Printed in: V. J. Rayward-Smith, I.H. Osman, C. R. Reeves and G. D. Smith, editors, Modern Heuristic Search Methods, chapter 4, pages 61–83. John Wiley and Sons Ltd, 1996.

Reactive Search: Toward Self-Tuning Heuristics

Roberto Battiti
Dip. di Matematica, Università di Trento
38050 Povo (Trento) - Italy
Email: battiti@science.unitn.it
WWW: **http://rtm.science.unitn.it/**

Abstract

Local (neighborhood) search can be guided to go beyond local minima through meta-heuristic methods that use the information obtained in the previous part of the run. The Reactive Search (RS) method proposes the integration of simple history-based feedback schemes in local search for the on-line determination of free parameters, therefore endowing the algorithm with the flexibility needed to cover a wide spectrum of problems but avoiding a human trial-and-error adjustment. In particular, in the Reactive Tabu Search (RTS) algorithm a simple feedback scheme determines the value of the prohibition parameter in Tabu Search, so that a balance of exploration versus exploitation is obtained that is appropriate for the local characteristics of the task. This paper summarizes the Reactive Search framework, reviews RTS and its applications, and presents novel results about the behavior of different Tabu Search schemes when escaping from a local attractor.

1 Reactive Search: feedback applied to heuristics

The Reactive Search (RS) framework proposes the introduction of **feedback** schemes in heuristics for discrete optimization problems. RS belongs to the *meta-heuristic* class of algorithms, where a basic scheme (local neighborhood search) is complemented to avoid its known defects. A simple form of **reinforcement learning** [33] is applied: the machine-learning agent is the discrete dynamical system that generates the search trajectory, whose internal parameters are changed through simple sub-symbolic mechanisms. Reactive Search is intensively **history-based**, in fact learning can be denoted as the influence on the behavior of a system of the past experienced events. Let us now summarize the context and the main motivations of the approach.

The *local search* heuristic for minimizing a cost function f starts from an admissible solution and tries to obtain a better one in an iterative way by "looking in the neighborhood." A *search trajectory* is generated in a *greedy* way if the algorithm chooses at each

step the best neighbor of the current tentative solution. Local search can be effective if the neighborhood structure matches the characteristics of the problem, but it stops when the current point is a *local minimizer*, i.e., when all neighbors have higher f values. Some possible actions to go beyond local minima while aiming at better suboptimal points are: i) the repetition of local search after restarting from different (possibly random) points, ii) the acceptance of upward-moving steps to exit local minima (e.g., in Simulated Annealing [32], Tabu Search [26]), iii) the consideration of more complex schemes based on the combination and selection of a set of suboptimal solutions (Genetic Algorithms [30] are based on this principle). There are at least a couple of potential drawbacks of simple implementations of the above schemes:

- **parameter tuning**: some schemes require a careful selection of parameters by the user before competitive results are obtained for specific problems. This requires either a deep knowledge of the problem structure or - simply - a lengthy "trial and error" tinkering process, that is not always fully reproducible (see for example the discussion in [4]).

- **search confinement**: after a local minimizer is encountered, all points in its attraction basin lose any interest for optimization. The search should avoid wasting excessive computing time in a single basin and *diversification* should be activated. On the other hand, in the assumptions that neighbors have correlated cost function values, some effort should be spent in searching for better points located close to the just found local minimum point (*intensification*). The two requirements are conflicting and finding a proper balance of diversification and intensification is a crucial issue in heuristics.

Can one obtain *wide spectrum* algorithms covering many applications without parameter tuning and, at the same time, avoiding excessive wastes of computing resources? We claim that this can be done in some cases by using **reactive** schemes, where the past history of the search is used for:

- **feedback-based parameter tuning**: the algorithm maintains the internal flexibility needed to cover many problems, but the tuning is automated, and executed while the algorithm runs and "monitors" its past behavior.

- **automated balance of diversification and intensification**: the "exploration versus exploitation" dilemma is present in many heuristics: is it better to intensify the search in the promising regions, or to diversify it to uncharted territories? An automated heuristic balance can be obtained through feedback mechanisms, for example by starting with intensification, and by progressively increasing the amount of diversification only when there is evidence that diversification is needed.

The focus of the RS method is on *wide spectrum* heuristic algorithms for discrete optimization, in which local search is complemented by feedback (**reactive**) schemes that use the past history of the search to increase its efficiency and efficacy. Related approaches in

a different context are, for example, the self-adaptation of the generating distributions of *Evolutionsstrategie* algorithms in [37], or the use of adaptive annealing schedules proposed by [31] and by other researchers.

The paper is organized as follows. Some underlying principles of Tabu Search (TS) are summarized in Sec. 2 and a classification of some TS-based algorithms is illustrated in Sec. 3. The application of reactive schemes in TS is summarized in Sec. 4. The results obtained by different versions of TS on a paradigmatic problem defined on binary strings are discussed in Sec. 5. The paper is concluded by a review of RS applications and future projects (Sec. 6–7).

2 Tabu Search: beyond local search

The Tabu Search meta-heuristic [26] is based on the use of *prohibition-based* techniques and "intelligent" schemes as a complement to basic heuristic algorithms like local search, with the purpose of guiding the basic heuristic beyond local optimality. It is difficult to assign a precise date of birth to these principles. For example, ideas similar to those proposed in TS can be found in the *denial* strategy of [38] (once common features are detected in many suboptimal solutions, they are *forbidden*) or in the opposite *reduction* strategy of [34] (in an application to the Travelling Salesman Problem, all edges that are common to a set of local optima are fixed). In very different contexts, prohibition-like strategies can be found in *cutting planes* algorithms for solving integer problems through their Linear Programming relaxation (inequalities that cut off previously obtained fractional solutions are generated) and in branch and bound algorithms (subtrees are not considered if the leaves cannot correspond to better solutions), see the textbook [35].

The renaissance and full blossoming of "intelligent prohibition-based heuristics" starting from the late eighties is greatly due to the role of F. Glover in the proposal and diffusion of a rich variety of meta-heuristic tools [26, 27], but see also [29] for an independent seminal paper. A growing number of TS-based algorithms has been developed in the last years and applied with success to a wide selection of problems [28, 41]. It is therefore difficult, if not impossible, to characterize a "canonical form" of TS, and classifications tend to be short-lived. Nonetheless, at least two aspects characterize many versions of TS: the fact that TS is used to complement **local (neighborhood) search**, and the fact that the main modifications to local search are obtained through the **prohibition** of selected moves available at the current point. Local search is effective if the neighborhood is appropriate to the problem structure but it clearly stops as soon as the first local minimizer is encountered, when no improving moves are available. TS acts to continue the search beyond the first local minimizer without wasting the work already executed, as it is the case if a new run of local search is started from a new random initial point, and to enforce appropriate amounts of diversification to avoid that the search trajectory remains confined near a given local minimizer.

In our opinion, the main competitive advantage of TS with respect to alternative heuristics based on local search like Simulated Annealing (SA) [32] lies in the intelligent

use of the past history of the search to influence its future steps. On the contrary, SA generates a Markov chain: the successor of the current point is chosen stochastically, with a probability that depends only on the current point and not on the previous history. The non-Markovian property is a mixed blessing: it permits heuristic results that are much better in many cases, but makes the theoretical analysis of the algorithm difficult. In fact, the practical success of TS techniques should motivate new search streams in Mathematics and Computer Science for its theoretical foundation. Incidentally, the often cited asymptotic convergence results of SA are unfortunately irrelevant for the application of SA to optimization. In fact, repeated local search [25], and even random search [19] has better asymptotic results. According to [1] "approximating the asymptotic behavior of SA arbitrarily closely requires a number of transitions that for most problems is typically larger than the size of the solution space ... Thus, the SA algorithm is clearly unsuited for solving combinatorial optimization problems to optimality." Of course, SA can be used in practice with fast cooling schedules, but then the asymptotic results are not directly applicable. The optimal finite-length annealing schedules obtained on specific simple problems do not always correspond to those intuitively expected from the limiting theorems [39].

Let us define the notation. $\mathcal{X}$ is the search space (as an example, the set of binary strings with a given length L: $\mathcal{X} = \{0,1\}^L$), $X^{(t)}$ is the current solution along the trajectory at iteration ("time") t. $N(X^{(t)})$ is the neighborhood of point $X^{(t)}$, obtained by applying a set of basic moves $\mu_0, \mu_1, ..., \mu_M$ to the current configuration:

$$N(X^{(t)}) = \{X \in \mathcal{X} \text{ such that } X = \mu_i \circ X^{(t)}, i = 0, ...M\}$$

In the case of binary strings, the moves can be those changing (complementing) the individual bits, and therefore M is qual to the string length L. Some of the neighbors are *prohibited*, a subset $N_A(X^{(t)}) \subset N(X^{(t)})$ contains the *allowed* ones. The general way of generating the search trajectory that we consider is given by:

$$X^{(t+1)} = \text{BEST-NEIGHBOR}(\ N_A(X^{(t)})\ ) \quad (1)$$

$$N_A(X^{(t+1)}) = \text{ALLOW}(N(X^{(t+1)}), X^{(0)}, ..., X^{(t+1)}) \quad (2)$$

The set-valued function ALLOW selects a subset of $N(X^{(t+1)})$ in a manner that depends on the entire search trajectory $X^{(0)}, ..., X^{(t+1)}$.

3 Some forms of Tabu Search

There are several vantage points from which to view heuristic algorithms. By analogy with the concept of *abstract data type* in Computer Science [2], and with the related *object-oriented* software engineering techniques [21], it is useful to separate the abstract concepts and operations of TS from the detailed implementation, i.e., realization with specific data structures. In other words, *policies* (that determine which trajectory is generated in the search space, what the balance of intensification and diversification is,

etc.) should be separated from *mechanisms* that determine *how* a specific policy is realized. An essential abstract concept in TS is given by the **discrete dynamical system** of eqn. 1-2 obtained by modifying local search. A taxonomy of Tabu Search is now presented where the abstract point of view of the dynamical systems is separated from the different implementation possibilities. Only a limited subset of the existing TS algorithms are cited in the presented classification.

3.1 Dynamical systems

A classification of some TS-related algorithms that is based on the underlying dynamical system is illustrated in Fig 1.

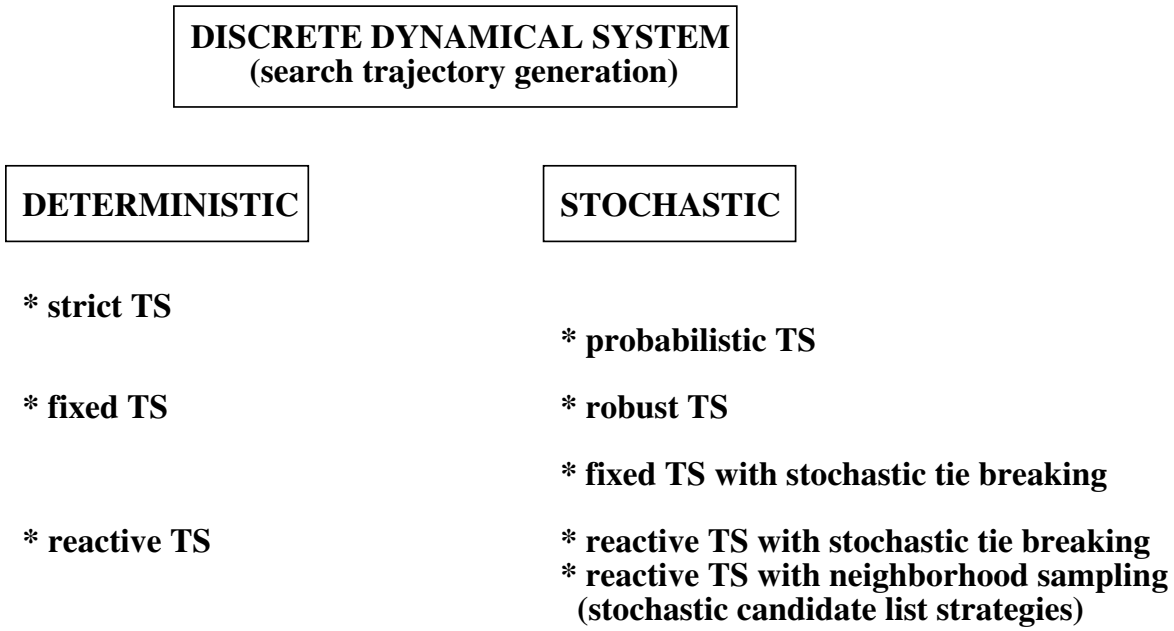

Figure 1: A classification based on discrete dynamical systems.

A first subdivision is given by the *deterministic* versus *stochastic* nature of the system. Let us first consider the deterministic versions. Possibly the simplest form of TS is what is called **strict-TS**: a neighbor is prohibited if and only if it has already been visited during the previous part of the search [26] (the term "strict" is chosen to underline the rigid enforcement of its simple prohibition rule). Therefore eqn. 2 becomes:

$$N_A(X^{(t+1)}) \quad = \quad \{X \in N(X^{(t+1)}) \quad \text{such that} \quad X \notin \{X^{(0)}, ..., X^{(t+1)}\}\} \tag{3}$$

Let us note that strict-TS is parameter-free.

Two additional algorithms can be obtained by introducing a **prohibition parameter**[1] T that determines how long a move will remain prohibited after the execution of its inverse.

[1]The term *prohibition parameter* is chosen instead of the more traditional *list size* because *list size* refers to a specific implementation: prohibitions can be obtained without using any list.

The **fixed-TS** algorithm is obtained by fixing T throughout the search [26]. A neighbor is allowed if and only if it is obtained from the current point by applying a move such that its inverse has not been used during the last T iterations. In detail, if LastUsed(μ) is the last usage time of move μ (LastUsed(μ) $= -\infty$ at the beginning):

$$N_A(X^{(t)}) = \{X = \mu \circ X^{(t)} \text{ such that } \textsc{LastUsed}(\mu^{-1}) < (t - T)\} \tag{4}$$

If T changes with the iteration counter depending on the search status (in this case the notation will be $T^{(t)}$), the general dynamical system that generates the search trajectory comprises an additional evolution equation for $T^{(t)}$, so that the three defining equations are now:

$$T^{(t)} = \textsc{T-React}(T^{(t-1)}, X^{(0)}, ..., X^{(t)}) \tag{5}$$

$$N_A(X^{(t)}) = \{X = \mu \circ X^{(t)} \text{ such that } \textsc{LastUsed}(\mu^{-1}) < (t - T^{(t)}) \tag{6}$$

$$X^{(t+1)} = \textsc{Best-Neighbor}(N_A(X^{(t)}))\} \tag{7}$$

Let us note that $\mu = \mu^{-1}$ for the basic moves acting on binary strings. Now, possible rules to determine the prohibition parameter by reacting to the repetition of previously-visited configurations have been proposed in [7] (**reactive-TS**, **RTS** for short). In addition, there are situations where the single reactive mechanism on T is not sufficient to avoid long cycles in the search trajectory and therefore a second reaction is needed [7]. The main principles of RTS are briefly reviewed in Sec. 4.

Stochastics algorithms related to the previously described deterministic versions can be obtained in many ways. For example, prohibition rules can be substituted with **probabilistic generation-acceptance rules** with large probability for allowed moves, small for prohibited ones, see for example the **probabilistic-TS** [26]. Stochasticity can increase the robustness of the different algorithms, in addition [26] "randomization is a means for achieving diversity without reliance on memory," although it could "entail a loss in efficiency by allowing duplications and potentially unproductive wandering that a more systematic approach would seek to eliminate." Incidentally, asymptotic results for TS can be obtained in probabilistic TS [24]. In a different proposal (**robust-TS**) the prohibition parameter is randomly changed between an upper and a lower bound during the search [40]. Stochasticity in fixed-TS and in reactive-TS can be added through a **random breaking of ties**, in the event that the same cost function decrease is obtained by more than one winner in the Best-Neighbor computation. At least this simple form of stochasticity should always be used to avoid external search biases, possibly caused by the ordering of the loop indices.

If the neighborhood evaluation is expensive, the exhaustive evaluation can be substituted with a partial **stochastic sampling**: only a partial list of candidates is examined before choosing the best allowed neighbor.

3.2 Implementations

While the above classification deals with dynamical systems, a different classification is based on the detailed data structures used in the algorithms and on the consequent

realization of the needed operations. Different data structures can possess widely different computational complexities so that attention should be spent on this subject before choosing a version of TS that is efficient on a particular problem.

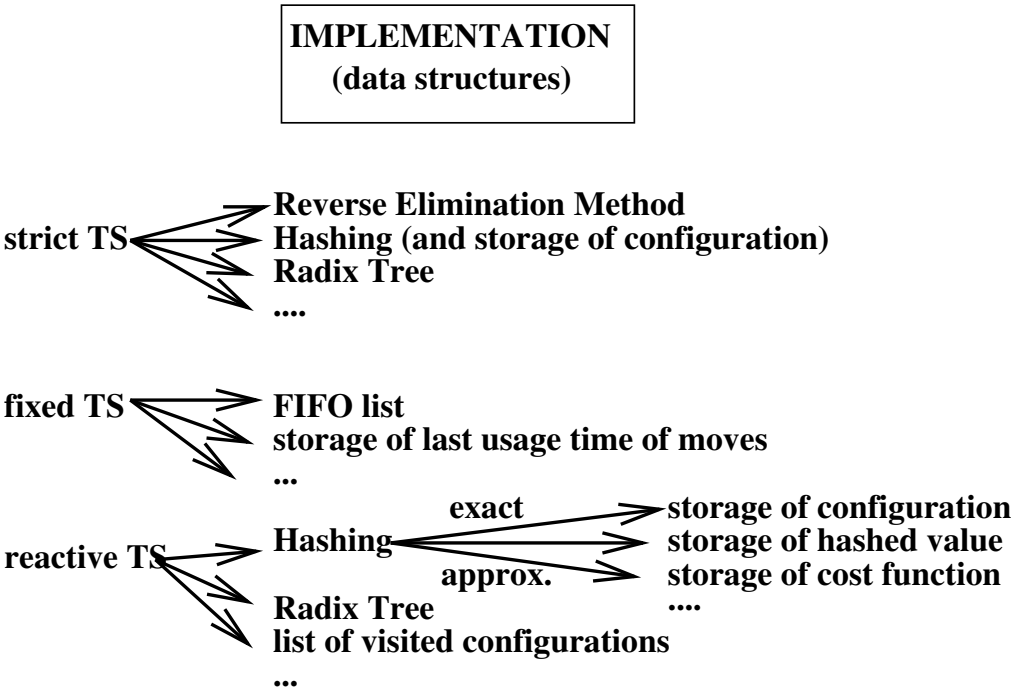

Figure 2: The same search trajectory can be obtained with different data structures.

Some examples of different implementations of the same TS dynamics are illustrated in Fig. 2. Strict-TS can be implemented through the reverse elimination method (REM) [27, 23], a term that refers to a technique for the storage and analysis of the ordered list of all moves performed throughout the search (called "running list"). The same dynamics can be obtained in all cases through standard **hashing** methods and storage of the configurations [42, 7], or, for the case of a search space consisting of binary strings, through the **radix tree** (or "digital tree") technique [7]. Hashing is an old tool in Computer Science: different hashing functions – possibly with incremental calculation – are available for different domains [20]. REM is not applicable to all problems (the "sufficiency property" must be satisfied [27]), in addition its computational complexity per iteration is proportional to the number of iterations executed, while the average complexity obtained through incremental hashing is $O(1)$, a small and constant number of operations. The worst-case complexity per iteration obtained with the radix tree technique is proportional to the number of bits in the binary strings, and constant with respect to the iteration. If the memory usage is considered, both REM and approximated hashing use $O(1)$ memory per iteration, while the actual number of bytes stored can be less for REM, because only changes (moves) and not configurations are stored.

Trivially, fixed-TS (alternative terms [26, 27] are: simple TS, static tabu search – STS – or tabu navigation method) can be realized with a first-in first-out list where the

prohibited moves are located (the "tabu list"), or by storing in an array the last usage time of each move and by using eqn. 4.

Reactive-TS can be implemented through a simple list of visited configurations, or with more efficient hashing or radix tree techniques. At a finer level of detail, hashing can be realized in different ways. If the entire configuration is stored (see also Fig. 4) an exact answer is obtained from the memory lookup operation (a repetition is reported if and only if the configuration has been visited before). On the contrary, if a "compressed" item is stored, like a hashed value of a limited length derived from the configuration, the answer will have a limited probability of error (a repetition can be reported even if the configuration is new, because the compressed items are equal by chance – an event called "collision"). Experimentally, small collision probabilities do not have statistically significant effects on the use of reactive-TS as heuristic tool, and hashing versions that need only a few bytes per iteration can be used. The effect of collision probabilities when hashing is used in other schemes is a subject worth investigating.

4 Reactive Tabu Search (RTS)

The RTS algorithm [7] represents the first realization of the principles introduced in Sec. 1 in the framework of F. Glover's Tabu Search (TS). Let us first note that the prohibition parameter T used in eq. 4 is related to the amount of **diversification**: the larger T, the longer the distance that the search trajectory must go before it is allowed to come back to a previously visited point. But T cannot be too large, otherwise no move will be allowed after an initial phase. In detail, if the search space is given by binary strings of length L, an upper bound $T \leq (L-2)$ guarantees that at least two moves are allowed at each iteration, so that the search does not get stuck and the move choice *is* influenced by the cost function value (it is not if only one move is allowed!). Now, if only allowed moves are executed, and T satisfies $T \leq (L-2)$, one obtains:

- The Hamming distance H between a starting point and successive points along the trajectory is strictly increasing for $T+1$ steps.

$$H(X^{(t+\tau)}, X^{(t)}) = \tau \quad \text{for} \quad \tau \leq T+1$$

- The minimum repetition interval R along the trajectory is $2(T+1)$.

$$X^{(t+R)} = X^{(t)} \;\Rightarrow\; R \geq 2(T+1)$$

Some problems arising in TS that are worth investigating are:

1. the determination of an appropriate prohibition T for the different tasks,
2. the robustness of the technique for a wide range of different problems,
3. the adoption of minimal computational complexity algorithms for using the search history.

The three issues are briefly discussed in the following subsections, together with the RTS methods proposed to deal with them.

4.1 Self-adjusted prohibition period

In RTS the *prohibition* T is determined through feedback (*reactive*) mechanisms during the search. T is equal to one at the beginning (the inverse of a given move is prohibited only at the next step), it increases only when there is **evidence** that diversification is needed, it decreases when this evidence disappears. In detail: the evidence that diversification is needed is signaled by the repetition of previously-visited configurations. All configurations found during the search are stored in memory. After a move is executed the algorithm checks whether the current configuration has already been found and it reacts accordingly (T increases if a configuration is repeated, T decreases if no repetitions occurred during a sufficiently long period).

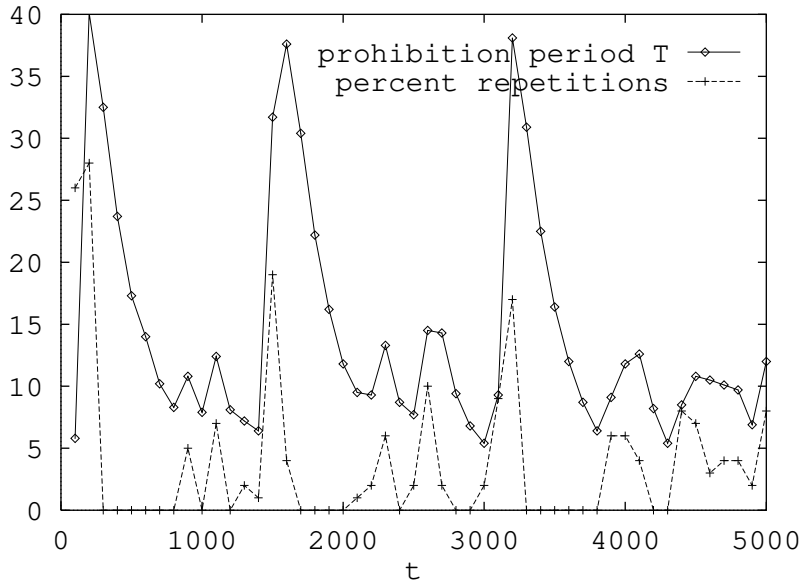

Figure 3: Dynamics of the the prohibition period T on a QAP task.

Let us note that T is not fixed during the search, but is determined in a dynamic way depending on the *local structure* of the search space. This is particularly relevant for "inhomogeneous" tasks, where the statistical properties of the search space vary widely in the different regions (in these cases a fixed T would be inappropriate).

An example of the behavior of T during the search is illustrated in Fig. 3, for a Quadratic Assignment Problem task [7]. T increases in an exponential way when repetitions are encountered, it decreases in a gradual manner when repetitions disappear.

4.2 The escape mechanism

The basic tabu mechanism based on prohibitions is not sufficient to avoid long cycles (e.g., for binary strings of length L, T must be less than the length of the string, otherwise all moves are eventually prohibited, and therefore cycles longer than $2 \times L$ are still possible). In addition, even if "limit cycles" (endless cyclic repetitions of a given set of configurations)

are avoided, the first reactive mechanism is not sufficient to guarantee that the search trajectory is not confined in a limited region of the search space. A "chaotic trapping" of the trajectory in a limited portion of the search space is still possible (the analogy is with *chaotic attractors* of dynamical systems, where the trajectory is confined in a limited portion of the space, although a limit cycle is not present).

For both reasons, to increase the robustness of the algorithm a second more radical diversification step (*escape*) is needed. The *escape* phase is triggered when too many configurations are repeated too often [7]. A simple *escape* consists of a number of random steps executed starting from the current configuration (possibly with a bias toward steps that bring the trajectory away from the current search region).

With a stochastic escape, one can easily obtain the **asymptotic convergence** of RTS (in a finite-cardinality search space, *escape* is activated infinitely often: if the probability for a point to be reached after escaping is different from zero for all points, eventually all points will be visited - clearly including the globally optimal points). The detailed investigation of the asymptotic properties and finite-time effects of different *escape* routines to enforce long-term diversification is an open research area.

4.3 Fast algorithms for using the search history

The storage and access of the past events is executed through the well-known **hashing** or radix-tree techniques in a CPU time that is approximately **constant** with respect to the number of iterations. Therefore the overhead caused by the use of the history is negligible for tasks requiring a non-trivial number of operations to evaluate the cost function in the neighborhood.

An example of a memory configuration for the hashing scheme is shown in Fig. 4. From the current configuration *phi* one obtains an index into a "bucket array." The items (configuration or hashed value or derived quantity, last time of visit, total number of repetitions) are then stored in linked lists starting from the indexed array entry. Both storage and retrieval require an approximately constant amount of time if: i) the number of stored items is not much larger than the size of the bucket array, and ii) the *hashing function* scatters the items with a uniform probability over the different array indices. More precisely, given a hash table with m slots that stores n elements, a load factor $\alpha = n/m$ is defined. If collisions are resolved by chaining searches take $O(1+\alpha)$ time, on the average.

5 Different ways of escaping from a local attractor

Local minima points are **attractors** of the search trajectory generated by deterministic local search. If the cost function is integer-valued and lower bounded it can be easily shown that a trajectory starting from an arbitrary point will terminate at a local minimizer. All points such that a deterministic local search trajectory starting from them terminates at a specific local minimizer make up its **attraction basin**. Now, as soon as a local minimizer is encountered, its entire attraction basin is not of interest for the optimization

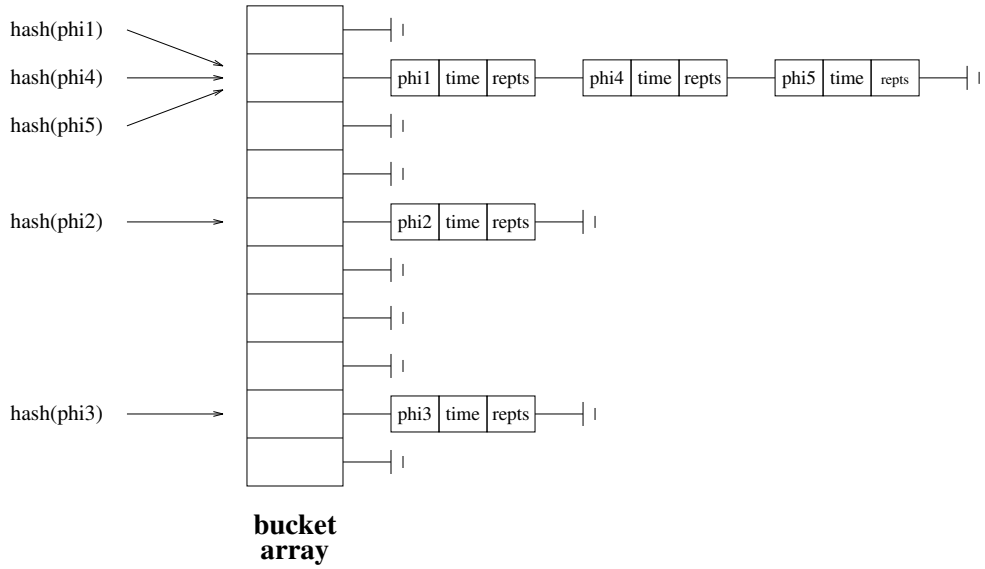

Figure 4: Open hashing scheme: items (configuration, or compressed hashed value, etc.) are stored in "buckets". The index of the bucket array is calculated from the configuration.

procedure, in fact its points do not have smaller cost values. It is nonetheless true that better points could be close to the current basin, whose boundaries are not known. One of the problems that must be solved in heuristic techniques based on local search is how to continue the search beyond the local minimizer and how to **avoid the confinement** of the search trajectory. Confinements can happen because the trajectory tends to be biased toward points with low cost function values, and therefore also toward the just abandoned local minimizer. The fact that the search trajectory remains close to the minimizer for some iterations is clearly a desired effect in the hypothesis that better points are preferentially located in the neighborhood of good suboptimal point rather than among randomly extracted points.

Simple confinements can be **cycles** (endless repetition of a sequence of configurations during the search) or more complex trajectories with no clear periodicity but nonetheless such that only a limited portion of the search space is visited (they are analogous to **chaotic attractors** in dynamical systems).

An heuristic prescription is that the search point is kept close to a discovered local minimizer at the beginning, snooping about better attraction basins. If these are not discovered, the search should gradually progress to larger distances (therefore progressively enforcing longer-term diversification strategies).

Some very different ways of realizing this general prescription are here illustrated for a "laboratory" test problem defined as follows. The search space is the set of all binary strings of length L. Let us assume that the search has just reached a (strict) local minimizer and that the cost f in the neighborhood is strictly increasing as a function of the number of different bits with respect to the given local minimizer (i.e., as a function of

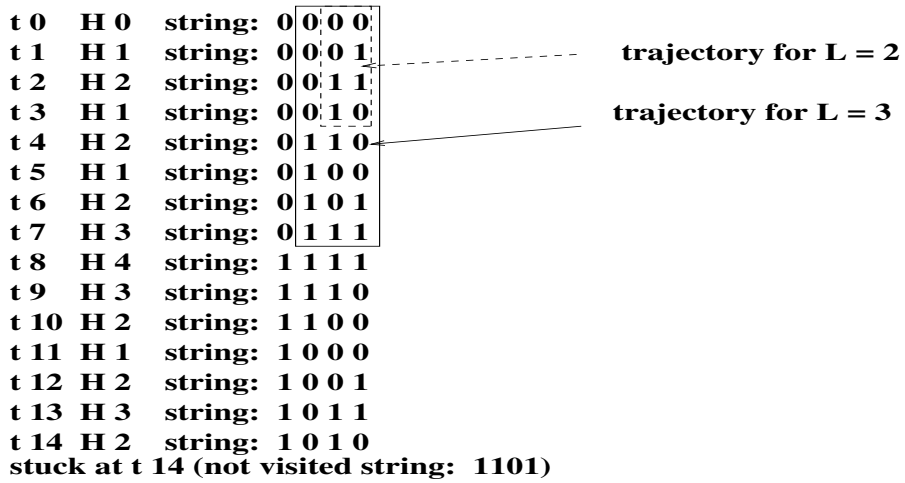

Figure 5: Search trajectory for deterministic strict-TS: iteration t, Hamming distance H and binary string.

the Hamming distance). Without loss of generality, let us assume that the local minimizer is the zero string ([00...0]) and that the cost is precisely the Hamming distance. Although artificial, the assumption is not unrealistic in many cases. An analogy in continuous space is the usual positive-definite quadratic approximation of the cost in the neighborhood of a strict local minimizer of a differentiable function. In the following parts the discussion is mostly limited to deterministic versions.

5.1 Strict-TS

In the deterministic version of strict-TS, if more than one basic move produce the same cost decrease at a given iteration, the move that acts on the right-most (least significant) bit of the string is selected.

The set of obtained configuration for $L = 4$ is illustrated in Fig 5. Let us now consider how the Hamming distance evolves in time, in the optimistic assumption that the search always finds an allowed move until all points of the search space are visited. If $H(t)$ is the Hamming distance at iteration t, the following holds true:

$$H(t) \leq \lfloor \log_2(t) \rfloor + 1 \tag{8}$$

This can be demonstrated after observing that a complete trajectory for an $(L-1)$-bit search space becomes a legal *initial part* of the trajectory for L-bit strings after appending zero as the most significant bit (see the trajectory for $L = 3$ in Fig 5). Now, a certain Hamming distance H can be reached only as soon as or after the H-th bit is set (e.g, H=4 can be reached only at or after $t = 8$ because the fourth bit is set at this iteration).

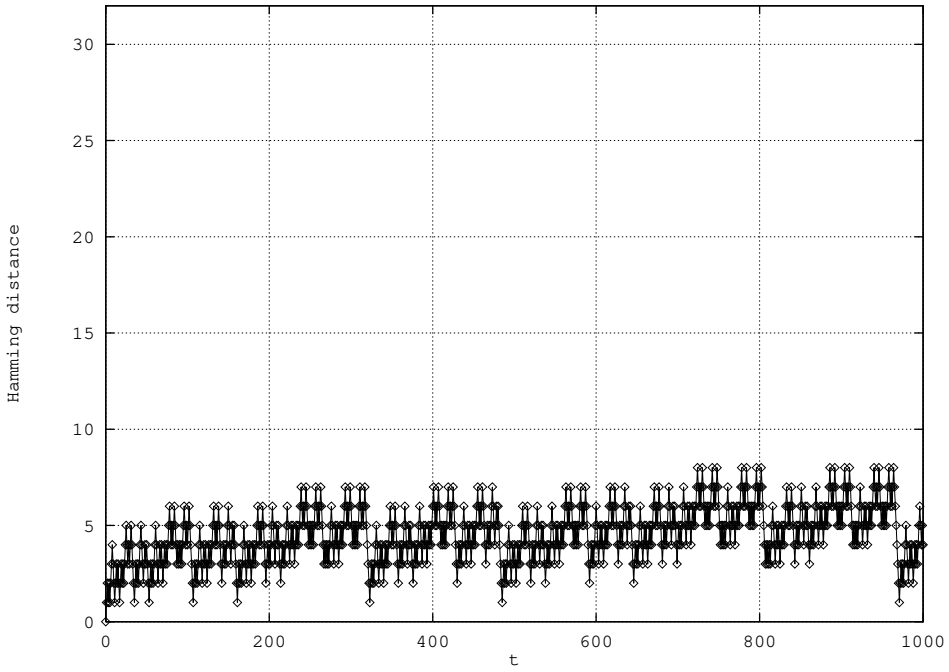

Figure 6: Evolution of the Hamming distance for deterministic strict-TS ($L = 32$).

Eqn. 8 trivially follows.

In practice the above optimistic assumption is not true: strict-TS can be stuck (trapped) at a configuration such that all neighbors have already been visited. In fact, the smallest L such that this event happens is $L = 4$ and the search is stuck at $t = 14$, so that the string [1101] is not visited. The problem worsens for higher-dimensional strings. For $L = 10$ the search is stuck after visiting 47 % of the entire search space, for $L = 20$ it is stuck after visiting only 11 % of the search space.

If the trajectory must reach Hamming distance H with respect to the local minimum point before escaping (i.e., before encountering a better attraction basin) the necessary number of iterations is at least exponential in H. Fig. 6 shows the actual evolution of the Hamming distance for the case of $L = 32$. The detailed dynamics is complex, as "iron curtains" of visited points (that cannot be visited again) are created in the configuration space and the trajectory must obey the corresponding constraints. The slow growth of the Hamming distance is related to the "basin filling" effect [7] of strict-TS: all points at smaller distances tend to be visited before points at larger distances (unless the iron curtains prohibit the immediate visit of some points). On the other hand, let us note that the exploration is not as "intensifying" as an intuitive picture of strict-TS could lead one to believe: some configurations at small Hamming distance are visited only in the last part of the search. As an example let us note that the point at $H = 4$ is visited at $t = 8$ while one point at $H = 1$ is visited at $t = 11$ (t is the iteration). This is caused by the fact that, as soon as a new bit is set for the first time, all bits to the right are progressively cleared (because new configurations with lower cost are obtained). In particular, the second configuration at $H = 1$ is encountered at $t > 2$, the third at $t > 4$,... the n-th at $t > 2^{(n-1)}$. Therefore, at least a configuration at $H = 1$ will be encountered only after

$2^{(L-1)}$ have been visited.

Let us note that the relation of eqn. 8 is valid only in the assumption that strict-TS is deterministic and that it is not stuck for any configuration. Let us now assume that one manages to obtain a more "intensifying" version of strict-TS, i.e., that all configurations at Hamming distance less than or equal to H are visited before configurations at distance greater than H. The initial growth of the Hamming distance is in this case much slower. In fact, the number of configurations C_H to be visited is:

$$C_H = \sum_{i=0}^{H} \left(\begin{array}{c} L \\ i \end{array} \right) \tag{9}$$

It can be easily derived that $C_H >> 2^H$, if $H << L$. As an example[2], for $L = 32$ one obtains from eqn. 9 a value $C_5 = 242\ 825$, and therefore this number of configurations have to be visited before finding a configuration at Hamming distance greater than 5, while $2^5 = 32$. An explosion in the number of iterations spent near a local optimum occurs unless the nearest attraction basin is very close. The situation worsens in higher-dimensional search spaces: for $L = 64$, $C_5 = 8\ 303\ 633$, $C_4 = 679\ 121$. This effect can be seen as a manifestation of the "curse of dimensionality:" a technique that works in very low-dimensional search space can encounter dramatic performance reductions as the dimension increases. In particular, there is the danger that the entire search span will be spent while visiting points at small Hamming distances, unless additional diversifying tools are introduced.

5.2 Fixed-TS

The analysis of fixed-TS is simple: as soon as a bit is changed it will remain prohibited ("frozen") for additional T steps. Therefore (see Fig. 7), the Hamming distance with respect to the starting configuration will cycle in a regular manner between zero and a maximal value $H = T + 1$ (only at this iteration the ice around the first frozen bit melts down and allows changes that are immediately executed because H decreases). All configurations in a cycle are different (apart from the initial configuration). The cycling T behavior is the same for both the deterministic and the stochastic version, the property of the stochastic version is that different configurations have the possibility of being visited in different cycles. In fact all configurations at a given Hamming distance H have the same probability of being visited if $H \leq T + 1$, zero probability otherwise.

The effectiveness of fixed-TS in escaping from the local attractor depends on the size of the T value with respect to the minimal distance such that a new attraction basin is encountered. In particular, if T is too small the trajectory will never escape, but if T is too big an "over-constrained" trajectory will be generated.

[2]Computations have been executed by Mathematica©.

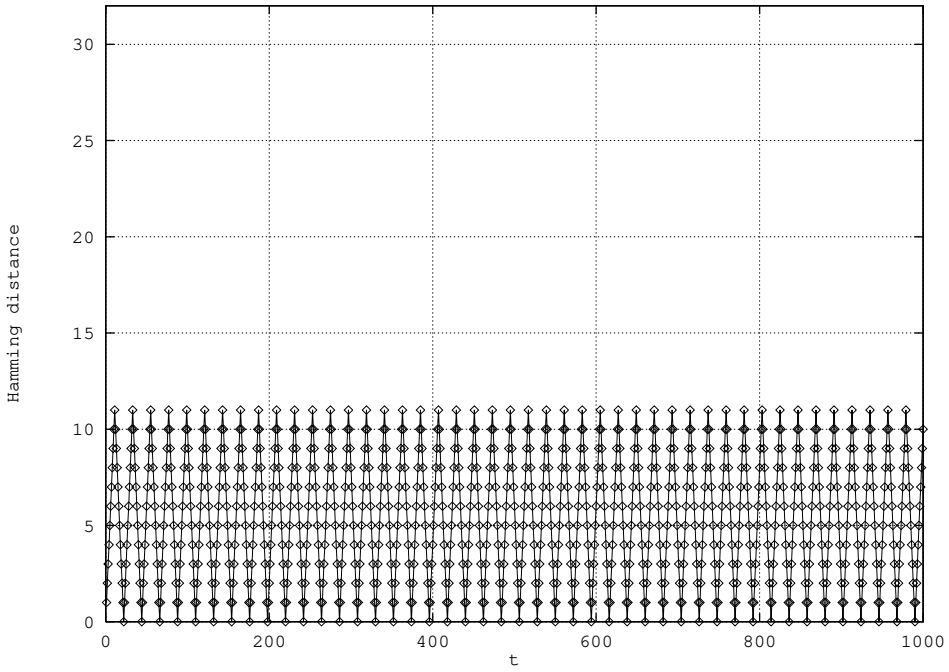

Figure 7: Evolution of the Hamming distance for both deterministic and stochastic fixed-TS ($L = 32$, $T = 10$).

5.3 Reactive-TS

The behavior of reactive-TS depends on the specific reactive scheme used. Given the previously illustrated relation between the prohibition T and the diversification, a possible prescription is that of gradually increasing T if there is evidence that the system is confined near a local attractor, until a new attractor is encountered. In particular, the evidence for a confinement can be obtained from the repetition of a previously visited configuration, while the fact that a new attraction basin has been found can be postulated if repetitions disappear for a suitably long period. In this last case, T is gradually decreased. This general procedure was used in the design of the RTS algorithm in [7], where specific rules are given for the entire feedback process.

In the present discussion we consider only the initial phase of the escape from the attractor, when increases of T are dominant over decreases. In fact, to simplify the discussion, let us assume that $T = 1$ at the beginning and that the reaction acts to increase T when a *local minimum* point is repeated, in the following manner:

$$\text{T-REACT}(T) \quad = \quad \min\{\max\{T \times 1.1, T + 1\}, L - 2\} \tag{10}$$

The initial value (and lower bound) of one implies that the system does not come back immediately to a just left configuration. The upper bound is used to guarantee that at least two moves are allowed at each iteration. Non-integral values of T are cast to integers before using them (the largest integer less than or equal to T).

The evolution of T for the deterministic version is shown in Fig. 8, repetitions of the local minimum point cause a rapid increase up to its maximal value. As soon as the value is reached the system enters a cycle. This limit cycle is caused by the fact that no

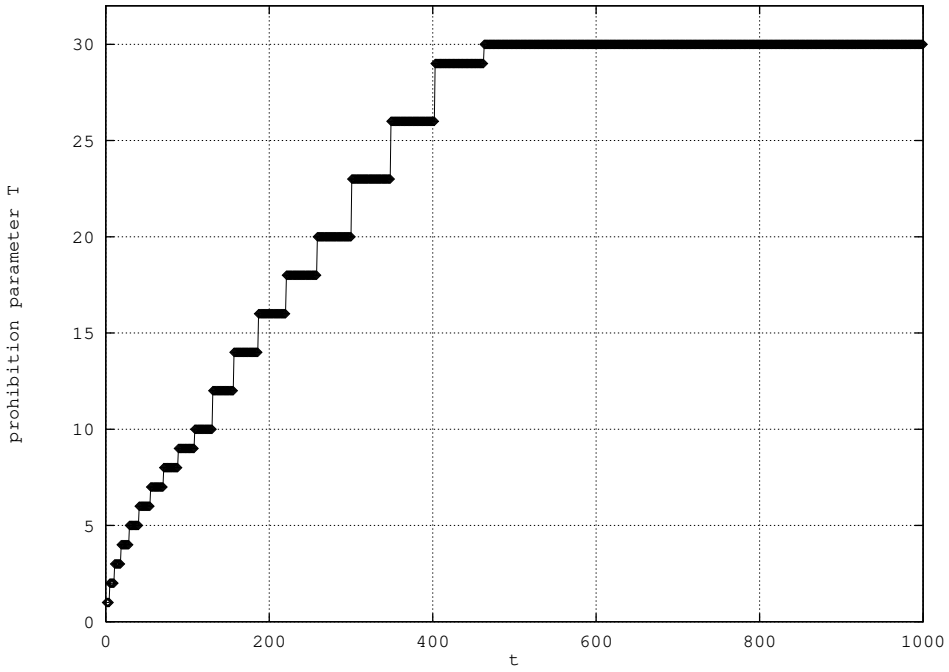

Figure 8: Evolution of the prohibition parameter T for deterministic reactive-TS with reaction at local minimizers ($L = 32$).

additional attraction basins exist in the test case considered, while in real-world *fitness surfaces* the prohibition T tends to be small with respect to its upper bound, both because of the limited size of the attraction basins and because of the complementary reaction that decreases T if repetitions disappear.

The behavior of the Hamming distance is illustrated in Fig. 9. The maximal Hamming distance reached increases in a much faster way compared to the strict-TS case.

Now, for a given $T^{(t)}$ the maximum Hamming distance that is reached during a cycle is $H_{max} = T^{(t)} + 1$ and the cycle length is $2(T^{(t)} + 1)$. After the cycle is completed the local minimizer is repeated and the reaction occurs. The result is that $T^{(t)}$ increases monotonically, and therefore the cycle length does also, as illustrated in Fig. 10 that expands the initial part of the graph.

Let us now consider a generic iteration t at which a reaction occurs (like $t = 4, 10, 18, ...$ in Fig. 10). At the beginning eqn. 10 will increase T by one unit at each step. If the prohibition is T just before the reaction, the total number t of iterations executed is:

$$t(T) = \sum_{i=1}^{T} 2(i+1) = 3T + T^2 \tag{11}$$

$$t(H_{max}) = \left(H_{max}^2 + H_{max} - 2\right) \tag{12}$$

$$H_{max}(t) = \frac{1}{2}\left(\sqrt{9+4t} - 1\right) \tag{13}$$

where the relation $T = H_{max} - 1$ has been used. Therefore the increase of the maximum reachable Hamming distance is approximately $O(\sqrt{t})$ during the initial steps. The increase is clearly faster in later steps, when the reaction is multiplicative instead of additive (when

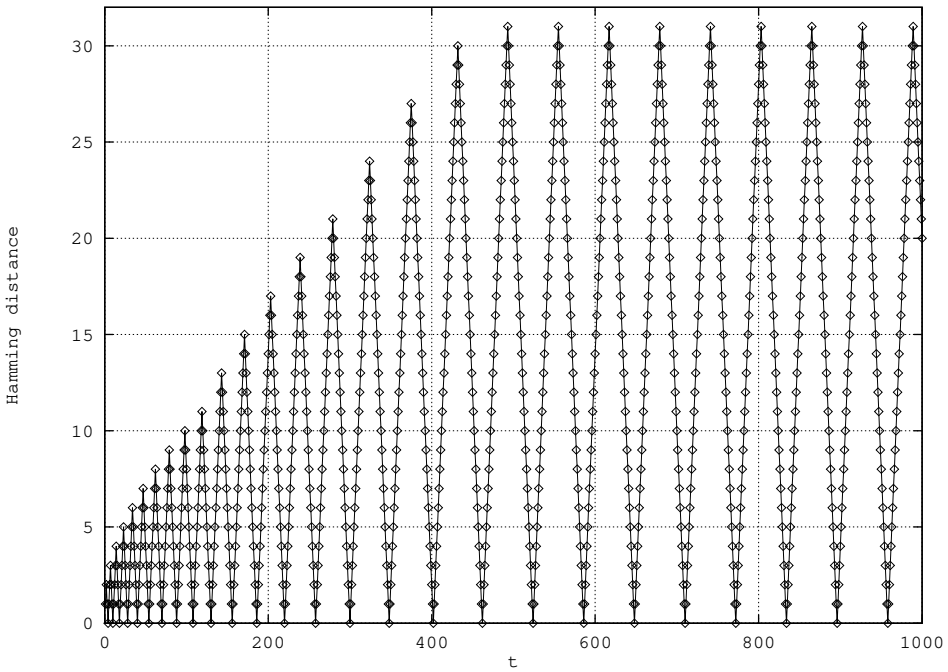

Figure 9: Evolution of the Hamming distance for simplified reactive-TS ($L = 32$).

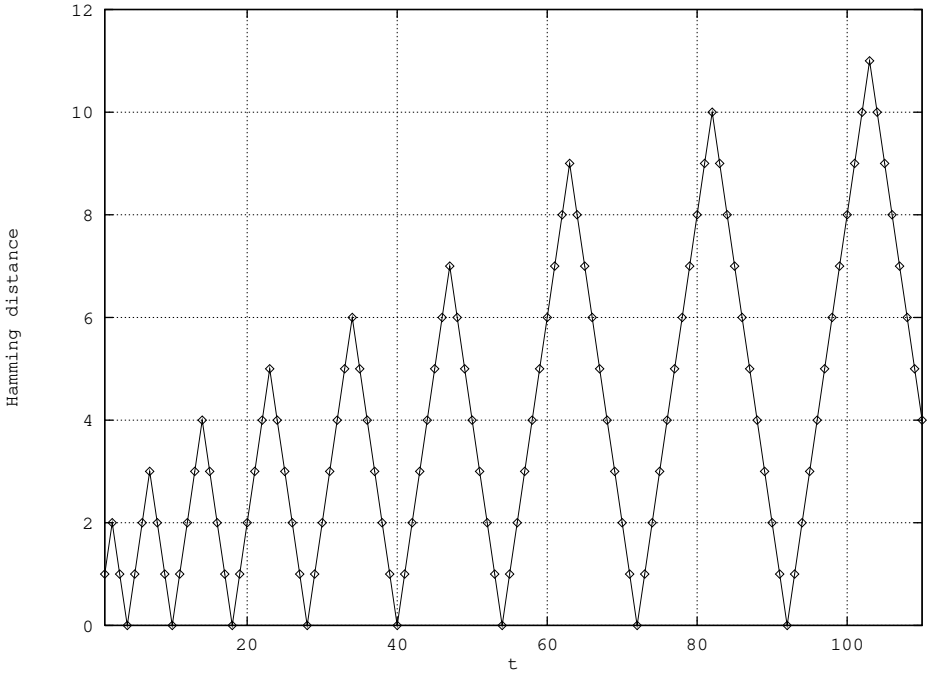

Figure 10: Evolution of the Hamming distance for reactive-TS, first 100 iterations ($L = 32$).

$\lfloor T \times 1.1 \rfloor > T + 1$ in eqn. 10), and therefore the above estimate of H_{max} becomes a lower bound in the following phase.

Let us note that the difference with respect to strict-TS is a crucial one: one obtains an (optimistic) logarithmic increase in the strict algorithm, and a (pessimistic) increase that behaves like the square root of the number of iterations in the reactive case. In this last case bold tours at increasing distances are executed until the prohibition T is sufficient to escape from the attractor. In addition, if the properties of the fitness surface change slowly in the different regions, and RTS reaches a given local minimizer with a T value obtained during its previous history, the chances are large that it will escape even faster.

6 Applications of Reactive Search: a review

The Reactive Search framework and, in particular, the RTS algorithm have been and are being applied to a growing number of discrete optimization problems (Sec. 6.1), continuos optimization tasks (Sec. 6.2), and different problems arising in the context of sub-symbolic machine learning (Sec. 6.3). Special purpose VLSI systems have been designed and realized for real-time *pattern recognition* applications with machine learning techniques (Sec. 6.4).

6.1 Combinatorial tasks

Reactive Search has been applied to the following list of problems.

- Quadratic Assignment Problem (QAP) [6, 7, 8]
- N-k model, a model of biological inspiration [6, 10]
- 0-1 Knapsack [7]
- Multi-Knapsack (with multiple constraints) [10]
- Max-Clique [18]
- Biquadratic Assignment Problem (Bi-QAP) [5]

In many cases the results obtained with alternative competitive heuristics have been duplicated with low computational complexity, and without intensive parameter and algorithm tuning. In some cases (e.g., in the Max-Clique and BiQAP problems) significantly better results have been obtained. A comparison of RTS with alternative heuristics (Repeated Local Minima Search, Simulated Annealing, Genetic Algorithms and Mean Field Neural Networks) is presented in [10]. A comparison with Simulated Annealing on QAP tasks is contained in [8]. A simple parallel implementation is presented in [6]. It has to be noted that RS is not a rigid algorithm but a general framework: specific algorithms have been introduced for unconstrained and constrained tasks, with different stochastic

mechanisms and prohibition rules. As it usually happens in heuristics, the more specific knowledge about a specific problem is used, the better the results. Nonetheless, it was often the case that simple RS versions realized with very limited effort could duplicate the performance of more complex schemes because of their simple embedded feedback (reinforcement learning) loop. A long-term goal of RS could be stated as the progressive shift of the learning capabilities from the algorithm user to the algorithm itself, through machine learning techniques.

6.2 Continuous optimization

A simple benchmark on a function with many suboptimal local minima is considered in [7], where a straightforward discretization of the domain is used.

A novel algorithm for the global optimization of functions (C-RTS) is presented in [9], in which a combinatorial optimization method cooperates with a stochastic local minimizer. The combinatorial optimization component, based on RTS, locates the most promising *boxes*, where starting points for the local minimizer are generated. In order to cover a wide spectrum of possible applications with no user intervention, the method is designed with adaptive mechanisms: in addition to the reactive adaptation of the *prohibition period*, the box size is adapted to the local structure of the function to be optimized (*boxes* are larger in "flat" regions, smaller in regions with a "rough" structure).

6.3 Sub-symbolic machine learning (neural networks)

While derivative-based methods for training from examples have been used with success in many contexts (*error backpropagation* [36] is an example in the field of neural networks), they are applicable only to differentiable performance functions and are not always appropriate in the presence of local minima. In addition, the calculation of derivatives is expensive and error-prone, especially if special-purpose VLSI hardware is used. We use a radically different approach: the task is transformed into a *combinatorial optimization* problem (the points of the search space are binary strings), and solved with the RTS algorithm [11]. To speed up the neighborhood evaluation phase a stochastic sampling of the neighborhood is adopted and a "smart" iterative scheme is used to compute the changes in the performance function caused by changing a single weight.

RTS escapes rapidly from local minima, it is applicable to non-differentiable and even discontinuous functions and it is very robust with respect to the choice of the initial configuration. In addition, by fine-tuning the number of bits for each parameter one can decrease the size of the search space, increase the expected generalization and realize cost-effective VLSI, as it is briefly described in the next section (representative papers are [13] [11] [12] [16]).

6.4 VLSI systems with learning capabilities

In contrast to the exhaustive design of systems for pattern recognition, control, and vector quantization, an appealing possibility consists of specifying a general architecture, whose parameters are then tuned through Machine Learning (ML). ML becomes a combinatorial task if the parameters assume a discrete set of values: the Reactive Tabu Search (RTS) algorithm permits the training of these systems with low number of bits per weight, **low computational accuracy**, no local minima "trapping", and limited sensitivity to the initial conditions [14, 15].

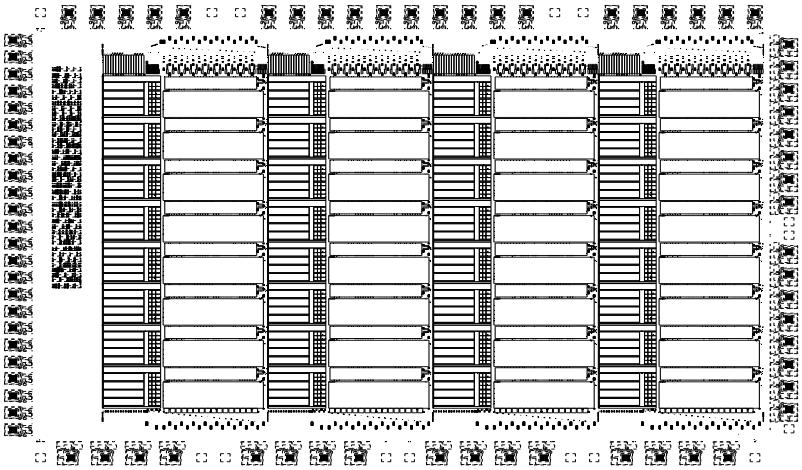

Figure 11: Layout of TOTEM chip: top level view of 32 processors, each containing the RAM cells (left), decoders (center), the MAC unit (bottom-right) and the output register (top-right). The control block is at the left.

Our project aims at developing special-purpose VLSI modules to be used as components of fully autonomous massively-parallel systems for **real-time "adaptive" applications**. Because of the intense use of parallelism at the chip and system level and the limited precision used, the obtained performance is competitive with that of state-of-the-art supercomputers (at a much lower cost), while a high degree of flexibility is maintained through the use of combinatorial algorithms.

In particular, *neural nets* can be realized. In contrast to many "emulation" approaches, the developed VLSI completely reflects the combinatorial structure used in the learning algorithms. The first chip of the project (TOTEM, funded by INFN and designed at IRST [15, 17]) achieves a performance of more than one billion multiply-accumulate operations. The chip layout is shown in Fig. 11. Applications considered are in the area of pattern recognition (Optical Character Recognition), events "triggering" in High Energy Physics [3], control of non-linear systems [11], compression of EEG signals [16].

```
1       void   RTSLoop(long     max_time,
2                      long     report_every)
3        {
4          while (Task.current.time < max_time)
5             {
6               if (check_for_repetitions(&Task.current) == DO_NOT_ESCAPE)
7                 {
8                    evaluate_neighborhood();
9                    choose_best_move();
10                   make_tabu(Task.chosen_bit);
11                   update_current_and_best();
12                }
13              else
14                 {
15                   escape();
16                 }
17              if ((Task.current.time % report_every) == 0)
18                PrintStateCompressed();
19            }
20       }
```

Figure 12: RTS basic loop, in the C programming language.

7 Software

A two-year project "Algorithms and software for optimization, and models of complex systems" started in 1995. The aim of the project is to apply optimization and machine learning algorithms in different domains and therefore particular attention is devoted to modern software engineering methods [22]. A code fragment is illustrated in Fig. 12, that shows the basic RTS loop in the C programming language. On-line information about the project as well as additional information about Reactive Search are available through an Internet WWW archive (http://rtm.science.unitn.it/).

Acknowledgements

This research was funded in part by the Special Project 1995-96 of the University of Trento (Math. Dept.), the initiative RTS of INFN (Istituto Nazionale di Fisica Nucleare), EU Esprit Project 7101 MInOSS. Dr. G. Di Caro provided assistance in the computational tests.

References

[1] E.H.L. Aarts, J.H.MN. Korst, and P.J. Zwietering (1995) Deterministic and randomized local search. In *Mathematical Perspectives on Neural Networks*, P. Smolenskij, M. Mozer and D. Rumelhart (Eds.). Lawrence Erlbaum Publishers: Hillsdale, NJ, to appear.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman (1983) *Data Structures and Algorithms*. Addison-Wesley.

[3] G. Anzellotti, R. Battiti, I. Lazzizzera, P. Lee, A. Sartori, G. Soncini, G. Tecchiolli, and A. Zorat (1995) TOTEM: a highly parallel chip for triggering applications with inductive learning based on the Reactive Tabu Search. Invited talk at: AIHENP95, Pisa - Italy.

[4] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. C. Resende, and W. Stewart (1995) Designing and Reporting on Computational Experiments with Heuristic Methods. In *Proceedings of MIC-95*, Breckenridge, CO, in press.

[5] R. Battiti and E. Çela (1995) Reactive Tabu Search for the Biquadratic Assignment Problem. Working Paper, Università di Trento.

[6] R. Battiti and G. Tecchiolli (1992) Parallel biased search for combinatorial optimization: Genetic algorithms and tabu. *Microprocessor and Microsystems* 16:351–367.

[7] R. Battiti and G. Tecchiolli (1994) The reactive tabu search. *ORSA Journal on Computing* 6(2):126–140.

[8] R. Battiti and G. Tecchiolli (1994) Simulated annealing and tabu search in the long run: a comparison on qap tasks. *Computer and Mathematics with Applications* 28(6):1–8.

[9] R. Battiti and G. Tecchiolli (1995) The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization. *Annals of Operations Research*, in press.

[10] R. Battiti and G. Tecchiolli (1995) Local search with memory: Benchmarking RTS. *Operations Research Spektrum* 17(2/3):67–86.

[11] R. Battiti and G. Tecchiolli (1995) Training nets with the reactive tabu search. *IEEE Transactions on Neural Networks* 6(5):1185–1200.

[12] R. Battiti, G. Tecchiolli, and P. Tonella (1995) Vector Quantization with the Reactive Tabu Search. In *Proceedings of MIC-95*, Breckenridge, CO, in press.

[13] R. Battiti (1993) The Reactive Tabu Search for Machine Learning. In *Atti del Quarto Workshop del Gruppo AI*IA di Interesse Speciale su Apprendimento Automatico* (G. Mauri Ed.), Milano, pp. 41-55.

[14] R. Battiti, P. Lee, A. Sartori and G. Tecchiolli (1994) Combinatorial Optimization for Neural Nets: RTS Algorithm and Silicon. Technical Report UTM 435-June 1994, Università di Trento.

[15] R. Battiti, P. Lee, A. Sartori and G. Tecchiolli (1994) TOTEM: A Digital Processor for Neural Networks and Reactive Tabu Search. In *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems - MICRONEURO'94*, Torino, Italy, IEEE Computer Society Press, CA, pp. 17–25.

[16] R. Battiti, A. Sartori, G. Tecchiolli, P. Tonella, and A. Zorat (1995) Neural Compression: an Integrated Approach to EEG Signals. In *Proc. of the International Workshop on Applications of Neural Networks to Telecommunications (IWANNT*95)*, Stockholm - Sweden, pp. 210-217.

[17] R. Battiti, P. Lee, A. Sartori, and G. Tecchiolli (1995) Special-Purpose Parallel Architectures for High-Performance Machine Learning. In *Proc. High Performance Computing and Networking, Milano - Italy*, Springer Verlag, pp. 944.

[18] R. Battiti and M. Protasi (1995) Reactive Local Search for the Maximum Clique Problem. Technical Report TR-95-052, ICSI, Berkeley, CA.

[19] T.-S. Chiang and Y. Chow (1988) On the convergence rate of annealing processes. *SIAM Journal on Control and Optimization* 26(6):1455–1470.

[20] T. H. Cormen, C. E. Leiserson, and R. L. Rivest (1990) *Introduction to Algorithms*. McGraw-Hill, New York.

[21] B. J. Cox (1990) *Object Oriented Programming, an Evolutionary Approach*. Addison-Wesley, Milano.

[22] G. Di Caro (1995) Regole di stile e di organizzazione per lo sviluppo di programmi in C. Technical Report UTM 462, Università di Trento.

[23] F. Dammeyer and S. Voss (1993) Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research* 41:31-46.

[24] U. Faigle and W. Kern (1992) Some Convergence Results for Probabilistic tabu Search. *ORSA Journal on Computing* 4(1):32–37.

[25] A. G. Ferreira and J. Zerovnik (1993) Bounding the probability of success of stochastic methods for global optimization. *Computer Math. Applic.* 25:1–8.

[26] F. Glover (1989) Tabu search - part I. *ORSA Journal on Computing* 1(3):190–260.

[27] F. Glover (1990) Tabu search - part II. *ORSA Journal on Computing* 2(1):4–32.

[28] F. Glover (1994) Tabu Search: Improved Solution Alternatives. In *Mathematical Programming, State of the Art 1994*, J. R. Birge and K. G. Murty (Eds.), The Univ. of Michigan, pp. 64–92.

[29] Hansen and B. Jaumard (1990) Algorithms for the maximum satisfiability problem. *Computing* 44:279–303.

[30] J. Holland (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI.

[31] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli (1986) An Efficient General Cooling Schedule for Simulated Annealing. In *Proceedingsof IICAD 86*, pp. 381–384.

[32] S. Kirkpatrick, C. D. Gelatt Jr. and M. P. Vecchi (1983) Optimization by simulated annealing. *Science* 220:671-680.

[33] L.-J. Lin (1992) Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning Journal, Special Issue on Reinforcement Learning* 8:293-322.

[34] S. Lin (1965) Computer Solutions of the Travelling Salesman Problems. *BSTJ* 44(10):2245–69.

[35] C. H. Papadimitriou and K. Steiglitz (1982) *Combinatorial Optimization, Algorithms and Complexity*. Prentice-Hall, NJ.

[36] D.E. Rumelhart, G.E. Hinton, and R.J. Williams (1986) Learning internal representations by error propagation. In *Parallel Distributed Processing Vol 1*. Cambridge, MA: MIT Press, pp. 318–362.

[37] H.-P. Schwefel, *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*. Interdisciplinary systems research; 26. Basel: Birkhäuser.

[38] K. Steiglitz and P. Weiner (1968) Some Improved Algorithms for Computer Solution of the Traveling Salesman Problem. In *Proceedings of the Sixth Allerton Conf. on Circuit and System Theory*, Urbana, Illinois, pp. 814–21.

[39] P. N. Strenski and S. Kirkpatrick (1991) Analysis of Finite Length Annealing Schedules. *Algorithmica* 6:346–366.

[40] E. Taillard (1991) Robust taboo search for the quadratic assignment problem. *Parallel Computing* 17:443–455.

[41] S. Voss (1993) *Intelligent search*. Manuscript, TH Darmstadt.

[42] D. L. Woodruff and E. Zemel (1993) Hashing vectors for tabu search. *Annals of Operations Research* 41:123–138.