Bargaining with limited computation: Deliberation equilibrium

An essay for course T-79.194, Seminar on Theoretical Computer Science

Based on the article by Kate Larson and Tuomas Sandholm published in Artificial Intelligence 132 (2001) 183-217

Ville Kotovirta, 41575d Ti
28 March, 2002

Abstract

We go through the main results from the article by Kate Larson and Tuomas Sandholm, published in Artificial Intelligence [2]. A normative theory of interaction for analyzing the non-cooperative game of two self-interested rational agents with limited computational capabilities is presented. Both agents have an intractable optimization problem, and a limited amount of time to carry out the computations. The agents can benefit from pooling the problems and implementing the joint problem. If no advantage is gained from pooling agents implement their individual solutions. The equilibria for the game differ based on what kind of uncertainties there are in the game, and whether the performance profiles are deterministic or stochastic. We conclude with some considerations about the algorithms Larson and Sandholm presented.

Contents

Introduction

Systems, especially in the Internet, are increasingly being used by multiple parties with their own preferences. There is no central control of all system components, only control of the mechanism, which means the rules of the game. The system can be seen as a setting of a game where different components, or agents, try to find the best individual solution to the problem at hand, and at the same time contribute to the overall operation of the system. Each agent participating chooses a strategy of its own.

The economic efficiency that a system achieves depends on the agents' strategies. By analysing the game using the Nash equilibrium solution concept, the designer of the system can make sure that each agent is motivated to behave in a desired way. No agent deviates from its strategy given that the others do not deviate.

However, the equilibrium for rational agents does not generally remain an equilibrium for computationally limited agents. Early on, in the field of economic theory, it was recognized that humans have bounded rationality, for example, due to limited cognition. The assumption of perfect rationality and actually observed human behavior has long been a source of economists' dissatisfaction. This is perhaps best evidenced in the work of Herbert Simon (see, for example [6] and [7]) . Considerable amount of research work has focused on developing a normative model to describe how the computationally limited agent should behave. This is a nontrivial task, containing numerous fundamental and technical difficulties. Game theorists have also realized the significance of computational limitations (see, for example, [4]).

In this paper we will present the main results given by Sandholm and Larson in [2] for an ultimatum game where agents use their limited computational resources to find a joint solution, or individual solutions for their intractable problem instances. All the proofs are not presented here, but the reader interested in deeper explanations is advised to consult the original paper. Also, some considerations about the results are given.

An example application

Larson and Sandholm (herein after referred to as "the authors") give a concrete example to clarify the problem setting [5]. Figure 1 depicts the example setting where there are two geographically dispersed dispatch centers that are self-interested companies. Each center is responsible for certain deliveries (tasks) and has a certain set of vehicles (resources) to take care of them. Each agent has to minimize the transportation costs, and handle the delivery tasks. In the figure the red arrows depict the delivery orders for the red dispatch center (to the left) that has trucks with red outlines, while the gray arrows are for gray dispatch center (to the right) with gray trucks.

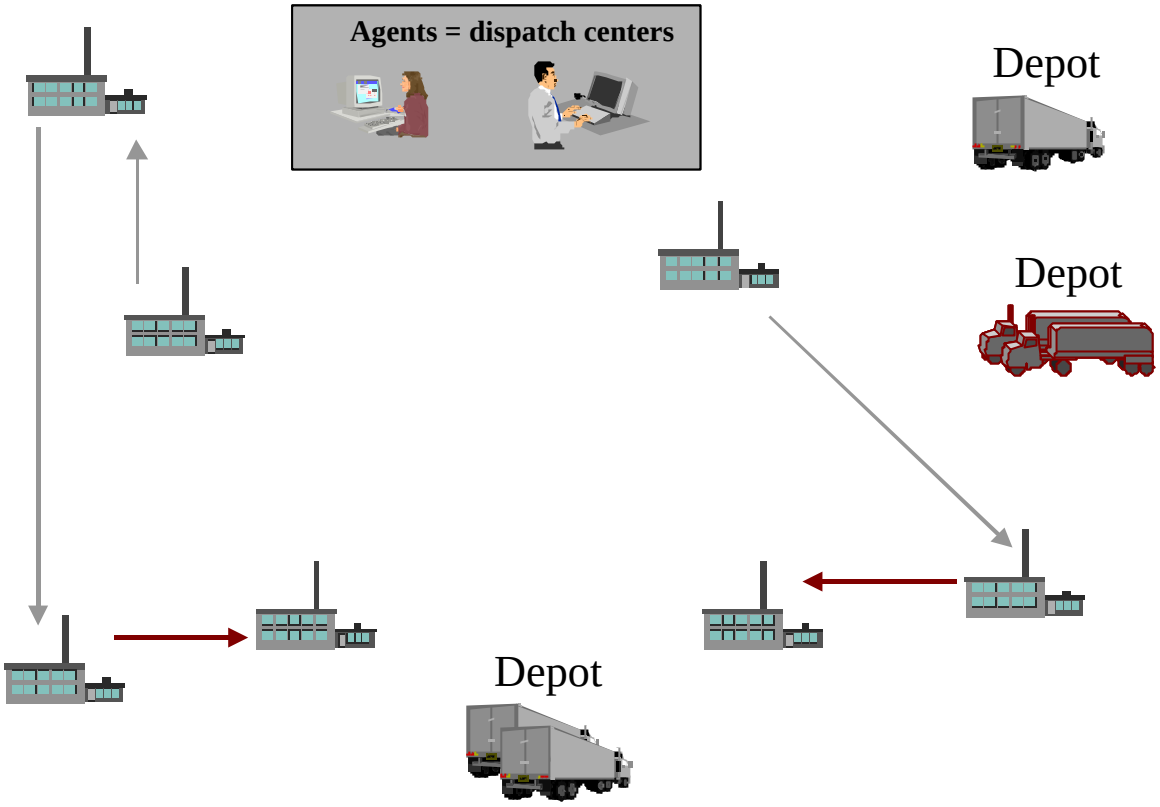

Figure 1. Example problem of the distributed vehicle routing problem [5]. See text for details.

The geographical operation areas overlap, and this creates a potential for savings in driven mileage by pooling the tasks and resources. One of the agents may be able to handle some of the tasks with less driving than the other. The agents must negotiate about whether to independently deliver their own packages, or whether to share their delivery tasks and resources in order to reduce costs.

The general setting

The distributed vehicle routing problem described above is only one example problem setting with two self-interested agents having an intractable individual problem, and with a potential savings gained from pooling the problems. The authors give a few other applications with these characteristics, including

- Manufacturing, where two companies have potential subcontract relationship
- Electric power negotiation between provider and consumer (participants need to construct their production and consumption schedules)
- Classroom scheduling
- Scheduling of scientific equipment
- Bandwidth allocation and routing in multi-provider multi-consumer computer networks

Agents need to determine the gain achieved by pooling the problem instead of each agent operating individually. It is assumed that the agents have anytime algorithms that produce some feasible solution whenever the algorithm is terminated. The solution improves as more computation time is allocated. Agent can allocate the

limited time it has to calculate its own problem, other agent's problem, or the joint problem. Agent must decide how and when to compute on the three problems, and based on the results of these computations decide which offer to make and which to accept.

The model

The model for the computational bargaining has two distinct parts: the deliberation control part and the bargaining part. Although the deliberation precedes bargaining, these two stages are interrelated.

In the deliberation part the agents try to determine which problem to calculate, and in the bargaining part they try to decide which value to offer, and which value to accept.

Let there be two agents, α and β, each with its own individual problem, and with the possibility to pool, giving rise to a joint problem. Time is discretized and limited, so agents can deliberate for at most T time steps.

Let $v_\alpha^\alpha(t)$ be the value of the solution to agent α's individual problem after agent α has computed on it for *t* time steps. Similarly, $v_\alpha^\beta(t)$ is the value of the solution to agent β's individual problem, and $v_\alpha^{joint}(t)$ is the value of the solution to the joint problem after agent α has computed the problems for *t* time steps.

A Performance profile tree

The agents have statistical performance profiles that describe how their anytime algorithms improve the solutions as a function of time. Agents use this information to decide how to allocate their computation at every step of the game.

A common representation of performance profiles is a table, which contains for each time step and each level of solution quality a probability that the solution will be of that quality (see Figure 2).

							.08	.19	.24
						.15	.30	.17	.39
				.16	.10	.16	.25	.30	.22
	.08	.04	.17	.20	.22	.30	.24	.19	.15
.09	.10	.20	.22	.23	.37	.31	.13	.15	
.11	.14	.33	.18	.21	.18	.08			
.22	.17	.25	.24	.15	.13				
.40	.31	.12	.19	.05					
.15	.20	.03							
.03									

Solution quality

Computing time

Figure 2. An example of table-based performance profile presentation.

Instead of a table, the authors propose storing the values in a tree structure, which allows conditioning along a path. The tree is constructed by collecting statistical data from previous runs of the algorithm on different problem settings. 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 are normalized to get the probabilities. If there is no node for certain value, the node is generated with an edge from the previous node to it.

Figure 3 depicts an example of a performance profile tree. Each depth corresponds to the time *t* of the run that the algorithm has executed. Each node at depth *t* is associated with a value representing a possible solution quality, v_i^z, that is obtained by running the algorithm for *t* time steps on that problem. The problems are indexed by z (α, β, or joint), and the agents are indexed by *i* (α, β). For each *z* there is a performance profile tree, τ_i^z, with which an agent i can condition its algorithm's performance on the problem instance.

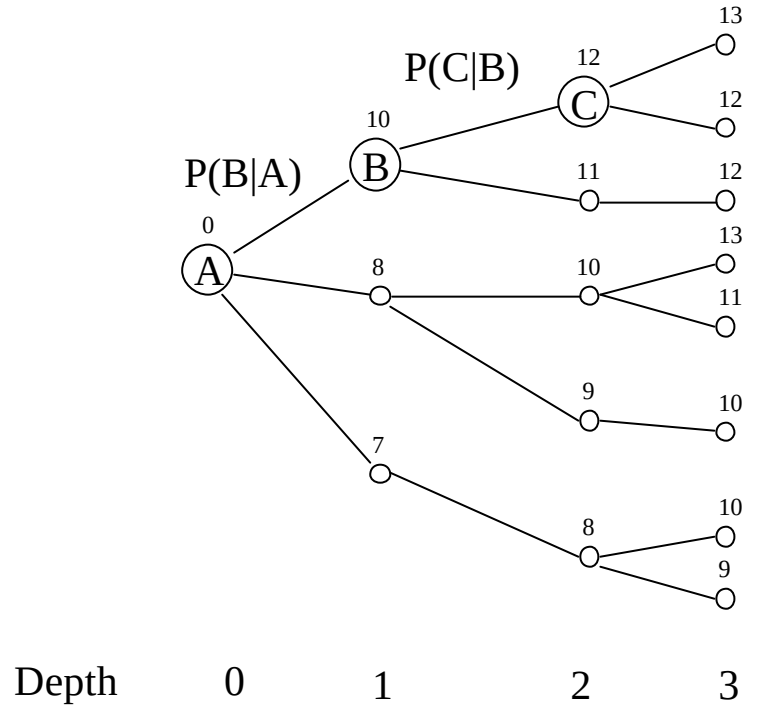

Figure 3. An example performance profile tree.

At any depth there may be more than one node with a certain value since the path taken to reach that value may be different depending on the problem instance.

A performance profile can be **deterministic** or **stochastic**. In Stochastic performance profile tree each edge is associated with a probability that the child is reached in the next step given that the parent has been reached. This makes it possible to calculate the probability to reach any particular future node in the tree by multiplying the probabilities on the path between two nodes. In figure 3, for example, P(B|A) is the probability that node B will be reached given that node A has been reached. The probabilities are calculated as described above.

A deterministic performance profile tree has only one path, which can be followed. All the conditional probabilities are equal to one, and a solution quality value can be determined exactly given the time of computation allocated on the problem.

Deliberation

In the model the state of deliberation of agent α at time step t is defined as

$$\theta_\alpha(t) = (n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{joint}) \quad (1)$$

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

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

$$\Theta_\alpha(t) = \{\theta_\alpha(t) \mid time(n_\alpha^\alpha) + time(n_\alpha^\beta) + time(n_\alpha^{joint}) = t\} \quad (2)$$

Bargaining

In the authors' setting agents bargain over how to divide the surplus or cost associated with implementing the joint problem. At some point in time, T, there is a deadline at which time both agents must stop deliberating and start the bargaining round. The goal of the bargaining is to decide whether to pool or to implement the individual solutions.

The model is restricted 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. If a proposal is accepted, the joint solution is implemented and the surplus is divided as agreed upon the proposal. If no agreement is reached the individual solutions are implemented without further interaction.

Agent α makes an offer, x_α^0, to agent β, about how much agent β's payoff will be if they pool. If agent β accepts the offer, agent α's solution for the joint problem is used and agent β gets x_α^0 as agreed. Agent α gets the rest of the value of the solution: $v_\alpha^{joint}(t) - x_\alpha^0$. If the offer is rejected, individual solutions are implemented, and the payoff for agent α is v_α^α and agent β's payoff is v_β^β. These combinations are presented in table 1.

Agent	Payoff if the offer is accepted	Payoff if the offer is rejected
α	$v_\alpha^{joint} - x_\alpha^0$	v_α^α
β	x_α^0	v_β^β

Table 1. Payoffs for the agents. See text for details.

Strategies

The agents' strategies include actions from both the deliberation part and the bargaining part of the game.

The deliberation part takes T time steps, and at every time step agents have to decide on which problem they allocate the limited computation time. The **deliberation strategy** for agent α is a vector, S_α^D, which contains the actions agent α will take at every time step *t* during the deliberation. Each element in the vector is a mapping from a deliberation state, $\theta_\alpha(t) = (n_\alpha^\alpha, n_\alpha^\beta, n_\alpha^{joint})$, at time *t* to a deliberation action A^z, which is the action of computing one time step on the solution for problem z.

$$S_a^D = (S_a^{D,t})_{t=0}^{T-1} \tag{3}$$

where

$$S_a^{D,t} : \Theta_a(t) \rightarrow \{A^a, A^b, A^{joint}\} \tag{4}$$

is the mapping described above. Equation (4) describes the mappings from all the possible deliberation states that can be achieved at time step *t*.

At the deadline T, each agent has to decide on its offer-accept vector, which captures agent's strategy. An offer-accept vector contains the amount that agent would offer if it were the proposer, and the value it would accept if the other agent made the proposal.

A **bargaining strategy** for agent α with deadline T is a mapping from a state of deliberations at time T to a two-dimensional offer-accept vectors, $(x_\alpha^0, x_\alpha^\alpha)$:

$$S_a^B = \Theta_a(T) \rightarrow R^2 \tag{5}$$

Since there are many deliberation states that can be achieved at the end of the deliberation part, we use the notation for deliberation set, Θ_α, in equation (5).

A strategy for agent α contains both the deliberation and bargaining strategies:

$$S_\alpha = (S_\alpha^D, S_\alpha^B) \tag{6}$$

Equilibria and algorithms

How to make the system behave in the desired way even though agents are designed to be self-interested real-world parties? One approach would be to make sure that no agent is motivated to deviate to another strategy. The strategy for each agent is the best strategy that agent has from its self-interested perspective.

This would be the *Nash equilibrium* concept from cooperative game theory. Actually, a stronger requirement is needed: at any point in the game, agent's strategy takes optimal actions from that point on, given agent's beliefs about what has happened so

far in the game, and the other agent's strategy. This type of equilibrium is called a *perfect Bayesian equilibrium* (PBE).

The authors introduce an equilibrium for computationally limited agents, which consists of a Nash, perfect Bayesian equilibrium for both deliberation and bargaining strategies.

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 for an agent is the value obtained for the solution for its individual problem. An agent will not accept any offer smaller than its fallback value.

The games differ significantly based on whether the proposer is known in advance or not, whether the deadline is known or not, and whether the performance profile is deterministic or stochastic. Figure 4 depicts the different possibilities for the games, and we will go through these settings in the following chapters.

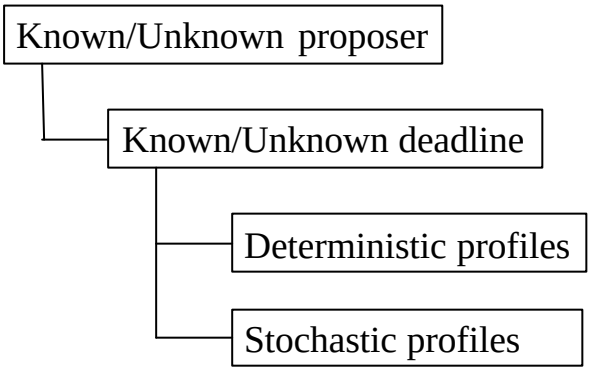

Figure 4. Different possibilities for the games.

Known proposer

If the proposer is known the agent not making the proposal is better off by computing only on its own problem. By this way it maximizes its fallback value, and it will accept any offer greater than its fallback value.

The strategies for the agent making the proposal differ based on whether the deadline is known or not.

Known deadline

In the simplest setting, both the deadline and proposer are common knowledge. The game differs based on whether the performance profiles are deterministic or stochastic.

Deterministic performance profiles

If the performance profile trees are deterministic (see chapter " A performance profile tree"), and common knowledge, the equilibria can be analytically determined. This is easy to conceive, since if agent α knows exactly what is the value agent β will get if it

allocates all the computation to its own problem, then α has two alternatives: either to compute its own problem, or the joint problem. Agent α makes the choice based on the β's fallback value, v_β^β, α's own fallback value, v_α^α, and the value of the joint problem, v_α^{joint}. These values can be determined from the deterministic profile trees: τ_α^α, τ_α^β, τ_α^{joint}, and τ_β^β.

Stochastic performance profiles

If the performance profiles are shared but stochastic, determining the equilibrium is more difficult. Agent α cannot be sure about the value agent β has reached as its fallback value.

If the agents use same algorithms and, therefore, have same kind of performance profiles, agent α may find it useful to deliberate on agent β's problem in order to refine its beliefs about the value agent β has obtained. On the other hand, if the agents do not share the algorithms, any deliberation that agent α does on agent β's problem may not correctly correspond to the value agent β has reached. Therefore, agent α gets no utility from computing on agent β's problem.

Without loss of generality we assume that agent α is the proposer and the deadline is at time T. If we consider that agent β's performance profile tree is stochastic agent α's expected utility at time T can be computed as:

$$E\{p_a\} = P_A(x_a^0)(V(n_a^{joint}) - x_a^0) + (1 - P_A(x_a^0))V(n_a^a) \quad (7)$$

where $P_A(x_\alpha^0)$ (A for accept) is the probability that agent β will accept an offer x_α^0, and accordingly, $(1 - P_A(x_\alpha^0))$ is the probability that the proposal will be rejected. The probabilities are determined by agent α's beliefs about what value agent β has computed for its own individual problem.

If we also take into account that agent α's performance profile tree is stochastic, there is uncertainty about in what deliberation state agent's deliberation strategy resulted. Agent α's expected utility function can be extended to include also the probability that the strategy resulted in a certain deliberation state.

$$E\{p_a(S_a^D)\} = \sum_{q_a(T) \in \Theta_a(t_a^a, t_a^b, t_a^{joint})} P(q_a(T) \mid S_a^D)\left(P_A(x_a^0)(V(n_a^{joint}) - x_a^0) + (1 - P_A(x_a^0))V(n_a^a)\right) \quad (8)$$

where $P_A(\theta_\alpha(T) \mid S_\alpha^D)$ is the probability of being in deliberation state $\theta_\alpha(T)$ after following deliberation strategy S_α^D. The sum is taken over all the possible deliberation states that can be achieved at time T.

The authors develop a dynamic programming algorithm to determine agent α's best response to agent β's strategy. Algorithm works backwards, and at each step it computes which deliberation action, a^z, is optimal if agent α finds itself in deliberation state $\theta_\alpha(t)$ at time *t*. The sequence of actions obtained is agent α's best-response deliberation strategy to agent β.

In order to understand the algorithm we have to define a couple of more denotations. The best offer that agent α can make to agent β, given that agent α is in state $\theta_\alpha(T)$, is

$$x_a^0(q_a(T)) = \arg\max_x \left(P_A(x)(V(n_a^{joint}) - x) + (1 - P_A(x))V(n_a^a)\right) \quad (9)$$

Let $\pi_\alpha^D(A^z, \theta_\alpha(t))$ denote the utility to agent α of computing on problem *z* (taking the action A^z) at time *t* + 1 when it finds itself in deliberation state $\theta_\alpha(t)$. Here, we assume that agent β follows a strategy S_β^D known to agent α. Algorithm 1 computes the best-response strategy for agent α.

For each deliberation state at time T

$$x_a^0(q_a(T)) \leftarrow \arg\max_x \left(P_A(x)(V(n_a^{joint}) - x) + (1 - P_A(x))V(n_a^a)\right)$$

For time *t* = T – 1 down to 0

$$A^z(q_a(t)) \leftarrow \arg\max_{A^z} E\left\{p_a^D(A^z, q_a(t))\right\}$$

Return $\left(A^z(q_a(t))_{t=0}^{T-1}, (x_a^0(q_a(T)), V(n_a^a))\right)$

Algorithm 1a. Pseudo-code presentation of the algorithm for the best response strategy for agent a**, given that** a **is the proposer, deadline is known, and the performance profiles are stochastic.**

For each deliberation state at time T

Find the offer, which will give the highest expected utility.

For time t = T – 1 down to 0

Find the action A^z, which will give the highest expected utility at time *t* + 1 when agent α finds itself in deliberation state $\theta_\alpha(t)$.

Return a vector, which consists of two vectors: one containing all the optimal actions for each deliberation state from time *t* = 0 to *t* = T –1, and an offer-accept vector containing the amount that agent would offer at time T if it were the proposer, and the value it would accept if the other agent made the proposal.

Algorithm 1b. "Plain English" explanation for the algorithm 1a.

The authors claim that algorithm 1 correctly computes a PBE strategy for agent α.

Unknown deadline

If the agents do not know the deadline, a new source of uncertainty, the probability that the deadline arrives at time *t*, is added to the problem setting. Agents update their beliefs about a probability distribution of the deadline whenever time *t* is reached but the deadline does not arrive. The authors present dynamic programming algorithms both for **deterministic** and **stochastic performance profiles** that incorporate the probability for the deadline, and correctly compute a PBE strategy for agent α.

Unknown proposer

If the proposer is unknown neither agent may have a dominant strategy. There are instances where in equilibrium neither agent has a pure strategy, but the game may have a mixed strategy PBE. See [2] for an example game setting.

No Pareto efficient outcome

It is often of interest to ask whether an outcome is optimal. An optimal outcome has the property of *Pareto efficiency*. It means that there is no alternative outcome where some agent is better off without making some other agent worse off.

In the setting where the proposer is unknown it is possible to have an outcome that is not Pareto efficient. Here, we present an example where agents allocate their deliberation resources in a non-optimal way in equilibrium.

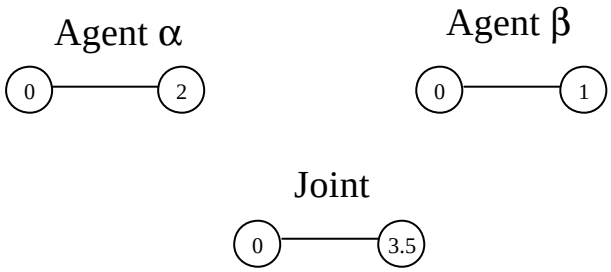

Figure 5. Performance profile trees where the equilibrium is not Pareto efficient.

Figure 5 depicts the performance profiles for both agents and the joint problem. Agents are allowed to make only one deliberation step, and the probability for agent α to be the proposer is ½. Each agent has three different strategies in the game, and the strategy to follow depends on what the other agent has done ($i = \{\alpha, \beta\}$):

S_i^1: OA_i=(null, 'own value'). Agent calculates it's own problem, offers null if it's the proposer, and accepts any offer greater than or equal to its own value.
S_i^2: OA_i=(0, 0). Agent calculates the joint problem, offers 0 if it's the proposer, and accepts any offer greater than or equal to 0.
S_i^3: OA_i=('other's value', 0). Agent calculates the joint problem, offers 'other agent's value' if it's the proposer, and accepts any offer greater than or equal to 0.

The game is presented in table 2. Here, we will demonstrate, as an example, how to calculate the value for a situation where both agents played S^3. If agent α is the

proposer it will offer 1, and gain 3.5 – 1 = 2.5. In this case agent β will accept the offer and it gets 1 as agreed. On the other hand, if agent β is the proposer, it will offer 2, and agent α will accept the offer. The value for agent β in this case is 3.5 – 2 = 1.5, and for agent α the value is 2. If we sum over the two possibilities we get ½ * (2.5 + 2) = 2.25 for agent α, and ½ * (1.5 + 1) = 1.25 for agent β.

There exists a unique pure Nash equilibrium where each agent computes its own problem. However, the equilibrium outcome is not Pareto efficient, since both agents would be better off if agent α played S_α^3 and agent β played S_β^3.

	$\mathbf{S_b^1}$	$\mathbf{S_b^2}$	$\mathbf{S_b^3}$
$\mathbf{S_a^1}$	2 1	2 0	2 0.75
$\mathbf{S_a^2}$	0 1	1.75 1.75	2.75 0.75
$\mathbf{S_a^3}$	1.25 1	1.25 2.25	2.25 1.25

Table 2. Example of setting with no Pareto efficient outcome. In each cell there are two values, one for agent a**, and the other one for agent** b**. The values indicate the utility for each agent, if the agents followed the strategies written in bold face type.**

In general, solving the unknown proposer problem is hard. The authors present a general method for solving the problem with an unknown proposer, but they remind that the general techniques presented do not take advantage of specific properties of the particular game. Specially designed algorithms, as the one presented above, may be more efficient than general techniques for computing equilibria. Readers interested in the general method are referred to [2].

Other sources of uncertainty

Randomized algorithms

The behavior of randomized algorithms is dependent both of their input and also values produced by a random number generator. Simulated annealing is an example of randomized algorithms [1, 3].

The authors present an *augmented performance profile tree*, which include two types of nodes, value nodes and random nodes. In figure 6, an example of an augmented performance profile tree is presented. Node B is a random node, and the edges emanating from it contain certain probabilities.

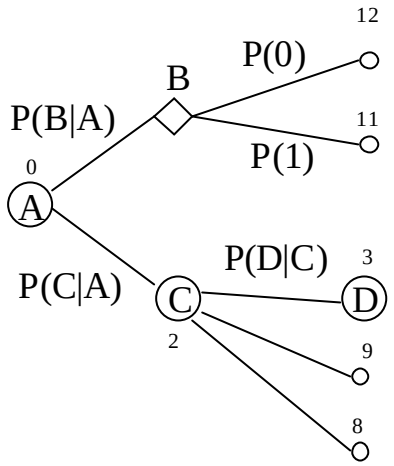

Figure 6. An augmented performance profile tree.

The roles of deliberation are slightly different when agents use randomized algorithms. Agent computing other agent's problem may not be sure that its random number generator produces the same numbers as the other agent's generator does. An agent can use its deliberation resources to emulate the run of a random algorithm by choosing appropriate "random" values. By emulating agent can find out what possible values the other agent may have achieved.

Agents with different algorithms

Agents do not necessarily use the same algorithms. Agent α may be very skilled at solving one problem type while agent β have different algorithms skills. In this case, if the algorithms do not correlate anyhow, there is no need for an agent to use its deliberation resources to compute solutions for the other agent's problem. If the algorithms correlate an agent may find it useful to deliberate on opponent's problems.

Agents do not know each others' problem instances

Another source of uncertainty occurs when agents do not know what problem instance, tasks and resources, the opponent has. In this case, each agent must have a probability distribution over possible problem instances of the opponent. These probabilities can be coded in an augmented performance profile tree as presented in figure 7. In order to update the probabilities, the agents must learn each others' problem instances at the end of the game.

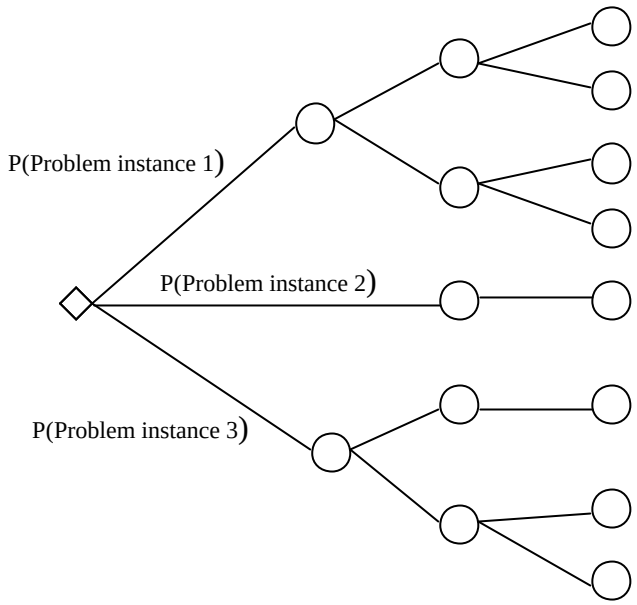

Figure 7. An augmented performance profile tree that one agent uses for the other agent's problem, when there is uncertainty about other agent's problem instance. The tree can be used to determine which node the other agent might have resulted in its calculations.

Conclusions

We have now shortly presented what the authors suggest as a framework for analyzing the non-cooperative game of self-interested rational agents with **limited computational** capabilities.

In the setting, each agent has an intractable optimization problem, and a limited amount of time to carry out the computations. The agents can benefit from pooling the problems and implementing the joint problem, but if the cooperation is not achieved agents implement their individual solutions. When the deadline comes one of the agents makes a proposal for the other. The game differs whether the proposer is known or not.

The authors analyzed the game with some sources of uncertainty presented. Agents may have randomized algorithms in use, they may use different algorithms, or one of the agents may not know exactly the opponent's problem instance.

The framework the authors presented seems to be well justified and robust for analyzing the game of two agents with limited computational capabilities. In simple cases an equilibrium is achieved, and agents have pure strategies which to follow.

The authors discuss the algorithms from a theoretical point of view. They also give considerations about the computational complexity, but they do not give any examples of using their algorithms in real-world problems. It would have been interesting to see how the algorithms are implemented, and how they manage some real problem.

The authors present a concept of a performance profile tree, where statistical data is gathered about how the games were played earlier. In simple cases the tree is

constructed efficiently and proven to be useful, but if the solution quality and time are discretized more finely, the number of possible runs increases, and more runs need to be seen to populate the space. The space should be populated densely to make the values in the tree correlate with the real probabilities of the game. This may be difficult if there are many sources of uncertainty in the game. The more complicated the setting is the more runs it takes for the performance profile trees to saturate to useful values, and the game to saturate towards an equilibrium.

One can imagine that in reality, there may be many sources of uncertainty involved, and the computation of equilibrium is not that straightforward. Especially in the Internet, where agents meet to solve common problems, these uncertainties must be taken into consideration. Agents may have different bandwidths in use, the communication line may be disconnected from time to time, some data may be lost, the agents may use different algorithms, have different problem instances, and the computational capabilities of the agents may vary.

The authors have plans to extend their framework in several directions. One interesting improvement would be to take more agents into the game. This makes the game setting a way more complex because of the combinatorial explosion of different ways to pool the problems. One can imagine that with more than two agents the equilibrium is harder to obtain, but it is important to understand multi-agent games in the environments of distributed computation, like the Internet.

References

[1] Kirkpatrick, S.; Gelatt, C.; Vecchi, M. Optimization by simulated annealing, Science 220 p. 671–680. 1983.

[2] Larson, K.; Sandholm T. Bargaining with limited computation: Deliberation equilibrium. Artificial Intelligence 132, p. 183-217. 2001.

[3] Luke, B. Simulated Annealing. http://members.aol.com/btluke/simann1.htm

[4] Rubinstein, A. Modeling bounded Rationality. MIT Press. Cambridge, MA. 1998.

[5] Sandholm, T; Lesser, V.R. Coalitions among computationally bounded agents, Artificial Intelligence 94 (1), p 99-137. Special issue on Economic Principles of Multiagent Systems. 1997.

[6] Simon, H. Rational Decision Making in Business Organizations. The American Economic Review, Volume 69, Issue 4, 493-513. September. 1979. http://www.sjcny.edu/~kaplan/pdf/simon_79.pdf

[7] Simon, H. Decision Making and Problem Solving. National Academy Press. Washington, DC. 1986. http://www.dieoff.org/page163.htm