Universal Parameter Optimisation in Games Based on SPSA

Levente Kocsis, Csaba Szepesvári[1]*, Mark H.M. Winands*[2]

ABSTRACT

Most game programs have a large number of parameters that are crucial for their performance. While tuning these parameters by hand is rather difficult, successful applications of automatic optimisation algorithms in game programs are known only the parameters that belong to certain components (e.g. for parameters of an evaluation function). The SPSA algorithm (Simultaneous Perturbation Stochastic Approximation) is an attractive choice for optimising any kind of parameters of a game program, both for its generality and its simplicity. However, the price is that SPSA can be very slow. In this article we propose several methods to speed of SPSA. In particular, we propose to combine it with RPROP, using common random numbers, antithetic variables and averaging. We test the resulting algorithm for tuning various types of parameters of our game programs in two domains, poker and lines of actions. From the experimental study, we conclude that using SPSA is a viable approach for optimisation in game programs, especially if no good alternative exist for the types of parameters considered.

1. INTRODUCTION

!!!!!!!!!!TODO (by Csaba): IMHO this section should be halved in length.

Any reasonable game program has several hundreds if not thousands of parameters. These parameters belong to various components of the program, such as the evaluation function or the search algorithm. Typically, the performance of the game program depends strongly on the settings of these parameters. While it is possible to make educated guesses about "good" values of certain parameters, hand-tuning the parameters is a difficult, cumbersome and time-consuming task. An alternatively approach is to find the "right" values of these parameters by means of some automatic parameter tuning algorithms.

The use of parameter optimisation methods for the performance tuning of game programs is made difficult by the fact that the objective function is rarely available analytically and hence methods that rely on the availability of an analytic expression for the gradient cannot be used. However, there exist several ways to tune parameters despite the lack of an analytic gradient. An important class of such algorithms is represented by the class of temporal-difference (TD) methods (Sutton, 1988) that are designed to tune the parameters of evaluation functions. Probably the best known success story in learning to play is Tesauro's TD-Gammon program that was trained by TD(λ) and achieved a level of play that exceeds that of the best human players (Tesauro, 1992). Another example is KnightCap, where a variant of TD(λ) was used to tune the parameters of the evaluation function of a chess program (Baxter, Tridgell, and Weaver, 2000). Recently, in (Kocsis, 2003a) several algorithms were considered for parameter tuning, some of them being adaptations of general-purpose algorithms (like tabu-search), whilst others were specifically designed to exploit the specific properties of the particular optimisation problems studied. Obviously, any general-purpose (gradient-free) global search method can be used for parameter optimisation in games. Just to mention a few examples, in (Chellapilla and Fogel, 1999) genetic algorithms were used to evolve a neural network to play the game of checkers, whilst in (Björnsson and Marsland, 2003) an algorithm similar to the Finite-Difference Stochastic Approximations (FDSA) algorithm that we will also consider in this article was used successfully for tuning the search-extension parameters of CRAFTY. Despite all the work done so far, we think that automatic tuning of game program parameters remains a largely unexplored area of game programming.

[1]MTA SZTAKI, Kende u. 13-17, Budapest, Hungary-1111 e-mails: {kocsis,szcsaba}@sztaki.hu
[2]CBS

In this article we investigate the use of SPSA (Simultaneous Perturbation Stochastic Approximation), a stochastic hill-climbing search algorithm for tuning the parameters of game programs. Since optimisation algorithms typically exhibit difficulties when the objective function (performance measure) is observed in heavy noise, for one test domain we chose a non-deterministic game, namely Omaha Hi-Lo Poker, one of the most complex poker variants (at least with regard to state-space complexity). For another variant of poker, Texas Hold'em, several years of research lead to a series of strong programs: Poki (Billings *et al.*, 2002), PsOpti (Billings *et al.*, 2003) and Vexbot (Billings *et al.*, 2004). Our program, MCRAISE, borrows several ideas from the above mentioned programs. The name of the program originates from the use of Monte-Carlo simulations and the program's aggressive style. Although MCRAISE is still in the development phase and its performance has not been tested extensively yet, preliminary results suggests that the program is much stronger than amateur players, and is probably competitive with professional players as well. Therefore, we believe that it constitutes a suitable environment for testing the behaviour of optimisation algorithms in games.

!!!!!!!!!!TODO by Csaba: add a paragraph here about LOA and MIA!!!!!

The article is organised as follows. In Section 2 we describe the SPSA algorithm, which is followed by a description of RSPSA, together with an argument supporting the potential advantages of RSPSA over SPSA. In Section 3 three ways to further enhance the performance of (R)SPSA are proposed together with a discussion of the various tradeoffs involved, supported by analytic arguments. Next, in Section 4 the test domains and the respective programs are described. Experiments with (R)SPSA in these domains are given in Section 5. Finally, we draw our conclusions in Section 6.

2. THE RSPSA ALGORITHM

2.1 Basic Setup

Consider the task of finding a maximiser $\theta^* \in \mathbb{R}^d$ of some real valued function $f = f(\theta)$. In our case f may measure the performance of a player in some environment (e.g. again a fixed set of opponents), or it may represent an auxiliary performance index of interest that is used internally in the algorithm such that a higher value of it might ultimately yield a better play. In any case, θ represents some parameters of the game playing program.

We assume that the algorithm whose task is to tune the parameters θ can query the value of f at any point θ, but the value received by the algorithm will be corrupted by noise . The noise in the evaluation of f can originate from randomised decisions of the players or the randomness of the environment. In a card playing game for example the randomness of cards represents a substantial source of randomness in the outcomes of rounds. We shall assume that the value observed in the tth step of the algorithm when the simulation is run with parameter θ_t is given by $f(\theta_t; Y_t)$ where Y_t is some random variable such that the expected value of $f(\theta_t; Y_t)$ conditioned on θ_t and given all past information equals to $f(\theta_t)$:

$$f(\theta_t) = \mathbb{E}\left[f(\theta_t; Y_t) \,|\, \theta_t, \mathcal{F}_t \right],$$

where $\mathcal{F}_t$ is the sigma-field generated by $Y_0, Y_1, \ldots, Y_{t-1}$ and $\theta_0, \theta_1, \ldots, \theta_{t-1}$. Stochastic gradient ascent algorithms work by changing the parameter θ in a gradual manner so as to increase the value of f on average:

$$\theta_{t+1} = \theta_t + \alpha_t \hat{g}_t(\theta_t). \tag{1}$$

Here θ_t is the estimate of θ^* in the tth iteration (time step), $\alpha_t \geq 0$ is a learning rate parameter that governs the size of the changes to the parameters and $\hat{g}_t(\theta_t)$ is some approximation to the gradient of f such that the expected value of $\hat{g}_t(\theta_t)$ given past data is equal to the gradient $g(\theta) = \partial f(\theta)/\partial\theta$ of f and $(\hat{g}_t(\theta_t) - g(\theta))$ has finite second moments.

2.2 SPSA

When f is not available analytically then in order to use gradient ascent one must resort to some approximation of the gradient of it. One such approximation that has some particularly appealing properties was introduced in the <u>S</u>imultaneous <u>P</u>erturbation <u>S</u>tochastic <u>A</u>pproximation (SPSA) algorithm in (Spall, 1992):

$$\hat{g}_{ti}(\theta_t) = \frac{f(\theta_t + c_t\Delta_t; Y_t^+) - f(\theta_t - c_t\Delta_t; Y_t^-)}{2c_t\Delta_{ti}}. \tag{2}$$

Here $\hat{g}_{ti}(\theta_t)$ is the estimate of the ith component of the gradient, $\Delta_{ti}, Y_t^+, Y_t^-$ are random variables: Y_t^+, Y_t^- are meant to represent the randomness of the evaluation of f, whilst $\Delta_{t\cdot}$ is a perturbation vector to be chosen by the user. Note that the numerator of this expression does not depend on the index i and hence evaluating (2) requires only two (randomised) measurements of the function f. Still, SPSA provides a good approximation to the gradient: Under the conditions that *(i)* the random perturbations Δ_t are independent of the past of the process, *(ii)* for any fixed t, $\{\Delta_{ti}\}_i$ is an i.i.d. sequence[3], *(iii)* the distribution of Δ_{ti} is symmetric around zero, *(iv)* $|\Delta_{ti}|$ is bounded with probability one and *(v)* $\mathbb{E}\left[\Delta_{ti}^{-1}\right]$ is finite and assuming that f is sufficiently smooth, it can be shown that the bias of estimating that gradient $g(\theta_t)$ by $\hat{g}_t(\theta_t)$ is of the order $O(c_t^2)$. Further, the associated gradient ascent procedure can be shown to converge to a local optima of f with probability one (Spall, 1992).

A simple way to satisfy the conditions on Δ_t is to choose its components to be independent ± 1-valued Bernoulli distributed random variables with each outcome occurring with probability $1/2$. One particularly appealing property of SPSA is that SPSA might need d times less measurements than the classical FDSA procedure and still achieve the same asymptotic statistical accuracy (see e.g. (Spall, 1992)). FDSA works by evaluating f at $\theta_t \pm c_t e_i$ and forming the appropriate differences – hence it requires $2d$ evaluations. For a more thorough discussion of SPSA, its variants and relation to other methods consult (Spall, 1992; Spall, 2000).

2.3 RPROP

SPSA, like other stochastic approximation algorithms has quite a few tunable parameters. These are the gain sequences α_t, c_t and the distribution of the perturbation Δ. When function evaluation is expensive, as is often the case in games, small sample behaviour of the algorithm becomes important. In that case the propoer tuning of these parameters becomes critical.

In practice, the learning rate α_t and the gain sequence c_t are often kept at a fixed value. Further, in all previous works on SPSA known to us it was assumed that the perturbations Δ_{ti}, $i = 1, \ldots, d$, have the same distribution. When different dimensions have different scales (which we believe is a very common phenomenon in practice) then it does not make too much sense to use the same scales for all the dimensions.

The issue is intimately related to the issue of scaling the gradient. Second and higher-order methods work by utilising information about higher order derivatives of the objective function (see e.g. (Spall, 2000; Dippon, 2003)). In general, these methods achieve a higher asymptotic rate of convergence, but, as discussed e.g. in (Theiler and Alper, 2004), their practical value might be limited in the small sample size case.

The RPROP ("resiliant backpropagation") (Riedmiller and Braun, 1993) algorithm and its variants are amongst the best performing first-order batch neural network gradient training methods and as such represent a viable alternative to higher-order methods.[4]. In particular, in practice RPROP methods were found to be very fast and accurate, robust to the choices of their parameters, scale well with the number of weights (as opposed to higher order methods whose memory and run-time requirements scale at least quadratically, RPROP scales linearly with the number of parameters). Further, RPROP is easy to implement, it is not sensitive to numerical errors and since the algorithm is dependent only the sign of the partial derivatives of the objective function[5] and not on the magnitude of these partial derivatives, it is thought to be suitable for applications where the gradient is numerically estimates and/or is noisy.

A particularly successful variant of RPROP is the iRPROP$^-$ algorithm due to Igel and Hüsken (2000). The update equations of iRPROP$^-$ when it is used to maximise a function $f = f(\theta)$ are as follows:

$$\theta_{t+1,i} = \theta_{t,i} + \operatorname{sign}(g_{ti})\delta_{ti}, \quad t = 1, 2, \ldots; i = 1, 2, \ldots, d. \tag{3}$$

Here $\delta_{ti} \geq 0$ is the step size for the ith component and $g_{t\cdot}$ is a gradient-like quantity:

$$g_{ti} = \mathbb{I}(g_{t-1,i} f_i'(\theta_t) \geq 0) f_i'(\theta_t), \tag{4}$$

i.e., g_{ti} equals the ith partial derivative of f at θ except when a sign reversal is observed between the current and the previous partial derivative, in which case g_{ti} is set to zero. Hence, in this case the iteration leaves the parameter $\theta_{t,i}$ unchanged.

[3]"i.i.d." is the shorthand of "independent, identically distributed"

[4]For a recent empirical comparison of RPROP and its variants with alternative, gradient optimisation methods such as BFGS, CG and others see e.g. (Igel and Hüsken, 2003)

[5]RPROP, though it was worked out for the training of neural networks, is applicable in any optimisation problem where the gradient can be computed (or approximated as proposed here).

The individual step-sizes δ_{ti} are updated in an iterative manner based on the sign of the product $g_{t-1,i}\, g_{ti}$:

$$\begin{aligned} \eta_{ti} &= (\mathbb{I}(g_{t-1,i} g_{ti} > 0)\eta^{+} + \mathbb{I}(g_{t-1,i} g_{ti} < 0)\eta^{-} + \mathbb{I}(g_{t-1,i} g_{ti} = 0), & (5) \\ \delta_{ti} &= P_{[\delta^{-},\delta^{+}]}\left(\eta_{ti}\delta_{t-1,i}\right), & (6) \end{aligned}$$

where $0 < \eta^{-} < 1 < \eta^{+}$, $0 < \delta^{-} < \delta^{+}$, $P_{[a,b]}$ clamps its argument to the interval $[a, b]$, and $\mathbb{I}(\cdot)$ is a $\{0, 1\}$-valued function working on Boolean values and $\mathbb{I}(\mathcal{L}) = 1$ if and only if $\mathcal{L}$ is true, and $\mathbb{I}(\mathcal{L}) = 0$, otherwise.

2.4 RSPSA

Given the success of RPROP in improving the finite sample performance of gradient ascent methods, here we propose a combination of SPSA and RPROP (in particular, a combination with iRPROP^{-}). We call the resulting combined algorithm RSPSA ("resiliant SPSA"). The algorithm works by replacing $f_i'(\theta_t)$ in (4) with its noisy estimates $\hat{g}_{t,i}(\theta_t)$. Further, the scales of the perturbation vector $\Delta_{t,\cdot}$ are coupled to the scale of the step-sizes of δ_{ti}.

Before motivating this coupling let us make a few observations on the expected behaviour of RSPSA. Since iRPROP^{-} depends on the gradient only through the sign of it, it is expected that if the sign of $\hat{g}_{t,i}(\theta_t)$ coincides with that of $f_i'(\theta_t)$ then the performance of RSPSA will be close to that of iRPROP^{-}. This can be backed up by the following simple argument: Assuming that $|f_i'(\theta)| > \varepsilon$, applying Markov's inequality, we get that

$$\mathbb{P}(\mathrm{sign}(\hat{g}_{t,i}(\theta)) \neq \mathrm{sign}(f_i'(\theta))) \leq \mathbb{P}(|\hat{g}_{t,i}(\theta)) - f_i'(\theta)| \geq \varepsilon) \leq \frac{M_{t,i}}{\varepsilon^2},$$

where $M_{t,i} = \mathbb{E}\left[(\hat{g}_{t,i}(\theta) - f_i'(\theta))^2 | \mathcal{F}_t\right]$ denotes the mean square error of the approximation of $f_i'(\theta)$ by $\hat{g}_{t,i}(\theta)$, conditioned on past observations. In fact, this error can be shown to be composed of a bias term dependent only on f, θ and c, and a variance term dependent on the random quantities in $\hat{g}_{t,i}(\theta)$. Hence, if this variance is small then the performance of RSPSA can be expected to be close to that of iRPROP^{-}.

Now, let us turn to the idea of coupling the scales of the perturbation vectors to the step-sizes of iRPROP^{-}. This idea can be motivated as follows: On "flat areas" of the objective function, where the sign of the partial derivatives of the objective function is constant and where the absolute value of these partial derivatives is small, a perturbation's magnitude along the corresponding axis should be large or the observation noise will dominate the computed finite differences. On the other hand, in "bumpy areas" where the sign of a partial derivative changes at smaller scales, smaller perturbations that fit the "scale" of desired changes can be expected to perform better. Since the step-size parameters of RPROP are larger in flat areas and are smaller in bumpy areas, it is natural to couple the perturbation parameters of SPSA to the step-size parameters of RPROP. A simple way to accomplish this is to let

$$\Delta_{ti} = \rho\, \delta_{ti},$$

where ρ is some positive constant, to be chosen by the user.

3. INCREASING EFFICIENCY

In this section we describe several methods that can be used to increase the efficiency of RSPSA. The first method, known as the "Method of Common Random Numbers", was proposed earlier to speed up SPSA (L'Ecuyer and Yin, 1998; Kleinman, Spall, and Neiman, 1999). The next method, averaging, was proposed as early as in (Spall, 1992). However, when averaging is used together with the method of common random numbers a new tradeoff arises. By means of a formal analysis this tradeoff is identified and resolved here for the first time. To the best of our knowledge, the last method, the use of antithetic variables has not been suggested earlier in this context to speed up SPSA. All these methods aim at reducing the variance of the estimates of the gradient. According to our experiments the "tricks" suggested here are essential to improve finite sample behaviour. In these sections we will drop the time index t in order to simplify the notation.

3.1 Common Random Numbers

In SPSA (and hence also in RSPSA) the estimate of the gradient relies on differences of the form $f(\theta+c\Delta; Y^{+}) - f(\theta - c\Delta; Y^{-})$. Denoting by F_i^{+} (resp. F_i^{-}) the first (resp. second) term, elementary probability calculus gives $\mathrm{Var}\left(F_i^+ - F_i^-\right) = \mathrm{Var}\left(F_i^+\right) + \mathrm{Var}\left(F_i^-\right) - 2\mathrm{Cov}\left(F_i^+, F_i^-\right)$. Hence the variance of the estimate of the gradient can be decreased by introducing some correlation between F_i^+ and F_i^-, provided that this does not increase the variance of F_i^+ and F_i^-. This is because by our assumptions Y^+ and Y^- are independent and hence $\mathrm{Cov}\left(F_i^+, F_i^-\right) = 0$. Now, if $F_i^\pm$ is redefined to depend on the *same* random value Y (i.e., $F_i^\pm = f(\theta \pm c\Delta; Y)$) then the variance of $F_i^+ - F_i^-$ will decrease when $\mathrm{Cov}\left(f(\theta + c\Delta; Y), f(\theta - c\Delta; Y)\right) > 0$. The larger this covariance is the larger the decrease of the variance of the estimate of the gradient will be.

When f is the performance of a game program obtained in some simulation that uses pseudorandom numbers then using the same random series Y can be accomplished by using the same seed value when computing the values of f at $\theta + c\Delta$ and $\theta - c\Delta$.

3.2 Using Averaging to Improve Efficiency

Another way to reduce the variance of the estimate of the gradient is to average many independent estimates of it. However, this variance reduction is not for free as evaluating $f(\theta; Y)$ can be extremely CPU-intensive, as it was mentioned earlier.

To study the involved tradeoffs let us define

$$\hat{g}_{q,i}(\theta) = \frac{1}{q}\sum_{j=1}^{q} \frac{f(\theta + c\Delta; Y_j) - f(\theta - c\Delta; Y_j)}{2c\,\Delta_i}, \tag{7}$$

where according to the suggestion of the previous section we use the same set of random number to evaluate f both at $\theta + c\Delta$ and $\theta - c\Delta$. Further, let $\hat{g}_{r,q,i}(\theta)$ be the average of r independent samples $\{\hat{g}^{(j)}_{q,i}(\theta)\}_{j=1,\ldots,r}$ of $\hat{g}_{q,i}(\theta)$ (in $\hat{g}_{q,i}(\theta)$ the underlying perturbations are independent). By the law of large numbers $\hat{g}_{r,q,i}(\theta)$ converges to $f_i'(\theta) + O(c^2)$ as $q, r \to +\infty$ (i.e. its ultimate bias is of the order $O(c^2)$). It follows then that increasing $p = rq$ above the value where the bias term becomes dominating does not improve the finite sample performance. This is because although increasing p increases the quality of approximation of the gradient, at the same time it also decreases the frequency of updates to the parameters.[6]

In order to gain further insight into how to choose p, q and r, let us consider the mean squared error of approximating the ith component of the gradient by $\hat{g}_{r,q,i}$: $M_{r,q,i} = \mathbb{E}\left[(\hat{g}_{r,q,i}(\theta) - f_i'(\theta))^2\right]$. By some lengthy calculations, the following expression can be derived for $M_{r,q,1}$:[7]

$$\begin{aligned} M_{r,q,i} = \frac{1}{r}\mathbb{E}\left[\Delta_1^2\right]\mathbb{E}\left[1/\Delta_1^2\right]\sum_{j=2}^{d}\left\{\left(1 - \frac{1}{q}\right)\mathbb{E}\left[f_j'(\theta, Y_1)^2\right] + \frac{1}{q}\mathbb{E}\left[f_j'(\theta, Y_1)\right]^2\right\} \\ + \frac{1}{rq}\,\mathbb{E}\left[(f_1'(\theta, Y_1) - f_1'(\theta))^2\right] + O(c^2). \end{aligned} \tag{8}$$

Here $f_i'(\theta; Y)$ is the partial derivative of f w.r.t. θ_i: $f_i'(\theta; Y) = \frac{\partial f(\theta; Y)}{\partial \theta_i}$. It follows from Eq. (8) that for a fixed budget of $p = qr$ function evaluations the smallest mean squared error is achieved by taking $q = 1$ and $r = p$ (disregarding the $O(c^2)$ bias term which we assume to be "small" as compared to the other terms).

Now the issue of choosing p can be answered as follows: Under mild conditions on f and Y (ensuring that the expectation and the partial derivative operators can be exchanged), $\sum_{j=2}^{d}\mathbb{E}\left[f_j'(\theta, Y_1)\right]^2 = \sum_{j=2}^{d} f_j'(\theta)^2$. Hence, in this case with the choices $q = 1$, $r = p$, $M_{p,1,1}$ becomes equal to

$$\frac{1}{p}\left\{\mathbb{E}\left[\Delta_1^2\right]\mathbb{E}\left[1/\Delta_1^2\right]\sum_{j=2}^{d} f_j'(\theta)^2 + \mathbb{E}\left[(f_1'(\theta, Y_1) - f_1'(\theta))^2\right]\right\} + O(c^2), \tag{9}$$

which is composed of two terms in addition to the bias term $O(c^2)$: the term $\sum_{j=2}^{d} f_j'(\theta)^2$ represents the contribution of the "cross-talk" of the derivatives of f to the estimation of the gradient, whilst the second term,

[6] In (Spall, 1992) a heuristic calculation is given that shows that using decreasing gains $\alpha_t = a/t^\alpha$ and $c_t = c/t^\gamma$ with $\beta = \alpha - 2\gamma > 0$, $0 < \alpha \le 1$, $0 < \gamma$, the optimal choice of p is governed by an equation of the form $p^{\beta-1}A + p^\beta B$, where $A, B > 0$ are some (unknown) system parameters. This equation has a unique minimum at $p = (1 - \beta)A/(\beta B)$, however, since A, B are unknown parameters this result has limited practical value besides giving a hint about the nature of the tradeoff in the selection of p.

[7] Without the loss of generality we consider the case $i = 1$.

$\mathbb{E}\left[(f_1'(\theta,Y_1)-f_1'(\theta))^2\right]$ gives the mean squared error of approximating $f_1'(\theta)$ with $f_1'(\theta,Y_1)$ (which is equal to the variance of $f_1'(\theta,Y_1)$ in this case). The first term can be large when θ is far from a stationary point of f, whilst the size of the second term depends on the amount of noise in the evaluations of f. When the magnitude of these two terms is larger than that of the bias term $O(c^2)$ then increasing p will increase the efficiency of the procedure, at least initially.

3.3 Antithetic Variables

In Section 3.1 we have advocated the introduction of correlation between the two terms of a difference to reduce its variance. The same idea can be used to reduce the variance of averages: Let $U_1, U_2, \ldots, U_n$ be i.i.d. random variables with common expected value I. Then the variance of $I_n = 1/n\sum_{i=1}^n U_i$ is $1/n\mathrm{Var}\,(U_1)$. Now, assume that n is even, say $n = 2k$ and consider estimate I by

$$I_n^A = \frac{1}{k}\sum_{i=1}^{k}\frac{U_i^+ + U_i^-}{2},$$

where now it is assumed that $\{U_1^+,\ldots,U_k^+\}$ are i.i.d., just like $\{U_1^-,\ldots,U_k^-\}$, $\mathbb{E}\left[U_i^+\right] = \mathbb{E}\left[U_i^-\right] = I$. Then $\mathbb{E}\left[I_n^A\right] = I$ and $\mathrm{Var}\left(I_n^A\right) = (1/k)\,\mathrm{Var}\left((U_1^+ + U_1^-)/2\right)$. Using the elementary identity $\mathrm{Var}\left((U_1^+ + U_1^-)/2\right) = 1/4\left(\mathrm{Var}\left(U_1^+\right) + \mathrm{Var}\left(U_1^-\right) + 2\mathrm{Cov}\left(U_1^+, U_1^-\right)\right)$ we get that if $\mathrm{Var}\left(U_1^+\right) + \mathrm{Var}\left(U_1^-\right) \le 2\mathrm{Var}\,(U_i)$ and $\mathrm{Cov}\left(U_1^+, U_1^-\right) \le 0$ then $\mathrm{Var}\left(I_n^A\right) \le \mathrm{Var}\,(I_n)$. One way to achieve this is to let U_i^+, U_i^- be *antithetic* random variables: U_i^+ and U_i^- are called antithetic if their distributions are the same but $\mathrm{Cov}\left(U_i^+, U_i^-\right) < 0$.

How can we introduce antithetic variables in parameter optimisation of game-programs? Consider the problem of optimising the performance of a player in a non-deterministic game. Let us collect all random choices external to the players into a random variable Y and let $f(Y;W)$ be the performance of the player in the game. Here W represents the random choices made by the players (f is a deterministic function of its arguments). For example in back-gammon, Y would collect the outcomes of dice-rolls, whilst in a card-game (e.g. in poker) Y can be chosen to be the cards in the deck at the beginning of the play after shuffling. The idea is to manipulate the random variables Y in the simulations by introducing a "mirrored" version, Y', of it such that $f(Y;W)$ and $f(Y';W')$ become antithetic. Here W' represents the player's choices in response to Y' (it is assumed that different random numbers are used when computing W and W').

Assuming that the player's play reasonably well, the influence of the random choices Y on the outcome of the game will be strong. By this we mean that the value of $f(Y;W)$ is largely determined by the value of Y. For example it may happen that in backgammon the dices roll in favour of one of the players. Another example is in poker when one player gets a strong hand, whilst the other gets a weak one. Staying with this example and assuming two-players, a natural idea to mitigate the influence of Y is to reverse the hands of the players: the hand of the first player becomes that of the second and vice versa. Denoting the cards in this new scenario by Y', it is expected that $\mathrm{Cov}\left(f(Y;W), f(Y';W')\right) < 0$. Since the distribution of Y and Y' are identical (the mapping between Y and Y' is a bijection), the same holds for the distributions of $f(Y;W)$ and $f(Y';W')$. When the random choices Y influence the outcome of the game strongly then we often find that $f(Y;W) \approx -f(Y';W')$. When this is the case then $\mathrm{Cov}\left(f(Y;W), f(Y';W')\right) \approx -\mathrm{Var}\left(f(Y;W)\right)$ and thus $f(Y;W)$ and $f(Y';W')$ are "perfectly" antithetic and thus $\mathrm{Var}\left(I_n^A\right) \approx 0$. Of course, $f(Y;W) = -f(Y';W')$ will never hold and hence the variance of I_n^A will not be eliminated entirely – but the above argument shows that it can be reduced to a large extent.

This method can be used in the estimation of the gradient (and also when the performance of the players is evaluated). Combined with the previous methods we get

$$\hat{g}_{p',i}(\theta) = \frac{1}{4cp'}\sum_{j=1}^{p'}\frac{1}{\Delta_i^{(j)}}\Big((f(\theta+c\Delta^{(j)};Y)+f(\theta+c\Delta^{(j)};Y')) - (f(\theta-c\Delta^{(j)};Y)+f(\theta-c\Delta^{(j)};Y'))\Big), \quad (10)$$

where $\Delta^{(1)},\ldots,\Delta^{(p')}$ are i.i.d. random variables. In our experiments (see Section **??**) we have observed that to achieve the same accuracy in evaluating the performance of a player (e.g. with a certain parameter) we could use up to 4 times less samples when antithetic variables were used. We expect similar speed-ups in other games where external randomness influences the outcome of the game strongly.

4. TEST DOMAINS

In this section we describe the two test domains, Omaha Hi-Lo Poker and Lines of Actions, together with the game playing programs used in the experiments.

4.1 Omaha Hi-Lo Poker

4.1.1 The rules

Omaha Hi-Lo Poker is a card game played by two to ten players. At the start each player is dealt four private cards, and at later stages five community cards are dealt face up (three after the first, and one after the second and third betting round). In a betting round, the player on turn has three options: *fold*, *check/call*, or *bet/raise*. After the last betting round, the pot is split amongst the players depending on the strength of their cards. The pot is halved into a high side and a low side. For each side, players must form a hand consisting of two private cards and three community cards. The high side is won according to the usual poker hand ranking rules. For the low side, a hand with five cards with different numerical values from Ace to eight has to be constructed. The winning low hand is the one with the lowest high card.

4.1.2 Estimating the Expected Proportional Payoff

Probably the most important element in the play of a poker player is to estimate his winning chances, or more precisely to predict how much share one will get from the pot. In order to fulfill this task, the player has to evaluate correctly the strength and potential of his own cards, as well as to guess the cards of the opponents. The former can be done by means of some static analysis, whilst for the latter, information can be gained from taking into account the previous betting actions taken by a particular opponent in the current game, and possibly in previous games.

Assume that in previous encounters a player was able to estimate the betting strategy of one of its opponent, i.e., the probability of the opponent's betting actions given a game state, including the public information (community cards dealt up to that moment, bet level, the betting history, and so on), and, if an opponent revealed his cards at showdown, the hole cards.

Such a model of an opponent can be used to derive information about the likely cards of the opponent in the current game based on his current betting actions. Obviously, guessing the opponent's hole cards should result not in a particular hand, but a probability distribution over the possible hands. Ideally, we would like to know the probability distribution over all possible hands (as it was done in Poki (Billings *et al.*, 2002)), but this is clearly not feasible since the number of possible opponent hands is overly large (recall that in Omaha Hi-Lo a player has four hole cards, compared to the two cards in Texas Hold'em). Alternatively, we may use a weighted random sample to represent this distribution and this is the strategy employed in McRaise. In particular, McRaise uses the following calculations to derive an estimate of the expected proportional payoff:

N random card configurations, cc, are generated each of which consists of the opponent hands $h_{opp}(cc)$ and the community cards that still have to be dealt. Then, given the betting history $history$ of the current game, the expected payoff as expressed as a proportion of the actual pot size (we call this quantity the expected proportional payoff or EPP) is approximated by

$$p_{win} = \frac{1}{N}\sum_{cc} win(cc) \prod_{opp} \frac{p(h_{opp}(cc)|history)}{p(h_{opp}(c))}.$$

where $win(cc)$ is the percentage of the pot won for a given card configuration cc and $p(h_{opp}(cc)|history)$ is the probability of cc given the observed history of betting actions. Here, it is assumed that the opponents choose their actions independently of each other. Further, for the sake of simplicity it was assumed that the opponents betting actions depend only on their own cards. Now, using Bayes' rule, we get

$$\frac{p(h_{opp}(cc)|history)}{p(h_{opp}(cc))} \propto p(history|h_{opp}(cc)),$$

where the omission of $p(history)$ is compensated by changing the calculation of p_{win} by normalising the individual weights of $w(cc) = p(history|h_{opp}(cc))$ so as they sum to 1, i.e., p_{win} is estimated by means of weighted importance sampling.

The only remaining unknown element is the probability of a betting sequence given the hole cards, $p(history|h_{opp}(cc))$. This can be computed using the probability of a certain betting action given the game state, $p(a|h_{opp}(cc))$, which is the core of the opponent model. If we would assume independence among the actions of an opponent, then $p(history|h_{opp}(cc))$ would come down to a product over the individual actions. This is obviously not the case. A simple way to include the correlation among the betting actions inside a round is given by the following equation:

$$p(history|h_{opp}(cc)) = \prod_{rnd} \frac{\sum_{a \in history_{opp,rnd}} \frac{p(a|h_{opp}(cc))}{p(a)}}{nractions(opp, rnd)},$$

where $nractions(opp, rnd)$ is the number of betting actions of an opponent opp in a particular round rnd.

Estimating $p(a|h_{opp}(cc))$ can be done in various ways. Currently, we use a generic opponent model, fitted to a game database that includes human games played on IRC, and games generated by self-play.

4.1.3 Action Selection

(!!!!!!TODO by Csaba: this section is too long and should be simplified!)

In the following we describe the mechanism used in the experiments for selecting the betting action based on the estimate of the winning chance. This is a rather simple mechanism that employs a 1-ply search. McRaise is capable to look further ahead, but the gain in performance is much smaller than in deterministic games such as chess. For this reason and for simplicity of the presentation we restrict ourselves here to search only 1-ply deep.

If we are looking just 1-ply ahead (i.e. only our action), then we do not need to deal with the uncertainty resulting from the opponents' future actions, nor with that resulting from the community cards that have yet to be dealt. Then action selection simplifies to a straightforward estimation of the expected value of each actions and selecting the best one. Given a the current situation s, the estimate of EPP, $p_{win} = p_{win}(s)$, the expected value of an action a is estimated by

$$Q(s, a) = p_{win}\Pi(a, s) - B(a, s),$$

where $\Pi(a, s)$ is the estimated pot size provided that action a is executed and $B(a, s)$ is the contribution to the pot. For estimating $\Pi(a, s)$ and $B(a, s)$, we assume that every player (including McRaise) checks from this point on. If the action under investigation is fold, then $\Pi(a, s) = 0$, $B(a, s) = B_0(s)$ ($B_0(s)$ represents the current contribution to the pot) and thus $Q(s, a) = -B_0(s)$. Using the simple models $\Pi(a, a) = \Pi_0(s) + PD(a)$ and $B(a, s) = B_0(s) + D(a)$, where P is the number of active players and $D(a)$ is the contribution of McRaise to the pot when action a is selected, it follows that McRaise will select the 'raise' action whenever $p_{win} > 1/P$.

(!!!!!!!!Csaba: Have read things up to this point!!!!!!!!!!)

4.2 Lines of Action

4.2.1 The rules

LOA is a two-person zero-sum game with perfect information; it is a chess-like game with a connection-based goal, played on an 8×8 board. LOA was invented by Claude Soucie around 1960. Sid Sackson (1969) described it in his first edition of *A Gamut of Games*.

LOA is played on an 8×8 board by two sides, Black and White. Each side has twelve pieces at its disposal. The players alternately move a piece, starting with Black. A move takes place in a straight line, exactly as many squares as there are pieces of either colour anywhere along the line of movement. A player may jump over its own pieces. A player may not jump over the opponent's pieces, but can capture them by landing on them. The goal of a player is to be the first to create a configuration on the board in which all own pieces are connected in one unit. The connections within the unit may be either orthogonal or diagonal. In the case of simultaneous connection, the game is drawn. If a player cannot move, this player has to pass. If a position with the same player to move occurs for the third time, the game is drawn.

4.2.2 MIA

The following series of experiments have been performed in the framework of the tournament program MIA 4++ (Maastricht In Action). This program has won the LOA tournament at the eighth (2003) and ninth (2004) Computer Olympiad and is therefore considered as the best LOA playing entity of the world (Winands, 2004). Besides a well-tuned evaluation function (Winands, van den Herik, and Uiterwijk, 2003), the program uses an $\alpha\beta$ depth-first iterative-deepening search to play the game. Several techniques are implemented to make the $\alpha\beta$ search efficient. The program performs PVS (Marsland and Campbell, 1982), *two-deep* transposition tables (Breuker, Uiterwijk, and van den Herik, 1996), Enhanced Transposition Cutoffs (ETC) (Schaeffer and Plaat, 1996), killer moves (Akl and Newborn, 1977) and the relative history heuristic (Winands *et al.*, 2004). Forward pruning techniques like (adaptive) null move (Heinz, 1999; Donninger, 1993) and multi-cut citebjornsson99,winands03 are successfully used in the program. In the leaf nodes of the tree a quiescence search is performed. Finally, the program uses realisation-probability search (RPS) (Tsuruoka, Yokoyama, and Chikayama, 2002), which we will investigate further in this section.

5. EXPERIMENTS

In poker, we tested the RSPSA algorithm by optimising two components of McRaise, the opponent model and the action selection. For both components, we compare the performance resulting by using SPSA with the performance given by an alternative (state-of-the-art) optimisation algorithm. The experiments for the opponent-model optimisation are described in section 5.1 and for the move-selection optimisation in Section 5.2. In Lines of Action, the RSPSA algorithm was employed for tuning the realisation-probability weights in MIA. It was noted in (Kocsis, 2003a) that these weights belong to a class of parameters (termed class-S search decisions) that can be evaluated using search trees. In Section 5.3 we show how this property is exploited for improving the efficiency of the learning.

5.1 Tuning the Opponent Model

The opponent model of McRaise is embodied in the estimation of $p(a|h_{opp}(cc))$ (see Section 4.1.2). The model uses in total six parameters.

For problems where the number of parameters is small FDSA can be a natural competitor to SPSA. We combined both SPSA and FDSA with RPROP (i.e. we have actually tested RSPSA here – early experiments with SPSA did not produce reasonable results due to perhaps the anisotropy of the underlying optimisation problems). For the sake of simplicity, we shall still write SPSA throughout this section.

In the process of estimating the derivatives we employed the “Common Random Numbers” method: in all cases the same decks were used for the two opposite perturbations. Since many of the decks produced zero SPSA differences, thus producing zero contribution to the estimation of the sign of the derivatives, those decks that resulted in no-zero differences were saved for reuse. In subsequent steps, half of the decks used for a new perturbation were taken from those previously stored, whilst the other half was generated randomly. The reuse was based on recency ensuring this way that certain decks are not persisting over a long period. The idea of storing and reusing decks that ‘make difference’ can be motivated using ideas from importance sampling, a method known to decrease the variance of Monte-Carlo estimates. Note, however, that the process just described can not be cast as an instance of importance sampling as it introduces some bias. According to our measurements, the variance reduction effect due to this biased procedure is significantly larger than if we used an unbiased variant.

The parameters of the algorithms are given in Table 1. Note that the performance corresponding to a single perturbation was evaluated by playing games in parallel on cluster of sixteen computers. The number of evaluations (games) for a given perturbation was kept in all cases above 100 to reduce the communication overhead. The parameters of the opponent model were initialised to the original parameter settings of McRaise.

The evolution of performance for the two algorithms is plotted in Figure 1 as a function of the number of iterations. The best performance obtained for SPSA was +0.170sb, whilst that of for FDSA it was +0.095sb. Since the performance resulting from the use of SPSA is almost twice as good as that of resulting from the use of FDSA, we conclude that despite the small number of parameters, SPSA is the better choice in this case.

	$\eta+$	$\eta-$	δ_0	δ^-	δ^+	$\Delta(\lambda)$	$batchsize$
RSPSA (OM)	1.1	0.85	0.01	1e-3	1.0	δ	$40 \times 2 \times 250$
FDSA (OM)	1.1	0.85	0.01	1e-3	1.0	δ	$6 \times 2 \times 1500$
RSPSA (EF)	1.2	0.8	0.05	1e-3	1.0	$\delta/0.7$	$100 \times 2 \times 100$
RSPSA (POL)	1.1	0.9	0.01	1e-3	1.0	$\delta/0.3$	$100 \times 2 \times 100$
TD (EF)	1.2	0.5	0.1	1e-6	1.0	0.9	10000

Table 1: Learning parameters of RSPSA and FDSA for opponent model (OM), SPSA and TD for evaluation function (EF) and SPSA for policy (POL) learning. $\eta+$, $\eta-$, δ_0 (the initial value of δ_{ti}), δ^- and δ^+ are the RPROP parameters; Δ is the SPSA (or FDSA) perturbation size, λ is the parameter of TD; $batchsize$ is the number of performance evaluations (games) in an iteration which, for SPSA and FDSA, is equal to the product of the number of perturbations (q), the number of directions (2) and the number of evaluations per perturbation (r).

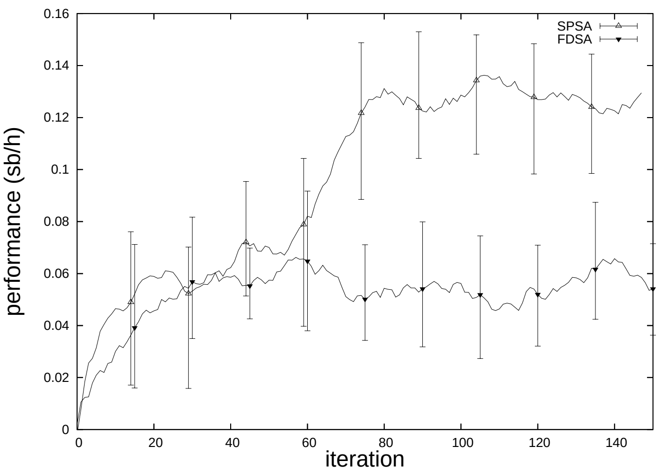

Figure 1: Learning curves for SPSA and FDSA as a function of the number of iteration. The graphs are obtained by smoothing the observed performance in windows of size 15. The error bars were obtained by dropping the smallest and largest values within the same windows centered around their respective oordinates.

5.2 Learning Policies and Evaluation Functions

The action selection mechanism of McRaise is based on a simple estimation of the expected payoffs of actions and selecting the best action (see Section 4.1.3). This can be cast as a 1-ply search w.r.t. the evaluation function V if s' is the situation after action a is executed from situation s and if we define $V(s') = Q(s, a)$. In the experiments described in this section we either represent the evaluation function with a neural network, or we replace the search (and evaluation function) with a neural network. In the first case the output of the neural network for a given situation s represents $V(s)$ that is used in the 1-ply search, whilst in the second case the neural network has three outputs that are used (after normalisation) as the probabilities of selecting the respective next actions. The input to the neural networks include EPP, the strength of the player's hand (as the a-priori chance of winning), the position of the player, the pot size, the current bet level and some statistics about the recent betting actions of the opponent.

Learning evaluation functions is by far the most studied learning task in games. One of the most successful algorithm for this task is TD-learning (Sutton, 1988) and the best known example of successfully training an evaluation function is TDGammon that learnt to play backgammon (Tesauro, 1992). By some, the success of TD-learning in backgammon can mostly be attributed to the highly stochastic nature of this game. Naturally, poker is similarly stochastic hence TD-algorithms might enjoy the same benefit. Temporal-difference learning had some success in deterministic games as well, e.g. it contributed significantly to the world champion LOA program, MIA (Winands *et al.*, 2002). In our experiment we use a similar design as the one used in MIA, combining TD(λ) with RRPOP (one source for the motivation of RSPSA comes from the success of combining TD-learning and RPROP).

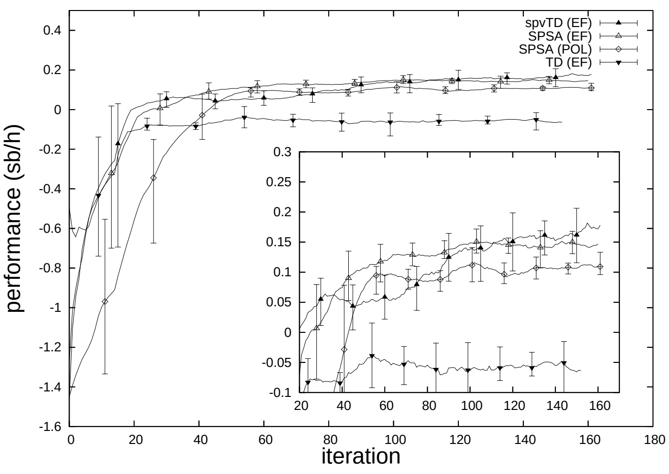

Figure 2: Learning curves for SPSA and TD as a function of the number of iteration. The graphs are obtained by smoothing the observed performance in windows of size 15. The error bars were obtained by dropping the smallest and largest values within the same windows centered around their respective coordinates. For an explanation of the symbols see the text.

The parameters of the algorithms are given in Table 1. For SPSA the same enhancements were used as in Section 5.1. We tested experimentally four algorithms: (1) SPSA for tuning the parameters of an evaluation function (SPSA(EF)), (2) SPSA for tuning a policy (SPSA(POL)), (3) TD for tuning an evaluation function (TD(EF)), and (4) TD for evaluation function tuning with a supervised start-up (spvTD(EF)). For the latter a simple supervised algorithm tuned the neural network used as the evaluation function to match the evaluation function that was described in Section 4.1.3. The learning curves of the four algorithms are given in Figure 2. The best performance obtained for SPSA(EF) was +0.194sb/h, for SPSA(POL) it was +0.152sb/h, for TD(EF) it was +0.015sb/h and for spvTD(EF) it was +0.220sb/h. It is fair to say that TD performed better than SPSA, which is a result one would expect given that TD uses more information about the gradient. However, we observe that for TD it was essential to start from a good policy (the one tuned by supervised learning) and this option might not be available in all cases. We note that although the two SPSA algorithms did not reach the performance obtained by the combination of supervised and TD-learning, they did reach a considerable performance gain even though they were started from scratch.

5.3 Tuning the Realisation-Probability weights

Generally the parameters of a game program is evaluated by playing a number of games against a (fixed) set of opponent. In (Kocsis, 2003a), it was noted that for parameters such as search extensions alternative performance measures exists as well. One such alternative is to measure the 'quality' of the move selected by the search algorithm (constrained by time, search depth or number of nodes). The quality of a move was defined in (Kocsis, van den Herik, and Uiterwijk, 2003b) as the minimax score returned by a sufficiently deep search for the position followed by the move.[8] In the following, we describe two experiments. In the first, the performance is evaluated by game result. The result is averaged over 500 (?) games played against five different opponents starting from 100 fixed positions with both colours. Each opponent is using a different evaluation function. Each player is searching a maximum of 250,000 nodes per move. In RSPSA the gradient is estimated with 500 perturbations, using one game per perturbation. The common random number technique is implemented in this case by using the same starting position, same opponent and same colour for both the positive and the negative sides. In the second experiment the performance is evaluated by move score. The move score is averaged over a fixed set of 10,000 positions. For selecting the move the search is limited to 250,000 nodes. For evaluating the move a deeper search is used with a maximum of 10,000,000 nodes. Since the score of a move does not depend on the Realisation-Probability weights, they are cached and reused when the same move is selected again (for the same position). This way, the deeper search is performed far less frequently then the shallower search. The RSPSA gradient is estimated with 5000 perturbations. Each side of a perturbation is evaluated using one position selected

[8]More precisely, it is the negative negamax score.

	$\eta+$	$\eta-$	δ_0	δ^-	δ^+	$\Delta(\lambda)$	$batchsize$
RSPSA (GR)	1.2	0.8	0.005	1e-3	1.0	$\delta/0.7$	$500 \times 2 \times 1$
RSPSA (MS)	1.2	0.8	0.005	1e-3	1.0	$\delta/0.7$	$5000 \times 2 \times 1$

Table 2: Learning parameters of RSPSA for Realisation-Probability weights using game result (GR) and move score (MS) for evaluation. See Table 1 for an explanation of the symbols.

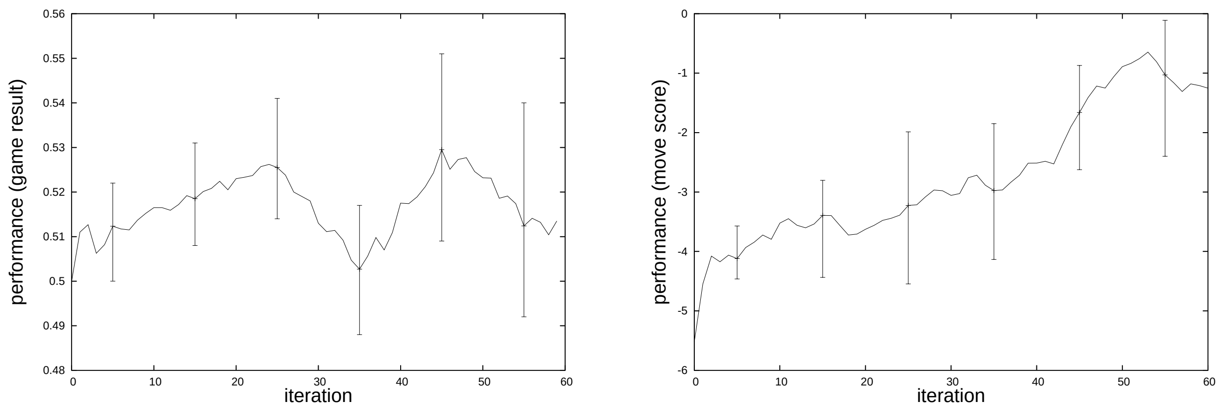

Figure 3: Learning curves for RSPSA on game result (left) and moves score (right) as a function of the number of iteration. The graphs are obtained by smoothing the observed performance in windows of size 10. The error bars were obtained by dropping the smallest and largest values within the same windows centered around their respective coordinates.

randomly from the set of 10,000 (the same position for both sides). Considering that the average game length in Lines of Action is approximately 40, in the second experiment the gradient is estimated approximately four times faster then in the first. Moreover, according to our observation, the estimates with move scores are less noisy as well. The parameters of the RSPSA algorithm for the two experiments are given in Table 2.

The learning curves for the two experiments are plotted in Figure 3. Since the two experiments are using different performance measures the performance for the two curves cannot be compared directly. Intuitively, the performance gain for the experiment using move scores (right) seems to be more significant than the one using game result. A more direct comparison can be performed by comparing the performance, measured as game result, of the best weight vector of each curve. The best performance obtained in the first experiment was 0.55, and the average game result corresponding to the best vector of the second experiment was 0.59. Therefore, we may conclude that using the move scores for estimating the performance improves the efficiency of the RSPSA algorithm.

6. CONCLUSIONS

This article investigated the value of a general purpose optimisation algorithm, SPSA, for the automatic tuning of game parameters. Several theoretical and practical issues were analysed, which in turn led to the design of a new variant of SPSA that we called RSPSA. RSPSA combines the strengths of RPROP and SPSA: SPSA is a gradient-free stochastic hill-climbing method that requires only function evaluations. RPROP is a first order method that is known to improve the rate of convergence of gradient ascent. The proposed combination couples the perturbation parameter of SPSA and the step-size parameters of RPROP. It was argued that this coupling is natural. By means of some preliminary experiments, it was shown that the combined method can indeed improve the convergence rate. This finding was also confirmed by experiments with tuning various parameters of our poker playing program, MCRAISE.

Several other methods were considered to improve the performance of SPSA (and thus that of RSPSA). The effect of performing a larger number of perturbations was analysed. An expression for the mean-square error of the estimate of the gradient was derived as the function of the number of (noisy) evaluations of the objective function per perturbation (q) and the number of perturbations (r). It was found that to optimise the mean-square error with a fixed budget $p = qr$, the number of perturbations should be kept at maximum.

We suggested that besides using the method of "common random numbers", the method of antithetic variables should be used for the further reduction of the variance of the estimates of the gradient. These methods together are estimated to yield a speed-up of factor larger than ten (since a smaller number of function evaluations is enough to achieve the same level of accuracy in estimating the gradient). The overall effect of these enhancements facilitated the application of SPSA for tuning parameters in several variants of our poker playing program McRaise.

The experiments where the performance of RSPSA was tested were performed in the game of Omaha Hi-Lo Poker. The optimisation of two components of the program were attempted: that of the opponent model and the action selection policy. The latter optimisation task was attempted both directly when the policy was represented explicitly and indirectly via the tuning of the parameters of an evaluation function. In addition to testing RSPSA, for both components an alternative optimiser was tested (resp. FDSA, and TD(λ)). On the task of tuning the parameters of the opponent model, RSPSA led to a significantly better performance as compared with the performance obtained when using FDSA. In the case of policy optimisation, RSPSA was competitive with TD-learning, although the combination of supervised learning followed by TD-learning outperformed RSPSA. Nevertheless, the performance of RSPSA was encouraging on this second task as well.

There are several lines of future research that look promising. Extensive numerical studies would be needed to gain more insight into the behaviour of SPSA (and RSPSA) as a function of its parameters. In particular, we believe that the coupling of the RPROP step-size and the SPSA perturbation size can be improved by using more information such as the learning "stage" and the variance of the gradient. Regarding the game environment, several components of McRaise should be improved. The opponent model has to be improved, especially by adapting it to the opponents during play. As we mentioned earlier deep searches are possible in McRaise, but the resulting gain is rather small. Foreseeing the future betting of the opponents plays an important part in human play, and therefore higher gains should be attainable by an improved search algorithm. Currently an important weakness of the program is its predictability. A potential solution to this problem is the game-theoretic approach that proved successful in Texas Hold'em (Billings *et al.*, 2003).

7. REFERENCES

Akl, S. and Newborn, M. (1977). The Principal Continuation and the Killer Heuristic. *1977 ACM Annual Conference Proceedings*, pp. 466–473, ACM Press, New York, NY, USA.

Baxter, J., Tridgell, A., and Weaver, L. (2000). Learning to Play Chess using Temporal Differences. *Machine Learning*, Vol. 40, No. 3, pp. 243–263.

Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., and Szafron, D. (2003). Approximating Game-Theoretic Optimal Strategies for Full-scale Poker. *Proceedings of Eighteenth International Joint Conference on Artificial Intelligence*, pp. 661–668.

Billings, D., Davidson, A., Schaeffer, J., and Szafron, D. (2002). The Challenge of Poker. *Artificial Intelligence*, Vol. 134, pp. 201–240.

Billings, D., Davidson, A., Shauenberg, T., Burch, N., Bowling, M., Holte, R., Schaeffer, J., and Szafron, D. (2004). Game Tree Search with Adaptation in Stochastic Imperfect Information Games. *Proceedings of Computers and Games (CG'04).*

Björnsson, Y. and Marsland, T. (2003). Learning Extension Parameters in Game-Tree Search. *Journal of Information Sciences*, Vol. 154, pp. 95–118.

Breuker, D., Uiterwijk, J., and Herik, H. van den (1996). Replacement Schemes and Two-Level Tables. *ICCA Journal*, Vol. 19, No. 3, pp. 175–180.

Chellapilla, K. and Fogel, D. B. (1999). Evolving Neural Networks to Play Checkers without Expert Knowledge. *IEEE Transactions on Neural Networks*, Vol. 10, No. 6, pp. 1382–1391. http://www.natural-selection.com/people/dbf/docs/ChellapillaAndFogelTransNN.zip.

Dippon, J. (2003). Accelerated randomized stochastic optimization. *Annals of Statistics*, Vol. 31, No. 4, pp. 1260–1281.

Donninger, C. (1993). Null Move and Deep Search: Selective-Search Heuristics for Obtuse Chess Programs. *ICCA Journal*, Vol. 16, No. 3, pp. 137–143.

Heinz, E. (1999). Adaptive Null-Move Pruning. *ICCA Journal*, Vol. 22, No. 3, pp. 123–132.

Igel, C. and Hüsken, M. (2003). Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, Vol. 50(C), pp. 105–123. citeseer.ist.psu.edu/igel03empirical.html.

Igel, C. and Hüsken, M. (2000). Improving the Rprop Learning Algorithm. *Proceedings of the Second International ICSC Symposium on Neural Computation (NC 2000)* (eds. H. Bothe and R. Rojas), pp. 115–121, ICSC Academic Press. citeseer.ist.psu.edu/igel00improving.html.

Kleinman, N., Spall, J., and Neiman, D. (1999). Simulation-based optimization with stochastic approximation using common random numbers. *Management Science*, Vol. 45, No. 11, pp. 1570–1578.

Kocsis, L. (2003a). *Learning Search Decisions.* Ph.D. thesis, Universiteit Maastricht, The Netherlands.

Kocsis, L., Herik, H. van den, and Uiterwijk, J. (2003b). Two learning algorithms for forward pruning. *ICGA Journal*, Vol. 26, No. 3, pp. 165–181.

L'Ecuyer, P. and Yin, G. (1998). Budget-Dependent Convergence Rate of Stochastic Approximation. *SIAM J. on Optimization*, Vol. 8, No. 1, pp. 217–247. ISSN 1052–6234.

Marsland, T. and Campbell, M. (1982). Parallel Search of Strongly Ordered Game Trees. *Computing Surveys*, Vol. 14, No. 4, pp. 533–551.

Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. *Proceedings of the IEEE International Conference on Neural Networks* (ed. E. Ruspini), pp. 586–591, IEEE Press.

Sackson, S. (1969). *A Gamut of Games.* Random House, New York, NY, USA.

Schaeffer, J. and Plaat, A. (1996). New Advances in Alpha-Beta Searching. *Proceedings of the 1996 ACM 24th Annual Conference on Computer Science*, pp. 124–130. ACM Press, New York, NY, USA.

Spall, J. C. (1992). Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, Vol. 37, pp. 332–341. citeseer.ist.psu.edu/spall92multivariate.html.

Spall, J. (2000). Adaptive Stochastic Approximation by the Simultaneous Perturbation Method. *IEEE Transactions on Automatic Control*, Vol. 45, pp. 1839–1853.

Sutton, R. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, Vol. 3, pp. 9–44.

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning*, Vol. 8, pp. 257–277.

Theiler, J. and Alper, J. (2004). On the Choice of Random Directions for Stochastic Approximation Algorithms. *IEEE Transactions on Automatic Control.* submitted.

Tsuruoka, Y., Yokoyama, D., and Chikayama, T. (2002). Game-tree Search Algorithm based on Realization Probability. *ICGA Journal*, Vol. 25, No. 3, pp. 132–144.

Winands, M. H. M. (2004). *Informed Search in Complex Games.* Ph.D. thesis, Universiteit Maastricht, Maastricht, The Netherlands.

Winands, M., Herik, H. van den, and Uiterwijk, J. (2003). An Evaluation Function for Lines of Action. *Advances in Computer Games 10: Many Games, Many Challenges* (eds. H. van den Herik, H. Iida, and E. Heinz), pp. 249–260. Kluwer Academic Publishers, Boston, MA, USA.

Winands, M., Kocsis, L., Uiterwijk, J., and Herik, H. van den (2002). Temporal Difference Learning and the Neural MoveMap Heuristic in the Game of Lines of Action. *Proceedings of 3rd International Conference on Intelligent Games and Simulation (GAME-ON 2002)*, pp. 99–103.

Winands, M., Werf, E. van der, Herik, H. van den, and Uiterwijk, J. (2004). The Relative History Heuristic. *Proceedings of the Fourth International Conference on Computers and Games (CG 2004)* (eds. H. van den Herik, Y. Björnsson, and N. Netanyahu). Accepted for publication.