Learning While Optimizing an Unknown Fitness Surface[*]

Roberto Battiti, Mauro Brunato, and Paolo Campigotto

DISI - Dipartimento di Ingegneria e Scienza dell'Informazione,
Università di Trento, Italy
battiti@disi.unitn.it

Abstract. This paper is about Reinforcement Learning (RL) applied to online parameter tuning in Stochastic Local Search (SLS) methods. In particular a novel application of RL is considered in the Reactive Tabu Search (RTS) method, where the appropriate amount of diversification in prohibition-based (Tabu) local search is adapted in a fast online manner to the characteristics of a task and of the local configuration. We model the parameter-tuning policy as a Markov Decision Process where the states summarize relevant information about the recent history of the search, and we determine a near-optimal policy by using the Least Squares Policy Iteration (LSPI) method. Preliminary experiments on Maximum Satisfiability (MAX-SAT) instances show very promising results indicating that the learnt policy is competitive with previously proposed reactive strategies.

1 Reinforcement Learning and Reactive Search

Reactive Search (RS) [1,2,3] advocates the integration of sub-symbolic machine learning techniques into search heuristics for solving complex optimization problems. The word *reactive* hints at a ready response to events during the search through an internal online feedback loop for the self-tuning of critical parameters. When Reactive Search is applied to local search (Reactive Local Search or RLS), its objective is to maximize a given function $f(x)$ by analyzing the past local search history (the trajectory of the tentative solution in the search space) and by learning the appropriate balance of intensification and diversification. In this manner the knowledge about the task and about the local properties of the *fitness surface* surrounding the current tentative solution can influence the future search steps to render them more effective.

Reinforcement Learning (RL) arises in the different context of machine learning, where there is no guiding teacher, but *feedback signals from the environment* which are used by the learner to modify its future actions. Think about bicycle riding: after some initial trials with positive or negative rewards, in the form of admiring friends or injuries to biological tissues, the goal is accomplished. The

[*] Work supported by the project CASCADAS (IST-027807) funded by the FET Program of the European Commission.

reinforcement learning context is more difficult than the one of supervised learning, where a teacher gives examples of correct outputs: in RL one has to make a *sequence of decisions* (e.g., about steering wheel rotation). The outcome of each decision is not fully predictable. In addition to an immediate *reward*, each action causes a change in the system state and therefore a different context for the next decisions. To complicate matters the reward is often delayed and one aims at maximizing not the immediate reward, but some form of *cumulative reward* over a sequence of decisions. This means that greedy policies do not always work. In fact, it can be better to go for a smaller immediate reward if this action leads to a state of the system where bigger rewards can be obtained in the future. Goal-directed learning from interaction with an (unknown) environment with trial-and-error search and delayed reward is the main feature of RL.

As it was suggested for example in [4], the issue of learning from an initially unknown environment is therefore shared by RS and RL. A basic difference is that RS optimizes a function and the environment is provided by a fitness surface to be explored, while RL optimizes the long-term reward obtained by selecting actions at the different states. The sequential decision problem and therefore the non-greedy nature of choices is also common. For example, in Reactive Tabu Search (the application of RS in the context of Tabu Search), steps leading to worse configurations need in some cases to be performed to escape from a basin of attraction around a local optimizer. It is therefore of interest to investigate the relationship in more detail, to see whether specific techniques of Reinforcement Learning can be profitably used in Reactive Search.

This paper is organized as follows. First the basics of RL learning and neuro-dynamic programming are summarized. Then the relationship between RL and RS are investigated, also with reference to existing work bridging the border between optimization and RL. Finally, the novel proposal is presented, together with the first obtained experimental results.

2 Reinforcement Learning and Neuro-dynamic Programming Basics

In this section, *Markov Decision Processes* are formally defined and the standard Dynamic Programming technique to determine the optimal policy is introduced in Sec. 2.2. In many practical cases exact solutions must be abandoned in favor of approximation strategies, which are the focus of Sec. 2.4.

2.1 Markov Decision Processes

A standard Markov process is given by a set of states $\mathcal{S}$ with transitions between them described by probabilities $p(i, j)$ (let us note the fundamental property of Markov models: earlier states do not influence the transition probabilities to the next state). Its evolution cannot be controlled, because it lacks the notion of decisions, *actions* taken depending on the current state and leading to a different state and to an immediate *reward*.

A Markov Decision Process (MDP) is an extension of the classical Markov process designed to capture the problem of *sequential decision making under uncertainty*, with states, decisions, unexpected results, and "long-term" goals to be reached. A MDP can be defined as a quintuple $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where $\mathcal{S}$ is a set of states, $\mathcal{A}$ a finite set of actions, $P(s, a, s')$ is the probability of transition from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ if action $a \in \mathcal{A}$ is taken, $R(s, a, s')$ is the corresponding reward, and γ is the discount factor, in order to exponentially decrease future rewards. This last parameter is fundamental in order to evaluate the overall value of a choice when considering its consequences on an infinitely long chain. In particular, given the following evolution of a MDP

$$s(0) \overset{a(0)}{\to} s(1) \overset{a(1)}{\to} s(2) \overset{a(2)}{\to} s(3) \overset{a(3)}{\to} \dots \tag{1}$$

the cumulative reward obtained by the system is given by

$$\sum_{t=0}^{\infty} \gamma^t R(s(t), a(t), s(t+1)).$$

Note that state transitions are not deterministic, nevertheless their distribution can be controlled by the action a. The goal is to control the system in order to maximize the expected cumulative reward.

Given a MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$, we define a *policy* as a probability distribution $\pi(\cdot|s) : \mathcal{A} \to [0, 1]$, where $\pi(a|s)$ is the probability of choosing action a when the system is in state s. In other words, π maps states onto probability distributions over $\mathcal{A}$. Note that we are only considering stationary policies. If a policy is deterministic, then we resort to the more compact notation $a = \pi(s)$.

2.2 The Dynamic Programming Approach

The intelligent component goal is to select a policy that maximizes a measure of the total reward accumulated during an infinite chain of decisions (infinite-horizon). To achieve this goal, let us define the *state-action value function* $Q^\pi(s, a)$ of the policy π as the expected overall future reward for applying a specified action a when the system is in state s, in the hypothesis that the ensuing actions are taken according to policy π. A straightforward implementation of the Bellman principle leads to the following definition:

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} P(s, a, s') \Bigl(R(s, a, s') + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s', a') \Bigr). \tag{2}$$

where the sum over $\mathcal{S}$ can be interpreted as an integral in case of a continuous state set. The interpretation is that the value of selecting action a in state s is given by the expected value of the immediate reward plus the value the future rewards which one expects by following policy π from the new state. These have to be discounted by γ (they are a step in the future w.r.t. starting immediately from the new state) and properly weighted by transition probabilities and action-selection probabilities given the stochasticity in the process.

The expected reward of a state/action pair $(s,a) \in \mathcal{S} \times \mathcal{A}$ is

$$R(s,a) = \sum_{s' \in \mathcal{S}} P(s,a,s') R(s,a,s'),$$

so that (2) can be rewritten as

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} \left(P(s,a,s') \sum_{a' \in \mathcal{A}} \pi(a'|s') Q^\pi(s',a') \right)$$

or, in a more compact linear form,

$$Q^\pi = \boldsymbol{R} + \gamma \boldsymbol{P} \boldsymbol{\Pi}_\pi Q^\pi \tag{3}$$

where $\boldsymbol{R}$ is the $|\mathcal{S}||\mathcal{A}|$-entry column vector corresponding to $R(s,a)$, $\boldsymbol{P}$ is the $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}|$ matrix of $P(s,a,s')$ values having (s,a) as row index and s' as column, while $\boldsymbol{\Pi}_\pi$ is a $|\mathcal{S}| \times |\mathcal{S}||\mathcal{A}|$ matrix whose entry $(s,(s,a))$ is $\pi(a|s)$.

Equation (3) can be seen as a non-homogeneous linear problem with unknown Q^π

$$(\boldsymbol{I} - \gamma \boldsymbol{P} \boldsymbol{\Pi}_\pi) Q^\pi = R \tag{4}$$

or, alternatively, as a fixed-point problem

$$Q^\pi = \boldsymbol{T}_\pi Q^\pi, \tag{5}$$

where $\boldsymbol{T}_\pi : x \mapsto R + \gamma \boldsymbol{P} \boldsymbol{\Pi}_\pi x$ is an affine functional.

If the state set $\mathcal{S}$ is finite, then (3-5) are matrix equations and the unknown Q^π is a vector of size $|\mathcal{S}||\mathcal{A}|$.

In order to solve these equations explicitly, a model of the system is required, i.e., full knowledge of functions $P(s,a,s')$ and $R(s,a)$. When the system is too large, or the model is not completely available, approximations in the form of *reinforcement learning* come to the rescue. As an example, if a *generative model* is available, i.e., a black box that takes state and action in input and produces the reward and next state as output, one can estimate $Q^\pi(s,a)$ through *rollouts.* In each rollout, the generator is used to simulate action a followed by a sufficiently long chain of actions dictated by policy π. The process is repeated several times because of the inherent stochasticity, and averages are calculated.

The above described state-action value function Q, or its approximation, is instrumental in the basic methods of dynamic programming and reinforcement learning.

2.3 Policy Iteration

A method to obtain the optimal policy π^* is to generate an improving sequence (π_i) of policies by building a policy π_{i+1} upon the value function associated to policy π_i:

$$\pi_{i+1}(s) = \arg\max_{a \in \mathcal{A}} Q^{\pi_i}(s,a). \tag{6}$$

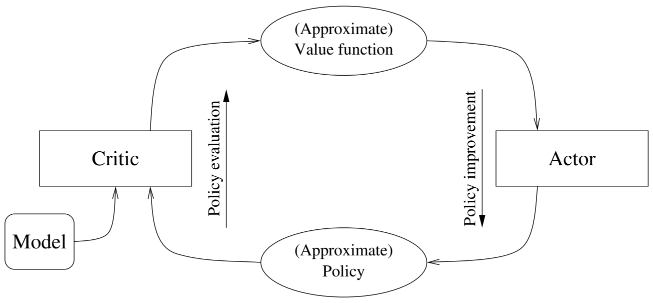

Fig. 1. The Policy Iteration (PI) mechanism

Policy π_{i+1} is never worse than π_i, in the sense that $Q^{\pi_{i+1}} \geq Q^{\pi_i}$ over all state/action pairs.

In the following, we assume that the optimal policy π^* exists in the sense that for all states it attains the minimum of the right-hand side of Bellman's equation, see [5] for more details.

The Policy Iteration (PI) method consists on the alternate computation shown in Fig. 1: given a policy π_i, the *policy evaluation* procedure (also known as the "Critic") generates its state-action value function Q^{π_i}, or a suitable approximation. The second step is the *policy improvement* procedure (the "Actor"), which computes a new policy by applying (6).

The two steps are repeated until the value function does not change after iterating, or when the change between consecutive iterations is less than a given threshold.

2.4 Approximations: Reinforcement Learning and LSPI

To carry out the above discussion by means of exact methods, in particular using (4) as the Critic component, the system model has to be known in terms of its transition probability $P(s, a, s')$ and reward $R(s, a)$ functions. In many cases this detailed information is not available but we have access to the system itself or to a *simulator*. In both cases, we have a black box which given the current state and the performed action determines the next state and reward. In both cases, more conveniently with a simulator, several sample trajectories can be generated, so that more and more information about the system behavior can be extracted aiming at optimal control.

A brute force approach can be that of estimating the system model functions $R(\cdot,\cdot,\cdot)$ and $R(\cdot,\cdot)$ by executing a very large series of simulations. The model-free *Reinforcement Learning* methodology bypasses the system model and directly learns the value function.

Assume that the system simulator (the "Model" box in Fig. 1) generates quadruples in the form

$$(s, a, r, s')$$

where s is the state of the system at a given step, a is the action taken by the simulator, s' is the state in which the system falls after the application of a, and

Variable	Scope	Description
$\mathcal{D}$	In	Set of sample vectors $\{(s,a,r,s')\}$
k	In	Number of basis functions
$\boldsymbol{\Phi}$	In	Vector of k basis functions
γ	In	Discount factor
π	In	Policy
$\boldsymbol{A}$	Local	$k \times k$ matrix
$\boldsymbol{b}$	Local	k-entry column vector
$\boldsymbol{w}^\pi$	Out	k-entry weight vector

```
1. function LSTDQ ( $\mathcal{D}$, $k$, $\boldsymbol{\Phi}$, $\gamma$, $\pi$)
2.     $\boldsymbol{A} \leftarrow 0$;
3.     $\boldsymbol{b} \leftarrow 0$;
4.     for each $(s,a,r,s') \in D$
5.         $\boldsymbol{A} \leftarrow \boldsymbol{A} + \boldsymbol{\Phi}(s,a)\big(\boldsymbol{\Phi}(s,a) - \gamma\boldsymbol{\Phi}(s',\pi(s'))\big)^T$
6.         $\boldsymbol{b} \leftarrow \boldsymbol{b} + r\boldsymbol{\Phi}(s,a)$
7.     $\boldsymbol{w}^\pi \leftarrow \boldsymbol{A}^{-1}\boldsymbol{b}$
```

Fig. 2. The LSTDQ algorithm [6]

r is the reward received. In the setting described by this paper, the (s,a) pair is generated by the simulator.

A viable method to obtain an approximation of the state-action value function is to approximate it with respect to a functional linear subspace having basis $\boldsymbol{\Phi} = (\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_k)$. The approximation $\hat{Q}^\pi \approx Q^\pi$ is in the form

$$\hat{Q}^\pi = \boldsymbol{\Phi}^T \boldsymbol{w}^\pi.$$

The weights vector $\boldsymbol{w}^\pi$ is the solution of the linear system $\boldsymbol{A}\boldsymbol{w}^\pi = \boldsymbol{b}$, where

$$\boldsymbol{A} = \boldsymbol{\Phi}^T(\boldsymbol{\Phi} - \gamma \boldsymbol{P}\boldsymbol{\Pi}_\pi \boldsymbol{\Phi}) \qquad \boldsymbol{b} = \boldsymbol{\Phi}^T R. \tag{7}$$

An approximate version of (7) can be obtained if we assume that a finite set of samples is provided by the "Model" box of Fig. 1:

$$\mathcal{D} = \{(s_1, a_1, r_1, s'_1), \dots, (s_l, a_l, r_l, s'_l)\}.$$

In this case, matrix $\mathcal{A}$ and vector $\boldsymbol{b}$ are "learned" as sums of rank-one elements, each obtained by a sample tuple:

$$\boldsymbol{A} = \sum_{(s,a,r,s') \in \mathcal{D}} \boldsymbol{\Phi}(s,a)\Big(\boldsymbol{\Phi}(s,a) - \gamma\boldsymbol{\Phi}(s',\pi(s'))\Big)^T, \qquad \boldsymbol{b} = \sum_{(s,a,r,s') \in \mathcal{D}} r\boldsymbol{\Phi}(s,a).$$

These approximations lead to the Least Squares Temporal Difference for Q (LSTDQ) algorithm proposed in [6], and shown in Figure 2, where the functions $R(s,a)$ and $P(s,a,s')$ are supposed to be unknown and are replaced by a finite sample set $\mathcal{D}$.

Note that the LSTDQ algorithm returns the weight vector that best approximates in the least-squares fixed-point sense (within the spanned subspace and

Variable	Scope	Description
$\mathcal{D}$	In	Set of sample vectors $\{(s, a, r, s')\}$
k	In	Number of basis functions
$\boldsymbol{\Phi}$	In	Vector of k basis functions
γ	In	Discount factor
ϵ	In	Weight vector tolerance
$\boldsymbol{w}_0$	In	Initial value function weight vector
$\boldsymbol{w}'$	Local	Weight vectors in subsequent iterations
$\boldsymbol{w}$	Out	Optimal weight vector

```
1. function LSPI ($D$, $k$, $\boldsymbol{\Phi}$, $\gamma$, $\epsilon$, $\boldsymbol{w}_0$)
2.     $\boldsymbol{w}' \leftarrow \boldsymbol{w}_0$;
3.     do
4.         $\boldsymbol{w} \leftarrow \boldsymbol{w}'$;
5.         $\boldsymbol{w}' \leftarrow$ LSTDQ ($D$, $k$, $\boldsymbol{\Phi}$, $\gamma$, $\boldsymbol{w}$);
6.     while $\|\boldsymbol{w} - \boldsymbol{w}'\| > \epsilon$
```

Fig. 3. The LSPI algorithm [6]

according to the sample data) the value function of a given policy π. It therefore acts as the "Critic" component of the Policy Iteration algorithm. The "Actor" component is straightforward, because it is an application of (6). The policy does not need to be explicitly represented: if the system is in state s and the current value function is defined by weight vector $\boldsymbol{w}$, the best action to take is

$$a = \arg\max_{a \in \mathcal{A}} \boldsymbol{\Phi}^T \boldsymbol{w}.$$

The complete LSPI algorithm is given in Fig. 3. Note that, because of the identification between the weight vector $\boldsymbol{w}$ and the ensuing policy π, the code assumes that the previously declared function `LSTDQ()` accepts its last parameter, i.e., the policy π, in form of a weight vector $\boldsymbol{w}$.

3 Reinforcement Learning for Optimization

Many are the intersections between optimization, Dynamic Programming and Reinforcement Learning. Approximated versions of DP/RL contain challenging optimization tasks, let's mention the maximization operations in determining the best action when an action value function is available, the optimal choice of approximation architectures and parameters in neuro-dynamic programming, or the optimal choice of algorithm details and parameters for a specific RL instance.

This paper, however, goes in the opposite direction: which techniques of RL can be used to improve heuristic algorithms for a standard optimization task such as minimizing a function? Interesting summaries of statistical machine learning for large-scale optimization are present in [7].

An application of RL in the area of local search for solving $\max_x f(x)$ is presented in [8]: the rewards from a local search method π starting from an initial configuration x are given by the size of improvements of the best-so-far value f_{best}. In detail, the value function $V^{\pi}(x)$ of configuration x is given by the expected best value of f seen on a trajectory starting from state x and following the local search method π. The curse of dimensionality discourages using directly x for state description: informative *features* extracted from x are used to compress the state description to a shorter vector $\boldsymbol{s}(x)$, so that the value function becomes $V^{\pi}(\boldsymbol{s}(x))$.

A second application of RL to local search is to supplement f with a "scoring function" to help in determining the appropriate search option at every step. For example, different basic moves or entire different neighborhoods can be applied. RL can in principle make more systematic some of the heuristic approaches involved in designing appropriate "objective functions" to guide the search process. An example is the RL approach to job-shop scheduling in [9,10], where a neural-network based $TD(\lambda)$ scheduler is demonstrated to outperform a standard iterative repair (local search) algorithm.

Also, tree-search techniques can profit from ML. It is well known that variable and value ordering heuristics (choosing the right order of variables or values) can noticeably improve the efficiency of complete search techniques, e.g. for constraint satisfaction problems. For example, RLSAT [11] is a DPLL solver for the Satisfiability (SAT) problem which uses experience from previous executions to learn how to select appropriate branching heuristics from a library of predefined possibilities, with the goal of minimizing the total size of the search tree, and therefore the CPU time. Lagoudakis and Littman [12] extend algorithm selection for recursive computation, which is formulated as a sequential decision problem. According to the authors, their work demonstrates that "some degree of reasoning, learning, and decision making on top of traditional search algorithms can improve performance beyond that possible with a fixed set of hand-built branching rules."

A different application is suggested in [5] in the context of constructive algorithms, which build a complete solution by selecting value for a component at a time. Let's assume that K fixed construction algorithms are available for the problem. The application consists of combining in the most appropriate manner the information obtained by the set of construction algorithms in order to fix the next index and value.

In the context of continuous function optimization, [13] uses RL for replacing a priori defined adaptation rules for the step size in Evolution Strategies with a reactive scheme which adapt step sizes automatically during the optimization process. The states are characterized only by the success rate after a fixed number of mutations, the three possible actions consists of increasing (by a fixed multiplicative amount), decreasing or keeping the current step size. SARSA learning with various reward functions is considered, including combinations of the difference between the current function value and the one evaluated at the last reward computation and the movement in parameter space (the distance traveled in the last phase). On-the-fly parameter tuning, or on-line calibration of parameters for evolutionary algorithms by reinforcement learning (crossover, mutation, selection operators, population size) is suggested in [14].

4 Reinforcement Learning for Reactive Tabu Search

This paper investigates a novel application of Reinforcement Learning in the framework of Reactive Tabu Search. An optimization algorithm operates a sequence of elementary actions (*local moves*, e.g., bit flips). The choice of the local move is driven by many different factors, in particular, most algorithms are *parametric*: their behavior (and their efficiency) depends on the values attributed to some free parameters, so that different instances of the same problem, and different configurations within the same instance, may require different parameter values.

This Section describes the proposed application of the LSPI algorithm to MAX-SAT: the Markov Decision Process (MDP) is described in Sec. 4.1, while the design of the basis function is described in Sec.4.2.

4.1 The Markov Decision Process Definition

The effect of a parameter change on the algorithm's behavior can only be evaluated after a significant number of local moves. As a consequence, also for performance reasons, algorithm parameters are not changed too often. We therefore divide the algorithm's trace into *epochs*, each composed of a suitable number of local moves, and to allow parameter changes only between epochs.

If the "state" of the system at the end of an epoch describes the algorithm's behavior during the last epoch, and an "action" is the modification of the algorithm's parameters before it enters the next epoch, then a local search algorithm can be modeled as a Markov Decision Process (MDP) and a Reinforcement Learning method such as LSPI can be used to control the evolution of its parameters.

The "state" should capture all criteria that we consider useful in order to decide how to change parameters in a proper way. Given the subdivision of the Local Search algorithm's trace into a sequence of epochs $(E_1, E_2, \dots)$, we define the state at the end of epoch E_i as a collection of features extracted from the algorithm's execution up to that moment in form of a tuple: $s(E_1, \dots, E_i) \in \mathbb{R}^d$, where d is the number of features that form the state. The features can be adequately normalized for better stability of the system. The cardinality of the action set $\mathcal{A}$ and the semantics of its elements changes according to the parameters required by the LS technique. Variable Neighborhood Search algorithms can define one action for each implemented neighborhood, or define just two actions (to be interpreted, e.g., as "widen" and "reduce") if the neighborhood set is ordered. Simulated Annealing, which basically depends on a continuous parameter T, can define two actions ("increase T" and "decrease T"). Likewise, a Tabu Search algorithm will increase or decrease the prohibition period T.

In this paper we consider a prohibition-based (Tabu) algorithm for the MAX-SAT problem [15]. It takes in input a CNF SAT instance (i.e., a Boolean formula being the conjunction of disjunctive clauses) and each algorithm step simply flips a variable. In particular, every variable is considered for flipping (i.e., non-prohibited) only if it hasn't been changed in the previous T iterations, T being the prohibition parameter to be controlled. At each iteration, the non-prohibited variable causing the largest increase in the number of satisfied clauses (or the lowest decrease, if no increase is possible) is selected for flipping. Ties are broken randomly. In this paper, T is assumed to take values over the interval $[T_{min,T_{max}}]$.

The Reinforcement Learning approach is exploited to adjust the prohibition parameter during the algorithm execution. Assume n and m the number of variables and clauses of the input SAT instance, respectively. Let $f(\boldsymbol{x})$ the score function counting the number of unsatisfied clauses in the truth assignment $\boldsymbol{x}$.

Each state of the MDP is created by observing the behavior of the Tabu search algorithm over an epoch of $2 * T_{max}$ consecutive variable flips. As in a prohibition mechanism with prohibition parameter T, during the first T steps, the Hamming distance keeps increasing and only in the subsequent steps it may decrease, an epoch is long enough to monitor the behavior of the algorithm also in the case of the largest allowed T value.

In particular, let us define the following:

- $\boldsymbol{x}_{\mathrm{bsf}}$ is the "best-so-far" configuration *before* the current epoch;
- T_{f} is the current fractional prohibition value (the actual prohibition period is
$$T = \lfloor nT_{\mathrm{f}} \rfloor \tag{8}$$
);
- $\overline{f}_{\mathrm{epoch}}$ is the average value of f during the epoch;
- $\overline{H}_{\mathrm{epoch}}$ is the average Hamming distance during the current epoch from the configuration at the beginning of the current epoch itself.

These variables have been chosen because of the Reactive Search paradigm's concern on the trade-off between diversification (the ability to explore new configurations in the search space by moving away from local minima) and bias (the preference for configurations with low objective function values), so that changes in f and the Hamming distance are good representatives of the current state. Many possible choices based on these considerations have been tested. Furthermore, for the purpose of addressing uniformly SAT instances with different number of variables, the fractional prohibition value T_{f} is used rather than the prohibition value T. The compact state representation chosen to describe an epoch is the following triplet:

$$\boldsymbol{s} \equiv \left(\Delta f, \frac{\overline{H}_{\mathrm{epoch}}}{n}, T_{\mathrm{f}} \right), \qquad \text{where} \qquad \Delta f = \frac{\overline{f}_{\mathrm{epoch}} - f(\boldsymbol{x}_{\mathrm{bsf}})}{m}.$$

The first component is the mean change of f in the current epoch with respect to the best value; all components of the state have been normalized.

The actions set is composed by two choices: $\mathcal{A} = \{\text{increase}, \text{decrease}\}$, with the following effects:

- if $a = \text{increase}$: $T_\text{f} \leftarrow \max\{T_\text{f} \cdot 1.1, T_\text{f} + 1/n\}$;
- if $a = \text{decrease}$: $T_\text{f} \leftarrow \min\{T_\text{f}/1.1, T_\text{f} - 1/n\}$.

Changes in T_f are designed in order to ensure variation of at least 1 in the actual prohibition period T. In addition, T_f is bounded between a minimum and a maximum value (0 and .2 in our experiments).

The reward signal is given by the normalized change of the best value achieved in the observed epoch with respect to the "*best so far*" value *before* the epoch: $(f(\boldsymbol{x}_\text{bsf}) - f(\boldsymbol{x}_\text{localBest}))/m$.

4.2 Basis Function Definition

Among the various tests that have been executed, in this paper we concentrate on the following 13-function basis function set:

$$\boldsymbol{\Phi}(s,a) = \begin{pmatrix} I_\text{increase}(a) & I_\text{decrease}(a) \\ I_\text{increase}(a) \cdot \Delta f & I_\text{decrease}(a) \cdot \Delta f \\ I_\text{increase}(a) \cdot \overline{H}_\text{epoch} & I_\text{decrease}(a) \cdot \overline{H}_\text{epoch} \\ I_\text{increase}(a) \cdot \overline{H}_\text{epoch} \cdot \Delta f & I_\text{decrease}(a) \cdot \overline{H}_\text{epoch} \cdot \Delta f \\ I_\text{increase}(a) \cdot (\Delta f)^2 & I_\text{decrease}(a) \cdot (\Delta f)^2 \\ I_\text{increase}(a) \cdot \overline{H}^2_\text{epoch} & I_\text{decrease}(a) \cdot \overline{H}^2_\text{epoch} \\ & T_\text{f} + \frac{I_\text{increase}(a) - I_\text{decrease}(a)}{n} \end{pmatrix}, \quad (9)$$

where I_increase and I_decrease are the indicator functions for the two actions (1 if the action is the indicated one, 0 otherwise), discerning the "state-action" features for the two different actions considered.

5 Experimental Results

In order to test the performance of Reinforcement Learning for on-line parameter tuning in Reactive Tabu Search (RTS), we have implemented C++ functions for the Tabu Search method described in Sec. 4.1 and interfaced them to the Matlab LSPI implementation found in [16].

The experimental work includes the generation of a training set of samples discussed in Sec. 5.1, the generation of an optimal policy and in the preliminary comparison with other relevant SLS heuristics for MAX-SAT in Sec. 5.2

5.1 Training Examples Generation

The training examples are created by running the Tabu search algorithm over selected MAX-3-SAT random instances defined in [17]. In detail, we selected two ($n = 500, m = 5000$) instances and 6 different initial prohibition periods ($T_\text{f} = .01, .02, .05, .1, .15, .2$), and performed 2 runs of the algorithm for each

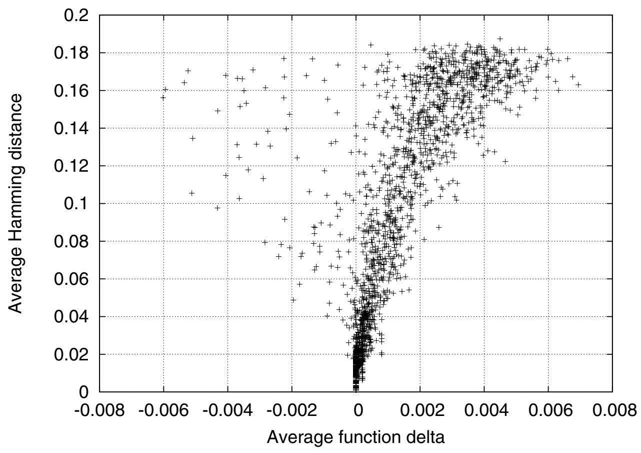

Fig. 4. Distribution of training sample states

combination with different randomly chosen starting truth assignments. Every run has been executed for 50 epochs to generate 50 training examples. The T_{f} parameter has been bounded in $[0, .2]$.

Each epoch is composed of 200 consecutive flips, as $T_{max} = 500 \cdot 0.2$ by Eq. 8.

Fig. 4 shows the distribution of examples states, projected onto the Δf and the $\overline{H}_{\mathrm{epoch}}$ state features.

5.2 Optimal Policy and Comparison

The LSPI algorithm has been applied to the training sample set, and with (9) as approximate space basis. The resulting approximate value function $\hat{Q}(s, a)$ is shown in Fig. 5 for the two actions, thus defining an approximation to the optimal policy. Note that the action "increase" is suggested in cases where the average Hamming distance between the configurations explored in the last epoch and the last local minimum does not exceed a certain value, provided that the current portion of landscape is not much worse than the previously explored regions. This policy is consistent with intuition: a higher value of T causes a larger differentiation of visited configurations (more different variables need to be flipped), and this is desired when the algorithm needs to escape the neighborhood of a local minimum; in this case, in fact, movement is limited because the configuration is trapped at the "bottom of a valley". On the other hand, when the trajectory is not within the attraction basin of a minimum, a lower value of T enables a better exploitation of the neighborhood.

To evaluate our novel MAX-SAT solver based on Reinforcement learning we report here a comparison with some of the best and famous SLS algorithms for MAX-SAT. In particular, the following SLS techniques are considered:

- GSAT/Tabu [18], which enriches the GSAT algorithm [19] via a prohibition-based search criterion;

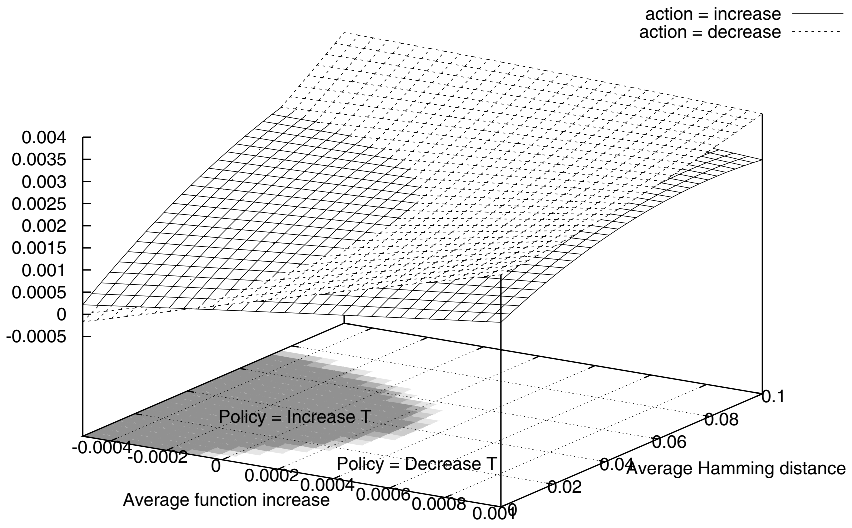

Fig. 5. Value function $\hat{Q}(s, a)$ for the two actions in the significant portion of the state space

- WalkSAT/Tabu [20], that adopts the same score function and the same variables selection mechanism of the WalkSAT/SKC algorithm [21], complemented by Tabu search;
- AdaptNovelty$^+$ [22], that exploits the concept of variable "age" and uses the same scoring function of GSAT.
- RSAPS, a reactive version of the Scaling and Probabilistic Smoothing (SAPS) [23] algorithm, on its turn, an accelerated version of the Exponentiated Subgradient algorithm [24] based on dynamic penalties;
- H_RTS ([25]), a prohibition-based algorithm that dynamically adjusts the prohibition parameter by monitoring the Hamming distance along the search trajectory.

While in this paper we base our comparisons on the solution quality after a given number of iterations, we note that the CPU time required by the proposed algorithm is analogous to that of the basic Tabu Search algorithm, with the overhead of two floating-point 13-element vector (Eq. 9) products in order to compute $\hat{Q}(s, a)$ for the two actions.

For each algorithm, 10 runs with different random seeds are performed for each of the 50 instances taken from the benchmark set described in [17], for a total of 500 tests. Fig. 6 shows the average results as a function of the number of iterations (flips), in the case of $(n = 500, m = 5000)$ instances. Among all the possible values for the prohibition parameter of the WalkSAT/Tabu algorithm, we plot the case $T_{\mathrm{f}} = .01$, as with this setting we obtain the best performance over the considered benchmark. The same for the GSAT/Tabu algorithm, whose curve is drawn for the optimal T_{f} value 0.05 over our benchmark set. Fig. 6

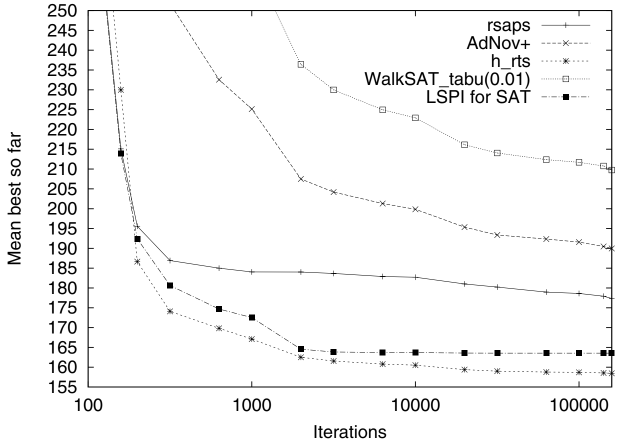

Fig. 6. Comparison among different algorithms

indicates that our RL-based approach is competitive with the other existing SLS MAX-SAT solvers.

6 Conclusions

This paper described preliminary results on the application of Dynamic Programming and Reinforcement Learning techniques to Reactive Search algorithms. In particular, the dependence of the algorithm on the prohibition parameter has been modeled as a Markov Decision Process and solved by means of the LSPI technique, achieving results that are comparable to the best algorithms in the literature.

Possible future improvements include the definition of alternative features for state description and of different reward functions. The optimal policy is currently learnt by means of the off-line generation of sample traces on a small number of instances, and the robustness of the learnt policy with respect to different problem instances has been tested. Another direction of research will cover on-line training where the optimal policy is determined by learning *while* the target optimization task is performed.

References

1. Battiti, R., Tecchiolli, G.: The reactive tabu search. ORSA Journal on Computing 6(2), 126–140 (1994)
2. Battiti, R., Brunato, M.: Reactive search: machine learning for memory-based heuristics. In: Gonzalez, T.F. (ed.) Approximation Algorithms and Metaheuristics, pp. 21–1 – 21–17. Taylor and Francis Books, CRC Press, Washington (2007)

3. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization. In: Operations research/Computer Science Interfaces. Springer, Heidelberg (in press, 2008)
4. Battiti, R.: Machine learning methods for parameter tuning in heuristics. In: 5th DIMACS Challenge Workshop: Experimental Methodology Day, Rutgers University (October 1996)
5. Bertsekas, D., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific (1996)
6. Lagoudakis, M., Parr, R.: Least-Squares Policy Iteration. Journal of Machine Learning Research 4(6), 1107–1149 (2004)
7. Baluja, S., Barto, A., Boese, K., Boyan, J., Buntine, W., Carson, T., Caruana, R., Cook, D., Davies, S., Dean, T., et al.: Statistical Machine Learning for Large-Scale Optimization. Neural Computing Surveys 3, 1–58 (2000)
8. Boyan, J.A., Moore, A.W.: Learning evaluation functions for global optimization and boolean satisfability. In: Press, A. (ed.) Proc. of 15th National Conf. on Artificial Intelligence (AAAI), pp. 3–10 (1998)
9. Zhang, W., Dietterich, T.: A reinforcement learning approach to job-shop scheduling. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, vol. 1114 (1995)
10. Zhang, W., Dietterich, T.: High-performance job-shop scheduling with a time-delay TD (λ) network. Advances in Neural Information Processing Systems 8, 1024–1030 (1996)
11. Lagoudakis, M., Littman, M.: Learning to select branching rules in the DPLL procedure for satisfiability. In: LICS 2001 Workshop on Theory and Applications of Satisfiability Testing, SAT 2001 (2001)
12. Lagoudakis, M., Littman, M.: Algorithm selection using reinforcement learning. In: Proceedings of the Seventeenth International Conference on Machine Learning, pp. 511–518 (2000)
13. Muller, S., Schraudolph, N., Koumoutsakos, P.: Step size adaptation in evolution strategies using reinforcementlearning. In: Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC 2002, vol. 1, pp. 151–156 (2002)
14. Eiben, A., Horvath, M., Kowalczyk, W., Schut, M.: Reinforcement learning for online control of evolutionary algorithms. In: Brueckner, S.A., Hassas, S., Jelasity, M., Yamins, D. (eds.) ESOA 2006. LNCS (LNAI), vol. 4335. Springer, Heidelberg (2007)
15. Battiti, R., Protasi, M.: Approximate algorithms and heuristics for MAX-SAT. In: Du, D., Pardalos, P. (eds.) Handbook of Combinatorial Optimization, vol. 1, pp. 77–148. Kluwer Academic Publishers, Dordrecht (1998)
16. Lagoudakis, M., Parr, R.: LSPI: Least-squares policy iteration (as of September 1, 2007),`http://www.cs.duke.edu/research/AI/LSPI/`
17. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distributions of SAT problems. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI 1992), San Jose, Ca, pp. 459–465 (July 1992)
18. Steinmann, O., Strohmaier, A., Stutzle, T.: Tabu search vs. random walk. In: KI - Kunstliche Intelligenz, pp. 337–348 (1997)
19. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI 1992), San Jose, Ca, pp. 440–446 (July 1992)
20. McAllester, D., Selman, B., Kautz, H.: Evidence for invariants in local search. In: Proceedings of the national conference on artificial intelligence (14), pp. 321–326. John Wiley & sons LTD., USA (1997)

21. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the national conference on artificial intelligence, vol. 12. John Wiley & sons LTD., USA (1994)
22. Tompkins, D.A.D., Hoos, H.H.: Novelty$^+$ and adaptive novelty$^+$. SAT 2004 Competition Booklet (solver description) (2004)
23. Tompkins, F.H.D., Hoos, H.: Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, p. 233. Springer, Heidelberg (2002)
24. Schuurmans, D., Southey, F., Holte, R.: The exponentiated subgradient algorithm for heuristic boolean programming. In: Proceedings of the international joint conference on artificial intelligence, vol. 17, pp. 334–341. Lawrence Erlbaum associates LTD., USA (2001)
25. Battiti, R., Protasi, M.: Reactive search, a history-sensitive heuristic for MAX-SAT. ACM Journal of Experimental Algorithmics 2 (ARTICLE 2) (1997), `http://www.jea.acm.org/`