Equilibrium Approximation in Simulation–Based Extensive–Form Games

Nicola Gatti
Politecnico di Milano
Piazza Leonardo da Vinci 32
Milano, Italy
ngatti@elet.polimi.it

Marcello Restelli
Politecnico di Milano
Piazza Leonardo da Vinci 32
Milano, Italy
restelli@elet.polimi.it

ABSTRACT

The class of *simulation–based games*, in which the payoffs are generated as an output of a simulation process, recently received a lot of attention in literature. In this paper, we extend such class to games in extensive form with continuous actions and perfect information. We design two convergent algorithms to find an approximate subgame perfect equilibrium (SPE) and an approximate Nash equilibrium (NE) respectively. Our algorithms can exploit different optimization techniques. In particular, we use: *simulated annealing*, *cross entropy* method, and *Lipschitz optimization*. We produce an extensive experimental evaluation of the performance of our algorithms in terms of approximation degree of the optimal solution and number of evaluated samples. Finding approximate NE and SPE requires exponential time in the game tree depth: an SPE can be computed in game trees with a small depth, while the computation of an NE is easier.

Categories and Subject Descriptors

I.2.11 [**Computating Methodologie**s]: Distributed Artificial Intelligence.

General Terms

Algorithms, Economics.

Keywords

Game Theory (cooperative and non-cooperative).

1. INTRODUCTION

Non–cooperative game theory provides formal tools to model situations wherein rational agents interact and describes the pertinent solution concepts [5]. The central solution concepts are the Nash equilibrium for games in which agents play simultaneously (said in *strategic form*) and the subgame perfect equilibrium when agents play sequentially (said in *extensive form*). Game theory proves that any finite game admits at least an equilibrium (Nash and subgame perfect), however it leaves open the problem to compute it. Equilibrium computation is currently one of the most challenging problem in computer science [17].

Cite as: Equilibrium Approximation in Simulation–Based Extensive–Form Games, Gatti, Restelli, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. 199-206.

The formal model of a game is based on the concept of *mechanism*. It defines the rules of the game, specifying the number of roles of the agents, the actions available to the agents, the sequential structure of the game, and the preferences of the agents over the outcomes (usually expressed as utility functions). Almost the entire game theory deals with games where the agents' utility functions are known in analytical form. Recently, the class of *simulation–based games* have been proposed, in which the agents' utility are not analytical known, but they are the result of a (usually continuous) simulation process. The main interest in studying these games lays in developing algorithms able to find agents' (approximate) equilibrium strategies without having any information about the simulation process.

The main work on equilibrium computation for simulation–based games is described in [21]. The authors provide algorithms based on best response iteration to find an approximate Nash equilibrium, where the best responses are approximated by using stochastic optimization techniques (precisely, simulated annealing). The authors discuss also the conditions that assure their algorithms to converge (in probability) to an equilibrium. However, this work is applicable only to games in strategic form. The study of simulation–based games in extensive form has not received enough attention. To the best of our knowledge, the unique pertinent result is provided in [20], where the authors employ the simulation–based framework for mechanism design.

In this paper, we provide the first study of simulation–based (continuous) extensive–form games. After having defined the class of simulation–based extensive–form games (Section 2), we provide two algorithms to compute an approximate Nash equilibrium (NE) and an approximate subgame perfect (SPE) respectively, that work directly on the game tree (Section 3). These algorithms exploit black–box (stochastic) optimization techniques. We develop our algorithms with three different optimization techniques: simulated annealing (as in [21]), cross entropy method, and Lipschitz optimization. Then, we experimentally evaluate the performance of our algorithms (and optimization techniques) in terms of: ϵ value of the found ϵ-approximate equilibrium, number of evaluated samples, and time (Section 5). We experimentally show that an NE can be found in game trees deeper than in the SPE case.

2. SIMULATION-BASED AND EXTENSIVE-FORM GAMES

The class of simulation–based games was introduced in [21] and captures situations where the agents' utility functions

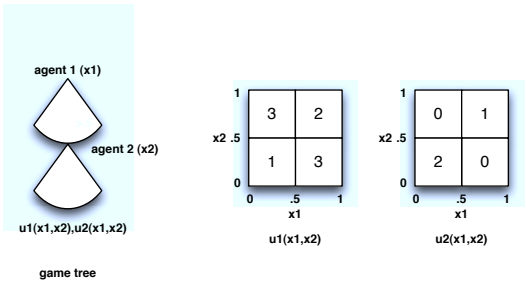

Figure 1: A two–level continuous game with two agents: game tree (left), and agents' utility functions (middle and right).

are not analytically known, but they are given as the result of a simulation process. Formally, there is an oracle $\mathcal{O}$ that, given an outcome of the game, produces a (possibly noisy) sample from the agents' joint utility functions. Since most of simulation processes are based on real–valued variables, a very interesting class of simulation–based games is that of games where the agents' actions are continuous. In these games, the actions available to the agents are the assignment of values to one or more real–valued variables. In what follows, we define the concept of simulation–based extensive–form games and we review the appropriate solution concepts. As customary in game theory, we distinguish the *mechanism*, that specifies the game rules, from the *strategies*, that specify the agents' behavior during the game.

2.1 Mechanism

A finite perfect–information extensive–form game is a tuple $(N, A, V, T, \iota, \rho, \chi, u)$, where: N is the set of n agents, A is a set of actions, V is the set of decision nodes of the game tree, T is the set of terminal nodes of the game tree, $\iota : V \to N$ is the agent function that specifies the agent that acts at a given decision node, $\rho : V \to \wp(A)$ returns the actions available to agent $\iota(v)$ at decision node v, $\chi : V \times A \to V \cup T$ assigns the next (decision or terminal) node to each pair composed of a decision node v and an action a available at v, and $u = (u_1, \ldots, u_n)$ is the set of agents' utility functions where $u_i : T \to \mathbb{R}$. An extensive–form game is with *imperfect–information* when some action of some agent is not perfectly observable by the agent's opponents. In this paper, we limit our study to perfect–information games. Furthermore, we focus on games with *perfect recall*, where every agent recalls all the previously undertaken actions.

In our work, we consider *continuous* perfect–information extensive–form games. In these games, sets A, V, and T are compact. Given a decision node v, agent $\iota(v)$ has a continuous set of actions $\subseteq A$. Each action can lead to a different decision or terminal node. (The model can be easily extended to the case in which A, V, and T are mixed sets composed of continuous and discrete elements.)

Example 2.1. *Consider Figure 1, there are two agents (i.e., agent 1 and agent 2) that play according to a two–level game tree where the first agent to play is agent 1. Agent 1 can assign a real value to $x1$ from the range $[0, 1]$. After the assignment by agent 1, agent 2 can assign a real value to $x2$ from the range $[0, 1]$. Agents' utility functions are defined on $x1$ and $x2$ as shown in figure.*

In a simulation–based extensive–form game, players' utility functions u are not known. Formally, the component u in the tuple $(N, A, V, T, \iota, \rho, \chi, u)$ is substituted by oracle $\mathcal{O}$ whose argument is $t \in T$. We say that $(N, A, V, T, \iota, \rho, \chi, \overline{u})$ where $\overline{u} = E[\mathcal{O}]$ is the *underlying game* of the simulation–based game where the oracle is $\mathcal{O}$. Given $u(t)$, we denote by $u_i(t)$ the component of u reporting the utility of agent i.

2.2 Strategies

In an extensive–form game, a *pure strategy* σ_i is a plan of actions specifying one action for each decision node of agent i. A mixed strategy σ_i is a randomization over pure strategies (plans). When the game is continuous, the definition of a single plan is extremely complex and cannot be conveniently used. An alternative and more compact representation is given by *behavioral strategies*. These define the behavior of an agent at each node independently of the choices taken at other nodes. Essentially, a behavioral strategy σ_i assigns each decision node $v \in V$ a probability distribution over the actions available at v. With perfect recall, the two representations (plans and behavioral) are equivalent. A *strategy profile* collects the strategies of all the agents and is defied as $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$.

With continuous games, behavioral strategies are usually expressed as functions that map actions played at previous nodes to an action (or a probability distribution) available at the current node.

Example 2.2. *Consider the game in Fig. 1, possible strategies for agent 1 and agent 2 are:*

$$\sigma_1 = \{x1 = .3$$

$$\sigma_2 = \begin{cases} x2 = .6 & \text{if } x1 \leq .2 \\ x2 = .9 & \text{if } x1 > .2 \end{cases}$$

Solving a game means to find a strategy profile in which agents' strategies are somehow in equilibrium. Under the assumption that information is complete and common we can define the concept of *Nash equilibrium* (NE) as a strategy profile $\boldsymbol{\sigma}$ such that for all $i \in N$: σ_i is a best response to σ_{-i} where σ_{-i} is given by $\boldsymbol{\sigma}$ once σ_i has been removed (easily, a strategy σ_i is a best response when no other strategy provides a larger utility). It is well known that in extensive–form games some Nash equilibria may be not reasonable with respect to the sequential structure of the game [5]. When information is perfect, the appropriate refinement of Nash for extensive–form games is the *subgame perfect equilibrium* (SPE) [5]. With perfect information, a *subgame* is a subtree of the game tree. Obviously, in continuous games, there are infinite subgames. A subgame perfect equilibrium is a strategy profile that is a Nash equilibrium in every subgame. Every finite extensive–form with perfect information has at least one subgame perfect equilibrium in pure strategies [5]. The same result holds when the game is continuous with bounded utility functions [8]. Since the subgame perfect equilibrium is a refinement of the Nash equilibrium, every continuous perfect–information extensive–form game always admits at least one Nash equilibrium (instead, continuous one–shot games may not admit any Nash equilibrium).

Example 2.3. *Consider the game depicted in Fig. 1. There is 'essentially' a unique SPE (rigorously speaking, there are infinite SPEs, but they are all equivalent in terms of utilities), represented in Fig. 2 where:*

$$\sigma_1 = \{x1 > .5$$

$$\sigma_2 = \begin{cases} x2 \leq .5 & \text{if } x1 \leq .5 \\ x2 > .5 & \text{if } x1 > .5 \end{cases}$$

This game does not admit additional NEs in pure strategies.

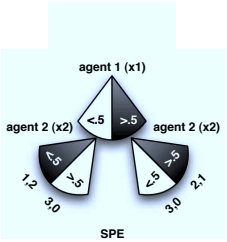

Figure 2: A representation of the unique SPE of the game reported in Fig. 1. The black slices represent the agents' optimal strategies.

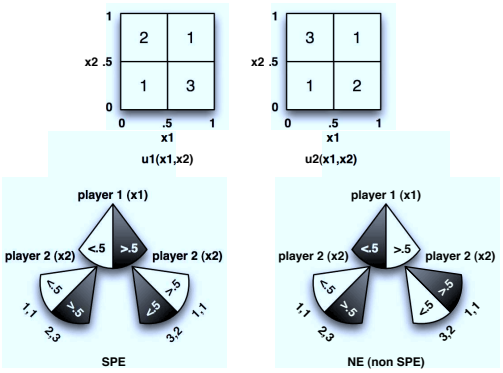

Figure 3: Agents' utility functions and equilibrium strategies (an SPE and an NE that is not an SPE). The black slices represent the agents' optimal strategies.

EXAMPLE 2.4. *Consider a variation of the game in Fig. 1, where utilities are reported in Fig. 3. There is 'essentially' a unique SPE and an additional NE in pure strategies. In the (non–SPE) NE , agent 2 makes a non–credible threat, committing to play $x2 = z$ with $z > .5$ when agent 1 play $x1 = w$ with $w > 0.5$, to force agent 1 to play $x1 = w$ with $w \leq 0.5$.*

Since our aim is to approximate equilibrium strategies in simulation–based games, we resort to the concepts of approximate equilibrium. Several concepts are available in the literature: ϵ–close, ϵ–Nash, and ϵ–perfect equilibria. An ϵ–close equilibrium is a strategy profile $\boldsymbol{\sigma}$ in which the distance (according to some metric) between every σ_i and σ_i^* is smaller than ϵ, where σ_i^* is the optimal strategy of i in an NE $\boldsymbol{\sigma}^*$. This concept provides a measure of the approximation degree of the strategy. However, it is usually considered non–satisfactory and it is preferred the provision of the approximation degree of the expected utility. An ϵ–Nash equilibrium is a strategy profile $\boldsymbol{\sigma}$ where no agent can improve more than ϵ her utility by a unilateral deviation, while an ϵ–perfect equilibrium takes into account also the deviations of the opponents at all the subgames.

3. ALGORITHMS

The approximation of an equilibrium in simulation–based extensive–form games cannot be tackled with the algorithms proposed for strategic–form games. To use these algorithms, we need to represent the agents' strategies as plans of actions and this is impractical.[1] This pushes for the development of *ad–hoc* algorithms that work directly on the game tree.

[1]Notice that the game cannot be solved by using the algorithm described [21] assuming that agent 1 and agent 2 assign values to x1 and x2 respectively. This would neglect the game sequential structure.

Algorithm 1 SPE_APPROXIMATION(v)

```
1: if v is terminal then
2:    return O(v)
3: else
4:    {σ^k(v)} ← SAMPLE_INITIALIZATION(ρ(v))
5:    while true do
6:       for each σ^k(v), assign u^k = SPE_APPROXIMATION(χ(v, σ^k(v)))
7:       if TERMINATION_CONDITION ({σ^k(v)}, {u^k}) then
8:          return arg max_{u∈{u^k}} u_{ι(v)}
9:       else
10:         {σ^k(v)} ← SAMPLES_GENERATION(ρ(v), {σ^k(v)}, {u^k})
11:      end if
12:   end while
13: end if
```

We provide two algorithms to compute an approximate NE and an approximate SPE respectively. For sake of exposition, at first we present the algorithm for finding an approximate SPE. The algorithm for finding an approximate NE is an extension of the previous one. In both algorithms, we use $\sigma(v)$ to specify the agent $\iota(v)$'s strategy at node v.

3.1 Computing an approximate SPE

Formally, for each node $v \in V$ an SPE is defined as follows:

$$\sigma^{SPE}(v) = \arg \max_{\sigma \in \rho(v)} u^{SPE}_{\iota(v)}(\chi(v, \sigma)), \quad (1)$$

where $u^{SPE}(v)$ is defined as

$$u^{SPE}(v) = \begin{cases} \mathcal{O}(v) & \text{if } v \text{ is terminal} \\ u^{SPE}\left(\chi\left(v, \sigma^{SPE}(v)\right)\right) & \text{otherwise.} \end{cases}$$

Algorithm 1 reports the pseudo–code to compute an approximate SPE. The algorithm, that works recursively, receives, as input, the current node v. Initially, the algorithm is called with v as the root node. If the current node v is terminal, then the oracle is called and a sample of agents' utilities are returned (Line 2). Otherwise, the optimal strategy at v is searched as follows. Initially, one or more action samples are generated (Line 4). Then, each sample $\sigma^k(v)$ is evaluated by recursively calling SPE_APPROXIMATION on the node reachable by application of action $\sigma^k(v)$ to node v (Line 6). Given the evaluation of all the samples, if a termination condition holds (Line 7), the largest utility among those of all the samples is returned (Line 8). Otherwise, new samples are generated (Line 11). Functions SAMPLE_INITIALIZATION, SAMPLES_GENERATION, and TERMINATION_CONDITION are based on black–box non–linear optimization techniques (their description is provided in Section 4).

EXAMPLE 3.1. *Consider the game in Fig. 1. Algorithm 1 works as follows. At first some samples of $x1$ are generated, e.g., $\{.2, .4, .8\}$. For each sample, the algorithm is recursively called and samples for $x2$ are produced, e.g., $\{.3, .5, .8\}$ for $x1 = .2$. Then, an iterative optimization process based on resampling is carried on to optimize the value of $x2$ (i.e., $\leq .5$ for $x1 = .2$) and, finally, to return the evaluation of utility associated with the sample of $x1 = .2$ (i.e., $u_1 = 1$). Once utility associated with all the generated samples of $x1$ are evaluated, an iterative optimization process based on resampling is carried on to optimize the value of $x1$ (i.e., $u_1 = 2$).*

Provided that the black–box non–linear optimization technique converges to the global optimum (see Section 4.4), it is easy to show that the proposed algorithm computes the solution of problem in Equation 1.

Algorithm 2 NE_APPROXIMATION (v)

1: **if** v is non–preterminal **then**
2: $\sigma(v) \leftarrow$ a random value uniformly distributed in $\rho(v)$
3: $u \leftarrow$ NE_APPROXIMATION$(\chi(v, \sigma(v)))$
4: **while** true **do**
5: $[\hat{u}, \hat{\sigma}(v)] \leftarrow$ NE_VALIDATION(v, v)
6: **if** $\hat{u}_{\iota(v)} \leq u(v)$ **then**
7: **return** u
8: **else**
9: $\sigma(v) \leftarrow \hat{\sigma}(v)$
10: $u \leftarrow$ NE_APPROXIMATION$(\chi(v, \sigma(v)))$
11: **end if**
12: **end while**
13: **else**
14: $\{\sigma^k(v)\} \leftarrow$ SAMPLE_INITIALIZATION$(\rho(v))$
15: **while** true **do**
16: for each $\sigma^k(v)$, assign $u^k = \mathcal{O}\left(\chi(v, \sigma^k(v))\right)$
17: **if** TERMINATION_CONDITION$\left(\{\sigma^k(v)\}, \{u^k\}\right)$ **then**
18: **return** $\arg\max_{u \in \{u^k\}} u_{\iota(v)}$
19: **else**
20: $\{\sigma^k(v)\} \leftarrow$ SAMPLES_GENERATION$\left(\rho(v), \{\sigma^k(v)\}, \{u^k\}\right)$
21: **end if**
22: **end while**
23: **end if**

3.2 Computing an approximate NE

Surprisingly, although the NE concept poses constraints less hard than the SPE concept, the algorithm computing an NE results to be more twisted than the one computing an SPE. An NE can be formulated as in Equation 1 as far as the nodes on the equilibrium path are considered, while off the equilibrium path other agents may be non-maximizers.

Algorithm 2 depicts the pseudo–code computing an approximate NE. In the computation of an NE, two phases can be recognized. During the first phase (entirely executed by Algorithm 2), a strategy profile $\boldsymbol{\sigma}$, limited to the equilibrium path, is searched. During the second phase (mainly executed by Algorithm 3), a possible strategy profile off the equilibrium path is searched such that $\boldsymbol{\sigma}$ is an NE. Essentially, the second phase validates that $\boldsymbol{\sigma}$ is an NE. The two phases iteratively alternate during the execution until an approximate NE has not been found. The details follow.

(First phase) In Algorithm 2, for each non–preterminal node[2] v a random strategy is assigned to $\sigma(v)$ (Line 2) and the algorithm is recursively called on the next node reached by applying $\sigma(v)$ to the current node (Line 3). If v is preterminal, the best strategy of agent $\iota(v)$ at v is searched by using black-box optimization techniques (Lines 14–22). Notice that this optimization works exactly as Lines 6-13 of Algorithm 1 except that here the evaluation of the sample is directly accomplished by calling the oracle without any recursive call of the algorithm. Once Line 18 is executed, a complete strategy assignment $\boldsymbol{\sigma}$ from the root of the tree to a terminal node (along a single path) is built.

EXAMPLE 3.2. *Consider the game depicted in Fig. 3. Algorithm 2 randomly assigns a value to $x1$, e.g., $x1 = 0.3$, and subsequently the optimal value of $x2$ is found, i.e., $x2 \geq .5$.*

(Second phase) The algorithm tries to validate that the found strategy profile $\boldsymbol{\sigma}$ is an NE (Line 5). This is accomplished by Algorithm 3. Given a node v, Algorithm 3 searches for a strategy in the subgames such that agent $\iota(v)$ cannot gain more by deviating from her strategy prescribed by $\boldsymbol{\sigma}$. Differently from what happens in the case of SPE, the strategy in the subgames of v does not need to be sequentially rational. Practically, this means that, while agent $\iota(v)$ will maximize her utility in such subgames, the behavior of her opponents is free, they do not necessarily maximize their utility. To assure that there is not any strategy such that $\iota(v)$ can gain more by deviating form her strategy prescribed by $\boldsymbol{\sigma}$ at v, we assume that all the opponents of $\iota(v)$ behave in the attempt to minimize the utility of $\iota(v)$. In this way, we find the maxmin value of agent $\iota(v)$ from the subgames. If the maxmin value is larger than the utility given by the strategy prescribed by $\boldsymbol{\sigma}$, then $\boldsymbol{\sigma}$ is not an NE, otherwise there is at least a strategy of the subgames such that the strategy prescribed by $\boldsymbol{\sigma}$ is optimal. The validation process must be repeated at every node v. If a strategy is not validated at a given node v, Algorithm 2 assigns the maxmin strategy of the subgames to $\sigma(v)$ (Line 8) and phase 1 is restarted.

[2]A node v is preterminal if, once applied an action to v, a terminal node is reached.

Algorithm 3 NE_VALIDATION (v, v_0)

1: **if** v is terminal **then**
2: **return** $\mathcal{O}(v)$
3: **else**
4: $\{\sigma^k(v)\} \leftarrow$ SAMPLE_INITIALIZATION$(\rho(v))$
5: **while** true **do**
6: for each $\sigma^k(v)$, assign $u^k =$ NE_VALIDATION$\left(\chi(v, \sigma^k(v)), v_0\right)$
7: **if** $\iota(v) = \iota(v_0)$ **then**
8: **if** TERMINATION_CONDITION$\left(\{\sigma^k(v)\}, \{u^k_{\iota(v)}\}\right)$ **then**
9: **return** $[\arg\max_{u \in \{u^k\}} u_{\iota(v)}, \text{best } \sigma^k(v)]$
10: **else**
11: $\{\sigma^k(v)\} \leftarrow$ SAMPLES_GENERATION $\left(\rho(v), \{\sigma^k(v)\}, \{u^k_{\iota(v)}\}\right)$
12: **end if**
13: **else**
14: **if** TERMINATION_CONDITION$\left(\{\sigma^k(v)\}, \{-u^k_{\iota(v_0)}\}\right)$ **then**
15: **return** $[\arg\min_{u \in \{u^k\}} u_{\iota(v_0)}, \text{best } \sigma^k(v)]$
16: **else**
17: $\{\sigma^k(v)\} \leftarrow$ SAMPLES_GENERATION$\left(\rho(v), \{\sigma^k(v)\}, \{-u^k_{\iota(v_0)}\}\right)$
18: **end if**
19: **end if**
20: **end while**
21: **end if**

Algorithm 3 works similarly to Algorithm 1 except that the opponents of agent $\iota(v_0)$, where v_0 is the node whose strategy we are validating, minimize agent $\iota(v_0)$'s utility (Line 15). We use the same black–box optimization techniques used in the previous algorithms, passing $\{-u^k_{\iota(v_0)}\}$ when we need to minimize (Lines 14 and 17).

EXAMPLE 3.3. *Consider the game depicted in Fig. 3. Suppose that Algorithm 2 has assigned $x1 = .3$ and $x2 = .1$, and that* NE_VALIDATION *is executed to validate the strategy of agent 1. A number of samples are generated for $x1$. For each sample of $x1$, the optimal value of $x2$ minimizing agent 1's utility is found. It can be easily observed that the maxmin utility of agent 1 from all the subgames is 1 and therefore the initial strategy is validated.*

The efficiency of Algorithm 3 can be improved by using a pruning procedure similar to the alpha–beta pruning. More precisely, call v_0 the node whose strategy is to validate. To assert that a given strategy $\boldsymbol{\sigma}$ is not an NE, we do not need to compute the maxmin value of v_0, but it is enough to find a strategy $\hat{\sigma}_{\iota(v_0)}$ such that agent $\iota(v_0)$'s utility is larger than

the one provided by the strategy to validate. Moreover, in all the subgames of v_0, we do not strictly need that opponents of agent $\iota(v_0)$ find the minimum of agent $\iota(v_0)$'s utility, but it is sufficient that they find an action such that agent $\iota(v_0)$'s utility is not larger than the one provided by the strategy to validate. Therefore, the termination condition at Line 8 can be safely modified in the following way: as soon as a sample provides agent $\iota(v_0)$ with a utility larger than the one provided by the strategy to be validated, the algorithm returns such utility and the corresponding sample. Similarly, at Line 14 it is possible to make the algorithm return as soon as a sample, that provides agent $\iota(v_0)$ with a utility non–larger than the one provided by the strategy to be validated, is found.

EXAMPLE 3.4. *Consider Example 3.3. Every time a sample of $x2$ is evaluated and it provides a utility of 1 to agent 1, no further samples are generated.*

4. OPTIMIZATION ALGORITHMS

As shown in the previous section, the computation of an SPE and an NE in extensive–form games requires to solve a number of recursive optimization problems. For each node v we need to maximize an unknown objective function which depends on the solutions of other optimization problems defined in the nodes of the sub–tree rooted at v. In a continuous extensive–form game, this means to solve a number of unconstrained continuous optimization problems. In the literature, researchers have proposed many optimization methods, which can be split into two main categories: deterministic methods (e.g., real algebraic geometry [2], Lipschitz optimization [18]) and non–deterministic (or stochastic) ones (e.g., evolutionary algorithms [7], simulated annealing [10], cross-entropy method [16], particle swarm optimization [9]). The optimization process which is common to all these algorithms is synthetically summarized in Algorithm 4. Starting from one or more initial samples within the search space D, at each iteration, new candidate solutions are generated and, on the basis of their scores (i.e., the corresponding objective function values) and eventually other information about the objective function, the search is directed towards the most promising regions in the search space. The search process iterates until some termination condition is met: for instance, many optimization algorithms stop when the improvement in a sequence of consecutive iterations falls below a predefined threshold or a predefined maximum number of iterations is reached. In the following subsections we will focus on one deterministic algorithm (Lipschitz optimization) and two random–search approaches (simulated annealing and cross–entropy method), and we will describe how the main steps of the search process are implemented in each of them. For sake of simplicity, we will present the algorithms for one–dimensional optimizations, but extensions to multiple dimensions are straightforward [1].

4.1 Lipschitz optimization

Lipschitz optimization is a deterministic approach to the global optimization problem. The basic assumption of this approach is that the objective function $u(x)$ satisfies the Lipschitz condition over the closed optimization interval $[a,b]$. A function is Lipschitz if there exists a finite bound α (called Lipschitz constant) to its rate of change:

$$| u(x_1) - u(x_2) | \leq \alpha | x_2 - x_1 |, \quad x_1, x_2 \in [a,b]. \tag{2}$$

Algorithm 4 Optimization Algorithm

1: $i := 0$
2: $\{x_0^k\}_{1\leq k\leq N}$ = SAMPLE_INITIALIZATION(D)
3: **repeat**
4: $\quad i := i + 1$
5: $\quad \{x_i^k\}_{1\leq k\leq N}$ = SAMPLE_G $\left(D, \{x_{[0:i-1]}^k\}_{1\leq k\leq N}, \{y_{[0:i-1]}^k\}_{1\leq k\leq N}\right)$
6: $\quad$ for $j = 1$ to N, assign $y_i^j = u(x_i^j)$
7: **until** TERMINATION_CONDITION $\left(\{x_{[0:i]}^k\}_{1\leq k\leq N}, \{y_{[0:i]}^k\}_{1\leq k\leq N}\right)$
8: **return** $\arg \max_{x\in\{x_i^k\}_{1\leq k\leq N}} u(x)$

The key idea of Lipschitz optimization is to select the next query point by maximizing an upper–bound function which is built on the sequence of samples generated by the search process.

Sample initialization. The algorithm starts by generating two samples placed at the boundary of the optimization interval $[a,b]$: $\{x_0^k\} = \{a,b\}$, and computes the corresponding scores $u(a)$ and $u(b)$.

Sample generation. In the following iterations, to select the next sample of the search process, the algorithm, given all the previously generated samples $\{x_{[0:i-1]}\}$ (sorted by value) and the associated scores $\{u(x_{[0:i-1]})\}$, determines the interval $[x^j, x^{j+1}]$ containing the highest value of the upper–bound function:

$$[x^{j^*}, x^{j+1^*}] = \arg \max_{x^j, x^{j+1}\in\{x_{[0:i-1]}\}} \frac{u(x^j) + u(x^{j+1})}{2} + \alpha \cdot \frac{x^{j+1} - x^j}{2}.$$

Once the interval has been identified, the new sample will be generated in the following point:

$$x_i = \frac{u(x^{j+1^*}) - u(x^{j^*})}{2\alpha} + \frac{x^{j^*} + x^{j+1^*}}{2}.$$

Termination condition. The stop criterion for Lipschitz optimization is that the difference between the actual objective-function value and the upper–bound value is lower than a specified global tolerance ϵ_{stop}.

4.2 Simulated Annealing

Simulated Annealing (SA) is a well–known probabilistic metaheuristics algorithm for finding global maximum (or minimum) of an objective function. It works by emulating the physics process whereby a solid is at first heated and then slowly cooled to find a configuration with lower internal energy than the initial one. The idea behind SA is to take a random walk through the search space at successively lower temperatures (i.e., less exploration), where the probability of taking a step is given by a Boltzmann distribution. Although SA is often used when the search space is discrete, here we consider its application to continuous global optimization problems [12].

Sample initialization. The SA algorithm starts from a random sample drawn from a uniform distribution over the search space.

Sample generation. At each iteration i, a new candidate sample $\hat{x}$ is generated from a kernel distribution $\mathcal{K}(\cdot, x_{i-1})$ (whose definition is critical to the convergence of SA [6]) centered in the last available sample x_{i-1} (we used Gaussian kernels). If the new sample $\hat{x}$ has a score higher than the one of x_{i-1} the sample is accepted. Otherwise, to avoid getting trapped in local maxima, it can be still accepted with

a likelihood that is proportional to the temperature parameter τ and the score difference $u(\hat{x}) - u(x_{i-1})$ (Metropolis algorithm [15]):

$$x_i = \begin{cases} \hat{x} & \text{if } p \leq \min\left\{1, e^{-\frac{u(\hat{x})-u(x_{i-1})}{\tau}}\right\} \\ x_{i-1} & \text{otherwise,} \end{cases}$$

where p is a random number drawn uniformly in $[0, 1]$. As the search process goes on, in order to converge to a solution the temperature parameter should decrease according to some cooling scheme [3].

Termination condition. Usually, SA algorithms stop when new samples are consecutively rejected for a fixed number of iterations. In this paper, we use a different criterion (similar to the one adopted in [19]) based on the score of the samples considered in the last M iterations. In particular, the algorithm is stopped at the i-th iteration if:

$$\max_{j\in[i-M,i]} u(x_j) - \frac{1}{M}\sum_{j=i-M}^{i} u(x_j) < \epsilon_{STOP}.$$

In words, the algorithm is stopped when its progress is considered too small.

4.3 Cross-entropy method

The Cross-Entropy (CE) method is a Monte Carlo technique to solve optimization problems. The method consists of two main steps: 1) generate samples according to a probability density defined over the search space, 2) update the probability density parameters by minimizing the cross-entropy (i.e., Kullback-Lieber divergence [11]) with respect to the best samples (*elite* samples). In this way, new samples will be generated in the most promising regions of the search space. Since considering distributions from an exponential family such minimization can be solved analytically, in this paper we use Gaussian densities parameterized by the mean μ and variance σ^2.

Sample initialization. At the first iteration, since no prior information is available, k samples are randomly generated using a uniform distribution over the search space.

Sample generation. At each iteration i, given the sample scores of the previous iteration $\{u(x_{i-1}^k)\}_{1\leq k\leq N}$, a percentage ρ_{CE} of the best samples is selected to estimate the new probability density from which new samples will be drawn in the next iteration. Using Gaussian densities, the minimization of the cross–entropy measure leads to update the parameters of the distribution by simply computing the sample mean and sample variance of the elite samples, i.e., the best $\lceil N \cdot \rho_{CE} \rceil$ samples.

Termination condition. The algorithm is stopped when the improvement in the $(1 - \rho_{CE})$-quantile (i.e., the score of the worst elite sample) does not exceed ϵ_{STOP} for d_{CE} successive iterations, or when the maximum number of iterations t_{max} is reached.

4.4 Convergence Properties

Lipschitz optimization allows one to approximate with arbitrary precision the global maximum of Lipschitz functions when an upper bound to the function derivative (the Lipschitz constant) is known. Since Lipschitz optimization is a deterministic method with very few parameters, there is no need for multiple runs and parameter tuning is minimized. On the other hand, when the Lipschitz constant is unknown or the objective function is not Lipschitz, no guarantees of accuracy can be given. Furthermore, when objective functions have large Lipschitz constants and/or are defined over multiple dimensions, the convergence is very slow.

When no information about the objective function is available, randomized–search methods (e.g., simulated annealing and cross–entropy method) are usually considered. Both the algorithms described in this section are simple and effective approaches to solve continuous optimization problems, even if simulated annealing is a local search algorithm (whose performance depends critically on a proper choice of the cooling scheme), while the cross–entropy method is a global optimization method. It is possible to show that, under quite mild conditions on the kernel distributions and the objective function, such methods converge to the optimum with probability approaching one as the number of samples grows to infinity (for details refer to [6, 13]).

5. EXPERIMENTAL EVALUATION

We implemented our algorithms with Matlab R10 and we executed them with a UNIX computer with dual quad-core 2.33GHz CPU and 8GB RAM. Our experimental activity is structured as follows. Initially, we apply our algorithms to a well–known practical economic problem (i.e., bargaining) modeled as a continuous extensive–form game which presents piecewise linear utility functions and whose exact solution is known in closed form. Subsequently, we apply our algorithms to *ad–hoc* games with a class of highly non–linear utility functions widely studied in multi–agent systems.

5.1 The bargaining case study

We chose the alternating–offers game as case study, being the principal model for strategic bargaining. The alternating–offers game prescribes that two agents, a buyer **b** and a seller **s**, play alternately at discrete time points. Time t is discrete and $\iota(t)$ is defined as follows: $\iota(0)$ is a parameter of the problem and for $t > 0$ it is such that $\iota(t) \neq \iota(t-1)$. The pure strategies available to agent $\iota(t)$ at $t > 0$ are: *offer*$(\overline{x})$, where $\overline{x} \in [0, 1]$; *accept*, that concludes the game with outcome $(\overline{x}, t)$, where $\overline{x}$ is the value offered at $t - 1$, and t is the time point at which the offer is accepted; and *exit*, that concludes the game with outcome $NoAgreement$. At $t = 0$ only actions *offer*$(\overline{x})$ and *exit* are available. Agents' utility functions are defined as follows. Each agent i has a deadline T_i. Before the deadlines, utility functions are defined as $U_{\mathbf{b}}(x, t) = (1 - x) \cdot (\delta_{\mathbf{b}})^t$ for the buyer and $U_{\mathbf{s}}(x, t) = x \cdot (\delta_{\mathbf{s}})^t$ for the seller. After the deadline of agent i, her utility is -1. δ_i and T_i are parameters. The alternating–offers game admits a unique SPE that prescribes that at each time t there is an optimal offer $x^*(t)$ for every $t \leq \min\{T_{\mathbf{b}}, T_{\mathbf{s}}\}$ such that agent $\iota(t)$ makes it and her opponent accepts it at $t+1$. (For the computation of x^* we point the interested reader to [4]) There is no optimal offer at $t > \min\{T_{\mathbf{b}}, T_{\mathbf{s}}\}$, so agents' optimal action is *exit*. That is, the minimal deadline essentially defines the depth of the game tree (exactly, the tree depth is $\min\{T_{\mathbf{b}}, T_{\mathbf{s}}\} + 1$).

We generated some game instances with different minimal deadlines from the range $\{3, 4, 5, 6\}$. We developed simple variations of our algorithms to capture the fact that actions are mixed, combining discrete and continuous actions. Moreover, we force agents to exit when the minimal deadline is expired (otherwise agents can indefinitely play).

At first, we have evaluated the performance of Algorithm 1 with different parameterizations when the minimal deadlines is 3. The used parameterizations and the obtained results

s.p.i.	best samples per iteration						
	2		4		6		
	$E[\cdot]$	std	$E[\cdot]$	std	$E[\cdot]$	std	
10	0.078	0.084	0.082	0.157	0.195	0.186	ϵ
	$3{\cdot}10^4$	$1{\cdot}10^4$	$3{\cdot}10^4$	$9{\cdot}10^3$	$5{\cdot}10^4$	$3{\cdot}10^4$	e.s.
	0.763	0.329	1.684	0.613	2.111	0.864	time (s)
20	0.042	0.045	0.015	0.009	0.017	0.013	ϵ
	$5{\cdot}10^4$	$2{\cdot}10^4$	$2{\cdot}10^5$	$3{\cdot}10^4$	$3{\cdot}10^5$	$7{\cdot}10^4$	e.s.
	2.677	0.730	5.725	0.836	7.987	2.295	time (s)
30	0.036	0.038	0.0048	0.003	0.010	0.003	ϵ
	$4{\cdot}10^5$	$9{\cdot}10^4$	$6{\cdot}10^5$	$9{\cdot}10^4$	$8{\cdot}10^5$	$6{\cdot}10^4$	e.s.
	10.431	2.290	13.991	1.811	20.511	2.038	time (s)
40	0.024	0.016	0.015	0.025	0.005	0.007	ϵ
	$9{\cdot}10^5$	$1{\cdot}10^5$	$1{\cdot}10^6$	$2{\cdot}10^5$	$1{\cdot}10^6$	$2{\cdot}10^5$	e.s.
	18.983	3.789	31.244	7.848	36.310	6.102	time (s)
50	0.013	0.009	0.009	0.008	0.003	0.003	ϵ
	$2{\cdot}10^6$	$3{\cdot}10^5$	$3{\cdot}10^6$	$4{\cdot}10^5$	$3{\cdot}10^6$	$4{\cdot}10^5$	e.s.
	32.801	6.005	49.719	7.551	54.853	7.412	time (s)

Table 1: Experimental results with a bargaining game with depth 3 obtained by applying CE to compute SPE ('s.p.i.' means samples per iteration and 'e.s.' means evaluated samples).

M	temperature (τ)						
	0.3		0.5		0.7		
	$E[\cdot]$	std	$E[\cdot]$	std	$E[\cdot]$	std	
10	0.042	0.030	0.082	0.133	0.031	0.033	ϵ
	$3{\cdot}10^4$	$1{\cdot}10^3$	$4{\cdot}10^4$	$1{\cdot}10^3$	$4{\cdot}10^4$	$2{\cdot}10^3$	e.s.
	0.915	0.051	0.946	0.024	0.956	0.055	time (s)
20	0.027	0.019	0.026	0.024	0.026	0.018	ϵ
	$3{\cdot}10^5$	$3{\cdot}10^3$	$3{\cdot}10^5$	$9{\cdot}10^3$	$3{\cdot}10^5$	$1{\cdot}10^4$	e.s.
	6.741	0.098	6.411	0.353	6.482	0.252	time (s)
30	0.023	0.017	0.020	0.017	0.029	0.024	ϵ
	$1{\cdot}10^6$	$3{\cdot}10^4$	$1{\cdot}10^6$	$3{\cdot}10^4$	$1{\cdot}10^6$	$3{\cdot}10^4$	e.s.
	22.650	0.782	22.820	0.910	24.390	0.623	time (s)
40	0.018	0.014	0.022	0.021	0.019	0.019	ϵ
	$2{\cdot}10^6$	$5{\cdot}10^4$	$2{\cdot}10^6$	$7{\cdot}10^4$	$2{\cdot}10^6$	$7{\cdot}10^4$	e.s.
	54.590	1.154	55.152	1.739	54.322	1.426	time (s)
50	0.018	0.017	0.013	0.181	0.035	0.062	ϵ
	$5{\cdot}10^6$	$1{\cdot}10^5$	$5{\cdot}10^6$	$8{\cdot}10^4$	$5{\cdot}10^6$	$1{\cdot}10^5$	e.s.
	108.627	1.903	107.689	2.865	106.613	2.781	time (s)

Table 2: Experimental results with a bargaining game with depth 3 obtained by applying SA to compute SPE ('M' is defined in Section 4.2and 'e.s.' means evaluated samples).

related to CE optimization are reported in Tab. 1, those related to SA optimization are reported in Tab. 2, and those related to Lipschitz optimization are reported in Tab. 3 (notice that the utilities in the bargaining game are no Lipschitz, not being continuous). We evaluated our algorithm in terms of ϵ value of the approximate ϵ–perfect equilibrium found by the algorithms, number of evaluated samples (e.s.), and computational time. The results reported in the tables are averaged over 10 executions ($\epsilon_{STOP} = 10^{-2}$ for CE and SA).

CE and SA exhibit similar performance in terms of ϵ value. This value reduces when the number of samples per iteration (s.p.i.) and M increase, while there are not optimal values of elite samples and temperature independently of the number of samples per generation. Only with a Lipshitz constant larger than 3, Lipshitz optimization returns a value of ϵ comparable to that returned by the other two optimization techniques. On the other hand, CE and SA evaluated a strictly smaller number of samples (ϵ being equal, CE always outperforms SA). Finally, computational times are proportional to the number of e.s. for all the optimization techniques in the same way (for this reason, we omit the computational time in the following evaluations).

Tab. 4 and Tab. 5 report the performance of Algorithm 1 with CE and SA with their fastest configurations (10 s.p.i. and 2 elite samples for CE, and $M = 2$ and $\tau = 0.3$ for SA) for different values of the minimal deadline. The results are

Lipschitz constant (α)				
1	2	3	4	
0.221	0.143	0.073	–	ϵ
138,359	1,127,485	6,692,909	$> 10^7$	e.s.
10.85	108.52	577.221	–	time (s)

Table 3: Experimental results when computing SPE with Lipshitz optimization in a bargaining game with depth 3 ('e.s.' means evaluated samples).

minimal deadline	samples per iteration					
	10		15		20	
	ϵ	e.s.	ϵ	e.s.	ϵ	e.s.
3	0.078	33,017	0.065	41,674	0.042	48,259
4	0.056	954,356	0.033	4,135,410	0.031	6,259,260
5	0.043	14,354,760	0.039	50,714,660	0.035	73,801,294
6	0.051	270,564,843	–	$> 10^9$	–	$> 10^9$

Table 4: Experimental results when computing SPE in a bargaining game using CE with elite samples equal to 2 ('e.s.' means evaluated samples).

averaged over 10 executions. We observe that the ϵ value is rather small even with minimal deadline equal to 6. The number of e.s. rises exponentially in the length of the minimal deadline. Also in this case CE outperforms SA in terms of evaluated samples.

We evaluate Algorithm 2 with the parameterizations used in Tab. 4 for CE. We report only e.s. because the ϵ–Nash value is zero for all the executions. Finding an NE requires less samples than finding an SPE, thus allowing one to approximate a bargaining problem with deadline 20.

5.2 Games with rugged utility functions

We evaluate the performance of our algorithms when utility functions are highly non–linear. Non–linear utility functions are deeply studied in the negotiation field and a class of utility functions that has received a lot of attention is said *rugged utility* [14]. A rugged utility function is defined as follows. Call $(x_1, \ldots, x_n)$ the arguments of utility function U where $x_i \in [0,1]$. Each domain $[0,1]$ is divided into k intervals where the j-th interval is $[\frac{j-1}{k}, \frac{j}{k}]$. Call $int : [0,1] \to \{1, \ldots, k\}$ the function that, given a continuous value x_i, returns the interval to which x_i belongs. Utility U is defined as $U(x_1, \ldots, x_n) = RAND\frac{1}{n}\sum_i^n int(x_i)$ where $RAND$ is a number randomly drawn from a uniform probability distribution over $[0,1]$. With these utility functions SPEs can be computed exactly.

We generated game trees with a depth $\in \{3,4,5\}$ and rugged utility functions. We executed 10 times Algorithm 1 with cross entropy for each configuration reported in Tab 7 (the number of elite samples is equal to 2). We compared the results returned by our algorithms with respect to the SPE of the game. We report in Tab. 7 the results with tree depth equal to 3 (the results with 4 and 5 are similar, but they require a much larger number of e.s.): the success percentage (suc.), the ϵ value of the associated ϵ–approximate equilibrium, and the number of evaluated samples.

It can be observed that the results with rugged utility functions are worse than those obtained in the bargaining case study. More precisely, the ϵ value and the number of e.s. are larger than those obtained in Section 5.1. The performance decreases with the increasing of k. In order to have a satisfactory success percentage, the number of s.p.i. must be rather larger than k. This poses severe limits to the size of the game trees solvable by the algorithm within reasonable time. We applied Algorithm 2 with CE using the same parameterization and with minimal deadline larger than 3. As

minimal deadline	samples per iteration					
	10		15		20	
	ϵ	e.s.	ϵ	e.s.	ϵ	e.s.
3	0.078	33,017	0.065	41,674	0.042	48,259
4	0.055	1,950,350	0.062	12,295,395	0.057	43,368,477
5	0.047	89,769,110	–	$> 10^9$	–	$> 10^9$
6	–	$> 10^9$	–	$> 10^9$	–	$> 10^9$

Table 5: Experimental results when computing SPE in a bargaining game using SA with $\tau = 0.3$.

minimal deadline	samples per iteration		
	10	15	20
3	532	874	1,146
4	1,361	1,983	3,214
5	2,841	3,324	5,482
6	5,362	8,641	9,215
10	60,316	93,325	341,513
15	301,142	762,413	1,356,234
20	1,882,582	6,241,562	13,646,221

Table 6: Evaluated samples when computing an NE in a bargaining game using CE with elite samples equal to 2 .

in the bargaining case, the computation of an NE requires a strictly smaller number of evaluated samples: (with 2 elite samples) 10^3 e.s. with deadlines 3, 10^4 e.s. with deadlines ≤ 7, 10^5 e.s. with deadlines ≤ 12.

6. CONCLUSIONS

Simulation–based games (i.e., games in which the agents' payoffs are provided as the result of a simulation process) have received a lot of attention in the scientific community. In this paper, we extended such class of games when the games are in extensive form and have continuous actions. We provided two convergent algorithms to compute an approximate subgame perfect and an approximate Nash equilibrium respectively. We used different black–box optimization techniques (simulated annealing, cross entropy, and Lipshitz optimization) in our algorithms and we experimentally evaluated them with different settings. A subgame perfect equilibrium can be computed in game trees with a small depth, while the computation of a Nash equilibrium is easier. Furthermore, the number of evaluated samples being equal, cross–entropy optimization demonstrated to outperform simulate annealing and Lipshitz optimization.

In future works, we try to improve the efficiency of our algorithms for all the situations wherein information on the structure of the problem is available, e.g., when games have an action–graphical structure.

7. REFERENCES

[1] H. Benson, R. Horst, and P. Pardalos. Handbook of Global Optimization, 1995.

[2] J. Bochnak, M. Coste, and M. Roy. *Real algebraic geometry*. Springer Verlag, 1998.

[3] I. Bohachevsky, M. Johnson, and M. Stein. Generalized simulated annealing for function optimization. *TECHNOMETRICS*, 28(3):209–217, 1986.

[4] F. Di Giunta and N. Gatti. Bargaining over multiple issues in finite horizon alternating-offers protocol. *ANN MATH ARTIF INTEL*, 47(3-4):251–271, 2006.

[5] D. Fudenberg and J. Tirole. *Game Theory*. The MIT Press, Cambridge, USA, 1991.

[6] A. Ghate and R. Smith. Adaptive search with stochastic acceptance probabilities for global optimization. *OPER RES LETT*, 36(3):285–290, 2008.

[7] D. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[8] C. Harris. Existence and characterization of perfect equilibrium in games of perfect information. *ECONOMETRICA*, 53:613–628, 1985.

[9] J. Kennedy and R. Eberhart. Particle swarm optimization. In *ICNN*, volume 4, pages 1942–1948, 1995.

[10] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

[11] S. Kullback and R. Leibler. On information and sufficiency. *ANN MATH STAT*, pages 79–86, 1951.

[12] M. Locatelli. Simulated annealing algorithms for continuous global optimization: convergence conditions. *J OPTIMIZ THEORY APP*, 104(1):121–133, 2000.

[13] L. Margolin. On the convergence of the cross-entropy method. *ANN OPER RES*, 134(1):201–214, 2005.

[14] I. Marsa-Maestre, M. Lopez-Carmona., J. Velasco, and E. de la Hoz. Avoiding the prisoner's dilemma in auction-based negotiations for highly rugged utility spaces. In *AAMAS*, pages 425–432, 2010.

[15] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, E. Teller, et al. Equation of state calculations by fast computing machines. *J CHEM PHYS*, 21(6):1087, 1953.

[16] R. Rubinstein and D. Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Springer-Verlag, 2004.

[17] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game Theoretic and Logical Foundations*. Cambridge University Press, 2008.

[18] B. Shubert. A sequential method seeking the global maximum of a function. *SIAM J NUMER ANAL*, 9(3):379–388, 1972.

[19] D. Vanderbilt and S. Louie. A Monte Carlo simulated annealing approach to optimization over continuous variables. *J COMPUT PHYS*, 56(2):259–271, 1984.

[20] Y. Vorobeychik, D. Reeves, and M. Wellman. Constrained automated mechanism design for infinite games of incomplete information. In *UAI*, 2007.

[21] Y. Vorobeychik and M. Wellman. Stochastic search methods for nash equilibrium approximation in simulation-based games. In *AAMAS*, pages 1055–1062, 2008.

k	samples per iteration								
	10			20			30		
10	40%	0.060	$2 \cdot 10^4$	80%	0.036	$2 \cdot 10^5$	90%	0.022	$1 \cdot 10^6$
20	10%	0.074	$2 \cdot 10^4$	30%	0.069	$2 \cdot 10^5$	50%	0.028	$1 \cdot 10^6$
30	0%	0.083	$3 \cdot 10^4$	10%	0.077	$3 \cdot 10^5$	20%	0.052	$1 \cdot 10^6$
40	0%	0.098	$3 \cdot 10^4$	0%	0.082	$3 \cdot 10^5$	10%	0.060	$1 \cdot 10^6$
50	0%	0.153	$4 \cdot 10^4$	0%	0.089	$3 \cdot 10^5$	0%	0.064	$1 \cdot 10^6$
100	0%	0.234	$6 \cdot 10^4$	0%	0.156	$5 \cdot 10^5$	0%	0.145	$1 \cdot 10^6$
	suc.	ϵ	e.s.	suc.	ϵ	e.s.	suc.	ϵ	e.s.

Table 7: Experimental results when computing SPE with rugged function using CE ('e.s.' means evaluated samples and 'suc.' means success probability).