The continuous reactive tabu search: Blending combinatorial optimization and stochastic search for global optimization

Roberto Battiti [1)]
Dipartimento di Matematica, Università di Trento,
Via Sommarive 14, I-38050 Povo (Trento), Italy
E-mail: battiti@science.unitn.it

Giampietro Tecchiolli
Istituto per la Ricerca Scientifica e Tecnologica and INFN,
Gruppo Collegato di Trento, Via Sommarive, I-38050 Povo (Trento), Italy
E-mail: tec@irst.it

A novel algorithm for the global optimization of functions (C-RTS) is presented, in which a combinatorial optimization method cooperates with a stochastic local minimizer. The combinatorial optimization component, based on the Reactive Tabu Search recently proposed by the authors, locates the most promising "boxes", in which starting points for the local minimizer are generated. In order to cover a wide spectrum of possible applications without user intervention, the method is designed with adaptive mechanisms: the box size is adapted to the local structure of the function to be optimized, the search parameters are adapted to obtain a proper balance of diversification and intensification. The algorithm is compared with some existing algorithms, and the experimental results are presented for a variety of benchmark tasks.

Keywords: Tabu search, local search, approximate algorithms, heuristics, global optimization, stochastic minimization, hybrid algorithms.

1. Introduction

In some challenging optimization tasks, the detailed properties of the function to be minimized (or maximized) are not known. The function can lack differentiability or it can present discontinuities. Even if the function is differentiable, one often encounters non-convex optimization problems: local minimizers like steepest descent will stop at the nearest local minimum. In addition, the local properties of the

[1)] To whom correspondence is to be sent.

function can be different in the different search regions: for example, the "basins of attraction" (i.e. the regions of initial search points that lead to the same local optima) can be of different sizes, the smoothness can vary, etc.

The above problems are well known and there is already a rich offering of different techniques to take care of them, some of which will be briefly discussed in the comparison part. The method that we propose is designed for general-purpose unconstrained global optimization (only function values are used, differentiability and continuity are not required) and it is characterized by an efficient use of *memory* during the search, in the framework of the tabu search heuristic. In addition, simple *adaptive* (*feedback*) mechanisms are used to tune the space discretization, by growing a *tree* of search boxes, and to adapt the prohibition period of the tabu search. This adaptation limits the amount of user intervention to the definition of an initial search region by setting upper and lower bounds on each variable, no parameters need to be tuned.

The paper is organized as follows. The motivations for considering a hybrid approach of combinatorial and continuous optimization are considered in section 2. The two basic components are described in section 3 (the combinatorial Reactive Tabu Search), and section 4 (the stochastic local minimizer Affine Shaker). The interface between the two components to obtain the hybrid C-RTS is described in section 5, together with the tree-like adaptive partition of the search space. Finally, the results of some experimental tests are discussed in section 6. In order to make this paper reasonably self-contained but to avoid cluttering the main text, some details about the RTS algorithm and about the test functions are contained in two appendices.

2. Fusing combinatorial and continuous optimization

Let us motivate the introduction of the C-RTS algorithm and define our notation. An instance of an optimization problem is a pair $(\mathcal{X}, f)$, where $\mathcal{X}$ is a set of feasible points and f is the cost function to be *minimized*: $f: \mathcal{X} \to R^1$. In the following, we consider both *combinatorial optimization* problems with a set $\mathcal{X}$ of finite cardinality, and *unconstrained continuous optimization* tasks where $\mathcal{X}$ is a compact subset of R^N, defined by bounds on the N independent variables $x_i : B_{L_i} \leq x_i \leq B_{U_i}$ (B_L and B_U are the lower and upper bounds, respectively). The specific meaning of $\mathcal{X}$ will be made clear from the context.

In many popular algorithms for continuous optimization, one identifies a "local minimizer" that locates a local minimum by descending from a starting point, and a "global" component that is used to diversify the search and to reach the global optimum. For example, steepest descent (SD) is one possible local minimizer for differentiable functions (although not always the best one!); in this scheme, the successor of the current point is chosen along the direction of the negative gradient. For a convex function f, SD will locate the global minimum, otherwise SD will

locate a *local* optimum corresponding to the *attraction basin* of the initial search point, and therefore supplementary strategies are needed to continue the search and identify additional optima, hopefully including the global optimum (for example, a trivial diversification component is given by repeated random starts). We define as *attraction basin* of a local minimum X_l the set of points that will lead to X_l when used as starting configurations for the local minimizer.

Combinatorial optimization algorithms must deal with hard tasks, often characterized by an abundance of local optima, and therefore they depend on efficient diversification strategies. For example, stochastic algorithms like the simulated annealing (SA) of [27] place non-zero probabilities for "upward moving" steps, genetic algorithms (GA) [24] use recombination and mutation mechanisms inspired by natural evolution, deterministic algorithms like the tabu search (TS) of [18,19] prohibit the repetition of previously-visited configurations in an explicit manner.

Now, a possible approach to solve a hard continuous optimization task consists of transforming it into a combinatorial task and solving it with state-of-the-art combinatorial algorithms. For example, the compact subset of R^N can be covered with a regular grid and the approximating solution can be searched among the grid points. The approach appears natural if one considers that limited precision numbers are actually used for computing the solution (e.g., floating point values represented with a given number of bits), and therefore the "continuous" task is numerically a combinatorial one. RTS has been used in the above way in [3] for the optimization of a continuous benchmark function with many deceiving local minima. Unfortunately, this paradigm is prone to inefficiencies and its actual success is doubtful for at least two reasons. First, the grid step of a fixed and uniform grid must be small enough so that the grid points represent the original function with sufficient precision, and therefore the number of grid points tends to be very large for non-trivial functions. The problem is serious especially if the problem dimension is high, or if the knowledge of f is limited and therefore a very fine grid has to be chosen. Second, a neighborhood that is appropriate for combinatorial optimization may be inappropriate for continuous optimization, so that the performance of a successful combinatorial algorithm can be seriously degraded.

One is therefore motivated to consider a *hybrid strategy*, whose local minimizer has the purpose of finding the local minimum with high precision and whose combinatorial component has the duty of discovering promising attraction basins for the local minimizer to be activated. Because the local minimizer is costly, it is activated only when the plausibility that a region contains a good local optimum is high. On the contrary, a fast evaluation of the search boxes is executed by the combinatorial component, and the size of the candidate boxes is adapted so that it is related to that of a single attraction basin (a box is split when there is evidence that at least two different local minima are located in the same box).

The main purpose of our work has been that of obtaining a sufficiently general-purpose and CPU-efficient algorithm, and this paper is dedicated to explaining

the detailed design of the basic C-RTS. In particular, the combinatorial component is based on the Reactive Tabu Search algorithm of [3] and is briefly summarized in section 3 and appendix I. The local minimizer is the Affine Shaker algorithm (AS) of [4], described in section 4. The AS is general-purpose and sufficiently efficient, although alternative local minimizers can be substituted for specific function classes. For example, if f is differentiable and well conditioned, steepest descent, conjugate gradient, or task-specific algorithms can be profitably used.

3. The combinatorial component: RTS

The *Reactive Tabu Search* (*RTS*) for combinatorial optimization was proposed by the authors in [2] and [3]. This section describes the basic principles of RTS, while the algorithm details and the modifications regarding the interface between combinatorial and continuous optimization are described in appendix I. RTS is a *local search* algorithm that generates a *trajectory* $X^{(t)}$ of points in the admissible search space. The successor of a point X is selected from a *neighborhood* $N(X)$ that associates to the current point X a subset of $\mathcal{X}$. A point X is *locally optimal with respect to N*, or a *local minimum* if $f(X) \leq f(Y)$ for all $Y \in N(X)$. For the following discussion, the consideration is limited to the case in which $\mathcal{X}$ is composed of binary strings with a finite length L: $\mathcal{X} = \{0, 1\}^L$ and the neighborhood is obtained by applying the *elementary moves* μ_i ($i = 1,\ldots,L$) that change the ith bit of the string $X = [x_1,\ldots,x_i,\ldots,x_L]$:

$$\mu_i([x_1,\ldots,x_i,\ldots,x_L]) = [x_1,\ldots,\bar{x}_i,\ldots,x_L], \tag{1}$$

where $\bar{x}_i$ is the negation of the ith bit: $\bar{x}_i \equiv (1 - x_i)$. Obviously, each move coincides with its inverse.

The RTS method uses an iterative *modified local search* algorithm to bias the search toward points with low f values and it incorporates *prohibition strategies* to discourage the repetition of already-visited configurations. At each step of the iterative process, the selected move is the one that produces the lowest value of the cost function f in the neighborhood. This move is executed even if f increases with respect to the value at the current point, to exit from local minima of f. As soon as a move is applied, the inverse move is temporarily prohibited (the names "tabu" or "taboo" derive from this prohibition).

In detail, at a given iteration t of the search, the set of moves $\mathcal{M}$ is partitioned into the set $\mathcal{T}^{(t)}$ of the *tabu* moves and the set $\mathcal{A}^{(t)}$ of the admissible moves. Superscripts with parentheses are used for quantities that depend on the iteration. At the beginning, the search starts from an initial configuration $X^{(0)}$, which is generated randomly, and all moves are admissible: $\mathcal{A}^{(0)} = \mathcal{M}$, $\mathcal{T}^{(0)} = \emptyset$. The search trajectory $X^{(t)}$ is then generated by applying the best admissible move $\mu^{(t)}$ from the set $\mathcal{A}^{(t)}$:

$$X^{(t+1)} = \mu^{(t)}(X^{(t)}) \quad \text{where } \mu^{(t)} = \arg \min_{v \in \mathcal{A}^{(t)}} f(v(X^{(t)})).$$

In isolation, the "modified local search" principle *can* generate cycles. For example, if the current point $X^{(t)}$ is a strict local minimum, the cost function at the next point must increase: $f(X^{(t+1)}) = f(\mu^{(t)}(X^{(t)})) > f(X^{(t)})$, and there is the possibility that the move at the next step will be its *inverse* ($\mu^{(t+1)} = \mu^{(t)^{-1}}$) so that the state after two steps will come back to the starting configuration

$$X^{(t+2)} = \mu^{(t+1)}(X^{(t+1)}) = \mu^{(t)^{-1}} \circ \mu^{(t)}(X^{(t)}) = X^{(t)}.$$

At this point, if the set of admissible moves is the same, the system will be trapped forever in a cycle of length 2. For this example, the cycle is avoided if the inverse move $\mu^{(t)^{-1}}$ is prohibited at time $t + 1$. In general, the inverse of the moves executed in the most recent part of the search are prohibited for a period T (also called *list size*). The period is finite because the tabu moves can be necessary to reach the optimum in a later phase of the search. In RTS, the prohibition period $T^{(t)}$ is time dependent: a move μ is prohibited if and only if its most recent use was at time $\tau \geq (t - T^{(t)})$.

The basic tabu search mechanism cannot guarantee the absence of cycles [3,5]. In addition, the choice of a fixed T without a priori knowledge about the possible search trajectories that can be generated in a given $(\mathcal{X}, f)$ problem is difficult. If the search space is inhomogeneous, a size T that is appropriate in a region of $\mathcal{X}$ may be inappropriate in other regions. For example, T can be too small and insufficient to avoid cycles, or too large, so that only a small fraction of the movements are admissible and the search is inefficient. Even if a constant T is suitable, its value depends on the specific problem.

RTS uses a simple mechanism to deal with cycles that are not avoided by using the basic tabu scheme and a way to change T during the search so that the value $T^{(t)}$ is appropriate to the local structure of the problem (hence the term "reactive").

The RTS scheme was designed under the constraints that the overhead (additional CPU time and memory) introduced in the search process by the reactive mechanisms had to be of a small number of CPU cycles and bytes and approximately constant for each step in the search process. By using *hashing* functions to store and retrieve the relevant data, the additional memory required can be reduced to some bytes per iteration, while the time is reduced to that needed to calculate a memory address from the current configuration and to execute a small number of comparisons and updates of variables [3]. RTS has been used for problems ranging from combinatorial optimization [2,3,7,8], minimization of continuous functions [3], and sub-symbolic machine learning tasks [5,6].

4. The local minimizer: affine shaker (AS)

This section summarizes the main features of the affine shaker algorithm of [4], an adaptive random search algorithm based on the theoretical framework of

[32]. The term "shaker" comes from the brisk movements of the search trajectory. The algorithm starts by choosing an initial point in the configuration space and an initial *search region* surrounding it, and it proceeds by iterating the following steps:

(i) A new tentative point is generated by sampling the search region with a uniform probability distribution.

(ii) The search region is adapted according to the value of the function at the new point. It is compressed if the new function value is greater than the current one (unsuccessful sample) or expanded otherwise (successful sample).

(iii) If the sample is successful, the new point becomes the current point, and the search region is translated so that the current point is at its center for the next iteration.

AS uses a simple search region Γ around the current point defined by a set of linearly independent vectors: $\Gamma = \{\boldsymbol{b}_j\}$ for $j = 1,\ldots,N$. To generate a random displacement, the basis vectors are multiplied by random numbers in the real range $(-1.0, 1.0)$ and added: $\boldsymbol{\delta} = \sum_{j=1}^{N} rand_j \times \boldsymbol{b}_j$.

The procedure is illustrated in figure 1. First the random displacement is generated. Then the *double-shot* strategy is employed: if the first sample $X + \boldsymbol{\delta}$ is not successful, the specular point $X - \boldsymbol{\delta}$ is considered. This choice drastically reduces the probability of generating two consecutive unsuccessful samples [12]. Finally, if a better point is found, the current point is updated and the search region Γ expanded; if not, the search region is compressed instead. The search region therefore undergoes an *affine* transformation after each iteration. The convergence speed after AS locates the local minimum region is very high: if the current point is close to the minimum, most trials will be unsuccessful and, in these cases, the volume of the search region is divided by two at each iteration.

The AS termination criterion is related to the fractional precision ε with which the user wants to determine the position of a local minimum (ε values are given in appendix II). In our tests, AS is terminated if for two consecutive steps $\boldsymbol{\delta}^{(t)}$ and $\boldsymbol{\delta}^{(t+1)}$, one has $\|\boldsymbol{\delta}\| < (\varepsilon/10)\,\|B_U - B_L\|$.

Let us note that the computational load of AS per iteration grows as $O(N^3)$ as a function of the dimension because of the affine transformations needed. A similar shaker algorithm is presented in [5] and [13] (Inertial Shaker), where the growth is of order $O(N)$. Although the inertial shaker tends to require more function evaluations to converge and larger amounts of memory and processing for a small number of independent variables, it can become competitive for high-dimensional tasks. The shaker algorithms have been used for problems ranging from nonlinear regression, computation of the minimum eigenvalue and functional minimization [13], training of neural networks [4,14].

procedure affine_shaker
comment: *The constant adjustment factors for the search region* Γ *are* $\rho_{expand} = 2$, $\rho_{compress} = 1/2$.
Initialize :
$t \leftarrow 0$ *(iteration counter)*;
$X \leftarrow$ *(initial random point)*;
$\forall j \quad \vec{b}_j \leftarrow \vec{e}_j \times (1/4) \times$ (range of j-th variable) where $\vec{e}_j$ is the canonical basis of R^N
repeat
("double shot" strategy:)
$\vec{\delta} = \sum_j rand_j \times \vec{b}_j$ *($rand_j$ = random numbers in (-1,1))*
if $f(X + \vec{\delta}) < f(X)$ **then**
$X \leftarrow X + \vec{\delta}$ *(first "shot" successful)*
$P \leftarrow I + (\rho_{expand} - 1)\frac{\vec{\delta}\vec{\delta}^T}{\|\vec{\delta}\|^2}$ *(expand* Γ*)*
$\forall i \quad \vec{b}_i \leftarrow P\,\vec{b}_i$
else if $f(X - \vec{\delta}) < f(X)$ **then**
$X \leftarrow X - \vec{\delta}$ *(second "shot" successful)*
$P \leftarrow I + (\rho_{expand} - 1)\frac{\vec{\delta}\vec{\delta}^T}{\|\vec{\delta}\|^2}$ *(expand* Γ*)*
$\forall i \quad \vec{b}_i \leftarrow P\,\vec{b}_i$
else *(no success)*
$P \leftarrow I + (\rho_{compress} - 1)\frac{\vec{\delta}\vec{\delta}^T}{\|\vec{\delta}\|^2}$ *(compress* Γ*)*
$\forall i \quad \vec{b}_i \leftarrow P\,\vec{b}_i$
$t \leftarrow (t + 1)$ *(increment iteration counter)*
until convergence criterion is satisfied

Figure 1. *Affine Shaker* algorithm.

5. The hybrid C-RTS

In the hybrid scheme, C-RTS must identify promising regions for the AS local minimizer to be activated. This section specifies how this goal is accomplished and how the two components are interfaced.

The initial search region is specified by bounds on each independent variable $x_i : B_{L_i} \le x_i \le B_{U_i}$, for $i = 1,\ldots,N$. The basic structure through which the initial search region is partitioned consists of a *tree of boxes*. The tree is born with 2^N equal-size leaves, obtained by dividing in half the initial range on each variable. Each box is then subdivided into 2^N equal-size children, as soon as two different local minima are found in it. Because the subdivision process is triggered by the local properties of f, after some iterations of C-RTS the tree will be of varying depth in the different regions, with boxes of smaller sizes being present in regions that require an intensification of the search. Only the *leaves* of the tree are admissible search points for the combinatorial component of C-RTS. The leaves partition the initial region: the intersection of two leaves is empty, the union of all leaves coincides with the initial

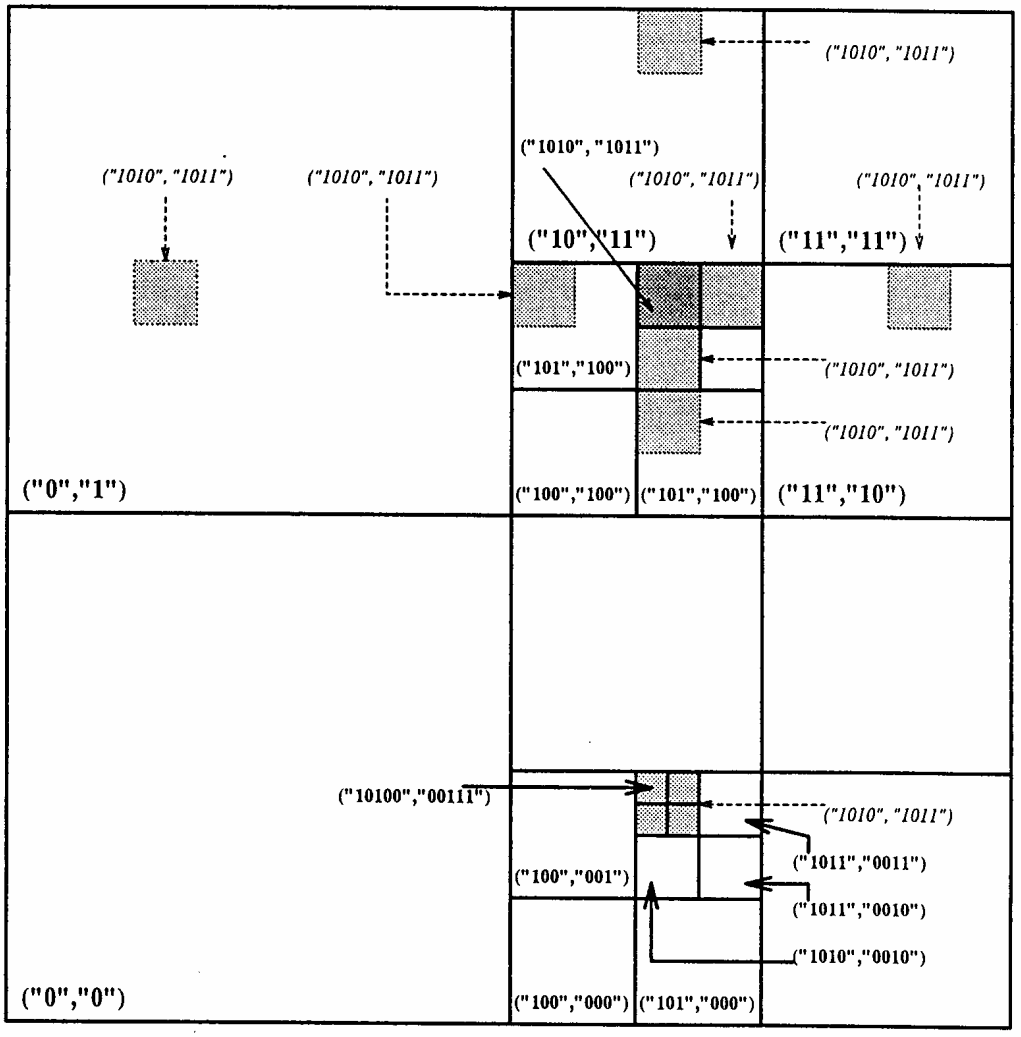

Figure 2. C-RTS: Tree of search boxes. Thick borders and bold strings identify existing leaf-boxes, hatched boxes show the neighborhood of box (1010, 1011).

search space. A typical configuration for a two-dimensional task is shown in figure 2, where each leaf-box is identified by thick borders and a bold binary string (hatched boxes refer to an example of the neighborhood, see section 5.2).

5.1. BOX EVALUATION

While the RTS algorithm for combinatorial optimization generates a search trajectory consisting of points $X^{(t)}$, C-RTS generates a trajectory consisting of *leaf-boxes* $B^{(t)}$. There are two important changes to be underlined: first, the function $f(X)$ must be substituted by a routine that measures the potentiality that the current box

contains good local optima; second, the tree is *dynamic* and the number of existing boxes grows during the search, so that the C-RTS prohibition rules must be changed accordingly. We use the term "potentiality", and not "probability", to emphasize the heuristic nature of the C-RTS algorithm, which does not use specific probabilistic models of f.

The combinatorial component must identify promising boxes quickly (the high precision of the local minimization is obtained from the affine shaker). In the absence of detailed models about the function f to be minimized, a simple evaluation of a box B can be obtained by generating a point X with a uniform probability distribution inside its region and by evaluating the function $f(X)$ at the obtained point. Let us use the same function symbol, the difference being evident from its argument: $f(B) \equiv f(rand\ X \in B)$. The potential drawback of this simple evaluation is that the search can be strongly biased toward a box in the case of a "lucky" evaluation (e.g., $f(X)$ close to the minimum in the given box), or away from a box in the opposite case. To avoid this drawback, when a box is encountered again during the search, a *new* point X is generated and evaluated, and some collective information is returned. Two possible ways of using the information from the points X_i evaluated in a given box have been tested:

1. The value $f(B)$ returned is the *average* of the evaluated X_i: $f(B) \equiv (1/N_B) \sum_{i=1}^{N_B} f(X_i)$, where N_B is the number of points.
2. The value returned is the *minimum* of the evaluated points: $f(B) \equiv \min_{i \in (1,\ldots,N_B)} f(X_i)$.

The two options clearly coincide at the beginning of the search; when $N_B = 1$, they may induce different dynamics as soon as some boxes are encountered again along the trajectory and new points are evaluated. The algorithm is robust with respect to this choice and the difference is hardly significant for the test functions considered, apart from the constrained task that was solved with the penalty method: in this case, the use of the minimum is more efficient, see section 6.5.

5.2. BOXES: IDENTIFYING BINARY STRINGS AND NEIGHBORHOOD

Each existing box for a problem of dimension N is identified by unique binary string B_S with $n \times N$ bits: $B_S = [g_{11},\ldots,g_{1n},\ldots,g_{N_1},\ldots,g_{N_n}]$. The value n is the depth of the box in the tree: $n = 0$ for the *root* box, $n = 1$ for the leaves of the initial tree (and therefore the initial string has N bits), n increases by one when a given box is subdivided. The length of the box edge along the ith coordinate is therefore equal to $(B_{U_i} - B_{L_i})/2^n$. The position of the box origin B_{O_i} along the ith coordinate is

$$B_{O_i} = B_{L_i} + (B_{U_i} - B_{L_i}) \sum_{j=1}^{n} g_{ij}/2^j.$$

The evaluated neighborhood of a given box consists only of existing leaf-boxes: no new boxes are created during the neighborhood evaluation. Now, after applying the elementary moves to the identifying binary string B_S of a given box B, one obtains $N \times n$ boxes of the same size distributed over the search space as illustrated in figure 2, for the case of B_S = (1010, 1011). Because the tree can have different depth in different regions, it can happen that some of the obtained strings do *not* correspond to leaf-boxes and others can cover *more* than a single leaf-box. In the first case, one evaluates the smallest *enclosing* leaf-box, in the second case, one evaluates a randomly-selected *enclosed* leaf-box. The random selection is executed by generating a point with uniform probability in the original box, and by selecting the leaf that contains the point. This ensures that the probability for a leaf to be selected is proportional to its volume. In both cases, the time needed to find the legal leaf-boxes is reduced to a small number of CPU cycles (bounded by the maximum depth n_{max}), by using a digital tree structure [28].

Let us consider the example of figure 2. The current box (1010, 1011) has the neighbors shown with a hatched pattern. The neighbor (0010, 1011) in the upper left part is not an existing leaf-box; it is therefore transformed into the enclosing existing leaf-box (0, 1). Vice versa, the neighbor (1010, 0011) in the lower right part contains four leaves; one of them (10100, 00111) is the output of a random selection. Figure 3 specifies the complete final neighborhood obtained for the given example.

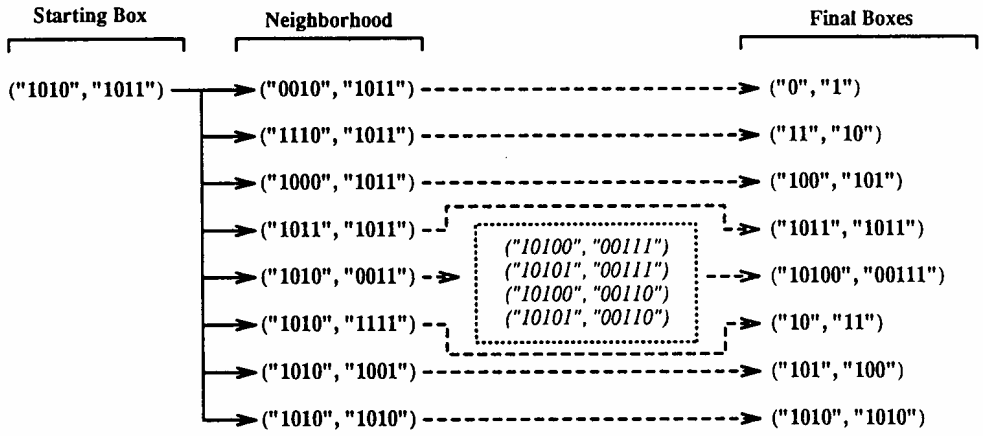

Figure 3. C-RTS: Evaluation of the neighborhood of box (1010, 1011).

According to the RTS dynamics illustrated in appendix I (figure 9), the neighborhood boxes obtained in the above way are evaluated only if the corresponding basic move from the current point is not prohibited. The prohibition rules guarantee that at least two moves are admissible, see section 5.3.

Only if the evaluation $f(B^{(t)})$ of the current box is less than all evaluations executed in the neighborhood is a decision taken about the possible triggering of the affine shaker. In other words, a necessary condition for activating high-precision – and therefore expensive – searches with the shaker algorithm is that there is a high plausibility that the current region can produce local minima that are better with respect to the given neighborhood of candidate boxes, the plausibility being measured by $f(B)$. Given the greedy nature of the combinatorial component, the current box $B^{(t)}$ on the search trajectory moves toward non-tabu locally optimal boxed; therefore, it will eventually become locally optimal and satisfy the conditions for triggering AS. Let us note that, if a given box B loses the above contest (i.e., it is not locally optimal for RTS), it *maintains* the possibility to win when it is encountered again during the search, because the evaluation of a different random point X can produce a better $f(B)$ value. Thanks to the evaluation method, C-RTS is *fast in optimal conditions*, when the f surface is smooth and $f(B)$ is a reliable indicator of the local minimum that can be obtained in region B, but it is *robust in harder cases*, when the $f(B)$ values have a high standard deviation or when they are unreliable indicators of good local minima obtainable with the affine shaker.

5.3. TABU RULE IN THE DYNAMIC CONTEXT OF C-RTS

The temporary prohibition of moves must enforce the appropriate degree of diversification but must ensure that at least two moves can be executed and that the search will not remain stuck. The RTS algorithm [3,5] guarantees that the above requirements are met when the space consists of fixed-length binary strings, but the rule for admitting or prohibiting moves must be changed in C-RTS, where the length of the binary string B_S varies because it is related to the depth of the current box in the tree. The natural modification is to introduce a "fractional" list size $T_F \in [0, 1]$ that regulates the fraction of prohibited moves in the possible set of $N \times n$ moves. In detail, the list size $T(n)$ that is used to determine the admissible moves from a box at depth n in the tree is obtained as follows:

$$T(n) = \mathrm{Min}\Big(\mathrm{Max}\big(1, \lfloor T_F\, N\, n \rfloor\big), nN - 2\Big). \tag{2}$$

The formula is easily explained. In the normal situation, the list is proportional to T_F: $T(n) = \lfloor T_F\, N\, n \rfloor$, the "Max" operator ensures that the list is at least one, the "Min" operator ensures that it is at most $Nn - 2$. In this way, the last executed move is always prohibited, and at least *two* moves are available from a given configuration, so that the chosen move is influenced by the values of f in the neighborhood. Trivially, there would be no choice with a unique available move. Naturally, the two requirements cannot always be met if $Nn - 2 < 1$: if the string length is $Nn \leq 2$, the list $T(n)$ becomes zero or negative and no moves are prohibited.

To summarize the basic reactive tabu search dynamics, the balance between intensification and diversification when candidate boxes are searched with C-RTS is solved in the following way:

(i) The balance starts by using only intensification, no moves are prohibited at the beginning, and the trajectory is driven toward locally optimal boxes.

(ii) As soon as boxes are repeated, the diversification starts by gradually increasing the list size until repetitions are avoided.

(iii) When repetitions are absent for a sufficiently long period, the diversification gradually disappears: the list size is gradually reduced until either it becomes minimal or repetitions are present again.

The second-level "escape" mechanism described in figure 9 (function **diversify_search**) is needed in exceptional situations, when the basic tabu dynamics is not sufficient, for example if cycles longer than $2(L-1)$ are present along the trajectory [5]. The fact that appropriate amounts of diversification are "administered" in an automated way during the search is the basic meta-strategy that permits the use of the same algorithm to solve in an efficient way a wide variety of optimization tasks.

5.4. ACTIVATION OF THE AFFINE SHAKER

As explained above, the local optimality of the current box B is a necessary condition for the activation of the shaker algorithm, but it is not sufficient unless B is locally optimal for the first time, a case in which AS is always triggered. Otherwise, if $r > 1$ is the number of times that box B has been locally optimal during the search, an additional AS run must be justified by a sufficient probability of finding a *new* local minimum in B. Bayesian stopping rules for multi-start global optimization methods are studied in [10]. These rules can be applied in the context of a single box, where a multi-start technique is realized with repeated activation of AS from uniformly distributed starting points. Because of our splitting criterion, at most one local optimum will be associated to a given box (a box is split as soon as two different local optima are found, see section 5.5). In addition, some parts of the box can be such that AS will exit the borders if the initial point belongs to these portions. One can therefore partition the box region into W components, the attraction basins of the local minima contained in the box and a possible basin that leads AS outside, so that the probabilities of the basins sum up to one ($\sum_{w=1}^{W} P_w = 1$). These are the cells (events) that define a multinomial probability distribution associated to a given box.

According to [10], if $r > W + 1$ restarts have been executed and W different cells have been identified, the total relative volume of the "observed region" (i.e., the posterior expected value of the relative volume Ω) can be estimated by

$$E(\Omega \mid r, W) = \frac{(r - W - 1)(r + W)}{r(r - 1)}; \quad r > W + 1. \tag{3}$$

Note that the above formula does not depend on the detailed probability values P_w but only on the number of cells and number of trials. The affine shaker is always triggered if $r \leq W + 1$, because the above estimate is not valid in this case; otherwise, the AS is executed again with probability equal to $1 - E(\Omega \mid r, W)$. In this way, additional runs of AS tend to be spared if the above estimate predicts a small probability to find a new local optimum, but a new run is never completely prohibited for the sake of robustness: it can happen that the Bayesian estimate of equation (3) is unreliable, or that the unseen portion $(1 - E(\Omega \mid r, W))$ contains a very good minimum with a small attraction basin. The probabilistic activation is obtained by generating a pseudo-random number *rand* in the range [0, 1] and firing AS only if $rand > E(\Omega \mid r, W)$.

The initial conditions for AS (see figure 1) are that the initial search point is extracted from the uniform distribution inside B, the initial search frame is $\boldsymbol{b}_i = \boldsymbol{e}_i \times (1/4) \times (B_{U_i} - B_{L_i})$, where $\boldsymbol{e}_i$ are the canonical basis vectors of R^N. The affine shaker generates a trajectory that must be contained in the box B enlarged by a border region of width $(1/2) \times (B_{U_i} - B_{L_i})$, and it must converge to a point contained in B. If AS exits the enlarged box or the root-box, it is terminated, and the function evaluations executed by AS are discarded. If it converges to a point outside the original box but inside the enlarged box, the point location is saved. In both cases, the C-RTS combinatorial component continues in the normal way: the next box $B^{(t+1)}$ is the best one in the admissible neighborhood of $B^{(t)}$. In any case, the "best so far" value is always updated by considering all admissible points evaluated (those that are inside the root-box).

A possible exception to the normal C-RTS evolution can happen only in the event that AS converges inside $B^{(t)}$ to a local minimum X_l. If X_l is the first local minimum found, it is saved in a memory structure associated to the box. If a local minimum Y_l was already present, and X_l corresponds to the same point, it is discarded, otherwise the current box is *split* until the "siblings" in the tree divide the two points. After the splitting is completed, the current box $B^{(t)}$ no longer corresponds to an existing leaf: to restore legality, a point is selected at random with uniform distribution in $B^{(t)}$ and the legal $B^{(t)}$ becomes the leaf-box that contains the random point. Therefore, each leaf-box in the partition of the initial box has a probability of being selected that is proportional to its volume. The splitting procedure is explained in the following section.

5.5. BOX SPLITTING

As stated before, as soon as two different local minima X_l and Y_l are identified in a given box B, the current box is subdivided into 2^N equal-sized boxes. If X_l and Y_l belong to two different leaf-boxes of the new partition, the splitting is terminated;

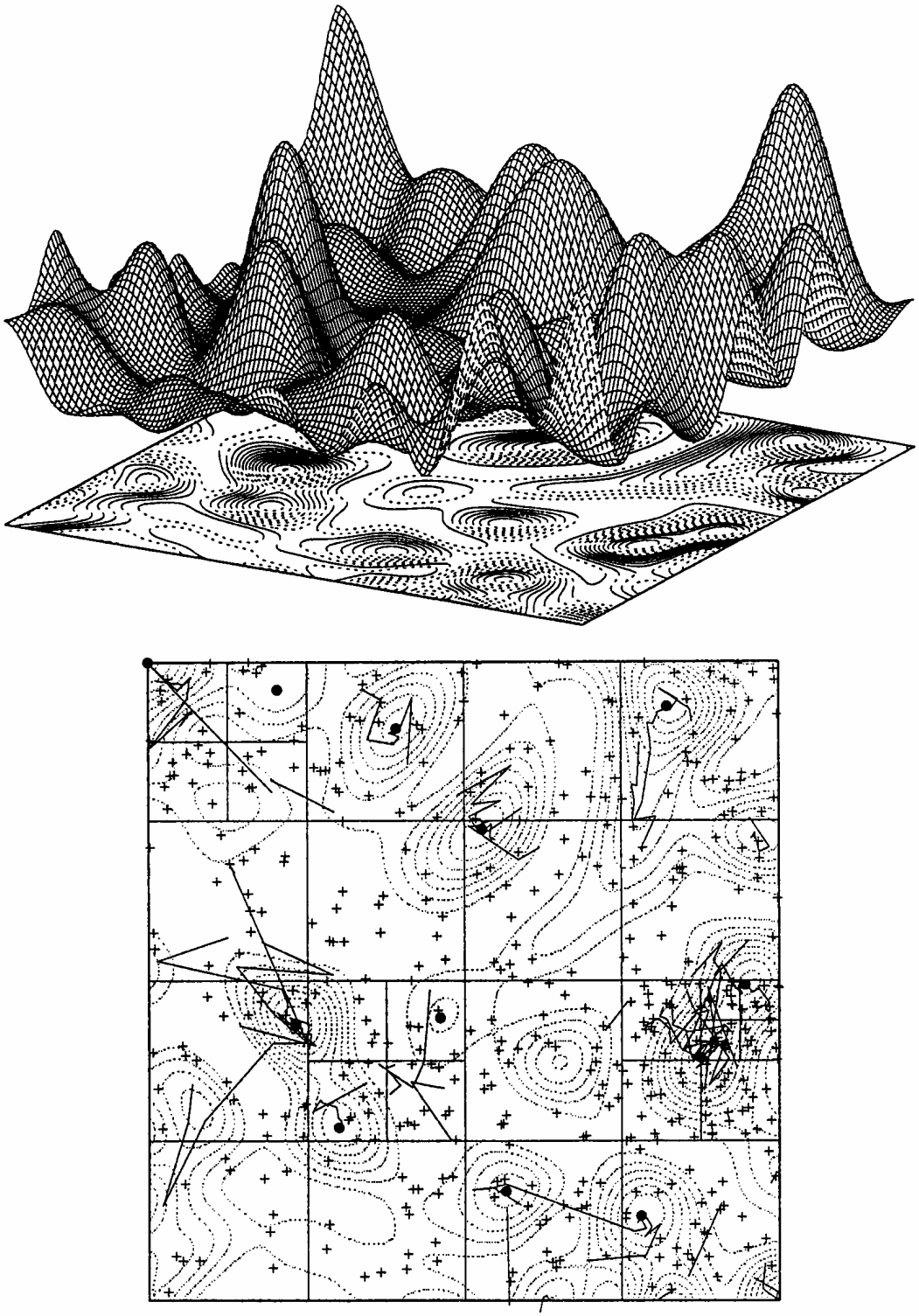

Figure 4. Strongin function upside-down (top), C-RTS tree structure and calculated points (bottom). Evaluated points (crosses), local minima (bullets), AS trajectories (jagged lines).

otherwise, the splitting is applied to the box containing X_l and Y_l until their separation is obtained.

In all cases, the old box ceases to exist and it is substituted with the collection obtained through the splitting. The local minima X_l and Y_l are associated with their new boxes. If a new box does not contain one of the two minima, it does not consume CPU time and RAM memory: the first point will be calculated only if (and when) the box is selected as a neighbor to be evaluated. Numerically, the criterion used in the tests for considering two *different* local minima X_l and Y_l is that $||X_l - Y_l|| < \varepsilon$, where ε is the precision requirement introduced in section 4.

All local minima identified are saved and reported when C-RTS terminates. This is useful in many engineering applications, if one is interested not only in obtaining the best minimum, but in having a series of alternatives with similar f values that can possibly be ranked according to additional criteria not contained in f.

An example of the tree structure produced during a run of C-RTS is shown in figure 4 for the case of a two-dimensional function (a "Strongin" function described in appendix II). The function of equation (9) is squared and plotted upside-down so that the local minima are clearly visible as "mountain tops". One notices that the points evaluated (the points used to calculate $f(B)$) are distributed quasi-uniformly over the search space: this is a result of our volume-proportional selection described in sections 5.2 and 5.4, and it guarantees that all regions of the search space are treated in a fair manner. The AS trajectories either converge to a local minimum (bullet) or are terminated when they exit from the enlarged box, as explained in section 5.4. Because of our splitting criterion, each box contains at most one local minimum. Although it is not visible from the figure, most points (about 85% in the example) are evaluated during the AS phases, which are the most expensive parts of the C-RTS algorithm.

6. Performance tests

Most tests presented are executed on widely known benchmark functions. Minimal information about them (function form, search region, references) is placed in appendix II, mainly to identify the test functions with no ambiguities (some editing mistakes and/or slightly different versions of the same functions are present in the literature). All tests have been executed without any individual tuning of the algorithm parameters.

6.1. TESTS ON "DIXON AND SZEGO" BENCHMARK

In this part, the performance of C-RTS is compared with that of various strategies considered in the collection of Dixon and Szego [15]. The original reference should be consulted for all details.

Table 1

Benchmark number of function evaluations. L indicates that a local minimum was found.

Method	SQRN5	SQRN7	SQRN10	Hartman3	Hartman6	GOLDPR	RCOS
Bremmermann	340 L	1700 L	500 L	505 L	L	210 L	250
Mod. Bremm.	375 L	405 L	336 L	420 L	515	300	160
Zilinskas	L	12121 L	8892 L	8641			5129
Gomulka/Branin	5500	5020	4860				
Torn	3679	3606	3874	2584	3447	2499	1558
Gomulka/Torn	6654	6084	6144				
Gomulka/V.M.	7085	6684	7352	6766	11125	1495	1318
Price	3800	4900	4400	2400	7600	2500	1800
Fagiuoli	2514	2519	2518	513	2916	158	1600
De Biase/Front.	620	788	1160	732	807	378	597
Mockus	1174	1279	1209	513	1232	362	189
C-RTS-ave	812	960	921	513	750	248	38
C-RTS-min	664	871	693	609	1245	171	41

The average number of function evaluations used by C-RTS obtained in a total of 1000 runs for each function is shown in table 1 and compared with the values obtained with alternative algorithms considered in [15] (the table is reported from [9]). The statistical error on the C-RTS averages is about 3%. The number of function evaluations used by C-RTS on the test problems is comparable to the lowest numbers obtained with the competitive techniques. C-RTS-ave and C-RTS-min refer to the two options for evaluating boxes (using the average or the minimum value). Note that the differences between the two versions of C-RTS are not very significant: the method is robust with respect to this choice. The C-RTS algorithm does not use the fact that the above functions are differentiable: an additional reduction in the number of function evaluations could be obtained by using a more appropriate derivative-based local minimizer (e.g., the conjugate gradient or pseudo-Newton techniques).

In [15], a "standard time unit" is defined to compare the CPU times needed by the different algorithms. A standard time unit is the time needed for 1000 evaluations of SQRN5. Table 2 reports the standard times used by C-RTS (average of 1000 runs) in comparison with those used by the different algorithms. Given the simplicity of the test functions, their evaluation is very fast and most time is spent in the initialization part, memory allocation, and search algorithm. One unit corresponds to about 5 milliseconds on state-of-the-art workstations. The average wall-clock time of C-RTS-min on our machine[2] is 183 milliseconds for the slowest

[2] Hp 747i, 100 MHz clock, C++ compiler and libraries from the GNU foundation.

Table 2

Performance measured in standard time units. L indicates that a local minimum was found.

Method	SQRN5	SQRN7	SQRN10	Hartman3	Hartman6	GOLDPR	RCOS
Bremmermann	11	8 L	17 L	2 L	L	0.5 L	1 L
Mod. Bremm.	1.5 L	1.5 L	2 L	2 L	3	0.7	0.5
Zilinskas	L	282 L	214 L	175			80
Gomulka/Branin	9	8.5	9.5				
Torn	10	13	15	8	16	4	4
Gomulka/Torn	17	15	20				
Gomulka/V.M.	19	23	23	17	48	2	3
Price	14	20	20	8	46	3	4
Fagiuoli	7	9	13	5	100	0.7	5
De Biase/Front.	23	20	30	16	21	15	14
MP package	96	258	1059	117	255	1407	1.5
C-RTS-ave	18	20	18	8	22	3.6	0.4
C-RTS-min	10	11	14	7	35	1.7	0.3

task (the Hartman6 function) and 1.5 milliseconds for the fastest one (the RCOS function), although a high-level programming language was used (C++).

6.2. COMPARISON ON DISCONTINUOUS FUNCTIONS

Stuckman [34] presents a global method that generalizes Kushner's one-dimensional method based on a "Brownian motion" model of the function to be minimized [29]. The method is meant to address functions with many extrema, possibly non-differentiable, with the aim of reducing the number of function evaluations at the expense of increased computation. The algorithm works by expanding the set of evaluated points: first, the vertices of the initial search domain are evaluated, then, at each iteration, all possible segments connecting two arbitrarily evaluated points are considered and the probabilistically-best point along these segments is evaluated and added to the set. Although the model is probabilistic, the algorithm is deterministic: a single sequence of points is visited for a given function. Unfortunately, if the model is not appropriate or if the global optimum lies in a "hole" that is not covered by the segments, the algorithm will converge to a suboptimum local minimum. In addition, its computational complexity grows as $O(m^3)$, where m is the total number of visited points, or as $O(m^2)$ if only a fixed number of segments is considered for each point (the ones that are closest to the newest point).

In the test phase, Stuckman introduces a class of functions designed to have more than one local minimum, with zero or non-existing gradient and frequent discontinuities (see appendix II), in order to "complete" Dixon and Szego's benchmark, which contains only everywhere-differentiable functions. The surface and contour plot for one function of this class are shown in figure 5. Let us note that the function presents narrow strips where the partial derivatives are equal to zero, separated by discontinuities. The surface plot highlights the overall structure but smooths over the narrow steps.

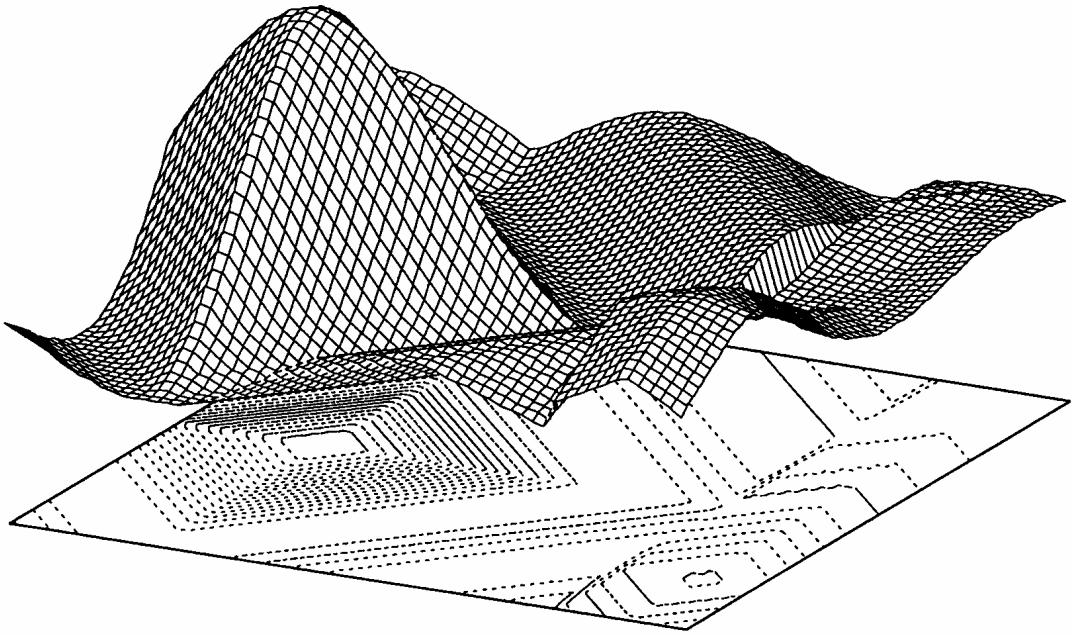

Figure 5. A Stuckman function: surface and contour plot.

On a set of 100 randomly generated functions, [34] found that 1000 evaluations were needed to reach the global minimum in 97 out of 100 cases. In a second series of tests (with different parameters) the algorithm always terminated in less than 1000 iterations and it found the global optimum in 96 out of 100 cases.

A set of 100 randomly generated Stuckman functions was created to test the C-RTS algorithm. For each function of the class, a total of 10 C-RTS runs were executed to average over the different random starts of C-RTS. The number of converged runs out of 1000 as a function of the number of function evaluations executed is illustrated in figure 6. Only 8 cases in 1000 require more than 1000 evaluations and the algorithm always locates the global optimum in less than 7000 iterations.

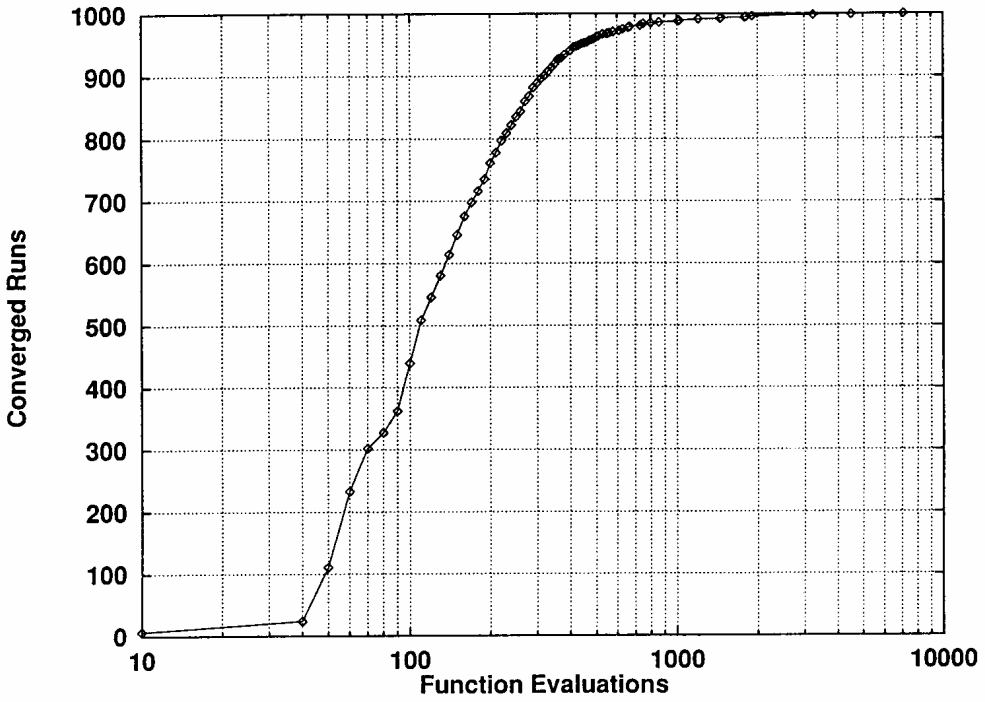

Figure 6. Solved problems for the Stuckman class out of a total of 1000 runs: 100 different functions, 10 runs each.

6.3. COMPARISON WITH TREE ANNEALING

A popular algorithm for combinatorial optimization is the simulated annealing algorithm of [27]. For the purpose of continuous optimization, the SA technique was adapted in [9] for the *Tree Annealing* (TA) algorithm. The TA algorithm bears some resemblance to our technique in that it is based on a locally adapted tree, although TA does not use a local continuous minimizer and therefore the splitting of a leaf is not triggered by the detection of local minima, but by the acceptance of contained points. In addition, the splitting in TA acts on each degree of freedom in turn (at depth n, the split is along dimension n mod N). The test results on the Dixon and Szego benchmark functions that were used in section 6.1 are shown in table 3. For convenience, the C-RTS-ave results have been duplicated.

The crucial differences between C-RTS and TA are both the RTS tabu dynamics and the use of the affine shaker as a local minimizer in promising boxes. In particular, while the basic SA algorithm is Markovian (the current step does not depend on the moves executed in the past), C-RTS is not (the current step *depends* on the past history) and therefore redundant exploration and/or the confinement of the search in a suboptimal portion of the search space can be effectively discouraged.

Table 3

Benchmark: C-RTS versus Tree Annealing number of function evaluations and percent success in finding the global optimum. Last line: standard time units of tree annealing.

Method	SQRN5	SQRN7	SQRN10	Hartman3	Hartman6	GOLDPR	RCOS
C-RTS-ave	812 100%	960 100%	921 100%	513 100%	750 100%	248 100%	38 100%
Tree annealing	3700 40%	2426 60%	3463 50%	1113 100%	17262 100%	6375 100%	4172 100%
TA std. time	11	5.8	8.6	1.4	46.4	6.3	10.4

For a fair comparison, let us note that TA is reasonably fast on this benchmark, although it does not always find the global optimum in the given CPU time.

6.4. COMPARISON WITH STRONGIN–SERGEYEV

A recent method for the global optimization of functions satisfying the Lipschitz condition $|f(X)-f(Y)| < L|X-Y|$ has been proposed in [33]. Multi-dimensional problems are reduced to one-dimensional tasks by using Peano-type space-filling curves. The experiments described in [33] for the two-dimensional function class of equation (9) in appendix II give an average number of function evaluations between 1575 and 1599 (four tests, each with 100 randomly-extracted functions of the class, were executed).

The number of evaluations required by C-RTS (averages over 100 functions of the class, for each function 10 different random initializations) is 1621 (std. dev. 105). Figure 7 reports the number of runs solved to optimality as a function of the number of evaluations. Some comments are appropriate. The technique of Strongin and Sergeyev guarantees the convergence to the global minimum, *provided* that the Lipschitz condition is satisfied with a certain L, and that the algorithm parameters are appropriately tuned. If either the Lipschitz condition is not satisfied (e.g., for discontinuous functions) or L is not known, it can be useful to try a parameter-free algorithm like C-RTS. We expected that C-RTS would use many more function evaluations, but the opposite conclusion had to be drawn for the given function class. Unfortunately, [33] does not report results for different and higher-dimensional problems.

6.5. TRANSFORMER DESIGN (CONSTRAINED TASK)

Although the C-RTS technique is not designed for constrained optimization tasks, we decided to run some tests on the "transformer design" task, described in appendix II, by using the *penalty method* to take care of constraints. The main

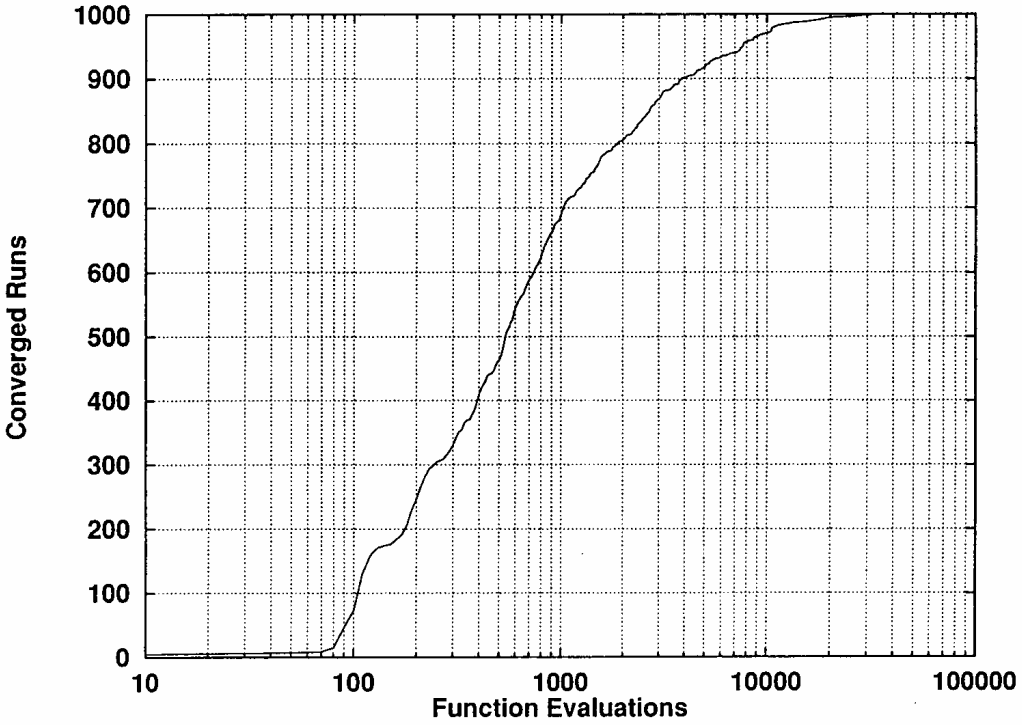

Figure 7. Solved problems for the Strongin–Sergeyev class out of a total of 1000 runs: 100 different functions, 10 runs each.

purpose of these tests was that of exploring the limits of C-RTS. By using the penalty function technique, the final objective function is defined as in [31]:

$$F = f + 1,000,000\,[\min(0, f_1)^2 + \min(0, f_2)^2], \tag{4}$$

where $f_1 \geq 0$ and $f_2 \geq 0$ are the two constraints.

The C-RTS algorithm needs an initial search region that, given the constraints, can be assumed to be contained in the domain $x_i \geq 0$. Tests have been executed with a growing initial search area, to measure how the number of function evaluations grows when the root-box is expanded. In detail, the root-box is centered at the point $X_B = (5.5, 4.5, 10.5, 12.5, 0.5, 0.5)$. The range on each independent variable is 1 for the smallest volume considered: $X_{B_i} - 1/2 \leq x_i \leq X_{B_i} + 1/2$. Then the range is increased in a series of steps s as follows: $X_{B_i} - s/2 \leq x_i \leq X_{B_i} + s/2$, for $i = 1, 2, 3, 4$, and $0 \leq x_i \leq X_{B_i} + s$, for $i = 5, 6$ (because of the constraint, the variables cannot become negative). The parameter s is varied from 2 to 9, therefore the search volume grows from 1 to $9^6 = 531{,}441$. Note that the initial points are chosen with a uniform distribution inside the root-box and therefore the fact that the global optimum is close to the center of the box is not a particular case. The first series of tests measures the effort to determine a point within 4% of the globally optimal

Table 4

Function evaluations as a function of search volume for the transformer task (within 4% of optimum, averages of 100 tests and std. dev. in parentheses). The success ratio is reported only if some runs are not successful.

Volume of search space	C-RTS-min	C-RTS-ave
1	311 (19)	334 (19)
3^6	745 (52)	24,381 (6802)
5^6	3,370 (305)	35,540 (12525) 53/100
7^6	7,604 (732)	71,044 (21873) 31/100
9^6	19,701 (1191)	59,135 (8729) 49/100

value as a function of the total search volume. Table 4 reports the average number of function evaluations over 100 runs for the two versions of C-RTS.

The increase in the number of function evaluations for growing search volumes is acceptable (when the volume increases from 1 to 531,411, the number of evaluations increases by a factor of about 185 for the C-RTS-min version). Part of the merit is due to the AS. In [32, 13], it is shown that the AS computation grows logarithmically with the volume of the search space. It can be noted that the version C-RTS-min is now significantly better than the version that uses the average. The version C-RTS-ave needs many more function evaluations and it is not always successful (in the allowed maximum of 50,000 C-RTS iterations), especially when the search space is large. The reason is that, because of the penalty in equation (4), the F values for points that violate the constraints are huge and dominate the average, so that the evaluation $F(B)$ is not a reliable indicator of the f values obtainable for admissible points. The use of the minimum value ameliorates the performance if at least one admissible point is used in the $F(B)$ evaluation.

In table 5, we report the average number of function evaluations (in 100 runs) needed to reach close approximations of the global optimum. It can be noted that the amount of effort to reach minimal values close to the optimum is acceptable up

Table 5

Function evaluations as a function of the desired precision of the optimal value (from 4% to 0.1% of the exact optimum).

% above exact optimum	C-RTS-min	C-RTS-ave
4	331 (19)	334 (19)
1	1,464 (124)	1,186 (111)
0.5	5,288 (608)	3,445 (535)
0.1	99,722 (4707)	172,485 (24127) 83/100

to about 1% and grows very rapidly to reach a much higher precision. This behavior is not typical for the unconstrained tasks: on the contrary, given the fast convergence of AS once the global optimum region has been identified, the effort needed to increase the relative precision is usually a small fraction of the total C-RTS work. After some investigation, it became clear that the explanation for the poor performance is again related to the box evaluation: because the global minimum lies on a constraint boundary, the box that contains it will also contain points that *violate* the constraints (unless the optimum lies at a vertex of the box, an event with zero probability). Therefore, the box evaluation $F(B)$ can be dominated by the illegal points (especially for the C-RTS-ave version), with the effect that suboptimal boxes that are *near* the constraints but do not contain points that violate them tend to get better evaluations with respect to the box containing the global optimum. The approximated solution approaches the feasibility border when more splits are executed, but the total effort required grows. A simple improvement would be that of changing the box evaluation so that the effect of non-admissible points is controlled.

While the design of a version of C-RTS for a constrained task is left for future work, the results obtained are promising. As a comparison, [31] used the Controlled Random Search (CRS3) algorithm for the same task. CRS3 combines a local optimization procedure (based on the Nelder–Mead simplex algorithm) and a global optimizer (CRS2). CRS2 starts by evaluating an empirical number of trial points chosen at random that are added to a set A of "working points". Then, at each iteration a new trial point P is selected by using the information in the current set A. P substitutes the worst point in A if it has a lower function value. The experimental performance of CRS3 is 3234 evaluations (average) to reach a solution within 4% of the optimal value, and 11,737 for the final convergence (the exact precision is not given).

6.6. SCALING WITH DIMENSION AND PROBLEM COMPLEXITY

The functions considered in the above tests were chosen because of their wide usage in the scientific literature. As was previously noted, most of these functions are relatively fast to compute (by current machines). On the other hand, the comparison of all cited algorithms on more complex functions is not an easy task, both because of the software effort required in duplicating the algorithms, and because some details are missing in the description of some methods.

In our opinion, this "reproducibility" issue is avoidable only by making the original software available, especially for complex algorithms. We decided to present some additional experiments about the use of C-RTS for problems of larger dimensions and/or more expensive functions (in terms of the CPU time to evaluate them), and to place our software in the public domain to permit independent tests by interested researchers (details are available from the authors).

Table 6

Function (F), gradient (G), and Hessian (H) evaluations (when needed) as a function of the dimension of Levy's functions.

Dimension	F evals. C-RTS (std. dev.)	F, G evals. tunneling	F, G, H evals. interval
3	278 (20)	7325 1328	177 146 57
5	341 (28)	7540 1122	300 310 99
8	858 (99)	19366 2370	464 405 150
10	1207 (135)	23982 3272	559 497 184

Table 7

Function evaluations and execution time as a function of the dimension of RBF functions. Standard deviation in parentheses.

Dimension	Function evaluations	Time in std. units	Cost of a function eval. in std, units
2	3218 (525)	1562 (253)	0.496 (0.080)
4	86778 (19914)	80061 (18287)	0.908 (0.083)
8	105403 (28282)	260642 (70279)	2.625 (0.122)
16	799354 (155905)	5436010 (1060166)	6.967 (0.236)

Table 6 compares the results obtained by C-RTS (C-RTS-min, 32 tests for each function) on the Levy's functions family (see [25] and appendix II) with those obtained by the Tunneling method of [25,26] and by the Interval method of [22]. C-RTS has a competitive number of function evaluations and does not need any gradient or Hessian computations. The scaling of the number of function evaluations with respect to the problem dimension makes C-RTS usable also for tasks of larger dimensions.

To test C-RTS with functions of a much higher complexity, we designed a family of radial basis functions (RBF), described in appendix II. A member of this function class is shown in figure 8 (512 Gaussians with different metrics and random positions and heights are superposed in the two-dimensional plane). Table 7 shows the average results obtained on 32 tests (dimensions 2, 4, and 8) and 16 tests (dimension 16). The average cost of a single function evaluation is also reported. Let us note that the evaluation time *depends* on the argument and has a fairly large standard deviation because of the floating point algorithms used by the machine (this variation should be taken into account in a proper definition of a "standard time unit"). Most of the CPU time is in this case consumed by the evaluation of the function, even for the low-dimensional problems. This result was expected: the

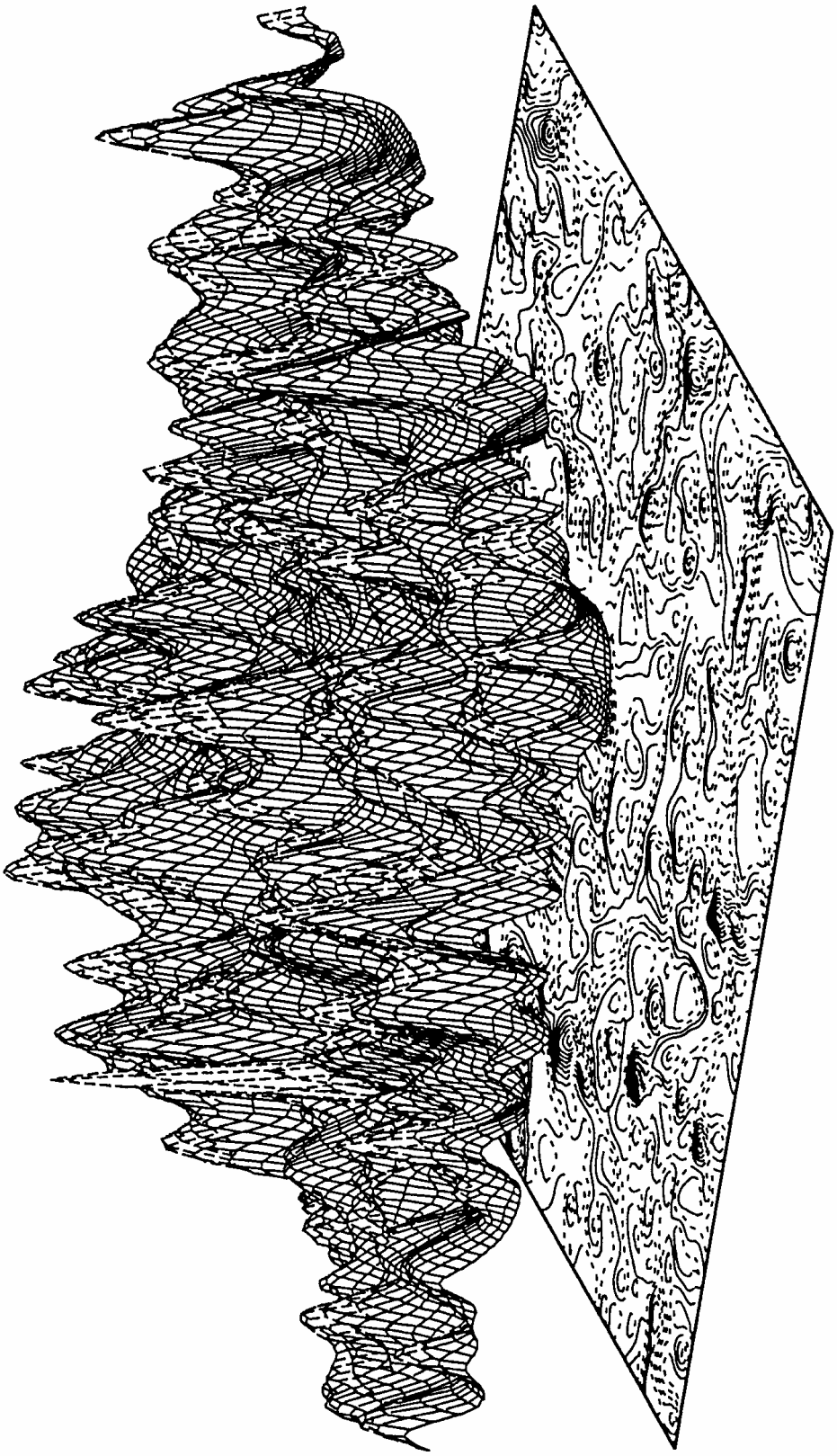

Figure 8. RBF function.

low *overhead* of C-RTS is negligible with respect to the function evaluation work, unless the function is very trivial to compute. The C-RTS algorithm is applicable without difficulty up to dimension 16.

7. Discussion and conclusion

We have presented a hybrid algorithm for the global optimization of functions, in which a fast combinatorial component (the reactive tabu search) identifies promising boxes in a tree-like partition of the initial search space, and a stochastic minimizer (the affine shaker algorithm) finds the local minimum in a given attraction basin with high precision.

Some characterizing points of the C-RTS algorithm are the following:

- **General-purpose optimization**: no requirements are placed on the function to be optimized. Although C-RTS is naturally faster and more efficient for smooth functions, it is effective for non-differentiable and even discontinuous functions.
- **Global optimization**: while the AS stochastic search component identifies a local optimum in a given attraction basin, the combinatorial component favors jumps between different basins, with a *bias* toward regions that plausibly contain good local optima.
- **Multi-scale search**: the use of grids at different scales in a tree structure is used to spare CPU time in slowly-varying regions of the search space and to intensify the search in critical regions.
- **Simplicity, reaction and adaptation**: the algorithmic structure of C-RTS is simple. The few parameters of the method are adapted in an automated way during the search by using the information derived from memory. The intensification–diversification dilemma is solved by using intensification until there is evidence that diversification is needed (when too many boxes are repeated excessively often along the search trajectory). The tree-like discretization of the search space in "boxes" is activated by evidence that the current box contains more than one attraction basin.
- **Tuneable precision**: the global optimum can be located with high precision both because of the local adaptation of the grid size and because of the decreasing sampling steps of the stochastic AS when it converges.
- **Minimal computational complexity and efficient memory use**: the CPU time and RAM size required for each point evaluated are minimal and approximately constant (of order $O(1)$).

Although C-RTS correctly solved a wide selection of test problems, naturally it is not guaranteed to find the global optimum for all possible functions. None-

theless, the algorithm tends to be robust even for "deceiving" functions, e.g., when the position of the global optimum is not related to the average function values in the different regions, or the attraction basin of the global optimum is small and hidden in a region with large average function values. In these cases, because of the escape mechanism, the diversification part will predominate in the long run and C-RTS will behave in a similar way to a "random walk with memory".

This work did not analyze termination criteria because they are dependent on specific (possibly probabilistic) function models and because this study focused on an algorithm that could cover a wide spectrum of tasks, including the cases in which the user has little knowledge of the detailed properties of the function to be minimized. In the absence of a specific model, C-RTS employs the allotted computing time in an efficient way: the waste of function evaluations in frequently visited regions is discouraged via the automated diversification and the escape mechanism. Nonetheless, model-based termination can be added (for example, by adapting [10, 16]). Termination criteria appropriate for specific function classes and the modification of C-RTS for constrained tasks are the themes of future work.

Some tests on functions with up to sixteen variables have been presented, in which the global optimum is always reached with competitive computing times and number of function evaluations. When the number of variables increases, one can expect a rapid growth of the computing resources required, unless the function is very smooth and the number of attraction basins grows slower than exponentially in the dimension. Given the rich variety of existing algorithms, the performance comparison had to be limited to a small selection of related algorithms and test functions. The authors will make the C-RTS software available to non-profit parties for testing C-RTS on specific problems of interest.

Acknowledgements

We acknowledge the support of INFN (Initiative RTS), and of the Special Project of the Department of Mathematics of the University of Trento. S. Struthers kindly checked the English. Thanks are due to the anonymous referees for clarifying some parts of the paper.

Appendix I: The RTS algorithm

To make this paper self-contained, this appendix summarizes the RTS algorithm of [3] and [5], and specifies the modifications executed to obtain the C-RTS version. Two background references for the basic tabu search strategy are [18] and [19].

To realize the RTS prohibition rules, the most recent iteration when each move μ_i has been applied is recorded and each configuration $X^{(t)}$ touched by the search trajectory is stored in memory with the most recent time when it was encountered. Let us introduce the functions:

- $\Lambda(\mu)$: the last iteration when μ has been used ($\Lambda(\mu) = -\infty$ if μ has never been used).
- $\Pi(X)$: the last iteration when X has been encountered ($\Pi(X) = -\infty$ if X has not been encountered or if it is not in the memory).
- $\Phi(X)$: the number of repetitions of configuration X in the search trajectory ("repetition counter"). At the beginning, $\Phi(X) = 0$ for all configurations.

The prohibition period $T^{(t)}$ is initialized with a small value (e.g., $T^{(0)} \leftarrow 1$), and then adapted by *reacting* to the occurrence of repetitions. Note that checking the *tabu* status of a move requires only a couple of CPU cycles if the function $\Lambda(\mu)$ is realized with an array in memory.

To simplify the description, the algorithm is illustrated with figures based on an elementary pseudo-language. Constructs of this pseudo-language are simple selection (conditional) and iteration keywords (**if ... then ... else**, **repeat ... until**, **repeat for ... to ...**), the assignment operator ($X \leftarrow Y$ means that the value of variable X is overwritten with the value of Y) and functions that can return values to the calling routine. Compound statements are indented, function names are in boldface, comments and descriptions are in italics.

Figure 9 describes the main structure of the *Reactive Tabu Search* for the case of a CO problem on binary strings of length L. Let us recall that in C-RTS, the length L depends on the depth n of the box in the tree and the number of independent variables N: $L = Nn$. The initialization part is followed by the main loop, that continues to be executed until a satisfactory solution is found or a limiting number of iterations is reached. In the first statement of the loop, the current configuration is compared with the previously visited points stored in memory by calling the function **memory_based_reaction** (figure 10), that returns two possible values (DO_NOT_ESCAPE or ESCAPE). In the first case, the next move is selected from the set of admissible moves (the rules for obtaining the list size ensure that this set is not empty); in the second case, the algorithm enters a diversification phase based on a short random walk, see the function **diversify_search** (figure 9). In this phase, one executes a number of random steps that is sufficient to obtain a substantially different string ($n_{\max} N/4$ steps, and in any case at least two, where $n_{\max}$ is the maximum depth of the tree). At each step, the complete neighborhood is considered and a random neighbor is selected. As soon as each step is executed, the corresponding move becomes tabu, so that, when the "escape" phase finishes, coming back to the starting configuration is discouraged. Each new configuration on the trajectory with a lower f value is saved with the associated configuration f, because otherwise this point could be lost when the trajectory escapes from a local minimum. The couple (X_b, f_b) is the suboptimal solution provided by the algorithm when it terminates.

When *RTS* is applied to "difficult" tasks (like NP-complete problems [17], for which no polynomial algorithms have been designed), one must settle for a

```
procedure reactive_tabu_search
(Initialize the data structures for tabu:)
t ← 0 (iteration counter); T_F^(0) ← 1/N (fractional prohibition period); t_T ← 0 (last time T_F was changed);
C ← ∅ (set of often-repeated configurations); R_ave ← 1 (moving average of repetition interval);
X^(0) ← random X ∈ 𝒳 (initial configuration); X_b ← X^(0) (best so far X); f_b ← f(X^(0)) (best so far f);
repeat
    (See whether the current configuration is a repetition:)
    escape ← memory_based_reaction(X^(t))    (see Fig. 10)
    if escape = DO_NOT_ESCAPE then
        μ ← arg min_{ν∈𝒜^(t)} f(ν(X^(t)))
        X^(t+1) = μ(X^(t))
        Λ(μ) ← t        (𝒜^(t) and T^(t) are therefore implicitly changed)
        (Update time, and best_so_far:)
        t ← (t + 1)
        if f(X^(t)) < f_b then
            f_b ← f(X^(t))
            X_b ← X^(t)
    else
        diversify_search
until f_b is acceptable or maximum no. of iterations reached

function diversify_search
comment: A sequence of random steps, that become tabu as soon as they are applied.
    repeat for i = 1 to Max(2, n_max N/4)    (at least two moves)
        σ_i ← move corresponding to a random step        (see text for details)
        X^(t+1) ← σ_i(X^(t))
        Λ(σ_i) ← t    (𝒜^(t) and T^(t) are therefore changed)
        (Update time, and best_so_far:)
        t ← (t + 1)
        if f(X^(t)) < f_b then
            f_b ← f(X^(t))
            X_b ← X^(t)
```

Figure 9. *RTS* algorithm: main structure.

suboptimal solution within the allotted amount of CPU time. The avoidance of cycles and confinements ensures that the available time is spent in an efficient exploration of the search space, and specific termination conditions (apart from the expiration of the given CPU time) are not necessary. Naturally, specific task-dependent termination criteria can be introduced, for example by setting a threshold on the quality of the solution.

```
function memory_based_reaction(X)
comment: The function returns ESCAPE when an escape is to be executed, DO_NOT_ESCAPE otherwise.
[ Search for configuration X in the memory:
if Π(X) > −∞ then      (if X was visited previously)
    [ Find the cycle length, update last_time and repetitions:
    R ← t − Π(X)      (R = repetition interval)
    Π(X) ← t
    Φ(X) ← Φ(X) + 1      (Φ(X) = repetitions of X)
    if Φ(X) > REP then
        [ C ← C ∪ X      (X is added to the set of often-repeated configurations)
        if |C| > CHAOS then
            [ C ← ∅
            T_F^(t+1) ← 1/N      (reset fractional list size)
            t_T ← t
            return ESCAPE (reaction III)
    if R < 2 (L − 1) and last visit time of X was before last escape then
        [ R_ave ← 0.1 × R + 0.9 × R_ave
        T_F^(t+1) ← Min(T_F^(t) × INCREASE, 1)      (reaction I)
        t_T ← t
else
    [ If the configuration is not found, install it and set:
    Π(X) ← t
    Φ(X) ← 1

if (t − t_T) > R_ave then
    [ T_F^(t+1) ← Max(T_F^(t) × DECREASE, 1/Nn)      (reaction II)
    t_T ← t
return DO_NOT_ESCAPE
```

Figure 10. *RTS* algorithm: the function **memory_based_reaction**.

The reactive mechanisms of the algorithm modify the discrete dynamical system that defines the trajectory so that *limit cycles* and confinements in limited portions of the search space are discouraged. The reaction is based on the past history of the search and it causes possible changes of $T^{(t)}$ or the activation of a diversifying phase. Short limit cycles are avoided by modifying $T^{(t)}$ in the appropriate way. In particular, see the function **memory_based_reaction** defined in figure 10, the current configuration X is compared with the configurations visited previously and stored in memory. If X is found, its last visit time $\Pi(X)$ and repetition counter $\Phi(X)$ are updated. Then, if its repetitions are greater than the threshold REP, X is included in set C, and if the size $|C|$ is greater than the threshold CHAOS, the

function returns immediately with the value ESCAPE. For all the presented tests, CHAOS = 3, REP = 3: the diversification is activated as soon as more than three configurations are encountered more than three times along the search. C-RTS is robust with respect to this choice. If the repetition interval R is sufficiently short (if $R < 2(L-1)$ so that the cycle *can* be avoided by using the tabu dynamics [5]), one can discourage cycles by increasing $T^{(t)}$ in the following way: $T^{(t+1)} \leftarrow T^{(t)} \times$ INCREASE. If X is not found, it is stored in memory, the most recent time when it was encountered is set to the current time ($\Pi(X) \leftarrow t$), and its repetition counter is set to one ($\Phi(X) \leftarrow 1$). In the case of C-RTS, the binary string length L depends on the depth n of the leaf-box in the tree. In detail, $L = N \times n$, N being the problem dimension. In addition, the cycles considered for increasing T and for calculating R_{ave} are those happening after the last escape.

If T is not allowed to decrease, there is the danger that its value will remain large after a phase of the search with many repetitions, even in later phases, when a smaller value is sufficient to avoid short cycles. Therefore, the statement labeled *reaction II* in figure 10 executes a reduction by the factor DECREASE < 1 if $T^{(t)}$ remained constant for a number of iterations greater than the moving average of repetition intervals R_{ave}. In our tests, the reaction parameters are INCREASE = 1.1, DECREASE = 0.9.

Let us note that the dynamic tabu rule illustrated in section 5.3 guarantees that $\mathcal{A}^{(t)}$ contains at least two admissible moves (if the binary string has at least two bits), see equation (2). The best move is selected by testing only the admissible moves.

When the reaction that modifies $T^{(t)}$ (*reaction I* and *II* in figure 10) is not sufficient to guarantee that the trajectory is not confined in a limited portion of the search space, the search dynamics enter a phase of "random walk" (*reaction III* in figure 10) that is specified by the statements in the function **diversify_search** of figure 9. The first time that a cycle is detected after last escape, R_{ave} becomes equal to this cycle length. Note that the execution time of the random steps is registered ($\Lambda(\sigma) \leftarrow t$), so that they become tabu. After the "random walk" phase terminates, the prohibition of the most recent random steps discourages the trajectory from returning to the old region.

Appendix II: Test functions

Here, we list the minimal amount of information sufficient to identify the various test functions used. See the appropriate references for additional details. The fractional precision ε is equal to 10^{-3} for all functions, apart from the RCOS function ($\varepsilon = 10^{-2}$) and the Stuckman class ($\varepsilon = 10^{-1}$). The random number generator used in all tests is the **drand48**( ) routine that is standard in the Unix operating system.

Shekel SQRNm

These are a family of four-dimensional functions [15] defined as

$$f(X) = -\sum_{i=1}^{m} \frac{1}{(X - A_i)^T (X - A_i) + c_i}, \tag{5}$$

where m gives the number of local minima ($m = 5, 7, 10$ in our tests), c_i are related to the local minima values, and A_i are four-dimensional vectors that determine the position of the local minima. The constants are given in table 8. The search region is $0 \leq X \leq 10$.

Table 8

Data for Shekel SQRN functions.

i	$(A_i)_1$	$(A_i)_2$	$(A_i)_3$	$(A_i)_4$	c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.01	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

Hartman

This family of functions [23] with different dimensions has the following form:

$$f(X) = -\sum_{i=1}^{m} c_i \exp\left(-\sum_{j=1}^{N} \alpha_{ij}(x_j - p_{ij})^2\right), \tag{6}$$

where N is the dimension, the vector p_i is the approximate location of the ith local minimum, the vector α_i is proportional to the eigenvalues of the Hessian at the ith local minimum, and $c_i > 0$ is related to the value of the ith local minimum (assuming that the interference with other local minima is weak). We consider the case $N = 3$, $m = 4$ (Hartman3) and $N = 6$, $m = 4$ (Hartman6). The defining constants are given in tables 9 and 10. The region of interest is $0 \leq X \leq 1$.

Goldstein and Price (GOLDPR)

This two-dimensional test function is derived from [20]:

Table 9

Data for Hartman3 function.

i	α_{i1}	α_{i2}	α_{i3}	c_i	p_{i1}	p_{i2}	p_{i3}
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.3815	0.5743	0.8828

Table 10

Data for Hartman6 function.

i	α_{i1}	α_{i2}	α_{i3}	α_{i4}	α_{i5}	α_{i6}	c_i	p_{i1}	p_{i2}	p_{i3}	p_{i4}	p_{i5}	p_{i6}
1	10.00	3.00	17.00	3.50	1.70	8.00	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10.00	17.00	0.10	8.00	14.00	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3.00	3.50	1.70	10.00	17.00	8.00	3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	17.00	8.00	0.05	10.00	0.10	14.00	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

$$f(X) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]. \quad (7)$$

The region of interest is $-2 \le X \le 2$.

Branin RCOS

The function is derived from [11]:

$$f(X) = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e, \quad (8)$$

where $a = 1$, $b = 5/(4\pi^2)$, $c = 5/\pi$, $d = 6$, $e = 10$, and $f = 1/(8\pi)$. The region of interest is $-5 \le x_1 \le 10$, $0 \le x_2 \le 15$.

Strongin

This two-dimensional function class is used in [33]:

$$\phi(X) = -\sqrt{\left(\sum_{i=1}^{7}\sum_{j=1}^{7}[A_{ij}a_{ij}(X) + B_{ij}b_{ij}(X)]\right)^2 + \left(\sum_{i=1}^{7}\sum_{j=1}^{7}[C_{ij}a_{ij}(X) - D_{ij}b_{ij}(X)]\right)^2}, \quad (9)$$

where $a_{ij}(X) = \sin(i\pi x_1)\sin(j\pi x_2)$, $b_{ij}(X) = \cos(i\pi x_1)\cos(j\pi x_2)$, and A_{ij}, B_{ij}, C_{ij}, D_{ij} are random coefficients in $[-1, 1]$. The search region is $0 \le x_1 \le 1$, $0 \le x_2 \le 1$.

Transformer design

The six-variable problem arises in connection with transformer design [1] and has been derived from [30] (the description in [31] presents editing mistakes). The function to minimize is:

$$f(X) = 0.0204x_1x_4(x_1 + x_2 + x_3) + 0.0187x_2x_3(x_1 + 1.57x_2 + x_4) + 0.0607x_1x_4x_5^2(x_1 + x_2 + x_3) + 0.0437x_2x_3x_6^2(x_1 + 1.57x_2 + x_4) \quad (10)$$

subject to the inequality constraints:

$$x_i \ge 0, \quad i = 1, \ldots, 6, \quad (11)$$

$$f_1 = x_1x_2x_3x_4x_5x_6 - 2070.0 \ge 0, \quad (12)$$

$$f_2 = 1.0 - 0.00062x_1x_4x_5^2(x_1 + x_2 + x_3) - 0.00058x_2x_3x_6^2(x_1 + 1.57x_2 + x_4) \ge 0. \quad (13)$$

The variables x_1, x_2, x_3, x_4 are physical dimensions of transformer parts, x_5 and x_6 are the magnetic flux density and current density.

The complete domain is $x_i \ge 0$, different search regions containing the minimum ($x_1 = 5.33$, $x_2 = 4.66$, $x_3 = 10.43$, $x_4 = 12.08$, $x_5 = 0.752$, $x_6 = 0.878$) are considered in the tests.

Stuckman

The two-variable function class introduced in [34] is defined as follows:

$$f(X) = -\begin{cases} \lfloor(\lfloor m_1 \rfloor + 1/2)(\sin(a_1)/a_1)\rfloor, & 0 \le x_1 \le b, \\ \lfloor(\lfloor m_2 \rfloor + 1/2)(\sin(a_2)/a_2)\rfloor, & b < x_2 \le 10, \end{cases} \quad (14)$$

where $a_1 = |x_1 - xr_{11}| + |x_2 - xr_{21}|$, $a_2 = |x_1 - xr_{12}| + |x_2 - xr_{22}|$. The defining constants are uniform random variables, defined as:

b	random	$\in [0, 10]$,
m_j	random	$\in [0, 100]$,
xr_{11}	random	$\in [0, b]$,
xr_{12}	random	$\in [b, 10]$,
xr_{ij}	random	$\in [0, 10]$ in the two remaining cases.

The search region is $0 \le X \le 10$.

Levy

The Levy benchmark functions are introduced in [25]. They are defined as follows:

$$f(X) = \sin^2(\pi y_1) + \sum_{i=1}^{N-1}(y_1 - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_N - 1)^2, \qquad (15)$$

where N is the dimension, $y_i = 1 + (x_i - 1)/4$. The search region is $-10 \le x_i \le 10$. The function has 125 local minima for $N = 3$, 10^5 for $N = 5$, 10^8 for $N = 8$, and 10^{10} for $N = 10$.

RBF

The Radial Basis Functions are a superposition of 512 Gaussian functions. The metric for the kth Gaussian is given by a positive-definite matrix $g^{(k)}$, its center is the vector $c^{(k)}$ with random components in the region [0, 1] and its height $h^{(k)}$ is random in [0, 100]. Each matrix $g^{(k)}$ is obtained by first computing a diagonal matrix $D^{(k)}$ with random components in the range [0, 8192] and then applying $r = N(N-1)/2$ rotations $R_1, R_2, \ldots, R_r$. Each R_i is a Givens matrix [21], representing a random rotation. In this way, one obtains a general (non-diagonal) positive-definite matrix. In detail:

$$\begin{aligned} f(X) &= \sum_{k=1}^{512} h^{(k)} \exp\left\{-\sum_{i,j=1}^{N}(x_i - c_i^{(k)})g_{ij}^{(k)}(x_j - c_j^{(k)})\right\}, \\ h^{(k)} &= rand(100), \\ c_i^{(k)} &= rand(1), \\ g^{(k)} &= R_r^T \ldots R_2^T R_1^T D^{(k)} R_1 R_2 \ldots R_r. \end{aligned} \qquad (16)$$

The search region considered in the tests is $0 \le x_i \le 1$.

References

[1] D.H. Ballard, C.A. Jelinek and R. Schinzinger, An algorithm for the solution of constrained generalized polynomial programming problems, Comp. J. 17(1974)261–266.

[2] R. Battiti and G. Tecchiolli, Parallel biased search for combinatorial optimization: genetic algorithms and TABU, Microproc. Microsyst. 16(1992)351–367.

[3] R. Battiti and G. Tecchiolli, The reactive tabu search, ORSA J. Comp. 6(1994)126–140.

[4] R. Battiti and G. Tecchiolli, Learning with first, second and no derivatives: A case study in high energy physics, Neurocomp. 6(1994)181–206.

[5] R. Battiti and G. Tecchiolli, Training neural nets with the reactive tabu search, IEEE Trans. Neural Networks 6(1995)1185–1200.

[6] R. Battiti, The reactive tabu search for machine learning, in: *Proc. GAA '93, Giornate dei Gruppi di Lavoro AI*AI, Apprendimento Automatico*, Milan (1993).

[7] R. Battiti and G. Tecchiolli, Local search with memory: Benchmarking RTS, Preprint UTM, University of Trento, Italy (October, 1993), submitted.
[8] R. Battiti and G. Tecchiolli, Simulated annealing and tabu search in the long run: A comparison on QAP tasks, Comp. Math. Appl. 28(6) (1994) 1–8.
[9] G.L. Bilbro and W.E. Snyder, Optimization of functions with many minima, IEEE Trans. Syst., Man, Cybern. SMC-21(1991)840–849.
[10] C.G.E. Boender and A.H.G. Rinnooy Kan, Bayesian stopping rules for multistart global optimization methods, Math. Progr. 37(1987)59–80.
[11] F.H. Branin, Jr., Widely convergent method for finding multiple-solutions of simultaneous nonlinear equations, IBM J. Res. Develop. (September 1992) 504–522.
[12] R. Brunelli and G. Tecchiolli, On random minimization of functions, Biol. Cybern. 65(1991) 501–506.
[13] R. Brunelli and G. Tecchiolli, Stochastic minimization with adaptive memory, J. Comp. Appl. Math. 57(1995)329–343.
[14] R. Brunelli, On training neural nets through stochastic minimization, Neural Networks (1994).
[15] L.C.W. Dixon and G.P. Szego (eds.), *Towards Global Optimization 2* (North-Holland, 1978).
[16] S. Fanelli and A. Ramponi, Computational experience with a multistart algorithm for global optimization based on new Bayesian stopping rules, in: *Proc. 14th Symp. A.M.A.S.E.S.*, Pescara (September 1990).
[17] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, 1979).
[18] F. Glover, Tabu search – Part I, ORSA J. Comp. 1(1989)190–206.
[19] F. Glover, Tabu search – Part II, ORSA J. Comp. 2(1990)4–32.
[20] A.A. Goldstein and I.F. Price, On descent from local minima, Math. Comp. 25 (July 1971).
[21] G.H. Golub and C.F. Van Loan, *Matrix Computations*, 2nd ed. (The Johns Hopkins University Press, 1990).
[22] E. Hansen, *Global Optimization Using Interval Analysis* (Marcel Dekker, 1992).
[23] J.K. Hartman, Technical Report NP55HH72051A, Naval Postgraduate School, Monterey, CA (1972).
[24] J.H. Holland, *Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control, Artificial Intelligence* (University of Michigan Press, 1975).
[25] A.V. Levy, A. Montalvo, S. Gomez and A. Galderon, *Topics in Global Optimization*, Lecture Notes in Mathematics No. 909 (Springer, 1981).
[26] A.V. Levy and S. Gomez, The tunneling method applied to global optimization, in: *Numerical Optimization 1984*, ed. P.T. Boggs, R.H. Bryd and R.B. Schnabel (SIAM Publications, 1985) pp. 213–244.
[27] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, Science 220(1983)671–680.
[28] D.E. Knuth, *The Art of Computer Programming*, Vol. 3: *Sorting and Searching* (Addison–Wesley, 1973).
[29] H.J. Kushner, A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise, in: *Proc. Joint Automatic Control Conf.* (1963).
[30] W.L. Price, Global optimization by controlled random search, J. Optim. Theory Appl. 40(1983) 333–348.
[31] W.L. Price, Global optimization algorithms for a CAD workstation, J. Optim. Theory Appl. 55(1987)133–146.
[32] F.J. Solis and R.J-B Wets, Minimization by random search techniques, Math. Oper. Res. 6(1981) 19–30.
[33] R.G. Strongin and Y.D. Sergeyev, Global multidimensional optimization on parallel computers, Parallel Comp. 18(1992)1259–1273.
[34] B.E. Stuckman, A global search method for optimizing nonlinear systems, IEEE Trans. Syst., Man, Cybern., SMC-18(1988)965–977.