An Investigation of Reinforcement Learning for Reactive Search Optimization

Roberto Battiti and Paolo Campigotto

Abstract Reactive Search Optimization advocates the adoption of learning mechanisms as an integral part of a heuristic optimization scheme. This work studies reinforcement learning methods for the online tuning of parameters in stochastic local search algorithms. In particular, the reactive tuning is obtained by learning a (near-)optimal policy in a Markov decision process where the states summarize relevant information about the recent history of the search. The learning process is performed by the Least Squares Policy Iteration (LSPI) method. The proposed framework is applied for tuning the prohibition value in the Reactive Tabu Search, the noise parameter in the Adaptive Walksat, and the smoothing probability in the Reactive Scaling and Probabilistic Smoothing (RSAPS) algorithm. The novel approach is experimentally compared with the original *ad hoc* reactive schemes.

1 Introduction

Reactive Search Optimization (RSO) [4, 7, 3] proposes the integration of subsymbolic 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 RSO is applied to local search, 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

Roberto Battiti
DISI - Dipartimento di Ingegneria e Scienza dell’Informazione, Università di Trento, Italy, e-mail: battiti@disi.unitn.it

Paolo Campigotto
DISI - Dipartimento di Ingegneria e Scienza dell’Informazione, Università di Trento, Italy, e-mail: campigotto@disi.unitn.it

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. In RL one has to make a sequence of decisions. The outcome of each decision is not fully predictable. In fact, 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 [2], the issue of learning from an initially unknown environment is therefore shared by RSO and RL. A basic difference is that RSO 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 RSO 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 RSO.

In this paper, we discuss the application of reinforcement learning methods for tuning the parameters of stochastic local search (SLS) algorithms. In particular, we select the Least Squares Policy iteration (LSPI) algorithm [22] to implement the reinforcement learning mechanism. This investigation is done in the context of the Maximum Satisfiability (MAX-SAT) problem. The reactive versions of the SLS algorithms for the SAT/MAX-SAT problem, like RSAPS, Adaptive Walksat or AdaptNovelty$^+$, perform better than their original non-reactive versions. While solving a single instance, the level of diversification of the reactive SLS algorithms is adaptively increased / decreased if search stagnation is / is not detected.

Reactive approaches in the above methods consist of *ad hoc* ways to detect the search stagnation and to adapt the value of one (or more) parameter determining the diversification of the algorithm.

In this investigation, a generic RL-based reactive scheme is designed, which can be customized to replace the *ad hoc* method of the algorithm considered.

To test its performance, we select three SAT/MAX-SAT SLS solvers: the (adaptive) Walksat, the Hamming Reactive Tabu Search (H_RTS) and the Reactive Scaling and Probabilistic Smoothing (RSAPS) algorithms. Their *ad hoc* reactive methods are replaced by our RL-based strategy and the results obtained over a random MAX-3-SAT benchmark and a structured MAX-SAT benchmark are discussed.

This paper is organized as follows. Previous approaches of reinforcement learning applied to optimization are discussed in Sec. 2, while the basics of reinforcement learning and dynamic programming are given in Sec. 3. In particular, the LSPI algorithm is detailed.

The considered Reactive SLS algorithms for the SAT/MAX-SAT problem are introduced in Sec. 4. Sec. 5 explains our integration of RSO with the RL strategy. Finally, the experimental comparison of the RL-based reactive approach w.r.t. the original *ad hoc* reactive schemes (Sec. 6) concludes the work.

2 Reinforcement learning for optimization

Many are the intersections between optimization, dynamic programming and reinforcement learning. Approximated versions of DP/RL contain challenging optimization tasks. Consider, for example, the maximization operations in determining the best action when an action value function is available, the optimal choice of approximation architectures and parameters in dynamic programming, or the optimal choice of algorithm details and parameters for a specific RL instance.

A recent paper about the interplay of optimization and machine learning (ML) research is [8], which mainly shows how recent advances in optimization can be profitably used in ML.

This work, 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 methods applied for large-scale optimization are presented in [1]. Autonomous search methods that adapt the search performance to the problem at hand are described in [15], which defines a metric to classify problem solvers based on their computation characteristics. The authors first identify a set of rules describing search processes, and then they classify existing search paradigms from different but related areas, such as the Satisfiability and the Constraint satisfaction problem.

In [10] RL is applied in the area of local search for solving $\max_x f(x)$: 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 directly using x for state description: informative *features* extracted from x are used to compress the state description to a shorter vector $s(x)$, so that the value function becomes $V^\pi(s(x))$. The introduced algorithm STAGE is a smart version of iterated local search, which alternates between two phases. In one phase, new training data for $V^\pi(F(x))$ are obtained by running local search from the given starting configuration. Assuming that local search is memory-less, the estimates of $V^\pi(F(x'))$ for all the points s' along the local search trajectory can be obtained from a single run. In the other phase, one optimizes the value function $V^\pi(F(x))$ (instead of the original f), so that a hopefully new and promising starting point is obtained. A suitable approximation architecture $V^\pi(F(x);w)$ and a supervised machine learning method to train $V^\pi(F(x);w)$ from the examples are needed. The memory-based version of the RASH (Reactive Affine Shaker) optimizer introduced in [11] adopts a similar strategy. An open issue is the proper definition of the features by the researcher, which are chosen by insight in [10]. Preliminary results are also presented about *transfer*, i.e., the possibility to use a value function learnt on one task for a different task with similar characteristics.

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 [31, 32], where a neural-network based $TD(\lambda)$ scheduler is demonstrated to outperform a standard iterative repair (local search) algorithm. The RL problem is designed to pick the best among two possible scheduling actions (moves transforming the current solution). The reward scheme is designed to prefer actions which *quickly* find a *good* schedule. For this purpose, a fixed negative reinforcement is given for each action leading to a schedule still containing constraint violations. In addition, a scale-independent measure of the final admissible schedule length gives feedback about the schedule quality. Features are extracted from the state, and are either hand-crafted or determined in an automated manner [32]; the δ values along the trajectory are used to gradually update a parametric model of the state value function of the optimal policy.

ML can be profitably applied also in tree-search techniques. 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 [21] is a DPLL solver for the 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 [20] extend algorithm selection for recursive computation, which is formulated as a sequential decision problem: the selection of an algorithm at a given stage requires an immediate cost – in the form of CPU time – and leads to a different state with a new decision to be executed. For SAT, features are extracted from each sub-instance to be solved, and TD learning with Least-Squares is used to learn an appropriate action value function, approximated as a linear function of the extracted features. Some modifications of the basic methods are needed. First, two new sub-problems are obtained instead of one, second an appropriate re-weighting of the samples is needed to avoid that the huge number of samples close to the leaves of the search tree practically hides the importance of samples close to the root. 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."

In [9] RL is applied in the context of the constructive search technique, which builds a complete solution by selecting the value for a solution component at a time.

One starts from the task data d and repeatedly picks a new index m_n to fix a value u_{m_n}, until values for all N components are fixed. The decision to be made from a partial solution $x_p = (d, m_1, u_{m_1}, ..., m_n, u_{m_n})$ is which index to consider next and which value to assign. Let us 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. The approximation architecture suggested is:

$$V(d, x_p) = \psi_0(x_p, w0) + \sum_{k=1}^{K} \psi_k(x_p, w_k) H_{k,d}(x_p)$$

where $H_{k,d}(x_p)$ is the value obtained by completing the partial solution with the k-th method, and ψ are tunable coefficients depending on the parameters w which are learned from many runs on different instances of the problem. While in the previously mentioned application one evaluates starting points for local search, here one evaluates the promise of partial solutions to lead to good complete solutions. The performance of this technique in terms of CPU time is poor: in addition to the separate training phase, the K construction algorithms must be run for each partial solution (for each variable-fixing step) in order to allow for picking the next step in an optimal manner.

A different parametric combination of costs of solutions obtained by two fixed heuristics is considered for the *stochastic programming problem* of maintenance and repair [9]. In the problem, one has to decide whether to immediately repair breakdowns by consuming the limited amount of spare parts, or to keep spare parts for breakdowns at a later phase.

In the context of continuous function optimization, [24] 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 [12]. The EA process is divided into episodes, the state describes the main properties of the current population (like mean fitness – or f values – standard deviation, etc.), the reward reflects the progress between two episodes (improvement of the best fitness value), while the action consists of determining the vector of control parameters for the next episode.

The trade-off between exploration and exploitation is another issue shared by RL and stochastic local search techniques. In RL an optimal policy is learnt by generating samples, in some cases samples are generated by a policy which is being evaluated and then improved. If the initial policy is generating states only in a limited portion of the entire state space, there will be no way to learn the true optimal policy. A paradigmatic example is the *n*-armed bandit problem [14], so named by analogy to a slot machine. In the problem, one is faced repeatedly with a choice among different options, or actions. After each choice a numerical reward is generated from a stationary probability distribution that depends on the selected action. The objective is to maximize the expected total reward over some time period. In this case the action value function depends only on the action, not on the state (which is unique in this simplified problem). In the more general case of many states, no theoretically sound ways exist to combine exploration and exploitation in an optimal manner. Heuristic mechanisms come to the rescue, for example, by selecting actions not in a greedy manner given an action value function, but based on a simulated-annealing-like probability distribution. Alternatively, actions which are within ε of the optimal value can be chosen with a non-zero probability.

A recent application of RL in the area of stochastic local search is [25]. The noise parameter of the Walksat/SKC algorithm [27] is self-tuned while solving a single problem instance by an average-reward reinforcement learning method: R-learning [26]. The R-learning algorithm learns a state-action value function $Q(s,a)$ estimating the reward for following an action a from a state s. To improve the trade-off between the exploitation of the current state of learning with exploration for further learning, the state-action value function selects with probability $1-\varepsilon$ the action a with the best estimated reward, otherwise it chooses a random action. Once the selected action is executed, a new state s' and a reward r are observed and the average reward estimate and the state-action value function $Q(s,a)$ are updated. In [25], the state of the MDP is represented by the current number of unsatisfied clauses and an action is the selection of a new value for the noise parameter from the set $(0.05, 0.1, 0.15, ..., 0.95, 1.00)$ followed by a local move. The reward is the reduction in the number of unsatisfied clauses since the last local minimum. In this context the average reward measures the average progress towards the solution of the problem, i.e., the average difference in the objective function before and after the local move. The modification performed by the approach in [25] to the Walksat/SKC algorithm consists of (re-)setting its noise parameter at each iteration. This approach is compared over a benchmark of SAT instances with the results obtained when the noise parameter is hand-tuned. Even if the approach obtains uniformly good results on the selected benchmark, the hand-tuned version of the Walksat/SKC algorithm performs better. Furthermore, the proposed approach is not robust with the setting of the parameters of the R-learning algorithm.

The work in [30] applies a reinforcement learning technique, Q-learning [28], in the context of the Constraint Satisfaction Problem (CSP). A CSP instance is usually solved by iteratively selecting one variable and assigning a value from its domain. The value assigned to the selected variable is then propagated, updating the domains of the remaining variables and checking the consistency of the constraints. If a conflict is detected, a backtracking procedure is executed. A complete assignment generating no conflicts is a solution to the CSP instance. The algorithm proposed in [30] learns which variable ordering heuristic should be used at each iteration, formulating the search process executed by the CSP solver as a reinforcement learning task. In particular, each state of the MDP process consists of a partial or total assignment of values to the variables, while the set of the actions is given by the set of the possible variable ordering heuristic functions. A reward is assigned when the CSP instance is solved. In this way, the search space of the RL algorithm corresponds to the search space of the input CSP instance and the transition function of the MDP process corresponds to the decision made when solving a CSP instance. However, several open questions related to the proposed approach remain to be solved, as pointed out by the authors.

Another example of the application of machine learning to the CSP is the Adaptive Constraint Engine introduced in [13], which learns to solve constrain satisfaction problems. The solving process of a CSP is modeled as a sequence of decisions, where each decision either selects a variable or assigns a value to a variable from its associated value set. After each assignment, propagation rules infer its effect on the domains of the unassigned variables. Each decision is obtained by the weighted combination of a set of pre-specified search heuristics. The learning task consists of searching the appropriate set of weights for the heuristics. The Adaptive Constraint Engine can learn the optimal set of weights for *classes* of CSP problems, rather than for a single problem.

3 Reinforcement learning and dynamic programming basics

In this section, *Markov decision processes* are formally defined and the standard dynamic programming technique is summarized in Sec. 3.2, while the policy iteration technique to determine the optimal policy is defined in Sec. 3.3. In many practical cases exact solutions must be abandoned in favor of approximation strategies, which are the focus of Sec. 3.4.

3.1 Markov decision processes

A standard Markov process is given by a set of states $\mathscr{S}$ 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. The process 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 $(\mathscr{S},\mathscr{A},P,R,\gamma)$, where $\mathscr{S}$ is a set of states, $\mathscr{A}$ a finite set of actions, $P(s,a,s')$ is the probability of transition from state $s \in \mathscr{S}$ to state $s' \in \mathscr{S}$ if action $a \in \mathscr{A}$ 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

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

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 $(\mathscr{S}, \mathscr{A}, P, R, \gamma)$, we define a *policy* as a probability distribution $\pi(\cdot|s) : \mathscr{A} \to [0,1]$, 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 $\mathscr{A}$. Note that we are only considering stationary policies. If a policy is deterministic, then we shall resort to the more compact notation $a = \pi(s)$.

3.2 The dynamic programming approach

The learning task consists of the selection of 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 status s, under the hypothesis that the ensuing actions are taken according to policy π. A straightforward implementation of the Bellman principle leads to the following definition:

$$Q^\pi(s,a) = \sum_{s' \in \mathscr{S}} P(s,a,s') \left(R(s,a,s') + \gamma \sum_{a' \in \mathscr{A}} \pi(a'|s') Q^\pi(s',a') \right) \tag{2}$$

where the sum over $\mathscr{S}$ can be interpreted as an integral in the 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 $(s,a) \in \mathscr{S} \times \mathscr{A}$ is

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

so that (2) can be rewritten as

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

or, in a more compact linear form,

$$Q^\pi = R + \gamma P \Pi_\pi Q^\pi \tag{3}$$

where R is the $|\mathscr{S}||\mathscr{A}|$-entry column vector corresponding to $R(s,a)$, P is the $|\mathscr{S}||\mathscr{A}| \times |\mathscr{S}|$ matrix of $P(s,a,s')$ values having (s,a) as row index and s' as column, while Π_π is a $|\mathscr{S}| \times |\mathscr{S}||\mathscr{A}|$ 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

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

or, alternatively, as a fixed-point problem

$$Q^\pi = T_\pi Q^\pi, \tag{5}$$

where $T_\pi : x \mapsto R + \gamma P \Pi_\pi x$ is an affine functional.

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

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.

3.3 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:

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

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 [9] for more details.

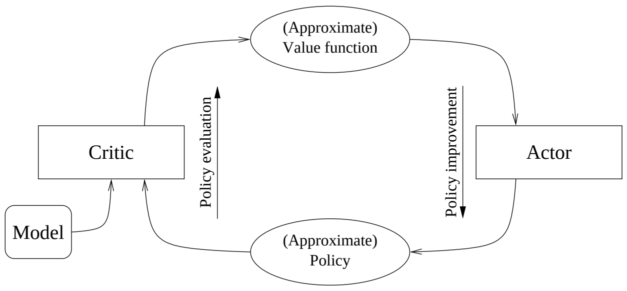

Fig. 1 the Policy Iteration (PI) mechanism

The Policy Iteration (PI) method consists in 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.

3.4 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 $P(\cdot,\cdot,\cdot)$ and $R(\cdot,\cdot)$ by executing a very large series of simulations. The *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 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 $\Phi = (\phi_1,\dots,\phi_k)$. The approximation $\hat{Q}^\pi \approx Q^\pi$ is in the form

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

The weights vector w^π is the solution of the linear system $Aw^\pi = b$, where

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

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:

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

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

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

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

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

Fig. 2 the LSTDQ algorithm

Such approximations lead to the Least Squares Temporal Difference for Q (LSTDQ) algorithm proposed in [22], and shown in Figure 2, where the functions

Variable	Scope	Description
$\mathscr{D}$	In	Set of sample vectors $\{(s,a,r,s')\}$
k	In	Number of basis functions
Φ	In	Vector of k basis functions
γ	In	Discount factor
ε	In	Weight vector tolerance
w_0	In	Initial value function weight vector
w'	Local	Weight vectors in subsequent iterations
w	Out	Optimal weight vector

```
function LSPI ($D$, $k$, $\Phi$, $\gamma$, $\varepsilon$, $w_0$)
    $w' \leftarrow w_0$;
    do
        $w \leftarrow w'$;
        $w' \leftarrow$ LSTDQ ($D$, $k$, $\Phi$, $\gamma$, $w$);
    while $\|w - w'\| > \varepsilon$
```

Fig. 3 the LSPI algorithm

$R(s,a)$ and $P(s,a,s')$ are assumed to be unknown and are replaced by a finite sample set $\mathscr{D}$.

Note that the LSTDQ algorithm returns the weight vector that best approximates the value function of a given policy π, within the spanned subspace and according to the sample data. 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 explicit: if the system is in state s and the current value function is defined by weight vector w, the best action to take is:

$$a = \arg\max_{a \in \mathscr{A}} \Phi(s,a)^T w. \tag{8}$$

The complete LSPI algorithm is given in Fig. 3. Note that, because of the identification between the weight vector 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 w.

4 Reactive SAT/MAX-SAT solvers

This investigation considers as starting point the following methods: the Walksat/SKC algorithm [27] and its reactive version, the Adaptive Walksat [16], the Hamming Reactive Tabu Search (H_RTS) [6] and the Reactive Scaling and Probabilistic smoothing algorithm (RSAPS) [19].

The Walksat/SKC algorithm adopts a two-stage variable selection scheme. First, one of the clauses which are violated by the current assignment is selected uniformly at random. If the selected clause contains variables that can be flipped without violating any other clause, one of these is randomly chosen. When such an improving step does not exist, a random move is executed with a certain probability p: a variable appearing in selected unsatisfied clause is selected uniformly at random and flipped. Otherwise with probability $1 - p$ the least worsening move is greedily selected. The parameter p is often referred to as noise parameter. The Adaptive Walksat algorithm automatically adjusts the noise parameter, increasing/decreasing it when search stagnation is/is not detected. Search stagnation is measured in terms of the time elapsed since the last reduction in the number of unsatisfied clauses has been achieved.

RSAPS is a reactive version of the Scaling and Probabilistic Smoothing (SAPS) [19] algorithm. In the SAPS algorithm, once the search process becomes trapped in a local minimum, the weights of the current unsatisfied clauses are multiplied by a factor bigger than 1 in order to encourage diversification. After updating, with a certain probability P_{smooth} the clause weights are smoothed back towards uniform values. The smoothing phase is to forget the collateral effects of the earlier weighting decisions, that affect the behaviour of the algorithm when visiting future local minima. The RSAPS algorithm dynamically adapts the smoothing probability parameter P_{smooth} during the search. In particular, RSAPS adopts the same stagnation criterion as Adaptive Walksat to trigger a diversification phase. If no reduction in the number of the unsatisfied clauses is observed in the last search iterations, in case of RSAPS the smoothing probability parameter is reduced while in the case of Adaptive Walksat the noise parameter is increased.

H_RTS is a prohibition-based algorithm that dynamically adjusts the prohibition parameter during the search. For the purpose of this investigation, we consider a “simplified” version of H_RTS, where the non-oblivious search phase[1] is omitted. As matter of fact, we are interested in evaluating the performance of different reactive strategies for adjusting the prohibition parameter. To the best of our knowledge, non-oblivious functions are defined only in the case of k-SAT instances. Furthermore, the benchmark used in the experimental part of this work is not limited to k-SAT instances. Finally, there is some evidence that the performance of H_RTS is determined by the reactive tabu search phase rather than the non-oblivious search phase [17]. For the rest of this work, the term H-RTS refers to the “simplified” version of H-RTS. Its pseudo-code is in Fig. 4. Once the initial truth assignment is generated in a random way, the search proceeds by repeating phases of local search followed by phases of tabu search (TS) (lines 6–12 in Fig. 4), until $10\,n$ iterations are accumulated. The variable t, initialized to zero, contains the current iteration and increases after a local move is applied, while t_r contains the iteration when the last random assignment was generated and f represents the score function counting the number of unsatisfied clauses. During each combined phase, first the local optimum X_I of f is reached, then $2(T+1)$ moves of Tabu Search are executed, where T is the prohibition parameter. The design principle underlying this choice is that prohibi-

[1] The non-oblivious search phase is a local search procedure guided by a non-oblivious function, i.e., a function that weights in different ways the satisfied clauses according to the number of matched literals. For example, the non-oblivious function for MAX-2-SAT problems is the weighted linear combination of the number of clauses with one and two matched literals. See [6] for details.

tions are necessary for diversifying the search only after local search (LS) reaches a local optimum. Finally, an "incremental ratio" test is executed to see whether in the last $T+1$ iterations the trajectory tends to move away or come closer to the starting point X_I. The test measures the Hamming distance from X_I covered in the last $T+1$ iterations and a possible reactive modification of T_f is executed depending on the tests results, see [6]. The fractional prohibition T_f (the prohibition T is obtained as $T_f\, n$) is therefore changed during the run to obtain a proper balance of diversification and bias.

The random restart executed after 10 n moves guarantees that the search trajectory is not confined to a localized portion of the search space.

```
procedure H_RTS
1.  repeat
2.      $t_r \leftarrow t$
3.      $X \leftarrow$ random truth assignment
4.      $T \leftarrow \lfloor T_f\, n \rfloor$
5.      repeat
6.          repeat                               { local search }
7.              $X \leftarrow$ BEST-MOVE $(LS, f)$
8.          until largest $\Delta f = 0$
9.          $X_I \leftarrow X$
10.         for $2(T+1)$ iterations   { reactive tabu search }
11.             $X \leftarrow$ BEST-MOVE $(TS, f)$
12.         $X_F \leftarrow X$
13.         $T \leftarrow$ REACT$(T_f, X_F, X_I)$
14.     until $(t - t_r) > 10\, n$
15. until solution is acceptable or maximum number
    of iterations reached
```

Fig. 4 The simplified version of the H_RTS algorithm considered in this work.

5 The RL-based approach for reactive SAT/MAX-SAT solvers

A local search algorithm operates through 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 search states within the same instance, may require different parameter values.

In this work we introduce a generic RL-based approach for adapting during the search the parameter determining the trade-off between intensification and diversification in the three reactive SLS algorithms considered in Sec. 4. From now on, the specific parameter modified by the reactive scheme is referred to as the *target* parameter. The target parameter is the noise parameter in the case of the Walksat algorithm, the prohibition parameter in the case of Tabu search and the smoothing probability parameter in the case of the SAPS algorithm.

Furthermore, in this paper the term *offline* defines an action executed before the search phase of the local search algorithm, while the term *online* describes an action executed during the search phase. This work proposes a RSO approach for the online tuning of the target parameter based on the LSPI algorithm, which is trained offline.

In order to apply the LSPI method introduced in Sec. 3.4 for tuning the target parameter, first the search process of the SLS algorithms is modelled as a Markov decision process. Each state of the MDP is created by observing the behavior of the considered algorithm over an epoch of $epoch\_length$ consecutive variable flips. In fact, the effect of changing the target parameter on the algorithm's behavior can only be evaluated after a reasonable number of local moves. Therefore the algorithms traces are divided into *epochs* $(E_1, E_2, \dots)$ composed of a suitable number of local moves, and changes of the target parameter are allowed only between epochs. Given the subdivision of the reactive local search algorithm's trace into a sequence of epochs $(E_1, E_2, \dots)$, the state at the end of epoch E_i is 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.

In the case of the Reactive Tabu search algorithm, $epoch\_length = 2 * T_{max}$, where T_{max} is the maximum allowed value for the prohibition parameter. Because 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. A preliminary application of RL to Reactive Tabu Search for SAT has been presented in [5].

In the case of the target parameter in the Walksat and the RSAPS algorithms, each epoch lasts 100 and 200 variable flips, respectively, including null flips (i.e., search steps where no variable is flipped) in the case of the RSAPS algorithm. These values for the epoch length are the optimal values selected during the experiments over a set of candidates ranging from the value 10 to the value 400.

The scope of this work is the design of a reactive reinforcement-based method that is independent of the specific SAT/MAX-SAT reactive algorithm considered. Therefore, the features selected for the definition of the states of the MDP underlying the reinforcement learning approach do not depend on the target parameter considered, allowing for generalization.

As specified above, the state of the system at the end of an epoch describes the algorithm's behavior during the epoch, and an "action" is the modification of the target parameter before the algorithm enters the next epoch. The target parameter is responsible for the diversification of the search process once stagnation is detected. The state features therefore describe the intensification-diversification trade-off observed in the epoch.

In particular, let us define the following:

- n and m are the number of variables and clauses of the input SAT instance, respectively;

- $f(x)$ is the score function counting the number of unsatisfied clauses in the truth assignment x;
- x_{bsf} is the "best-so-far" (BSF) configuration *before* the current epoch;
- $\overline{f}_{\text{epoch}}$ is the average value of f during the current epoch;
- $\overline{H}_{\text{epoch}}$ is the average Hamming distance during the current epoch from the configuration at the beginning of the current epoch.

The compact state representation chosen to describe an epoch is the following couple:

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

The first component of the state is the mean change of f in the current epoch with respect to the best value. It represents the preference for configurations with low objective function values. The second component describes the ability to explore new configurations in the search space by moving away from local minima. In the literature, this behaviour is often referred to as the diversification-bias trade-off. For the purpose of addressing uniformly SAT instances with different number of variables, the first and the second state components have been normalized.

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: $(f(x_{\text{bsf}}) - f(x_{\text{localBest}}))/m$.

The state of the system at the end of an epoch describes the algorithm's behavior during the last epoch, while an action is the modification of the algorithm's parameters before it enters the next epoch. In particular, the action consists of setting the target parameter from scratch at the end of the epoch. The noise and the smoothing probability parameter are set to one of 20 uniformly distributed values in the range $[0.01, 0.2]$. In the Tabu search context, we consider the fractional prohibition parameter (the prohibition parameter is $T = \lfloor nT_{\text{f}} \rfloor$), equal to one of 25 uniformly distributed values in the range $[0.01, 0.25]$. Therefore the actions set $A = \{a_i, i = 1..n\}$ is composed of $n = 20$ choices in the case of the noise and the smoothing probability parameters and $n = 25$ choices in the case of the prohibition parameter. The effect of the action a_i consists of setting target parameter to $0.01 * i$, $i \in [1, 20]$ in the case of the noise and the smoothing probability parameters and $i \in [1, 25]$ in the case of the prohibition parameter.

Once a reactive local search algorithm is modeled as a Markov decision process, a reinforcement learning method such as LSPI can be used to control the evolution of its parameters. To tackle large state and action spaces, the LSPI algorithm approximates the value function as a linear weighted combination of basis functions (see Eq. 8). In this work, we consider the value function space spanned by the following finite set of basis functions:

$$\Phi(s,a) = \begin{pmatrix} I_{a==a_1}(a) \\ I_{a==a_1}(a) \cdot \Delta f \\ I_{a==a_1}(a) \cdot \overline{H}_{\text{epoch}} \\ I_{a==a_1}(a) \cdot \overline{H}_{\text{epoch}} \cdot \Delta f \\ I_{a==a_1}(a) \cdot (\Delta f)^2 \\ I_{a==a_1}(a) \cdot \overline{H}^2_{\text{epoch}} \\ \vdots \\ I_{a==a_n}(a) \\ I_{a==a_n}(a) \cdot \Delta f \\ I_{a==a_n}(a) \cdot \overline{H}_{\text{epoch}} \\ I_{a==a_n}(a) \cdot \overline{H}_{\text{epoch}} \cdot \Delta f \\ I_{a==a_n}(a) \cdot (\Delta f)^2 \\ I_{a==a_n}(a) \cdot \overline{H}^2_{\text{epoch}} \end{pmatrix} \tag{9}$$

The set is composed of $6*n$ elements and $I_{a==a_i}$, with $i = 1..n$, is the indicator function for the actions, evaluating to 1 if the action is the indicated one, 0 otherwise. The indicator function is used to discern the “state-action” features for the different actions considered: the learning for an action is entirely decoupled from the learning for the remaining actions. For example, consider the case of $n = 20$ actions. The vector $\Phi(s,a)$ has 120 elements and for each of the 20 possible actions only 6 different elements are not zero.

The adoption of LSPI for reactive search optimization requires a training phase that identifies the optimal policy for the tuning of the parameter(s) of the considered SLS algorithm. During the training phase, a given number of runs of the SLS algorithm are executed over a subset of instances from the considered benchmark. The initial value for the target parameter is selected uniformly at random over the set of the allowed values. Each run is divided into epochs and the target parameter value is modified only at the end of each epoch using a random policy. An example (s,a,r,s') is composed of the states s and s' of the Markov process observed at the end of two consecutive epochs, the action a modifying the target parameter at the end of the first epoch and the reward r observed for the action. The collected examples are used by the LSPI algorithm to identify the optimal policy (see Fig. 3). In this work, the training phase is executed off-line.

During the testing phase, the state observed at the end of the current epoch is given as input to the LSPI algorithm. The learnt policy determines the appropriate modification of the target parameter (see Eq. 8). For the testing phase, the difference between the RL-based and the original version of the considered SLS algorithm consists only of the modification of the target parameter at the end of each epoch.

6 Experimental results

To measure the performance of our Reinforcement-based approach, we implemented C++ functions for the Walksat/SKC, RSAPS and Reactive Tabu search methods described in Sec. 4 and interfaced them to the Matlab LSPI implementation by Lagoudakis and Parr, available at http://www.cs.duke.edu/research/AI/LSPI/ (as of Dec 15, 2009).

The experiments performed in this work were carried out using a 2GHz Intel Xeon processor machine, with 6GB RAM. The efficient UBCSAT [29] implementation of the Walksat and (R)SAPS algorithms is considered, while for the H_RTS algorithm the original code of its authors is executed.

While in this paper we base our comparisons on the solution quality after a given number of iterations, the CPU time required by RTS_RL, Walksat_RL and RSAPS_RL is analogous to that of the basic H_RTS, Walksat and SAPS algorithms, respectively. The only negligible CPU time overhead is due to the computation of 20 floating-point 120-element vector products in order to compute $\hat{Q}(s,a)$ (see Eq. 8) for the 20 actions at the end of each epoch.

For brevity, the applications of our Reinforcement-based approach to the tuning of the noise parameter, the prohibition value and the smoothing probability parameter are termed “Walksat_RL”, “RTS_RL” and “RSAPS_RL”, respectively.

We consider two kinds of instances: random MAX-3-SAT instances and instances derived from relevant industrial problems.

For the training phase, we selected one instance from the considered benchmark and performed four runs of the algorithm with different randomly chosen starting truth assignments. We created 8000 examples for the Walksat_RL and RSAPS_RL algorithms and 4000 examples for the RTS_RL method. The instance selected for the training is not included in the set of instances used for the testing phase.

6.1 Experiments on difficult random instances

The experiments performed are over selected MAX-3-SAT random instances defined in [23]. Each instance has 2000 variables and 8400 clauses, thus lying in the satisfiability threshold for 3-SAT instances.

The comparison of our approach based on reinforcement learning w.r.t. the original Walksat/SKC, RTS and RSAPS algorithms is in Table 1. Each entry in the table contains the mean and the standard error of the mean over 10 runs of the best-so-far number of unsatisfied clauses at iteration 210000.

The Walksat/SKC algorithm has been executed with the default 0.5 value for the noise parameter. The setting for the SAPS parameters is the default one described in [19], as empirical results in [19] show it is the most robust configuration.

Table 1 shows the superior performance of the RSAPS algorithm compared to the SAPS algorithm, motivating the need of the reactive parameters tuning over this benchmark. Over all the benchmark instances, the RSAPS algorithm exhibits a lower best-so-far number of unsatisfied clauses at iteration 210000 compared to the SAPS algorithm. The Kolmogorov-Smirnov test for two independent samples indicates that there is statistical evidence (with a confidence level of 95%) for the superior performance of RSAPS compared to SAPS (except in the case of the in-

Instance	H_RTS	RTS_RL	Walksat	Walksat_RL	SAPS	RSAPS	SAPS_RL
sel_01.cnf	17.7 (.349)	8.6 (.142)	4.8 (.308)	7.3 (.163)	21.6 (.347)	16.2 (.329)	3.9 (.268)
sel_02.cnf	12.2 (.454)	6.1 (.179)	3.1 (.087)	4.5 (.190)	21.7 (.343)	12.3 (.235)	1.1 (.112)
sel_03.cnf	16.5 (.427)	7.3 (.194)	7.5 (.347)	8.6 (.320)	26.0 (.294)	18.2 (.364)	4.5 (.195)
sel_04.cnf	12.9 (.351)	7.6 (.183)	5.6 (.134)	8.0 (.298)	20.7 (.388)	16.2 (.304)	3.1 (.172)
sel_05.cnf	17.5 (.279)	8.0 (.169)	7.4 (.206)	8.6 (.271)	28.4 (.313)	20.2 (.244)	6.9 (.317)
sel_06.cnf	22.4 (.347)	8.3 (.262)	7.0 (.418)	10.2 (.274)	29.4 (.275)	21.4 (.291)	6.7 (.249)
sel_07.cnf	16.8 (.507)	7.7 (.200)	6.0 (.182)	8.2 (.345)	24.5 (.535)	18.2 (.198)	4.4 (.236)
sel_08.cnf	14.6 (.283)	6.7 (.216)	4.5 (.177)	8.6 (.356)	21.9 (.260)	16.6 (.231)	3.9 (.166)
sel_09.cnf	15.9 (.462)	9.0 (.274)	6.1 (.237)	8.6 (.291)	23.9 (.341)	17.2 (.322)	6.3 (.249)
sel_10.cnf	15.5 (.330)	8.5 (.263)	6.2 (.385)	6.4 (.189)	23.0 (.298)	16.2 (.161)	3.8 (.139)

Table 1 A comparison of the mean BSF values of selected SAT/Max-SAT solvers. The values in the table are the mean and the standard error of the mean over 10 runs of the best-so-far number of unsatisfied clauses at iteration 210000.

stance sel_04.cnf). In the case of sel_04.cnf instance the Kolmogorov-Smirnov test accepts the null hypothesis , while, e.g., in the case of instance sel_03.cnf it rejects the null hypothesis.

Over the considered benchmark, our RL-based approach improves the performance of the underlying algorithm, when considering RSAPS and H_RTS.

In the case of the smoothing probability parameter, for each benchmark instance the RL-based algorithm shows a lower number of unsatisfied clauses in the best-so-far as of iteration 210000 compared to to the original RSAPS algorithm. For example, over the instance sel_07.cnf, RSAPS_RL on average reaches 4.4 unsatisfied clauses, while the original RSAPS algorithm 18.2 unsatisfied clauses. The Kolmogorov-Smirnov test for two independent samples confirms this observation. Figure 5 shows the evolution of the value $\|\mathbf{w}-\mathbf{w}'\|$ during the training phase of the LSPI algorithm in the case of RSAPS_RL. The LSPI algorithm converges in 7 iterations, when the value $\|\mathbf{w}-\mathbf{w}'\|$ becomes smaller than 10^{-6}.

In the case of the RL-based tuning of the prohibition parameter, the RTS_RL algorithm outperforms H_RTS over all the benchmark instances. For example, consider the 7.3. unsatisfied clauses reached on average by the RTS_RL algorithm compared to the 16.5 clauses of the H_RTS algorithm over the instance sel_03.cnf. Again, the Kolmogorov-Smirnov test for two independent samples rejects the null hypothesis at the standard significance level of 0.05 for all the instances of the considered benchmark.

However, over the MAX-3-SAT benchmark considered, no improvement is observed for the RL-based reactive tuning of the noise parameter of the Walksat/SKC algorithm. The Kolmogorov-Smirnov test does not show statistical evidence (with a confidence level of 95%) for the different performance of the Walksat_RL compared to the Walksat/SKC, except in the case of the instances 1 and 4 where the latter algorithm reaches on average a lower number of unsatisfied clauses compared to the former one. In the case of the instances 1 and 4, the Kolmogorov-Smirnov test rejects the null hypothesis.

Note that during the offline learning phase of the RL-based version of Walksat, the LSPI algorithm does not converge even after 20 iterations over the set of ex-

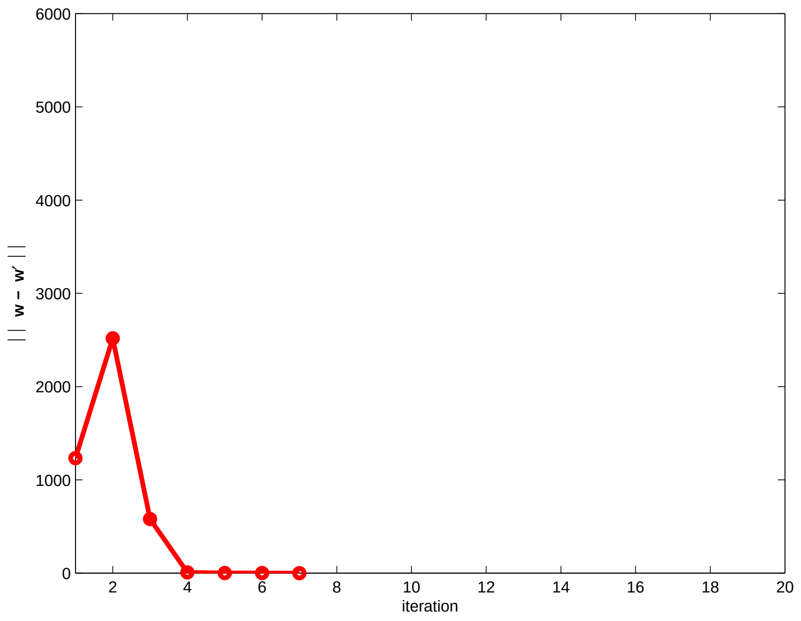

Fig. 5 Training of the LSPI algorithm over the MAX-3-SAT random instances considered in the case of RSAPS_RL.

amples extracted from the considered benchmark (even if 64000 instead of 8000 examples are generated).

6.2 Experiments on structured industrial instances

We repeat the experiment over a benchmark of MAX-SAT industrial instances (Table 2). Again, note first the superior performance of the RSAPS algorithm compared to the SAPS algorithm (the Kolmogorov-Smirnov test indicates that there is statistical evidence of different performance, except in the case of the instance en_6_nd.cnf). This observation motivates the selection of this benchmark to test the smarter reactive approach based on RL.

Instance	**H_RTS**	**RTS_RL**	**Walksat**	**Walksat_RL**	**SAPS**	**RSAPS**	**SAPS_RL**
en_6.cnf	1073.7 (.006)	1034.3 (.542)	67.0 (.875)	81.9 (.369)	26.3 (.563)	11.8 (.742)	33.5 (.256)
en_6_case1.cnf	1091.3 (.776)	1059.5 (.112)	65.6 (.228)	92.9 (.299)	25.5 (.343)	6.0 (.222)	25.8 (.997)
en_6_nd.cnf	1087.9 (.242)	1050.7 (.117)	75.6 (.358)	99.0 (.709)	27.8 (.597)	21.7 (.964)	28.3 (.562)
en_6_nd_case1.cnf	1075.9 (.469)	1063.9 (.343)	68.9 (.563)	83.1 (.785)	20.0 (.878)	5.0 (.728)	55.9 (.930)

Table 2 A comparison of the mean BSF values of selected SAT/Max-SAT solvers over a benchmark of big industrial unsat instances. The values in the table are the mean and the standard error of the mean over 10 runs of the best-so-far number of unsatisfied clauses at iteration 210000.

Over this benchmark, the RL-based reactive approach does not show appreciable improvement.

In the case of the prohibition parameter, the Kolmogorov-Smirnov test for two independent samples indicates that the difference among the results of the RTS_RL approach compared to the H_RTS approach is not statistically significant (at the standard significance level of 0.05) in the case of instances en_6_nd.cnf and en_6_nd_case1.cnf. However, over the instances en_6.cnf and en_6_case1.cnf, RTS_RL shows superior performance compared to H_RTS. E.g., in the case of the instance en_6_nd.cnf the Kolmogorov-Smirnov test accepts the null hypothesis since the *p*-value is 0.11 while in the case of the instance en_6.cnf it rejects the null hypothesis since the *p*-value is 0.031.

In the case of the smoothing probability and the noise parameters, the RL-based approach shows poor performance even compared to the non-reactive versions of the original methods considered (the Walksat and the SAPS algorithms). The Kolmogorov-Smirnov test do not reject the null hypothesis comparing the results of SAPS compared to RSAPS_RL algorithm (except in the case of instance en_6_nd_case1.cnf where SAPS outperforms RSAPS_RL). Furthermore, the statistical test confirms the worse results of Walksat_RL compared to the Walksat algorithm (except in the case of the instance en_6_nd_case1.cnf where there is no statistical evidence of different performance).

Note that during the offline learning phase of the RL-based version of Walksat and RSAPS, the LSPI algorithm does not converge even after 20 iterations over the set of examples extracted from the considered benchmark (even if 64000 instead of 8000 examples are generated). Figure 6 shows the evolution of the value $\|\mathbf{w}-\mathbf{w}'\|$ during the training phase of the LSPI algorithm in the case of Walksat_RL. The algorithm does not converge: from iteration 12, the L2-norm of the vector $\mathbf{w}-\mathbf{w}'$ oscillates from the value 4150 to the value 5700.

Furthermore, qualitatively different results w.r.t. Tab. 2 are not observed if a different instance is used for training, or if the training set includes two instances rather than one.

Table 2 shows a poor performance of prohibition-based reactive algorithms. To understand possible causes for this lack of performance we analyzed the evolution of the prohibition parameter during the run. Figure 7 depicts the evolution of the fractional prohibition parameter T_f during a run of the H_RTS algorithm over two instances of the Engine benchmark.

The fact that the maximum prohibition value is reached suggests that the prohibition mechanism is not sufficient to diversify the search in these more structured instances. The performance of H_RTS is not significantly improved even when smaller values for the maximum prohibition value are considered.

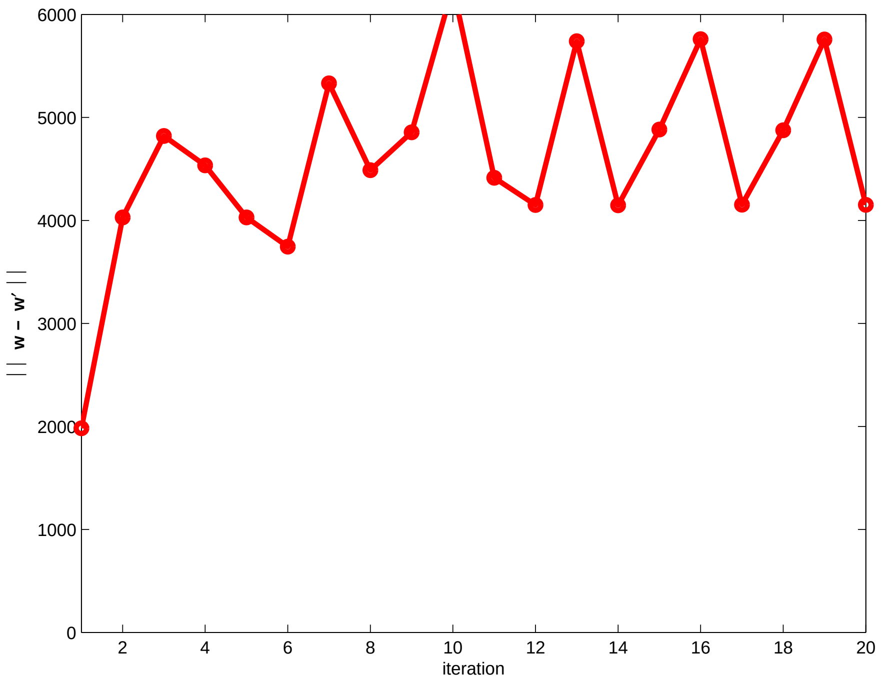

Fig. 6 Training of the LSPI algorithm over the industrial instances considered in the case of Walksat_RL.

6.3 A comparison with offline parameter tuning

The experimental results show that over the MAX-3-SAT benchmark the RL-based approach outperforms the *ad hoc* reactive approaches in the case of the RSAPS and H_RTS algorithms. The values in Table 1 and 2 are obtained by using the original default settings of the algorithms parameters. In particular, the default configuration for SAPS parameters is the following:

- the scaling parameter is set to 1.3;
- the smoothing parameter is set to 0.8;
- the smoothing probability parameter is set to 0.05;
- the noise parameter is set to 0.01.

This configuration is also the original default setting of RSAPS, where the reactive mechanism adapts the smoothing probability parameter during the search history. The initial default value for the fractional prohibition parameter in H_RTS is 0.1. The reactive mechanism implemented in H_RTS is responsible for the adaptation of the prohibition parameter during the algorithm execution.

For many domains, a significant improvement in the performance of the SAT solvers can be achieved by setting their parameters differently w.r.t the original de-

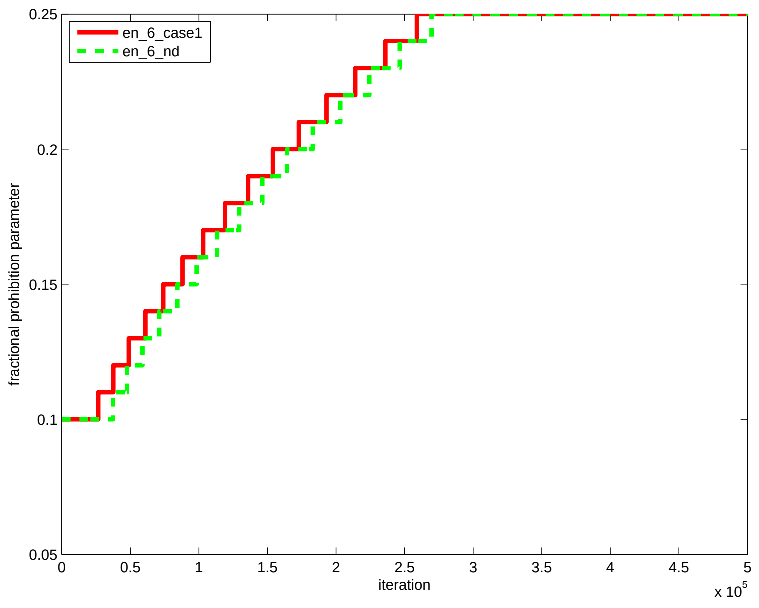

Fig. 7 Evolution of the T_f parameter during a run of the H_RTS algorithm over two selected instances of the Engine benchmark.

fault configuration. The tuning is executed offline, and the determined parameters are then fixed while the algorithm runs on new instances.

In order to measure the beneficial effect of the RL-based reactive mechanism observed for the smoothing probability parameter and the prohibition over MAX-3-SAT benchmark, we compare RTS_RL and RSAPS_RL with the best fixed parameter tuning of SAPS and H_RTS algorithm for those instances. The best fixed configuration for SAPS and H_RTS parameters are obtained by ParamILS [18], an automatic tool for the parameter tuning problem. In particular, the best fixed setting for SAPS parameters:

- the scaling parameter is set to 1.256;
- the smoothing parameter is set to 0.5;
- the smoothing probability parameter is set to 0.1;
- the noise parameter is set to 0.05.

The best fixed value for the fractional prohibition parameter over the MAX-3-SAT benchmark is 0.13. Let us note that, by fixing the prohibition parameter in H_RTS, one obtains the standard prohibition-based approach known as GSAT/tabu. Table 3 shows the results of the best fixed parameters setting approach.

The Kolmogorov-Smirnov test shows that there is no statistical evidence of different performance between SAPS_RL and SAPS$_{best}$, except in the case of instance

Instance	**SAPS$_{best}$**	**SAPS_RL**	**GSAT/tabu$_{best}$**	**RTS_RL**
sel_01.cnf	2.9 (.099)	3.9 (.268)	5.8 (.187)	8.6 (.142)
sel_02.cnf	2.1 (.099)	1.1 (.112)	4.0 (.163)	6.1 (.179)
sel_03.cnf	2.8 (.103)	4.5 (.195)	5.9 (.119)	7.3 (.194)
sel_04.cnf	2.7 (.094)	3.1 (.172)	6.2 (.161)	7.6 (.183)
sel_05.cnf	3.6 (.084)	6.9 (.317)	7.3 (.094)	8.0 (.169)
sel_06.cnf	4.8 (.261)	6.7 (.249)	7.2 (.122)	8.3 (.262)
sel_07.cnf	2.9 (.110)	4.4 (.236)	6.5 (.108)	7.7 (.200)
sel_08.cnf	3.5 (.117)	3.9 (.166)	5.9 (.179)	6.7 (.216)
sel_09.cnf	4.2 (.220)	6.3 (.249)	6.7 (.115)	9.0 (.274)
sel_10.cnf	2.5 (.135)	3.8 (.139)	5.1 (.137)	8.5 (.263)

Table 3 Best fixed tuning of SAPS and and GSAT/tabu algorithms. The values in the table are the mean and the standard error of the mean over 10 runs of the BSF value observed at iteration 210000.

sel_5.cnf where the latter algorithm outperforms the former one (the *p*-value is 0.030). Therefore, over the MAX-3-SAT benchmark SAPS_RL is competitive with *any* fixed setting of the smoothing probability parameter.

In the case of the prohibition parameter, a better performance of the GSAT/tabu$_{best}$ algorithm compared to RTS_RL is observed over three instances (sel_1.cnf, sel_9.cnf and sel_10.cnf) where the Kolmogorov-Smirnov test for two independent samples rejects the null hypothesis at the standard significance level of 0.05. Over the remaining instances, the performance of RTS_RL is competitive with the performance of GSAT/tabu$_{best}$. E.g., in the case of the instance sel_8.cnf the Kolmogorov-Smirnov test accepts the null hypothesis since the *p*-value is 0.974, while in the case of the instance sel_1.cnf it rejects the null hypothesis since the *p*-value is 0.006.

6.4 Interpretation of the learnt policy

For RTS_RL, Walksat_RL and SAPS_RL, the LSPI algorithm has been applied to the training sample set, and with (9) as approximate space basis. Fig. 8 and 9, 10 and 11 show the policy learnt when executing RTS_RL and Walksat_RL over the MAX-3-SAT benchmark, respectively.

Each point in the graph represents the action determined by Eq. 9 for the state $s \equiv \left(\Delta f, \frac{\overline{H}_{\text{epoch}}}{n}\right)$ of the Markov decision process. On the x and y-axis, the features of the state are represented. In particular, the x-axis shows the first component on the MDP state $\frac{\overline{H}_{\text{epoch}}}{n}$, which represents the normalized mean Hamming distance during the current epoch from the configuration at the beginning of the current epoch itself (see Sec. 5). The y-axis refers to Δf, the second component of the state, representing the mean change of f in the current epoch with respect to the best value. The range for the x and y-axis are loose bounds on the maximum and minimum values for the $\frac{\overline{H}_{\text{epoch}}}{n}$ and Δf features observed during the offline training phase. The z-axis contains the values for the action.

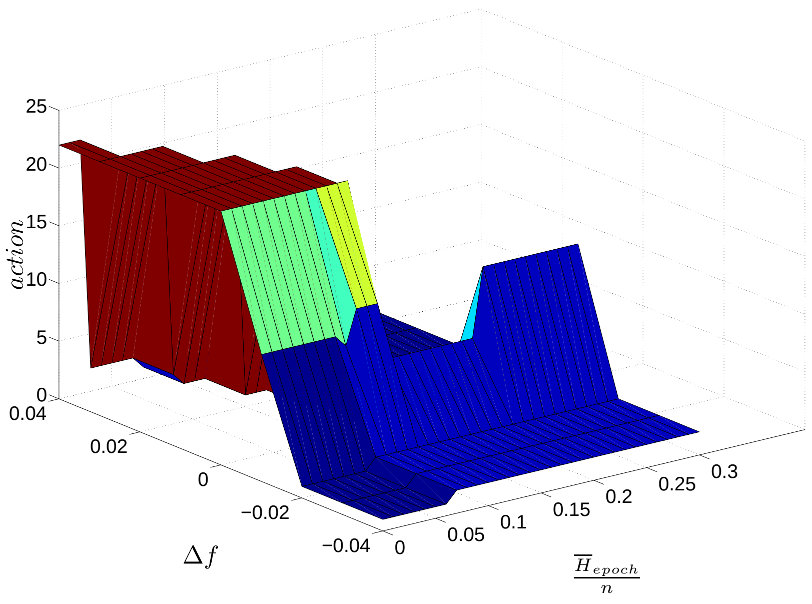

Fig. 8 Distribution of the actions in the significant portion of the state space for the RTS_RL algorithm over the Selman benchmark.

In the case of RTS_RL, the LSPI algorithm converges and the distribution of the actions over the state space is consistent with the intuition. A high value for the prohibition parameter is suggested in cases where the mean Hamming distance between the configurations explored in the last epoch and the configuration at the beginning of the epoch does not exceed a certain value, provided that the current portion of landscape is 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. Furthermore, small values for the prohibition parameter are suggested when the current portion of landscape is better than the previously explored regions: the algorithm is currently investigating a promising region of the search space. These observations, consistent with the intuition, confirm the positive behavior of RTS_RL over the Selman benchmark, improving the performance of the H_RTS algorithm.

For Walksat_RL over the MAX-3-SAT instances, the LSPI algorithm does not converge. The distribution of the best action obtained after 20 iterations of the LSPI algorithm is in Fig. 10 and Fig. 11. A very similar diversification level is suggested when the current portion of landscape is both better and worse than the previously explored regions. In particular, a counter-intuitive diversification action is suggested

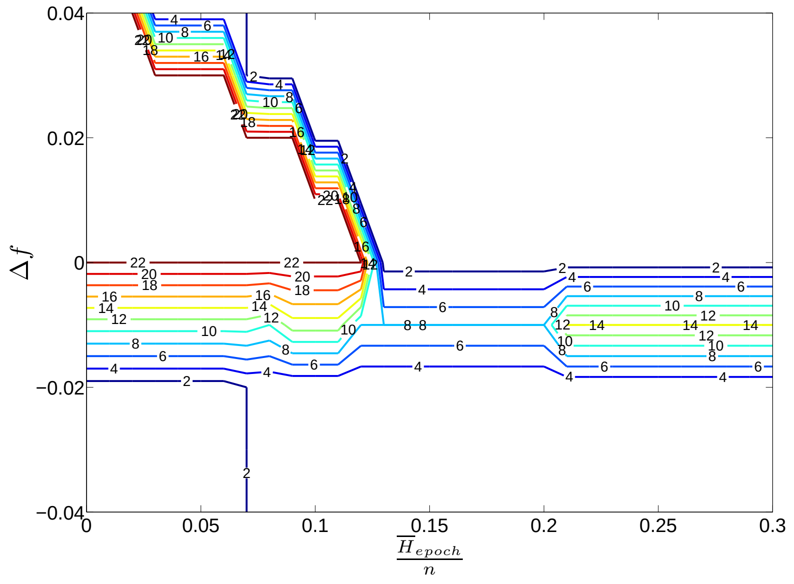

Fig. 9 Distribution of the actions in the significant portion of the state space for the RTS_RL algorithm over the Selman benchmark (contour lines).

when the Δf feature assumes negative values while high values are observed for the $\frac{\overline{H}_{\text{epoch}}}{n}$ feature. These observations confirm the poor results of the Walksat_RL algorithm, performing worse even than the non-reactive Walksat algorithm.

7 Conclusion

This paper describes an application of reinforcement learning for Reactive Search Optimization. In particular, the LSPI algorithm is applied for the online tuning of the prohibition value in H_RTS, the noise parameter in Walksat and the smoothing probability in the SAPS algorithm.

On one side, the experimental results are promising: over the MAX-3-SAT benchmark considered, RTS_RL and SAPS_RL outperform the H_RTS and the RSAPS algorithms, respectively. On the other side, this appreciable improvement is not observed over the structured instances benchmark. Apparently, the wider differences among the structured instances render the adaptation scheme obtained on some instances inappropriate and inefficient on the instances used for testing.

Some more weaknesses of the proposed reinforcement learning approach were observed during the experiments: the LSPI algorithm does not converge over both the benchmarks considered in the case of Walksat_RL and over the structured in-

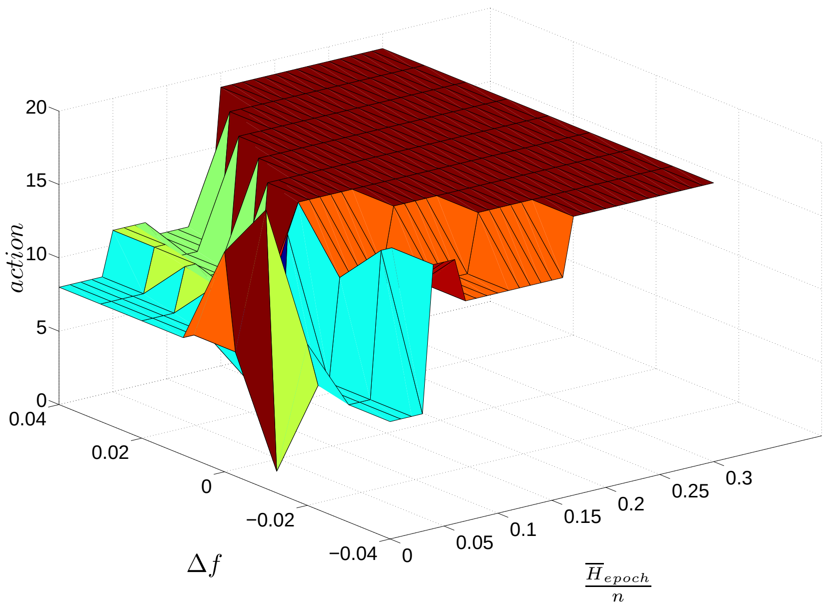

Fig. 10 Distribution of the actions in the significant portion of the state space for the Walksat_RL algorithm over the Selman benchmark.

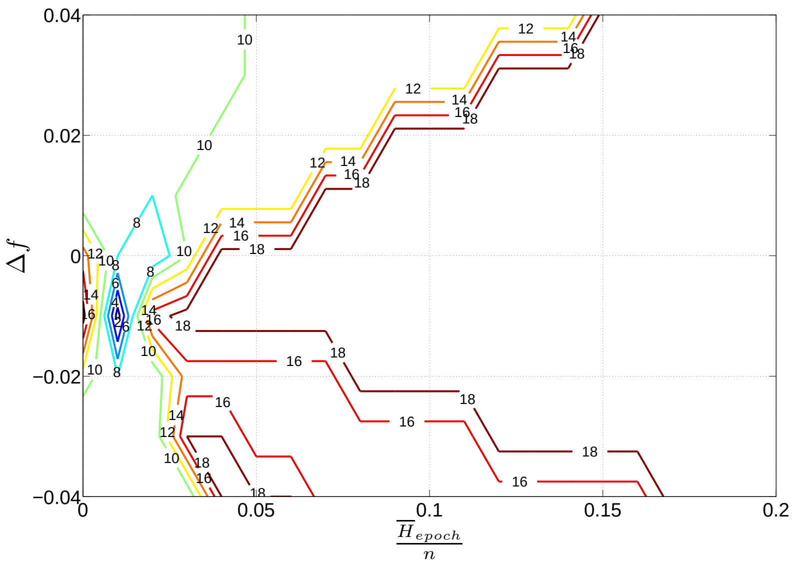

Fig. 11 Distribution of the actions in the significant portion of the state space for the Walksat_RL algorithm over the Selman benchmark (contour lines).

stances benchmark in the case of SAPS_RL. Furthermore, the interval size and the discretization interval of the target parameter in the case of Walksat_RL and SAPS_RL have to be hand-tuned.

Finally, the diversification-bias trade-off determining the performance of SLS algorithms is encountered also during the exploration of the states of the Markov decision process performed by any RL schema.

The application of RL in the context of Reactive Search Optimization is far from trivial and the number of research problems generated appears in fact to be larger than the number of problems it solves. This is positive if considered from the point of view of the future of this RSO and Autonomous Search fields: the less one can adapt techniques from other areas, the larger the motivation to develop novel and independent methods. Our preliminary results seem to imply that more direct techniques specifically designed for the new RSO context will show a much greater effectiveness than techniques inherited from the RL context.

References

1. 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)
2. Battiti, R.: Machine learning methods for parameter tuning in heuristics. In: 5th DIMACS Challenge Workshop: Experimental Methodology Day, Rutgers University (1996)
3. Battiti, R., Brunato, M.: Reactive search: machine learning for memory-based heuristics. In: T.F. Gonzalez (ed.) Approximation Algorithms and Metaheuristics, chap. 21, pp. 21–1 – 21–17. Taylor and Francis Books (CRC Press), Washington, DC (2007)
4. Battiti, R., Brunato, M., Mascia, F.: Reactive Search and Intelligent Optimization, *Operations research/Computer Science Interfaces*, vol. 45. Springer Verlag (2008)
5. Battiti, R., Campigotto, P.: Reinforcement Learning and Reactive Search: an adaptive MAX-SAT solver. In: Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence, pp. 909–910. IOS Press (2008)
6. 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/
7. Battiti, R., Tecchiolli, G.: The reactive tabu search. ORSA Journal on Computing **6**(2), 126–140 (1994)
8. Bennett, K., Parrado-Hernández, E.: The Interplay of Optimization and Machine Learning Research. The Journal of Machine Learning Research **7**, 1265–1281 (2006)
9. Bertsekas, D., Tsitsiklis, J.: Neuro-Dynamic Programming. Athena Scientific (1996)
10. Boyan, J.A., Moore, A.W.: Learning evaluation functions for global optimization and boolean satisfability. In: A. Press (ed.) In Proc. of 15th National Conf. on Artificial Intelligence (AAAI), pp. 3–10 (1998)
11. Brunato, M., Battiti, R., Pasupuleti, S.: A memory-based rash optimizer. In: A.F.R.H.H. Geffner (ed.) Proceedings of AAAI-06 workshop on Heuristic Search, Memory Based Heuristics and Their applications, pp. 45–51. Boston, Mass. (2006). ISBN 978-1-57735-290-7
12. Eiben, A., Horvath, M., Kowalczyk, W., Schut, M.: Reinforcement learning for online control of evolutionary algorithms. In: Brueckner, Hassas, Jelasity, Y. (eds.) (eds.) Proceedings of the 4th International Workshop on Engineering Self-Organizing Applications (ESOA'06), LNAI. Springer Verlag (2006). To appear
13. Epstein, S.L., Freuder, E.C., Wallace, R.J.: Learning to support constraint programmers. Computational Intelligence **21**(4), 336–371 (2005)

14. Fong, P.W.L.: A quantitative study of hypothesis selection. In: International Conference on Machine Learning, pp. 226–234 (1995). URL citeseer.ist.psu.edu/fong95quantitative.html
15. Hamadi, Y., Monfroy, E., Saubion, F.: What is autonomous search? Tech. Rep. MSR-TR-2008-80, Microsoft Research (2008)
16. Hoos, H.: An adaptive noise mechanism for WalkSAT. In: Proceedings of the national conference on artificial intelligence, vol. 18, pp. 655–660. AAAI Press; MIT Press (1999)
17. Hoos, H., Stuetzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
18. Hutter, F., Hoos, H.H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proc. of the Twenty-Second Conference on Artifical Intelligence (AAAI '07), pp. 1152–1157 (2007)
19. Hutter, F., Tompkins, D., Hoos, H.: Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In: Proc. Principles and Practice of Constraint Programming - CP 2002, Ithaca, NY, Sept 2002, Springer LNCS, pp. 233–248 (2002)
20. Lagoudakis, M., Littman, M.: Algorithm selection using reinforcement learning. Proceedings of the Seventeenth International Conference on Machine Learning pp. 511–518 (2000)
21. Lagoudakis, M., Littman, M.: Learning to select branching rules in the DPLL procedure for satisfiability. LICS 2001 Workshop on Theory and Applications of Satisfiability Testing (SAT 2001) (2001)
22. Lagoudakis, M., Parr, R.: Least-Squares Policy Iteration. Journal of Machine Learning Research **4**(6), 1107–1149 (2004)
23. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distributions of SAT problems. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), pp. 459–465. San Jose, Ca (1992)
24. Muller, S., Schraudolph, N., Koumoutsakos, P.: Step size adaptation in evolution strategies using reinforcementlearning. Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC'02. **1**, 151–156 (2002)
25. Prestwich, S.: Tuning local search by average-reward reinforcement learning. In: Proceedings of the 2nd Learning and Intelligent OptimizatioN Conference (LION II), Trento, Italy, Dec 10-12, 2007, Lecture Notes in Computer Science. Springer (2008)
26. Schwartz, A.: A reinforcement learning method for maximizing undiscounted rewards. In: ICML, pp. 298–305 (1993)
27. 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)
28. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
29. Tompkins, D.: Ubcsat (as of Oct 1, 2008). Http://www.satlib.org/ubcsat/#introduction
30. Y. Xu D. Stern, H.S.: Learning adaptation to solve constraint satisfaction problems. In: Proceedings of the 3rd Learning and Intelligent OptimizatioN Conference (LION III), Trento, Italy, Jan 14-18, 2009, Lecture Notes in Computer Science. Springer, in press (2009)
31. Zhang, W., Dietterich, T.: A reinforcement learning approach to job-shop scheduling. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence **1114** (1995)
32. 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)