Tabu Search

Introduction

Webster's dictionary defines the word *tabu* or *taboo* to mean "banned on grounds of morality or taste or as constituting a risk...". Tabu Search (TS) is an optimization method designed to help a search negotiate difficult regions (i.e. to escape from local minima or to cross infeasible regions of the search space) by imposing restrictions. It was originally developed as a method for solving combinatorial optimization problems (these are problems where the control variables are some form of ordered list — the *Travelling Salesman Problem* is the classic example), but variants to solve continuous and integer optimization problems have also been developed. We will focus here on TS for continuous optimization problems.

The power of TS derives from its use of flexible memory cycles. These memory cycles control the local search pattern and intensify and diversify the search in the quest for a suitable solution.

Search Control

Local Search

Local search in the particular variant of TS under discussion is performed using a modified version of Hooke and Jeeves search. The local search process is now:

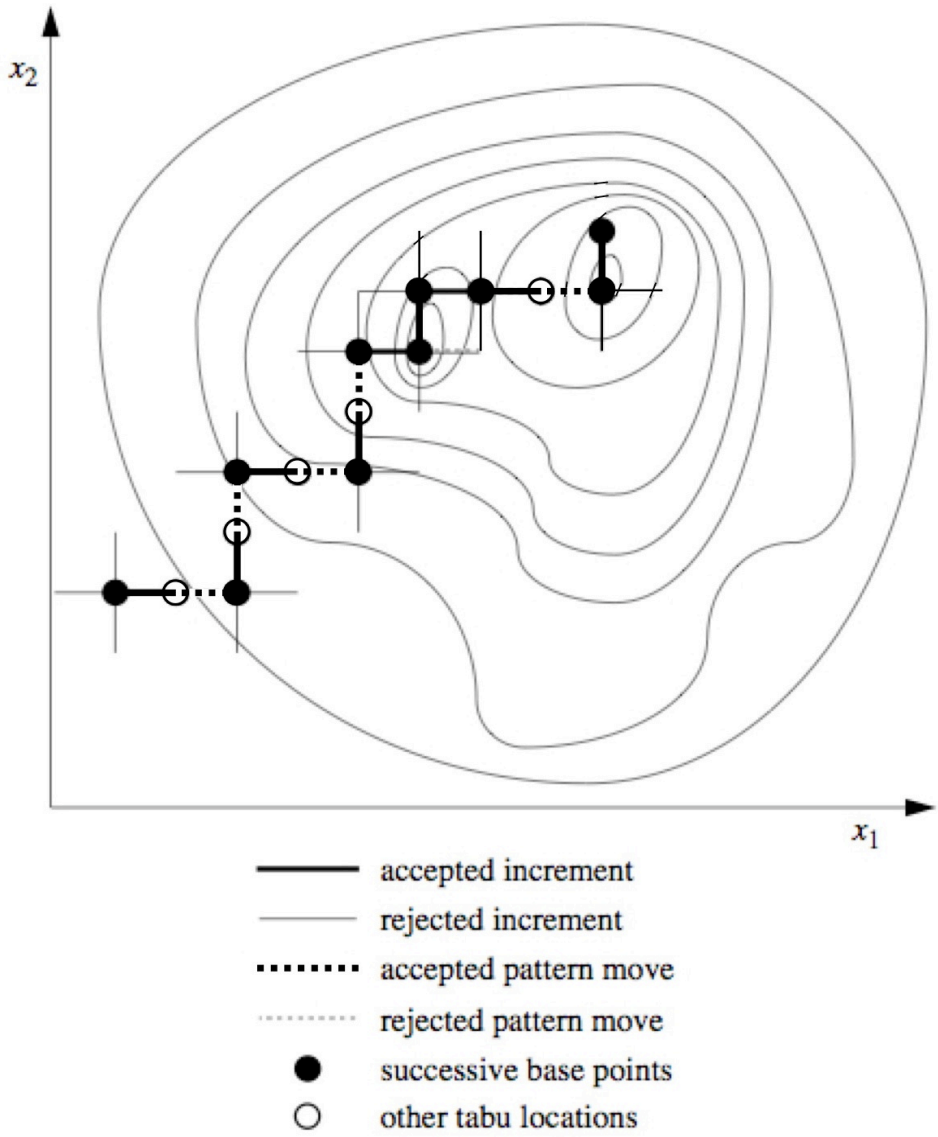
1. Choose a *starting point*. Evaluate the objective function $f(\mathbf{x})$. Choose a suitably-sized increment for each variable $x_1, x_2, \ldots, x_n$.
2. Record the current point as the base point.
3. For each variable in turn:
   - Increase it by its increment, i.e. $x_i + \delta_i$, while keeping the other variables at their base point values, and evaluate the objective function.
   - Decrease it by its increment, i.e. $x_i - \delta_i$, while keeping the other variables at their base point values, and evaluate the objective function.
4. Make the best *allowed* (i.e. non-tabu) move of those evaluated in 3.
5. If the objective function was reduced by the move in 4, perform a *pattern move* by repeating the vector from the last base point. If this reduces the value of the objective function further, retain it, otherwise discard it.
6. Record the current point as the new base point. Return to step 3.

Whether a move is allowed or not is determined by the *Short Term Memory* (STM). This records the last N locations successfully visited (i.e. accepted in steps 4 or 5 above). This list is updated every time a new location is successfully visited (on a first in, first out basis). Loca-

tions within the STM are *tabu*, i.e. may not be visited again. The appropriate value for N depends on the number of control variables. $N = 7$ has been found to give good performance on problems with up to a dozen control variables. If the number of control variables is greater than this N should be increased appropriately (a value slightly greater than half the number of control variables is about right). Note that this requirement that the best allowed move be made means that often this move results in an objective function increase.

The effect of the STM is that the algorithm behaves like a normal hill descending algorithm until it reaches a minimum. It is then forced to climb out of the hollow and explore further.

Figure 1: An Example of TS Local Search.



This is illustrated by the example shown in Figure 1. This shows a contour plot of a two-dimensional optimization problem containing one local and one global optimum. From the starting point the TS algorithm rapidly locates the neighbourhood of the local minimum, without the STM having any effect. However, once the search has reached this local minimum, it is

forced by the tabu restrictions to move away from the minimum in the direction which increases the objective function least. Because the last N visited locations are tabu, the search cannot leave the way it came (although in the example shown this would not happen anyway) and once it has left the minimum it cannot return directly to it. The algorithm therefore forces the search onwards and, in this example, it soon locates the neighbourhood of the global minimum.

Note that in this local search the step sizes associated with the control variables are not reduced, as they are in the traditional Hooke and Jeeves method. Step size reduction is implemented differently in TS.

Search Intensification

Search Intensification (SI) in TS is a process associated with the *Medium Term Memory* (MTM). The MTM records the best M solutions located thus far, i.e. the M solutions with the lowest objective function values. For small scale problems (12 or fewer control variables) $M = 4$ has been found to give good performance.

The MTM is exploited as follows. If there has been no improvement in the best solution found for a specified number (perhaps 10 for small scale problems) of local search iterations, then the search is *intensified*. This entails moving the current search location $\mathbf{x}_{k+1}$ to an "average best" position, e.g.

$$\mathbf{x}_{k+1} = \frac{1}{M} \sum_{j=1}^{M} \mathbf{y}_j \,, \tag{1}$$

where $\mathbf{y}_j$ are the control variable values of the M solutions in the MTM.

SI is performed in order to focus the search in neighbourhood of the best solutions found without visiting any of them again.

Search Diversification

If SI does not succeed in finding an improved best solution, i.e. further (perhaps 5 for small scale problems) local search iterations pass without a new best solution being located, then the search is diversified. *Search Diversification* (SD) is associated with the *Long Term Memory* (LTM). The LTM records the areas of the search space which have been searched reasonably thoroughly, perhaps by dividing the search space into sectors and tallying the number of locatsions successfully visited in each sector. When SD takes place the current search location is moved to a randomly selected part of the search space which has not been thoroughly searched.

SD is performed in order to investigate previously unexplored parts of the search space, if no further progress is being made in areas already well explored.

Step Size Reduction

After prolonged lack of success, i.e. if no new best solution has been found for perhaps 25 (for small scale problems) local search iterations despite SI and SD, then *Step Size Reduction* (SSR) takes place. In SSR the step sizes associated with each control variable are reduced, the search is returned to the location of the best solution found thus far and the counter controlling SI, SD and SSR is reset.

SSR is carried out in order to ensure that the neighbourhood of the best solution is searched carefully.

Convergence

Like all optimization algorithms, TS is terminated when a specified convergence criterion is met. As in the traditional Hooke and Jeeves algorithm, the convergence criterion can be that the step sizes have been reduced below specified thresholds. An alternative possible criterion is that, despite SI, SD and SSR, the same best location is repeatedly being found.

Search Control Summary

The overall search control strategy in TS can be summarised as follows:

1. Set `COUNTER` = 0
2. Test convergence
3. If converged, stop;

   If not, perform a local search iteration
4. If a new best solution has been found, go to step 1;

   If not, increment `COUNTER`
5. If `COUNTER` = `INTENSIFY`, intensify search
6. If `COUNTER` = `DIVERSIFY`, diversify search
7. If `COUNTER` = `REDUCE`, reduce step size and go to step 1
8. Go to step 2

For a small scale problem (up to 12 control variables) suitable values of the three control thresholds might be `INTENSIFY` = 10, `DIVERSIFY` = 15, and `REDUCE` = 25. For larger scale problems these parameters should be increased appropriately.

Local Search Variants

The local search at the heart of the basic TS implementation can make the algorithm particularly computationally expensive if the evaluation of the problem functions (objective and con-

straints is expensive) and the number of control variables is large. In these circumstances it may be better to conduct a partial rather than an exhaustive local search at each iteration. There are a number of ways of doing this.

Random Subset

In this approach, at each iteration a random subset of the possible non-tabu moves are evaluated and the best allowed amongst these moves is made.

Successive Random Subsets

In this approach, at each iteration a random subset of the possible non-tabu moves are evaluated. If any of these moves improve on the current solution the best allowed move is made. If none of the moves represents an improvement, another subset of the possible non-tabu moves are evaluated. This process continues until either an improving move is found or all possible moves have been evaluated. If no improving, non-tabu move is found the best allowed (non-improving) move is made.

Variable Prioritisation

In variable prioritisation schemes an attempt is made to identify which control variables have the greatest influence on the objective function and then these are prioritised in the local search. One such scheme works as follows:

1. Conduct an exhaustive search around the current search point.
2. Calculate the sensitivity of the objective function to each control variable, i.e.

$$s_i = \left\| \frac{f(\mathbf{x} + \delta_i) - f(\mathbf{x} - \delta_i)}{2\delta_i} \right\| \tag{2}$$

3. Rank the control variables according to the values of s_i. Select the $n/2$ highest ranked variables to be free variables, and fix the values of the others at their current values.
4. Continue the search using this reduced set of variables for a defined number of iterations. Then go to 1.

Constraints

Some thought needs to be given to the handling of constraints when using the TS algorithm. In many cases the routine can simply be programmed to reject any proposed changes which result in constraint violation, so that a search of feasible space only is executed. However, there are three important circumstances in which this approach cannot be followed:

- if there are any equality constraints defined on the system,
- if it is suspected that the problem is highly constrained, so that generating feasible solutions

is not easy, or

- if the feasible space defined by the constraints is (suspected to be) disjoint, so that it is not possible to move between all feasible solutions without passing through infeasible space.

In any of these cases the problem should be transformed into an unconstrained one by constructing an augmented objective function incorporating any violated constraints as penalty functions:

$$f_A(\mathbf{x}) = f(\mathbf{x}) + \mathbf{w}^T \mathbf{c}_V(\mathbf{x}) , \tag{3}$$

where $\mathbf{w}$ is a vector of nonnegative weighting coefficients and the vector $\mathbf{c}_V$ quantifies the magnitudes of any constraint violations. It may also improve algorithm performance if the degree of penalisation is increased as the search progresses, so that it is biased increasingly heavily towards feasible space.

TS has been found to perform particularly well (in comparison to other heuristic algorithms) on highly constrained problems. This is due to the fact that the local search mechanism at its heart is much more likely to maintain feasibility of solutions (once feasible space has been located) than an algorithm that progresses by making larger changes in solutions.

Example: Rosenbrock's Function $f(\mathbf{x}) = 100\,(x_2 - x_1^2)^2 + (1 - x_1)^2$

Figures 2 to 5 show the progress of a TS run on Rosenbrock's function. Although one would not ordinarily choose to use TS on a problem which is amenable to solution by more efficient methods, it is interesting to do so for purposes of comparison.

Figure 2 shows the initial local search, the successfully visited locations being displayed. Starting from (–1, 1) the initial local search efficiently locates the optimum at (1, 1) and is then forced onwards by the tabu restrictions. [Note that many of the individual steps made result in objective function increases.] After 10 further iterations, in which of course no improvement in the best solution proves possible (the optimum has already been found!). The search is intensified, i.e. relocated to the average of the best four (in this case) solutions found at (0.925, 0.875). The progress from this point is shown in Figure 3 (marked by the square symbols). After 5 further iterations, the search is diversified (because no new best solution has been found) and leaps to (0.285, –0.455). The progress from this point is also shown in Figure 3 (marked by the triangular symbols). After 10 further iterations, step size reduction takes place (again because no new best solution has been found).

Figure 2: Minimization of Rosenbrock's Function by TS — Initial Local Search.

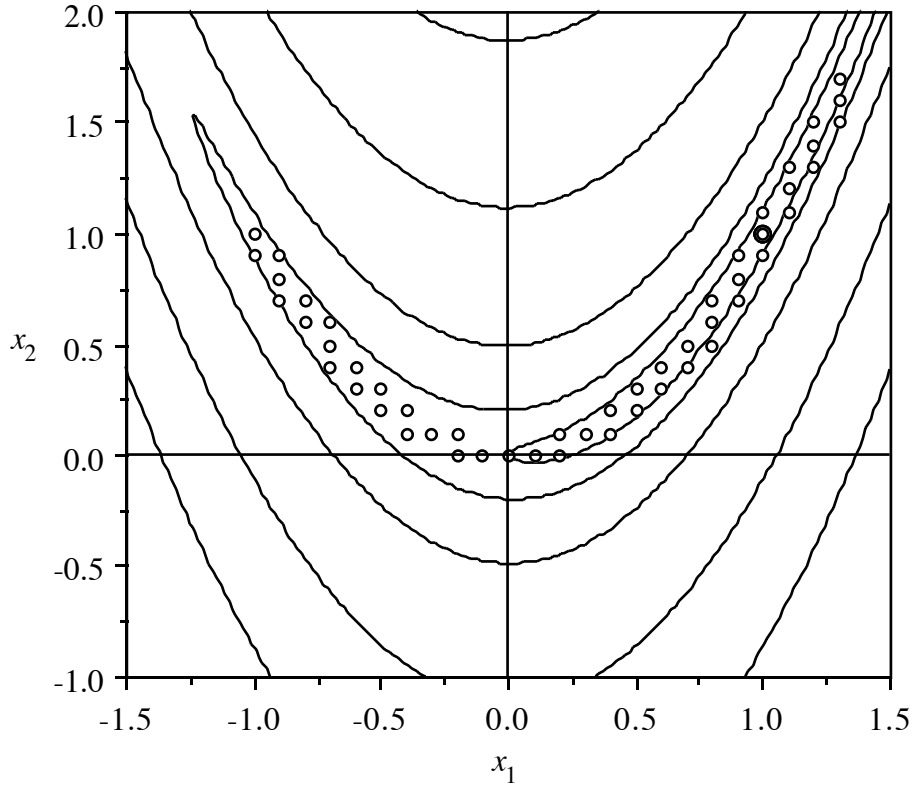


Figure 3: Minimization of Rosenbrock's Function by TS — SI and SD.

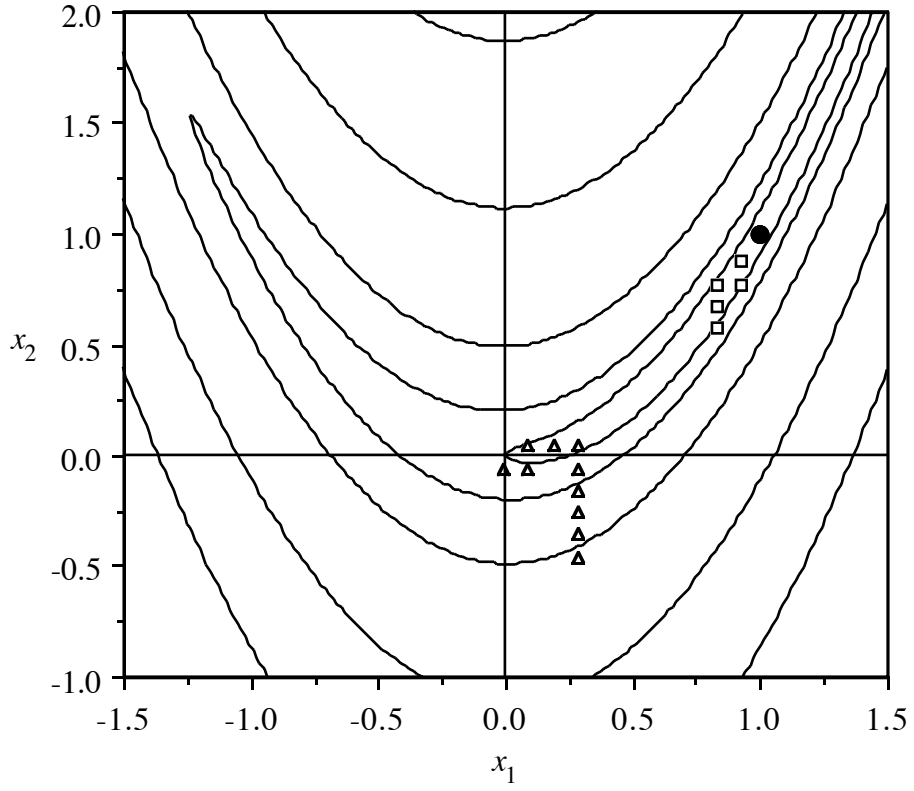


Figure 4: Minimization of Rosenbrock's Function by TS — SSR, SI and SD.

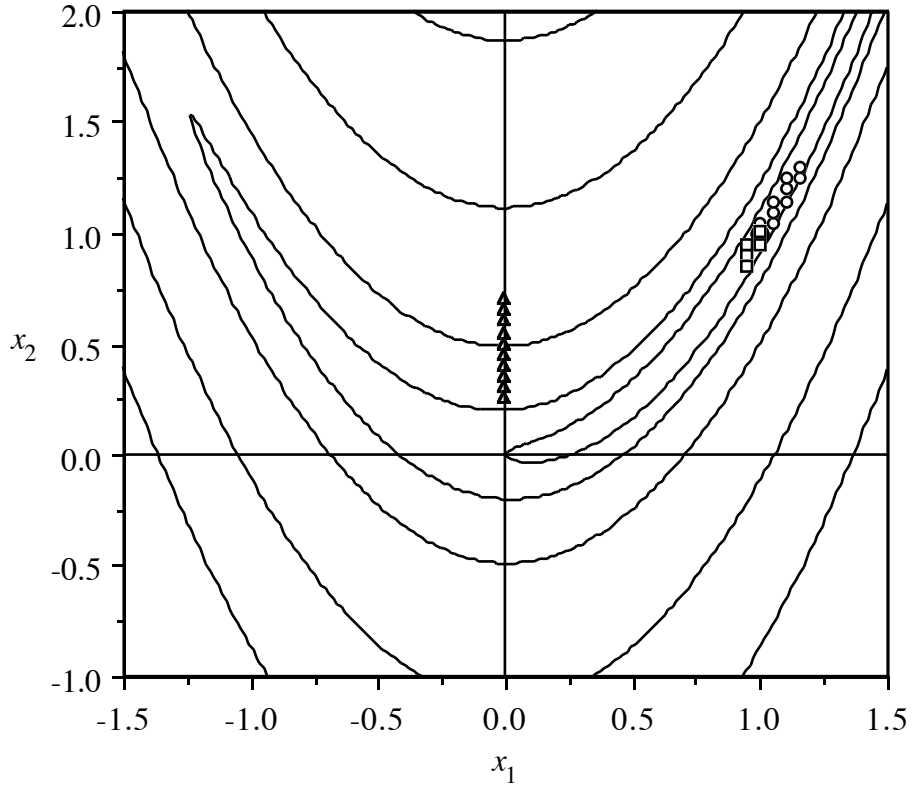
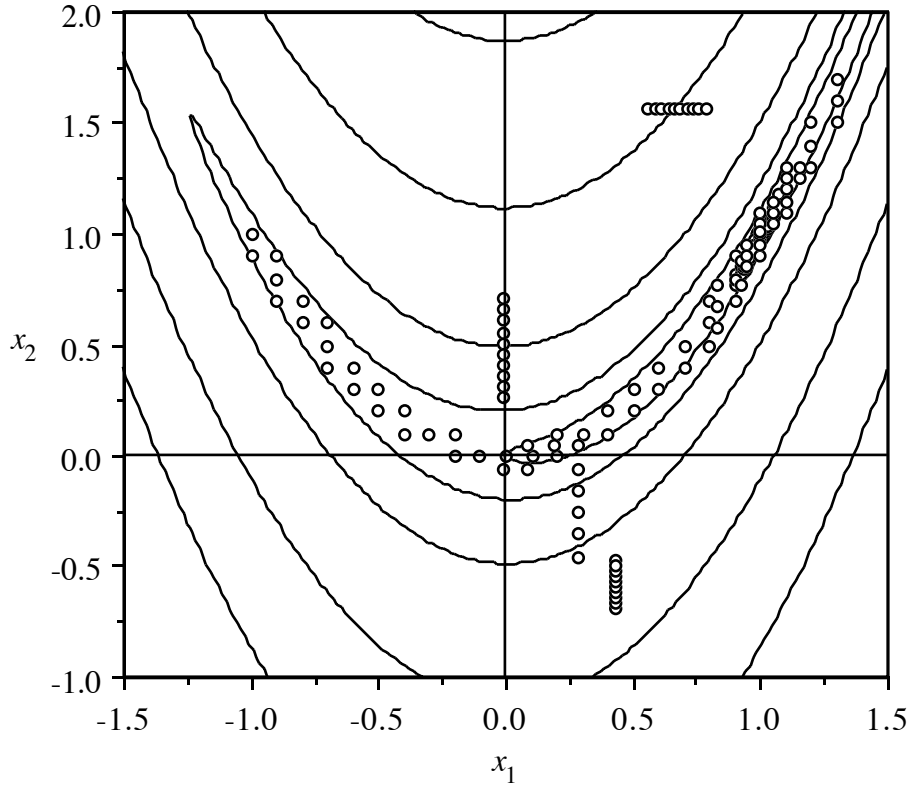


Figure 5: Minimization of Rosenbrock's Function by TS — Complete Search.



Following SSR the search restarts from the best solution found $(1, 1)$ with the step sizes halved. As shown in Figure 4, a cycle of initial local search (circular symbols) is followed by SI and further search (square symbols) and then SD and further search (triangular symbols).

Figure 5 shows the complete search pattern performed by TS before convergence was achieved (the step sizes were sufficiently small). The patterns produced by SI, SD and SSR cycles are readily apparent.

Obviously in this case, because the exact optimum was found on the first local search, the SI procedure achieves nothing, but this would not be the case in general. Similarly, the SSR procedure achieves nothing except enabling the convergence criterion to be met. Again this would not be the case in general. The SD procedure increases confidence that the minimum which has been found is indeed the global minimum by sampling other parts of the search space.