of being in deliberation state $\theta_\alpha(T)$ after following deliberation strategy S_α^D .

The game differs based on whether the performance profiles are deterministic or stochastic.

⁸ If the performance profiles are shared by the agents, agent α 's beliefs are based on the node n_α^β it has reached in the performance profile tree, \mathcal{T}^β , given that agent α has reached node n_α^β in the tree, \mathcal{T}_α^β .

6.1.1. Deterministic performance profiles

In an environment where the performance profiles are deterministic, the equilibria can be analytically determined.

Proposition 2. *Assume that the agents' performance profiles $\mathcal{T}_\alpha^\alpha$, \mathcal{T}_α^β , $\mathcal{T}_\alpha^{\text{joint}}$, \mathcal{T}_β^α , \mathcal{T}_β^β , and $\mathcal{T}_\beta^{\text{joint}}$ are deterministic and agent α knows agent β 's performance profiles. Then there exists a PBE where agent β will only compute on its own problem, and agent α will never split its computation. It will either compute solely on its own problem or solely on the joint problem. The PBE payoffs to the agents are unique, and the PBE is unique unless the performance profile that an agent is computing on flattens, after which time it does not matter where the agent computes since that does not change its payoff or bargaining strategy. The PBEs are also the only Nash equilibria.*

Proof. Let $\eta_\alpha^{\text{joint}}$ be the node in $\mathcal{T}_\alpha^{\text{joint}}$ that agent α reaches after allocating all of its computation on the joint problem. Let η_α^α be the node in $\mathcal{T}_\alpha^\alpha$ that agent α reaches after allocating all of its computation on its own problem. Let η_β^β be the node in \mathcal{T}_β^β that agent β reaches after allocating all of its computation on its own problem.

By Proposition 1, agent β has a dominant strategy to compute on its own solution (unless its performance profile flattens after which time it does not matter where the agent computes since that does not change its payoff). Agent α 's strategies are more complex since they depend on agent β 's final fallback value, $V(\eta_\beta^\beta)$, and also on what potential values the joint solution and α 's individual solution may have.

- (1) *Case 1:* $V(\eta_\alpha^{\text{joint}}) - V(\eta_\beta^\beta) > V(\eta_\alpha^\alpha)$. Agent β will accept any offer greater than or equal to $V(\eta_\beta^\beta)$ since that is its fallback. If agent α makes an offer that is acceptable to agent β , then the highest payoff that agent α can receive is $V(\eta_\alpha^{\text{joint}}) - V(\eta_\beta^\beta)$. If this value is greater than $V(\eta_\alpha^\alpha)$ —that is, the highest fallback value agent α can have—then agent α will make an acceptable offer. To maximize the amount it will get from making the offer, agent α must compute only on the joint problem. Any deviation from this strategy will result in agent α receiving a lesser payoff (and strictly less if its performance profile has not flattened).
- (2) *Case 2:* $V(\eta_\alpha^{\text{joint}}) - V(\eta_\beta^\beta) < V(\eta_\alpha^\alpha)$. Any acceptable offer that agent α makes results in agent α receiving a lesser payoff than if it had computed on its own solution solely, and made an unacceptable offer (and strictly less if its performance profile has not flattened). Therefore agent α will compute only on its own problem until that performance profile flattens, after which it does not matter where it allocates the rest of its computation.
- (3) *Case 3:* $V(\eta_\alpha^{\text{joint}}) - V(\eta_\beta^\beta) = V(\eta_\alpha^\alpha)$. By computing only on its own problem, agent α 's payoff is $V(\eta_\alpha^\alpha)$. By computing only on the joint problem, the payoff is $V(\eta_\alpha^{\text{joint}}) - V(\eta_\beta^\beta)$. These payoffs are equal. However, by dividing the computation across the problems, both payoffs decrease (unless at least one of the two performance profiles has flattened, after which it does not matter where the agent allocates the rest of its computation).

The above arguments also hold for Nash equilibrium. \square

6.1.2. Stochastic performance profiles

If the performance profiles are shared but stochastic, determining the equilibrium is more difficult. By Proposition 1, agent β has a dominant strategy, S_β , and only computes on its individual problem.⁹

However, based on the results it has obtained so far, agent α may decide to switch between problems on which it is computing—possibly several times. The problem is similar to computing values and policies for a sequential decision problem with stochastic actions, except that the deadlines mean that the game has a finite horizon. The payoffs can be seen as state-dependent reward values and the accessibility functions can be modeled as the probability of transferring into a deliberation state, given the action taken.

There are two different cases that affect agent α 's capabilities when it comes to speculating as to what value agent β has obtained from deliberation. If the two agents share algorithms and, therefore, performance profiles, then agent α can deliberate on agent β 's problem, and be sure that the results obtained are related to those that agent β has obtained. Agent α might then find it useful to deliberate on agent β 's problem in order to refine its beliefs as to what value agent β has obtained. On the other hand, if the agents have different algorithms and, therefore, different performance profiles, any deliberation that agent α does on agent β 's problem, using its own algorithm, may not correctly reflect the solutions that agent β has achieved. It gets no utility from computing on agent β 's problem since its beliefs can not be updated.

In this section we do not assume that the performance profiles are common knowledge. However, it is required that at least agent α can observe the performance profiles for agent β . Agent β does not need to know that agent α can view its performance profiles. This knowledge does not change agent β 's behavior as it has a dominant strategy.

We use a dynamic programming algorithm to determine agent α 's best response to agent β 's strategy. The base case involves looping through all possible deliberation states $\theta_\alpha(T)$ for agent α at the deadline T . Each $\theta_\alpha(T)$ determines a probability distribution over the set of nodes agent β reached by computing T time steps. For any offer x that agent α may make, the probability that agent β will accept is

$$P_a(x) = \sum_{\{n^\beta | n^\beta \text{ in subtree } \mathcal{T}^\beta(n_\alpha^\beta) \text{ at depth } T - \text{time}(n_\alpha^\beta) \text{ s.t. } V(n^\beta) \leq x\}} P(n^\beta). \quad (3)$$

The best offer, x_α^o , that agent α can make to agent β , given that α is in the state of deliberation $\theta_\alpha(T) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle$ is

$$x_\alpha^o(\theta_\alpha(T)) = \arg \max_x [P_a(x)(V(n_\alpha^{\text{joint}}) - x) + (1 - P_a(x))V(n_\alpha^\alpha)]. \quad (4)$$

We denote the optimal bargaining strategy, given the deliberation state $\theta_\alpha(T)$ by

$$S_\alpha^{B*}(\theta_\alpha^T) = (x_\alpha^o(\theta_\alpha(T)), V(n_\alpha^\alpha)). \quad (5)$$

The expected utility to agent α from following such a bargaining strategy while in deliberation state $\theta_\alpha(T)$ is

$$E[\pi_\alpha((S_\alpha^D, S_\alpha^{B*}(\theta_\alpha(T))), S_\beta)] = P_a(x_\alpha^o)(V(n_\alpha^{\text{joint}}) - x) + (1 - P_a(x_\alpha^o))V(n_\alpha^\alpha). \quad (6)$$

⁹ If that performance profile has flattened and agent β has computed on agent α 's or the joint problem thereafter, this does not change agent β 's fallback, and this is the only aspect of agent β that agent α cares about.

It is possible to work backwards and to compute which deliberation action, a^z , is optimal if agent α finds itself in deliberation state $\theta_\alpha(t)$ at time t once the optimal offers have been computed for each final deliberation state. Let $\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)$ denote the utility to agent α of computing on problem z at time $t + 1$, given that at time t it is in deliberation state $\theta_\alpha(t)$ and agent β is following strategy S_β . The expected value is

$$\begin{aligned} & E[\pi_\alpha^D((a^z, \theta_\alpha(T-1)), S_\beta)] \\ &= \sum_{\theta_\alpha(T) \in \Theta_\alpha(T)} P(\theta_\alpha(T) | \theta_\alpha(T-1), a^z) E[\pi_\alpha^D((S_\alpha^D, S_\alpha^{B*}(\theta_\alpha(T))))], \end{aligned} \quad (7)$$

and

$$\begin{aligned} & E[\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)] \\ &= \sum_{\theta_\alpha(t+1) \in \Theta_\alpha(t+1)} P(\theta_\alpha(t+1) | \theta_\alpha(t), a^z) \max_{a^z} E[\pi_\alpha^D((a^z, \theta_\alpha(t+1)), S_\beta)] \end{aligned} \quad (8)$$

for $t < T - 1$ where $P(\theta_\alpha(t) | \theta_\alpha(t-1), a^z)$ is the probability of reaching deliberation state $\theta_\alpha(t)$ given that upon reaching deliberation state $\theta_\alpha(t-1)$ the agent computes one step on problem z .

The optimal action in deliberation state $\theta_\alpha(t)$ is

$$a^z(\theta_\alpha(t)) = \arg \max_{a^z} E[\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)]. \quad (9)$$

The sequence of actions $(a^z(\theta_\alpha(t)))_{t=0}^{T-1}$ is agent α 's best-response deliberation strategy to agent β , and the offer-accept vectors $(x_\alpha^o(\theta_\alpha(T)), V(n_\alpha^\alpha))_{\theta_\alpha(T)}$ define its best-response bargaining strategy. Therefore, $S_\alpha = ((a^z(\theta_\alpha(t)))_{t=0}^{T-1}, (x_\alpha^o(\theta_\alpha(T)), V(n_\alpha^\alpha))_{\theta_\alpha(T)})$. The following algorithm computes the best-response strategy for agent α .

Algorithm 1. StratFinder1(T)

For each deliberation state $\theta_\alpha(T)$ at time T

$$x_\alpha^o(\theta_\alpha(T)) \leftarrow \arg \max_x [P_\alpha(x)[V(n_\alpha^{\text{joint}}) - x] + (1 - P_\alpha(x))V(n_\alpha^\alpha)].$$

For time $t = T - 1$ down to 0

For each deliberation state $\theta_\alpha(t)$

$$a^z(\theta_\alpha(t)) \leftarrow \arg \max_{a^z} E[\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)].$$

Return $((a^z(\theta_\alpha(t)))_{t=0}^{T-1}, (x_\alpha^o(\theta_\alpha(T)), V(n_\alpha^\alpha))_{\theta_\alpha(T)})$

Proposition 3. Algorithm 1 correctly computes a PBE strategy for agent α .¹⁰ Assume that the number of children of any node in $\mathcal{T}_\alpha^\alpha$, \mathcal{T}_α^β and $\mathcal{T}_\alpha^{\text{joint}}$ is at most k . Algorithm 1 runs in $O(k^{T-1}T^3)$ time.

¹⁰ By keeping track of equally good actions at every step, Algorithms 1, 2, and 3 can return all PBE strategies for agent α . Again, the dominant strategy of agent β is to compute on its own problem (unless the performance profile flattens out after which it does not matter what agent β computes on.

Proof. The nonproposing agent, β , has a dominant strategy, S_β . Algorithm 1 computes the best-response for agent α , at every time and for every state of deliberation.

Let k be the maximum number of children of any node in the performance profiles. At time t the number of states of deliberation is at most $k^t \binom{t+2}{2} = k^t (t+2)(t+1)/2$. Since

$$\begin{aligned} \sum_{t=0}^{T-1} k^t \frac{(t+1)(t+2)}{2} &\leq k^{T-1} \sum_{t=0}^{T-1} \frac{(t+1)(t+2)}{2} \\ &= k^{T-1} \frac{T(T+1)(T+2)}{6} \end{aligned}$$

the algorithm runs in $O(k^{T-1}T^3)$ time. \square

6.2. Known proposer, unknown deadline

There are situations where agents may not know the deadline. We represent this by a probability distribution $Q = \{q(i)\}_{i=1}^T$ over possible deadlines. Q is assumed to be common knowledge.

Whenever time t is reached but the deadline does not arrive, agents update their beliefs about Q . The new distribution is $Q' = \{q'(i)\}_{i=t}^T$ where $q'(t) = q(t)/(\sum_{j=t}^T q(j))$.

6.2.1. Deterministic performance profiles

Since there is no uncertainty as to agent β 's fallback value, agent α need never compute on agent β 's problem. Therefore, agent α will only be in deliberation states $\langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle$ where $\text{time}(n_\alpha^\beta) = 0$. Therefore, strategies that include computation actions a^β need not be considered. This, and the lack of uncertainty in which deliberation state action a leads to, greatly reduce the space of deliberation states to consider. Denote by $\Gamma_\alpha(t)$ any deliberation state of agent α where $\text{time}(n_\alpha^\alpha) + \text{time}(n_\alpha^{\text{joint}}) = t$ and $\text{time}(n_\alpha^\beta) = 0$. The algorithm for determining agent α 's equilibrium strategy differs from Algorithm 1 in that it incorporates the probability that the deadline may arrive at any time, and considers only the restricted space of deliberation states.

Algorithm 2. StratFinder2(Q)

For each deliberation state $\Gamma_\alpha(T)$ at time T

$$x_\alpha^o(\Gamma_\alpha(T)) \leftarrow \arg \max_x [P_a(x)[V(n_\alpha^{\text{joint}}) - x] + (1 - P_a(x))V(n_\alpha^\alpha)].$$

For $t = T - 1$ down to 0 $q'(t) \leftarrow q(t)/(\sum_{j=t}^T q(j))$

For each deliberation state $\Gamma_\alpha(t)$

$$x_\alpha^o(\Gamma_\alpha(t)) \leftarrow \arg \max_x [P_a(x)[V(n_\alpha^{\text{joint}}) - x] + (1 - P_a(x))V(n_\alpha^\alpha)],$$

$$\begin{aligned} a^z(\Gamma_\alpha(t)) &\leftarrow \arg \max_{a^z} [q'(t)E[\pi_\alpha((S_\alpha^D, S_\alpha^{B*}(\Gamma_\alpha(t))), S_\beta)] \\ &\quad + (1 - q'(t))E[\pi_\alpha^D((a^z, \Gamma_\alpha(t)), S_\beta)]]]. \end{aligned}$$

Return $((a^z(\Gamma_\alpha(t)))_{t=0}^{T-1}, (x_\alpha^o(\Gamma_\alpha(t), V(n_\alpha^\alpha)))_{t=0}^T)$

Proposition 4. *With deterministic performance profiles, Algorithm 2 correctly computes a PBE strategy for agent α in $O(T^2)$ time.*

Proof. The nonproposing agent, β , has a dominant strategy, S_β . Algorithm 2 computes the best-response for agent α , at every time and for every state of deliberation.

Since the setting is deterministic, agent α need only consider deliberation states at time t of the form $\theta_\alpha(t) = \langle n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{\text{joint}} \rangle$ where $\text{time}(n_\alpha^\beta) = 0$. At time t there are $t + 1$ deliberation states of this form. Each calculation in the algorithm takes constant time. The first loop is repeated $T + 1$ times. In the second loop, the calculations are done $t + 1$ times for $t = 0$ to $T - 1$, or $T(T + 1)/2$ times. Therefore, the algorithm takes $O(T^2)$ time. \square

6.2.2. Stochastic performance profiles

The algorithm differs from Algorithm 1 in that it considers the probability that the deadline might arrive at any time.

Algorithm 3. StratFinder3(Q)

For each deliberation state $\theta_\alpha(T)$ at time T

$$x_\alpha^o(\theta_\alpha(T)) \leftarrow \arg \max_x [P_a(x)[V(n_\alpha^{\text{joint}}) - x] + (1 - P_a(x))V(n_\alpha^\alpha)].$$

For $t = T - 1$ down to 1 $q'(t) \leftarrow q(t)/(\sum_{j=t}^T q(j))$.

For each deliberation state $\theta_\alpha(t)$

$$x_\alpha^o(\theta_\alpha(t)) \leftarrow \arg \max_x [P_a(x)[V(n_\alpha^{\text{joint}}) - x] + (1 - P_a(x))V(n_\alpha^\alpha)],$$

$$a^z(\theta_\alpha(t)) \leftarrow \arg \max_{a^z} [q'(t)E[\pi_\alpha((S_\alpha^D, S_\alpha^{B*}(\theta_\alpha(t))), S_\beta)] \\ + (1 - q'(t))E[\pi_\alpha^D((a^z, \theta_\alpha(t)), S_\beta)]].$$

Return $((a^z(\theta_\alpha(t)))_{t=0}^{T-1}, (x_\alpha^o(\theta_\alpha(t)))_{t=0}^T, V(n_\alpha^\alpha)_{t=0}^T)$

Proposition 5. *Algorithm 3 correctly computes a PBE strategy for agent α . Assume that the number of children of any node in $\mathcal{T}_\alpha^\alpha$, \mathcal{T}_α^β and $\mathcal{T}_\alpha^{\text{joint}}$ is at most k . Algorithm 3 runs in $O(k^{T-1}T^3)$ time.*

Proof. The nonproposing agent, β , has a dominant strategy, S_β . Algorithm 3 computes the best-response for agent α , at every time and for every state of deliberation.

Let k be the maximum branching factor for the performance profiles. At time t the number of states of deliberation is $k^t \binom{t+2}{2} = k^t(t+2)(t+1)/2$. Since

$$\sum_{t=0}^{T-1} k^t \frac{(t+1)(t+2)}{2} \leq k^{T-1} \sum_{t=0}^{T-1} \frac{(t+1)(t+2)}{2} \\ = k^{T-1} \frac{T(T+1)(T+2)}{6}$$

the algorithm runs in $O(k^{T-1}T^3)$ time. \square

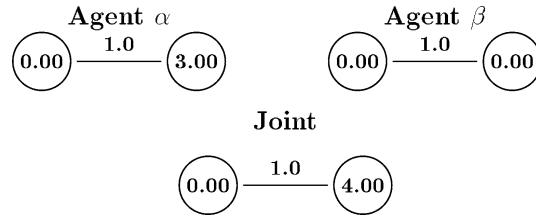


Fig. 3. Performance profile trees for the example where there is no pure strategy Nash equilibrium.

7. Unknown proposer

This section discusses the case where the proposer is unknown but the probability of each agent being the proposer is common knowledge. The deadline may be common knowledge. Alternatively, the deadline is not known but its distribution is common knowledge. This is a more complex setting since neither agent may have a dominant strategy. In fact, there exist instances where in equilibrium neither agent has a pure strategy.

7.1. Nonexistence of a pure strategy equilibrium

In this subsection we show that in some cases, there is no pure strategy equilibrium.

Proposition 6. *There exist instances (defined by the performance profile trees) of the game that have a unique mixed strategy PBE, but no pure strategy PBE (not even a pure strategy Nash equilibrium).*

Proof. Assume that the agents have the performance profiles in Fig. 3 and are allowed to take only one deliberation action ($T = 1$). Assume, also, that with equal probability either agent may be named as the proposer, that is, agent α is the proposer with probability $\frac{1}{2}$. Let \emptyset represent a null offer, where the proposer does not want to implement a joint solution.

The undominated strategies for agent α are

- $S_\alpha^1 = \{a^\alpha, (\emptyset, 3.0)\}$: Agent α computes one time step on its own problem. If it is chosen as the proposer then it makes a null offer, otherwise it accepts any offer that is greater than or equal to the fallback value of 3.0.
- $S_\alpha^2 = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent α computes one time step on the joint problem. If it is chosen as the proposer then it makes an offer of 0.0, otherwise it accepts any offer greater than or equal to the fallback value of 0.0.

The undominated strategies for agent β are

- $S_\beta^1 = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent β computes on the joint problem. If it is chosen proposer then it offers 0.0, otherwise it accepts anything greater than or equal to the fallback value of 0.0.
- $S_\beta^2 = \{a^{\text{joint}}, (3.0, 0.0)\}$: Agent β computes on the joint problem. If it is chosen as the proposer, then it offers 3.0, otherwise it accepts anything greater than or equal to the fallback value of 0.0.

Table 2
 Reduced normal form of the bargaining game with performance profiles in Fig. 3. There is no pure strategy Nash equilibrium

	S_β^1	S_β^2
S_α^1	3.0, 0.0	3.0, 0.5
S_α^2	2.0, 2.0	3.5, 0.5

The game can be represented in normal form (Table 2). There is no pure strategy Nash equilibrium for this game. It is easy to prove this by checking each strategy profile.

- (1) (S_α^1, S_β^1) is not a Nash equilibrium since agent β would respond with S_β^2 if α played S_α^1 .
- (2) (S_α^1, S_β^2) is not a Nash equilibrium since agent α would respond with S_α^2 if agent β played S_β^2 .
- (3) (S_α^2, S_β^2) is not a Nash equilibrium since agent β would respond with S_β^1 if agent α played S_α^2 .
- (4) (S_α^2, S_β^1) is not a Nash equilibrium since agent α would respond with S_α^1 if agent β played S_β^1 .

There does exist a mixed strategy Nash equilibrium. If agent α plays S_α^1 with probability γ and if agent β plays S_β^1 with probability δ then α 's expected payoff is

$$u_\alpha = \gamma(3.0\delta + 3.0(1 - \delta)) + (1 - \gamma)(2.0\delta + 3.5(1 - \delta)) = 1.5\gamma\delta - 0.5\gamma - 0.5\delta + 3.5.$$

The first order condition is

$$0 = \frac{du_\alpha}{d\gamma} = 1.5\delta - 0.5 \Rightarrow \delta = \frac{1}{3}.$$

Similarly for agent β

$$u_\beta = \delta(2.0(1 - \gamma)) + (1 - \delta)(0.5\gamma + 0.5(1 - \gamma)) = -2.0\gamma\delta + 1.5\delta + 0.5.$$

The first order condition is

$$0 = \frac{du_\beta}{d\delta} = -2.0\gamma + 1.5 \Rightarrow \gamma = \frac{3}{4}.$$

Therefore, in Nash equilibrium, agent α plays S_α^1 with probability 3/4 and agent β plays S_β^1 with probability 1/3. \square

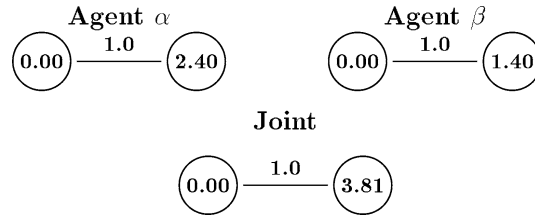


Fig. 4. Performance profile trees where the equilibrium outcome is not Pareto efficient.

7.2. Suboptimal outcome

It is often of interest to ask whether an outcome is “optimal”. An essential requirement for any optimal outcome is that it possess the property of *Pareto efficiency*. An outcome is Pareto efficient if there is no alternative outcome where some agent is better off without making some other agent worse off. Unfortunately, as we show below, in the setting where there is uncertainty as to which agent will be the proposer, agents may allocate their deliberation resources in a nonoptimal manner in equilibrium, so the outcome will not be Pareto efficient. In other words, if the agents would use different deliberation strategies, they would both be better off.

Proposition 7. *There exist instances (defined by T^α , T^β , and T^{joint}) of the game where the outcome of the unique Nash equilibrium is not Pareto efficient.*

Proof. Consider the performance profiles in Fig. 4. Let the probability that agent α will be the proposer be $1/2$. The agents are allowed only one deliberation step each, ($T = 1$). Let \emptyset represent a null offer, where the proposer does not want to implement a joint solution.

The undominated strategies for agent α are

- $S_\alpha^1 = \{a^\alpha, (\emptyset, 2.4)\}$: Agent α computes on its own problem and makes a null offer if it is the proposer. Otherwise, it accepts anything greater than or equal to 2.4.
- $S_\alpha^2 = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent α computes on the joint problem and offers nothing if it is the proposer. Otherwise, it accepts anything greater than or equal to 0.0.
- $S_\alpha^3 = \{a^{\text{joint}}, (1.4, 0.0)\}$: Agent α computes on the joint problem and offers 1.4 if it is the proposer. Otherwise, it accepts anything greater than or equal to 0.0.

The undominated strategies for agent β are

- $S_\beta^1 = \{a^\beta, (\emptyset, 1.4)\}$: Agent β computes on its own problem and makes a null offer if it is named as the proposer. Otherwise it accepts any offer greater than or equal to 1.4.
- $S_\beta^2 = \{a^{\text{joint}}, (0.0, 0.0)\}$: Agent β computes on the joint problem and offers nothing if it is the proposer. Otherwise, it accepts anything greater than or equal to 0.0.
- $S_\beta^3 = \{a^{\text{joint}}, (2.4, 0.0)\}$: Agent β computes on the joint problem and makes an offer of 2.4 if it is the proposer. Otherwise it accepts anything greater than or equal to 0.0.

The game can be represented in normal form (Table 3). There is a unique pure Nash equilibrium where agent α plays strategy S_α^1 and agent β plays strategy S_β^1 , that is, both agents compute on their own problems. However, the equilibrium outcome is not Pareto efficient. Both agents would be strictly better off if agent α played S_α^3 and agent β

Table 3

Normal form representation of the bargaining game where the performance profiles are found in Fig. 4, and the probability of agent α being the proposer is 1/2. The pure strategy Nash equilibrium is (S_α^1, S_β^1) . The Pareto efficient outcome is (S_α^3, S_β^3)

	S_β^1	S_β^2	S_β^3
S_α^1	2.4, 1.4	2.4, 0.0	2.4, 0.705
S_α^2	0.0, 1.4	1.905, 1.905	3.105, 0.705
S_α^3	1.205, 1.4	1.205, 2.605	2.405, 1.405

played S_β^3 . Unfortunately, the strategies S_α^3 and S_β^3 are not in equilibrium. If agent α played S_α^3 then agent β would deviate to S_β^2 . Similarly, if β played S_β^3 then agent α would deviate to S_α^2 . \square

7.3. A general method for solving the game with an unknown proposer

In general, solving an unknown proposer problem is hard, as neither agent may have a dominant strategy. Instead, the strategy of one player depends on the strategy of the other. One approach of solving for perfect Bayesian equilibria is to convert the game into its *normal form* by considering all pure strategies for each player and the resulting payoffs when these strategies are employed. There are relatively efficient algorithms for solving normal form games [2,37], but the conversion itself usually incurs an exponential blowup since the number of pure strategies is often exponential in the depth of the game tree because a pure strategy specifies a move for each information set of the player.

A more recent approach is to represent the extensive form game in its *sequence form* [36]. In the rest of this subsection we show how that technique can be used in our setting. A sequence of choices of a player corresponds to a node a in the game tree. The sequences replace the set of pure strategies in the normal form. In our setting a sequence is either;

- (1) \emptyset , the empty sequence,
- (2) a sequence of deliberation actions,
- (3) a sequence of deliberation actions followed by a proposal, or
- (4) a sequence of deliberation actions followed by either an accept or reject action.

All nodes in an information set, I , of an agent are defined by the same sequence σ_I . If, after reaching information set I , an agent then makes action c , the new sequence is denoted $\sigma_I c$. The set SQ_α denotes the set of all sequences for agent α . Similarly the set SQ_β is the set of all sequences for agent β .

Payoffs to agents α and β are represented by matrices A and B respectively. Each row corresponds to a sequence of agent α and each column corresponds to a sequence of agent β . Every leaf in the game tree defines a pair of sequences, that is, actions that both agents must have taken in order to reach that node. For each sequence pair defined by a leaf node, the agent's payoff is the payoff it received at the leaf node if there are no chance moves. If there are chance moves, as there are in our setting, then a pair of sequences may

correspond to more than one leaf node. The payoff entry is the sum of the payoffs over all leaves that correspond to the sequence pair weighted by the probabilities of reaching the leaves given the sequence pair. If a pair of sequences does not correspond to a leaf node, then the payoff entry is zero. The matrices, A and B , are sparse as the only (possible) nonzero entries occur at sequence pairs defined by leaf nodes which is linear in the size of the game tree.

Both agents also have realization plans, which are nonnegative vectors that represent the realization probabilities for the sequences of the agent when it is playing a mixed strategy. Let x be the realization plan for agent α and let y be the realization plan for agent β . The plans for agent α are characterized by the following constraints.

$$\begin{aligned} x(\emptyset) &= 1, \\ -x(\sigma_I) + \sum_{c \in C_I} x(\sigma_I c) &= 0, \end{aligned}$$

for all information sets I of agent α where C_I is the set of possible moves that agent α can make at information set I . This means that at any information set, I , the probability of reaching I is the same as the sum of the probabilities of taking an action that leaves I . The constraints for the realization plan, y , for agent β are similarly defined.

The realization plans can be represented by

$$Ex = e \quad \text{and} \quad Fy = f,$$

where E and F are constraint matrices. The first row is $(1, 0, 0, \dots)$ and corresponds to the empty sequence having probability one. The other rows correspond to the information sets of the respective agent. The vectors e and f are equal to the vector $(1, 0, 0, \dots, 0)^T$ which is of the appropriate size.

The Nash equilibrium of the game is a solution to a linear programming problem. The vectors x and y are in Nash equilibrium if they are mutual best responses. If y is fixed, then x is a best response if and only if it is an optimal solution to the linear program

$$\begin{aligned} &\text{maximize}_x \quad x^T(Ay) \\ &\text{subject to} \quad x^T E^T = e^T, \\ &\quad \quad \quad x \geq 0. \end{aligned}$$

The dual linear program is

$$\begin{aligned} &\text{minimize}_p \quad e^T p \\ &\text{subject to} \quad E^T p \geq Ay, \end{aligned}$$

where p is an unconstrained vector of variables.

Similarly, y is a best response to x if it is an optimal solution to

$$\begin{aligned} &\text{maximize}_y \quad y^T(Bx) \\ &\text{subject to} \quad y^T F^T = f^T, \\ &\quad \quad \quad y \geq 0, \end{aligned}$$

with dual

$$\begin{aligned} &\text{minimize}_q \quad f^T q \\ &\text{subject to} \quad F^T q \geq Bx, \end{aligned}$$

where q is an unconstrained vector of variables.

The feasible solutions to the linear programming problems are optimal only if the two objective function values are equal. That is, x is a best response to y only if

$$x^T(-Ay + E^T p) = 0$$

and y is a best response to x only if

$$y^T(-Bx + F^T q) = 0.$$

Any Nash equilibrium x, y is part of a solution x, y, p, q to the previous constraints. These constraints define a linear complementarity problem and therefore the solution to the linear complementarity problem is also a Nash equilibrium [2]. The standard linear complementarity problem is to find a vector $z \in \mathbb{R}^n$ so that

$$\begin{aligned} z &\geq 0, \\ b + Mz &\geq 0, \\ z^T(b + Mz) &= 0, \end{aligned}$$

where $b \in \mathbb{R}^n$ and M is an $n \times n$ matrix.

It is possible to create a linear complementarity problem that is equivalent to the linear programming problems [14]. First, set

$$M = \begin{pmatrix} & -A & E^T & -E^T & & \\ -B^T & & & & F^T & -F^T \\ -E & & & & & \\ E & & & & & \\ & -F & & & & \\ & F & & & & \end{pmatrix}$$

and

$$b = \begin{pmatrix} 0 \\ 0 \\ e \\ -e \\ f \\ -f \end{pmatrix}.$$

Let $z = (x, y, p', p'', q', q'')^T$ where p', p'', q', q'' are nonnegative vectors such that $p = p' - p''$ and $q = q' - q''$.

Using this representation, the equilibria for the extensive form game can be determined by similar algorithms that are known for the normal form, such as Lemke's algorithm for solving linear complementarity problems. Often, these algorithms run exponentially faster

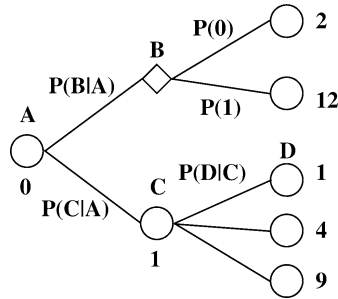


Fig. 5. An augmented performance profile tree. Node B is a random node. All other nodes are value nodes. The edges emanating from node B correspond to a random number generator producing different values.

than with the standard approach since the size of the sequence form is linear, and not exponential, in the size of the game tree [14].

However, these are general techniques which do not take advantage of specific properties of the particular game. It can be the case that the performance profiles will affect the payoffs in such a way that finding the equilibrium strategies is straightforward (for example, in situations where it is always better not to coordinate actions and implement the joint solution). Thus, specially designed algorithms can sometimes take advantage of the special structure and be more efficient than general techniques for computing equilibria. Earlier in the paper we presented such specialized algorithms for the setting with a known proposer, but currently we only have general algorithms for the setting with an unknown proposer.

8. Other sources of uncertainty

So far in this paper we have discussed settings where uncertainty in deliberation stems from different performance of the algorithm on different problem instances. In this section we discuss other sources of uncertainty that may also arise.

We address settings where agents are running randomized algorithms, agents have different algorithms and are uncertain about what algorithm the opponent is using, and agents that may not know each others' problem instances exactly.

8.1. Randomized algorithms

An algorithm is randomized if its behavior is determined by both its input and also values produced by a random number generator. Examples of randomized algorithms include simulated annealing [13], and some variants of hill climbing [34].

We can capture randomized algorithms in our deliberation control framework by using an *augmented performance profile tree*. An example of an augmented performance profile tree is presented in Fig. 5. An augmented performance profile tree can include two different types of nodes, *value nodes* and *random nodes*. Value nodes are similar to the nodes in the earlier definition of a performance profile tree. They hold the value of the solution, given that the algorithm has followed a path which would reach the node. The children of value

nodes are the nodes reached by taking one more deliberation step. The edges emanating from a value node all have a weight of one time step. Each edge is also associated with a probability that the child is reached in the next computation step given that the parent has been reached.

Random nodes represent places where random numbers are used during the algorithm run. Edges emanating from the random node are associated with the possible random numbers that might be generated. Each edge is labeled with the probability that its associated random number was used. All edges emanating from a random node have a weight of zero time steps.

The roles of deliberation are slightly different when agents have randomized algorithms. Agents must still use their deliberation resources in order to determine what their solutions to the various problems will be. However, if, for example, agent α computes on agent β 's individual problem, then there is no guarantee that agent α 's random number generator will produce the same numbers as agent β 's random number generator. Therefore, the results obtained by the agents on the same problem instance may be different. In this setting an agent can use its deliberation resources to *emulate* the run of a random algorithm. Emulation is different from running a random algorithm. If an agent is running an algorithm, whenever a random node in the performance profile is reached, a random number generator would generate some number which would specify the path the algorithm should take. In emulation, agents take a more active role. Whenever a random node is encountered, the agent can *choose* a “random” number instead of using a number generated by a random number generator. This allows the agent doing the emulation to learn what solution would have been obtained if the random numbers generated were the same as the ones chosen by the agent itself. This means that the agent can emulate different random numbers that the opponent may have used. An agent can use emulation to get a better idea of what solutions the opponent may have seen (and thus also a better idea of how the opponent may have allocated its computation as a function of the results it has obtained). An agent can emulate on the opponent's problem, the joint problem and even its own problem. As a side effect of emulating on one's own problem or on the joint problem, the agent obtains solutions that it can use. The agent can also obtain solutions without emulation, that is, by using actual random numbers in its randomized algorithm rather than emulating different edges emanating from random nodes.

The deliberation state has to be modified when agents are allowed to emulate algorithms. When emulating, an agent may follow different paths in a performance profile tree. Agents are allowed to revisit random nodes, and choose different random numbers in order to see where that new path of computation will lead. Therefore, an agent may reach several different nodes in a performance profile tree. This means that an agent may have several different possible solutions to a problem, among which it can select the best (as a solution to its own problem or the joint problem) or use all of the solutions and the associated distribution information (when speculating about the opponent's solutions to the three problems). The new deliberation state for agent i at time t is defined as

$$\overline{\theta}_i(t) = ((n_i^{\alpha,j}), (n_i^{\beta,j}), (n_i^{\text{joint},j})),$$

where $(n_i^{z,j})$ is a list of nodes in performance profile \mathcal{T}^z that agent i has reached. The time, t , is equal to the amount of time it took to reach all nodes listed in the deliberation state. That is,

$$t = \sum_j \text{time}(n_i^{\alpha,j}) + \sum_j \text{time}(n_i^{\beta,j}) + \sum_j \text{time}(n_i^{\text{joint},j}).$$

A new technique for computing the time function is used. If the agent followed only one path in the performance profile tree, then the time for any node on this path would merely be the sum of the weights of the edges of the path to that node. However, if there are multiple paths produced by an agent choosing different random numbers at a node, then the technique of summing the edge weights no longer works. It leads to overcounting since an agent does not need to start at the root of the performance profile tree each time it emulates a run. Instead it can back up to a random node, choose a new random number and continue emulation from there. Thus, when determining the emulation time for each node in the emulation component, the following approach is used. For any random node r , $\text{time}(r)$ is equal to the sum of the weights of the edges on the path to the node r . Let n_1 and n_2 be two value nodes in the same performance profile tree. Let r' be the least common random node ancestor of nodes n_1 and n_2 . Then $\text{time}(n_1)$ is equal to $\text{time}(r')$ plus the sum of the weights of the edges on the path from node r' to node n_1 . Since the paths to nodes n_1 and n_2 are the same from the root to node r' , the agent performing the emulation need only backtrack to node r' and continue deliberating from there. Thus, the time to reach node n_2 is only the additional time needed from node r' , that is, the sum of the weights of the edges on the path from node r' to node n_2 .

We also define a *deliberation set* which corresponds to the earlier definition of deliberation set.

Definition 7. The *deliberation set* for agent i at time t is

$$\overline{\Theta}_i(t) = \{\overline{\theta}_i(t)\}.$$

The agents' strategies also change slightly, as there is a larger action space. Agents must first decide which problem they wish to devote a deliberation step on and also they must select which "random" numbers to use, and whether to start the algorithm at an earlier random node with a new "random" number. These "random" numbers can be actual random numbers or numbers chosen by the agent for emulation purposes.

Definition 8. A *deliberation strategy* for agent i with deadline T is

$$S_i^D = (S_i^{D,t})_{t=0}^{T-1},$$

where

$$S_i^{D,t} : \overline{\Theta}_i(t) \rightarrow z \times n(z) \times \text{ran-numb}(n(z))$$

is a mapping from a deliberation state at time t to a tuple which specifies that the agent should take one deliberation step starting at the node defined by

- (1) a problem $z \in \{\alpha, \beta, \text{joint}\}$ on which to devote one deliberation step,
- (2) a node, $n(z)$, which is either
 - (a) a random node that the agent has reached earlier when deliberating on problem z , or
 - (b) a value node which is at the end of a path that the agent has followed when deliberating on problem z ,
- (3) a chosen “random” number, $\text{ran-num}(n(z))$, that will be used to determine the direction of the path of computation if the node chosen is a random node.

The bargaining strategy does not change. As before, it is a mapping from the deliberation state at the deadline T to an offer accept vector.

Definition 9. A bargaining strategy for agent i with deadline T

$$S_i^B : \Theta_i(T) \rightarrow \mathbb{R}^2$$

is a mapping from a state of deliberation at time T , to an offer accept vector (x_i^o, x_i^a) .

An agent’s strategy consists of a deliberation strategy and a bargaining strategy.

Definition 10. A strategy for agent i with deadline T is

$$S_i = (S_i^D, S_i^B).$$

If the proposer is known in advance, then the agent not proposing has a dominant strategy which is independent of its performance profiles.

Proposition 8. Assume agent i knows that it cannot make a proposal. Then it has a dominant strategy where it will emulate on its own problem in such a way as to maximize its fallback value. It will accept any offer that is greater than or equal to its computed fallback. If the performance profile does not flatten out before the deadline ($V(n') < V(n)$ for every value node n' on the path to value node n) then this is the unique dominant strategy.

The proof is identical to that of Proposition 1.

Since the nonproposing agent has a dominant strategy, the proposing agent can determine a best response. Algorithms 1 and 3 that were presented earlier for finding equilibrium strategies for the proposing agent when the agents have stochastic performance profiles need to be slightly generalized in order to find the equilibrium strategies when the agents have randomized algorithms. First, the new definition of a deliberation state must be used. Secondly, the deliberation action space must be enlarged to coincide with the action space specified in the definition for the deliberation strategy. As defined earlier, an action is a tuple which specifies what problem, what node to start from, and what “random” node to use as the agent takes one deliberation step. Once these modifications are made, both Algorithm 1 and Algorithm 3 correctly return the PBE strategy for the proposing agent when the deadline is known in advance (Algorithm 1) and when the deadline is known with some probability (Algorithm 3).

If the proposer is not known in advance then, in general, neither agent has a dominant strategy. In Section 7.3 a general method for solving the game with an unknown proposer was discussed. This same approach can be used if agents have random algorithms, where the new definition of a deliberation state is used, and the deliberation action space is enlarged to include the selection of random nodes and numbers.

8.2. Agents with different algorithms

Another setting where uncertainty arises is the situation where agents do not necessarily use the same algorithms. For example, agent α may be very skilled at solving one problem type while agent β may have different algorithmic skills.

If the agents, α and β , do not have access to the same algorithms, and the algorithms are not related (that is, one agent's algorithm performing well on a problem instance is no indication that the other agent's algorithm would have also performed well on the same problem instance) then an agent has no incentive to use any of its deliberation resources to compute solutions for the other agent's problem. Deliberating (or emulating) on an opponent's problem reduces the amount of time (and thus solution quality) that an agent can obtain for its own problem, or for the joint problem, while providing no signals as to what values the opponent has obtained for the problems.

However, often the agents' algorithms will be correlated in that if one agent's algorithm returns a certain solution value to a problem, another agent's algorithm would return a similar solution value. In such a setting, an agent may want to use some of its deliberation resources to compute on its opponent's problems. While the agent must use its own algorithms which differ from its opponents, the results obtained from deliberation (or emulation) can signal information about the solution quality that the opponent has obtained for its problems.

Agents may also have collections of algorithms, and use different algorithms to compute solutions to different types of problem instances. They may select which algorithm to use on which problem instance and may switch between algorithms even while deliberating on the same problem instance, starting the new algorithm with the solution computed by an earlier algorithm.

A collection of algorithms can be captured in an augmented performance profile tree as seen in Fig. 6. Our results that use augmented performance profile trees apply to the case of algorithm collections and agents having different algorithms as long as the choice of the algorithms is done at random (for example, each agent happens to have some algorithm from a collection of algorithms) rather than strategically by the agents. Specifically, the random nodes in the augmented performance profile tree will then represent randomness that stems from uncertainty regarding the opponent's algorithm.¹¹

¹¹ The setting where agents strategically switch between algorithms is more complicated. An agent may use partial results from one algorithm as a starting point for another. Each agent's single deliberation problem, of how much time to allocate to its own problem, when to switch between algorithms, and what solution to use as a starting point for the new algorithm is complicated in itself. Therefore, in this paper we do not attempt to analyze such a situation. However, the single agent problem is interesting in itself and is worth future investigation.

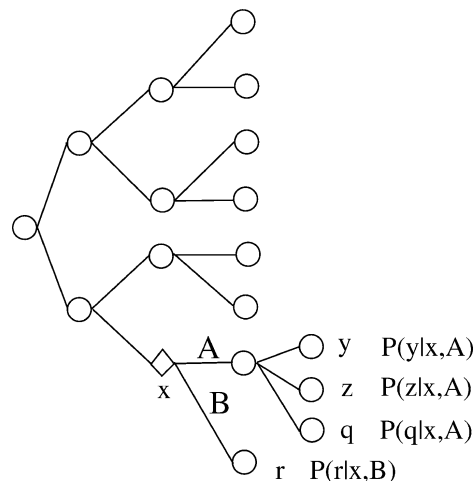


Fig. 6. An agent's augmented performance profile tree where the opponent may have either of two algorithms, Algorithm A or Algorithm B. At random node x , the two algorithms start to differ in how well they are likely to solve the problem instance. In one step of computation, Algorithm B would reach a solution represented by node r . On the other hand, Algorithm A could lead to node y , z , or q , each with a certain probability.

8.3. Agents that do not know each others' problem instances

Another situation where uncertainty can enter the picture occurs when agents do not actually know what problem instance (tasks, resources, etc.) the opponent has and that affects the opponent's individual problem and the joint problem. Instead, each agent has a probability distribution over possible problem instances of the opponent, and performance profile trees which describe how each algorithm performs on the different instances. Now, an agent can deliberate (or emulate) several steps on the solution to one problem instance, and then several more steps on another problem instance, etc., using the performance profile trees (see, for example, Fig. 7) to guide the deliberation process. This allows the agent to get a better probabilistic estimate of the opponent's solutions to the three problems as well as the agent's own solutions to the three problems.¹² However, this deliberation does not remove any of the uncertainty as to what problem instance the opponent has. That is, the agent cannot update its probability distribution over the possible problem instances by deliberating on the opponent's problem.

8.4. Uncertainty that arises from combinations of sources

As mentioned in the previous subsections, uncertainty may arise for many different reasons:

¹² We assume that if the agents agree to go with the joint solution, the agents learn each others' problem instances at the end of the game. That is, no deliberation can occur after that time. We also assume that the proposer offers the other agent a fixed value (chosen by the proposer), so that the proposer carries all the risk related to the uncertainty about the other's problem instance.

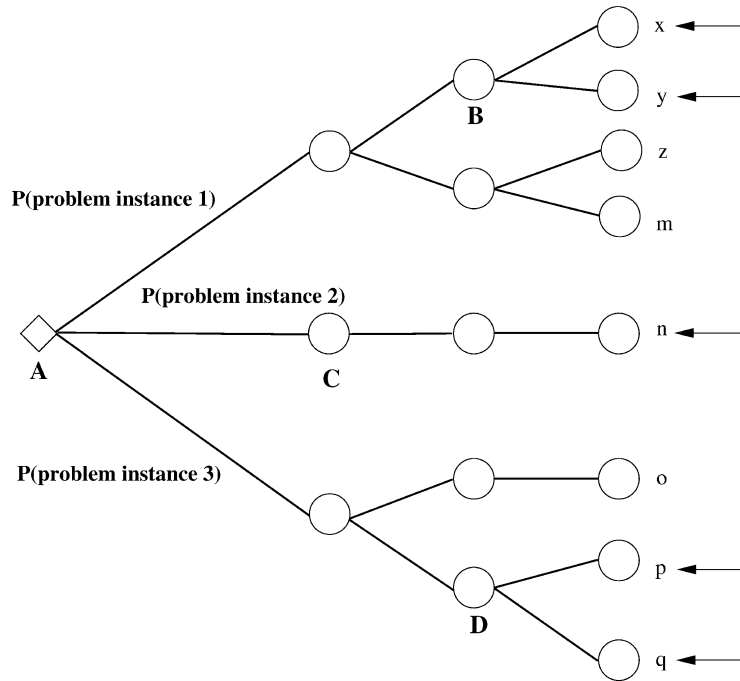


Fig. 7. An augmented performance profile tree that agent α uses for agent β 's problem. Agent α has uncertainty as to which problem instance agent β has encountered. Instead, agent α has a probability distribution over the three possible problem instances, problem instance 1, problem instance 2, and problem instance 3. At node A agent α selects which problem instance to begin computing on. After deliberating for several steps, it can return to node A and select another problem instance to deliberate on. Say, for example, agent α computed on all three problem instances and reached node B for problem instance 1, node C for problem instance 2, and node D for problem instance 3. Then, agent α knows that if agent β has computed on its own problem and the problem instance was 1, then the solution agent β obtained was either node x or node y . If the problem instance was problem instance 2, then agent β would have reached node n and if the problem instance was problem instance 3, then agent β would have reached either node p or node q . It is not possible for agent β to have obtained solutions z , m or o .

- (1) The algorithms may perform differently on different problem instances.
- (2) Agents may have randomized algorithms.
- (3) Agents may have different algorithms and might not know which algorithm the opponent uses for sure.
- (4) Agents may have uncertainty as to what problem instance the opponent has.

The results and techniques of this paper apply to combinations of the sources of uncertainty as well. For example, in any setting where there are randomized algorithms, agents can emulate the algorithms' runs. The emulation will still be effective even if there are other sources of uncertainty such as uncertainty about the problem instance.

In the most general setting, agents have different randomized algorithms and do not know exactly what each others' problem instances are or what algorithm the opponent is running exactly. The results of this paper apply to that setting as well, when augmented performance profile trees are used. The random nodes in those trees capture the different types of uncertainty.

9. Conclusions and future research

Noncooperative game-theoretic analysis is necessary to guarantee nonmanipulability of systems that consist of self-interested agents. However, the equilibrium for rational agents does not generally remain an equilibrium for computationally limited agents. This leaves a potentially hazardous gap in theory. This paper presented a framework and the first steps toward filling that gap.

We studied a setting where each agent has an intractable optimization problem, and the agents can benefit from pooling their problems and solving the joint problem. We presented a fully normative model of deliberation control that allows agents to condition their projections of their anytime algorithms' performance on the problem instance and path of solutions seen so far. We show how this approach can be generalized to handle uncertainty that may arise in deliberation, either from the problem instance itself, from the use of random algorithms, from the use of different algorithms, and uncertainty about the opponent's problem instance. Using that model, we solved the equilibrium of the bargaining game. This is, to our knowledge, the first piece of research to treat deliberation actions strategically via noncooperative game-theoretic analysis. We call this solution concept the *deliberation equilibrium*.

In ultimatum games where the agents know which one gets to make a take-it-or-leave-it offer to the other, the receiver of the offer has a dominant strategy of computing on its own problem, independent of the algorithm's statistical performance profiles. It follows that these games have pure strategy equilibria. In equilibrium, the proposer can switch multiple times between computing on its own, the other agents, and the joint problem. The games differ based on whether or not the deadline is known and whether the performance profiles are deterministic or stochastic. We presented algorithms for computing a pure strategy equilibrium in each of these variants. For games where the proposer is not known in advance, we use a general algorithm for finding a mixed strategy equilibrium in a 2-person game [14]. This generality comes at the cost of potentially being slower than our algorithms for the other cases and only guarantees that some equilibrium will be found, not necessarily all.

In situations where there is uncertainty as to which agent will make the final proposal, we show that there exist instances, defined by the performance profiles, where there is a unique mixed strategy perfect Bayesian equilibrium but no pure strategy one. This means that in equilibrium agents would have to randomized over which action to take. We also show that there exist instances where, in equilibrium, agents do not allocate their deliberation resources optimally which leads to non-Pareto efficient outcomes.

Our approach of combining a normative model of bounded rationality with a noncooperative solution concept leads the way to building systems where the agents are self-interested and computationally limited. This area is filled with promising future research possibilities and can be extended in several directions. We plan to analyze richer bargaining settings. In particular we are interested in allowing negotiation to occur amidst computation, not just after it [16]. In such settings, proposals signal about what problems the other agent has computed on, what solutions it has found, and what solutions it expects to find if it computes further. We are also interested in extending the model to settings where there are different models of bounded rationality (e.g., costly but unlimited computation as well

as computation which is both costly and limited), and to settings where there are more than two agents involved in bargaining.

An exciting research path which we plan to explore is the design of mechanisms that lead to as efficient as possible outcomes with as little redundant computation as possible. While it has been pointed out that the central *revelation principle* from noncooperative game theory [19] ceases to apply when computational complexity limits each agent's rationality [23,32], our model provides a framework for actually deriving results in settings where the revelation principle fails to hold. This framework allows one to analyze problems beyond bargaining as well, including auctions where each agent needs to potentially solve an intractable problem to determine its valuations [17,18], and voting where each agent needs to potentially solve an intractable problem to determine its preferences over outcomes.

Acknowledgements

This material is based upon work supported by the National Science Foundation under CAREER Award IRI-9703122, Grant IIS-9800994 and Grant ITR IIS-0081246. We thank Dov Monderer for helpful discussions.

References

- [1] D. Abreu, A. Rubinstein, The structure of Nash equilibrium in repeated games with finite automata, *Econometrica* 56 (6) (1988) 1259–1281.
- [2] H.M. Amman, D.A. Kendrick, J. Rust (Eds.), *Handbook of Computational Economics*, Elsevier Science, Amsterdam, 1996.
- [3] E.B. Baum, W.D. Smith, A Bayesian approach to relevance in game playing, *Artificial Intelligence* 97 (1–2) (1997) 195–242.
- [4] M. Boddy, T. Dean, Deliberation scheduling for problem solving in time-constrained environments, *Artificial Intelligence* 67 (1994) 245–285.
- [5] A. Garvey, V. Lesser, A survey of research in deliberative real-time artificial intelligence, *Real-Time Systems* 6 (1994) 317–347.
- [6] I. Gilboa, D. Samet, Bounded versus unbounded rationality: The tyranny of the weak, *Games and Economic Behavior* (1989) 213–221.
- [7] I. Good, Twenty-seven principles of rationality, in: V. Godambe, D. Sprott (Eds.), *Foundations of Statistical Inference*, Holt, Rinehart, Winston, Toronto, Ontario, 1971.
- [8] E. Hansen, S. Zilberstein, Monitoring and control of anytime algorithms: A dynamic programming approach, *Artificial Intelligence* 126 (2001) 139–157.
- [9] E. Horvitz, Principles and applications of continual computation, *Artificial Intelligence* 126 (2001) 159–196.
- [10] E.J. Horvitz, Reasoning about beliefs and actions under computational resource constraints, in: *Proc. 3rd Workshop on Uncertainty in Artificial Intelligence*, Seattle, WA, 1987, pp. 429–444. American Association for Artificial Intelligence. Also, in: L. Kanal, T. Levitt, J. Lemmer (Eds.), *Uncertainty in Artificial Intelligence* 3, Elsevier, Amsterdam, 1989, pp. 301–324.
- [11] E.J. Horvitz, Reasoning under varying and uncertain resource constraints, in: *Proc. AAAI-88*, St. Paul, MN, Morgan Kaufmann, San Mateo, CA, 1988, pp. 111–116.
- [12] P. Jehiel, Limited horizon forecast in repeated alternate games, *J. Economic Theory* 67 (1995) 497–519.
- [13] S. Kirkpatrick, C. Gelatt, M. Vecchi, Optimization by simulated annealing, *Science* 220 (1983) 671–680.
- [14] D. Koller, N. Megiddo, B. von Stengel, Efficient computation of equilibria for extensive two-person games, *Games and Economic Behavior* 14 (2) (1996) 247–259.

- [15] S. Kraus, J. Wilkenfeld, G. Zlotkin, Multiagent negotiation under time constraints, *Artificial Intelligence* 75 (1995) 297–345.
- [16] K. Larson, T. Sandholm, An alternating offers model of bargaining with deliberation actions, submitted for review, 2001.
- [17] K. Larson, T. Sandholm, Computationally limited agents in auctions, in: AGENTS-01 Workshop of Agents for B2B, Montreal, Quebec, 2001, pp. 27–34.
- [18] K. Larson, T. Sandholm, Costly valuation computation in auctions, in: *Theoretical Aspects of Rationality and Knowledge (TARK VIII)*, Siena, Italy, 2001, pp. 161–182.
- [19] A. Mas-Colell, M. Whinston, J.R. Green, *Microeconomic Theory*, Oxford University Press, Oxford, 1995.
- [20] J. Nash, Equilibrium points in n -person games, *Proc. Nat. Acad. Sci.* 36 (1950) 48–49.
- [21] M. Osborne, A. Rubinstein, *Bargaining and Markets*, Academic Press, New York, 1990.
- [22] C.H. Papadimitriou, M. Yannakakis, On complexity as bounded rationality, in: *STOC-94*, 1994, pp. 726–733.
- [23] D.C. Parkes, Optimal auction design for agents with hard valuation problems, in: *Proc. Agent-Mediated Electronic Commerce Workshop at the International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.
- [24] J.S. Rosenschein, G. Zlotkin, *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*, MIT Press, Cambridge, MA, 1994.
- [25] A. Rubinstein, *Modeling Bounded Rationality*, MIT Press, Cambridge, MA, 1998.
- [26] S. Russell, Rationality and intelligence, *Artificial Intelligence* 94 (1) (1997) 57–77.
- [27] S. Russell, D. Subramanian, Provably bounded-optimal agents, *J. Artificial Intelligence Research* 1 (1995) 1–36.
- [28] S. Russell, E. Wefald, *Do the Right Thing: Studies in Limited Rationality*, MIT Press, Cambridge, MA, 1991.
- [29] S. Russell, E. Wefald, Principles of metareasoning, *Artificial Intelligence* 49 (1991) 361–395.
- [30] L. Samuelson, Bounded rationality and game theory, *Quarterly Rev. Econom. Finance* 36 (1996) 17–35.
- [31] T. Sandholm, An implementation of the contract net protocol based on marginal cost calculations, in: *Proc. AAAI-93*, Washington, DC, 1993, pp. 256–262.
- [32] T. Sandholm, Issues in computational Vickrey auctions, *Internat. J. Electronic Commerce* 4 (3) (2000) 107–129. Special Issue on Applying Intelligent Agents for Electronic Commerce. A short, early version appeared at the Second International Conference on Multi-Agent Systems (ICMAS), 1996, pp. 299–306.
- [33] T. Sandholm, V.R. Lesser, Coalitions among computationally bounded agents, *Artificial Intelligence* 94 (1) (1997) 99–137. Special issue on Economic Principles of Multiagent Systems. Early version appeared at the International Joint Conference on Artificial Intelligence (IJCAI), 1995, pp. 662–669.
- [34] B. Selman, H. Levesque, D. Mitchell, A new method for solving hard satisfiability problems, in: *Proc. AAAI-92*, San Jose, CA, 1992, pp. 440–446.
- [35] H.A. Simon, *Models of Bounded Rationality*, Vol. 2, MIT Press, Cambridge, MA, 1982.
- [36] B. von Stengel, Efficient computation of behavior strategies, *Games and Economic Behavior* 14 (1996) 220–246.
- [37] B. von Stengel, Computing equilibria for two-person games, in: R.J. Aumann, S. Hart (Eds.), *Handbook of Game Theory*, Vol. 3, North-Holland, Amsterdam, 2002.
- [38] S. Zilberstein, F. Charpillet, P. Chassaing, Real-time problem solving with contract algorithms, in: *Proc. IJCAI-99*, Stockholm, Sweden, 1999, pp. 1008–1013.
- [39] S. Zilberstein, S. Russell, Optimal composition of real-time systems, *Artificial Intelligence* 82 (1–2) (1996) 181–213.