Tabu searching for robust solutions

Theoretical framework

Kenneth Sörensen
University of Antwerp UA
Faculty of Applied Economic Sciences UFSIA–RUCA
Middelheimlaan 1
B-2020 Antwerp — Belgium
kenneth.sorensen@ua.ac.be

September 6, 2002

Abstract

In this paper, we investigate how tabu search techniques can be adapted so that they find solutions that (1) have a good solution quality and (2) are more robust than other solutions. We show that there is a need for robust solutions in many practical problems and discuss different types of robustness. We show that tabu search and other local search techniques can be effectively used to find solutions that are both robust and of high quality. The concept of *robust evaluation function* is introduced. In a robust evaluation function, a number of solutions are first perturbed and then combined into a single measure of both robustness and solution quality. We show that this concept extends tabu search so that it searches for robust solutions without requiring large modifications to the tabu search algorithm itself, thus rendering the technique very flexible and practically useable.

The proposed procedure is tested for a simple tabu search procedure, the optimisation of a continuous function of a single variable on a finite domain. For this type of problems, we find an approximation of the number of perturbed evaluations that are needed to create a good robust evaluation function.

Contents

1 Introduction

Tabu search is a *metaheuristic* that guides a local heuristic search procedure to efficiently explore the search space of a problem. It uses both short-term and long-term memory structures to prevent the search from getting stuck in local optima. For a detailed description of this technique, we refer to Glover (1989), Glover (1990) and Glover and Laguna (1999). The best-known solutions to many problems have been found by cleverly using a tabu search algorithm.

Research on tabu search and metaheuristics in general, however, has neglected the need for robust solutions that exists in many real-life problems. We define robustness as *the insensitivity of a solution with respect to changes in the environment in which this solution is implemented*. If a solution is very sensitive to small perturbations in the input data, it might not be a good solution at all. Therefore, we develop a technique that modifies a tabu search heuristic so that it searches for solutions that not only are of good quality but are also robust.

In this paper, we examine some very simple examples that show the effectiveness of the procedure, leaving extensive tests on real-life examples for future work. Since the principles of the procedure described in this paper are independent of the tabu search technique used (and to some extent even of the search technique used), we limit ourselves to very basic tabu searches. The aim of this paper is mainly to introduce the concept of robust evaluation function and derive some theoretical results.

The aspect of robustness has been studied in the past and some techniques to evaluate and improve the robustness of the solution to a problem with regard to changes in input data exist. One of the first attempts to estimate the robustness of the solution to a mathematical programming problem is sensitivity analysis (Dupačová, 1987; Dupačová, 1990). Although very useful, the main drawback of this technique is that it does not search for robust solutions. Sensitivity analysis techniques are by definition a-posteriori techniques that are only used to evaluate the robustness of a solution found. Some other techniques include robust optimisation (Mulvey et al., 1995), stochastic (linear) programming (Dantzig, 1955; Beale, 1955; Wets, 1974; Kall and Wallace, 1994), surface response techniques (Box and Wilson, 1951; Myers et al., 1989; Box and Draper, 1987; Giovannitti-Jensen and Myers, 1989). These techniques all have their roots in operational research and offer powerful tools to find robust solutions to several problems. They are however very hard to use in conjunction with local search techniques such as tabu search.

Others (Taguchi methods (Kackar, 1985; Krottmaier, 1993)) find their roots in other sciences (mainly engineering).

A recent and closely related way to improve the robustness of solutions found by genetic algorithms is GAs/RS^3 (genetic algorithms with a robust solution searching scheme) (Tsutsui et al., 1996; Tsutsui and Ghosh, 1997). This technique finds robust solutions by adding a Gaussian noise to the *phenotypic parameters* (the decoded *genotypes*). Doing this reduces the height of narrow peaks in the evaluation function and therefore allows the GA to find "wider" peaks, that correspond to more robust solutions. The amount of noise to add to the solution can be approximately determined by assuming a certain width for the sharp peaks and calculating a *reduction factor*. The reduction factor determines the amount by which the height of the narrow peaks will be reduced. Tsutsui and Jain (1998) later expand GA/RS^3 to problems with a multi-dimensional search space. The effect of adding several perturbations to the phenotypic parameters (instead of a single one) is examined in Tsutsui (1999). It is found that adding a single perturbation is preferable.

A similar approach is found in Branke (1998). Here also, a (normally distributed) noise is added to the phenotypes before they are evaluated. The main emphasis of this research is also on genetic algorithms and optimisation of continuous functions of one variable.

This paper can be regarded as an extension of the research of Branke (1998) and Tsutsui et al. (1996), extending it for use with local search algorithms such as tabu search and combinatorial optimisation problems.

Taillard's (1990) robust taboo search and Battiti and Tecchiolli's (1994) reactive tabu search are techniques that dynamically update the length of the tabu list. These procedures can be called "robust" because they *consistently* find good solutions with less parameters having to be set. The robust tabu search method described in the next section is called "robust" because it finds *robust solutions*.

The outline of this paper is the following. In section 2 an overview of the tabu search technique is given and some of its features are discussed. Section 3 attempts to define and create a taxonomy of robustness and examines why real-world applications exhibit a need for robustness. In section 4, we examine how a tabu search algorithm can be adapted to search for robust solutions through the use of a *robust evaluation function*. In section 5 a simple illustration of the technique is presented. It is shown how the proposed technique can be used to find broad peaks of a mathematical function. This setting is used in section 6 to derive an estimate of the number of evaluations per robust evaluation that are needed to find a broad peak with a certain degree of confidence. Section 7 presents some conclusions and pointers for future research.

2 Tabu Search

2.1 Overview

Given a function $f(x)$ to be optimised over a solution space X, tabu search uses a local search algorithm to move from one solution to the next until an arbitrarily chosen stopping criterion has been satisfied (e.g. the maximal number of iterations has been performed or the quality of the best solution found is sufficient). Each solution $x \in X$ has a neighbourhood $N(x)$ associated with it. A solution $x' \in N(x)$ is reached from x by an operation called a *move*.

To explore regions of the search space that would be left unexplored by the local search procedure and—by doing this—escape local optimality, tabu search modifies the neighbourhood structure of each solution as the search progresses. The solutions admitted to $N^*(x)$, the new neighbourhood, are determined through the use of special memory structures. The search now progresses by iteratively moving from a solution x to a solution $x' \in N^*(x)$.

Tabu search uses both long-term and short-term memory, and each type of memory has its own special strategies.

2.2 Short-term memory

Perhaps the most important type of short-term memory to determine the solutions in $N^*(x)$—and the one that gives its name to tabu search—is the use of a *tabu list*. In its simplest form, a tabu list contains the solutions that have been visited in the recent past (less than n moves ago, where n is the *tabu tenure*). Solutions in the tabu list are excluded from $N^*(x)$. Other tabu list structures prohibit solutions that have certain *attributes* (e.g. TSP tours that include certain arcs) or prevent certain moves (e.g. an arc that was added to a TSP tour cannot be removed in the next n moves). Selected attributes in solutions recently visited are labelled *tabu-active*. Solutions that contain tabu-active elements are *tabu*. This type of short-term memory is also called *recency-based* memory.

To prevent the recency-based memory from preventing extremely good solutions being found, *aspiration levels* are commonly introduced. The tabu status of a solution can be overruled if its solution quality exceeds a certain aspiration level. For example, if a solution is better than any solution previously found, it clearly deserves to be admitted, regardless of its tabu status.

In situations where $N^*(x)$ is very large and/or its elements are expensive to evaluate, *candidate list strategies* can be used to reduce the number of eval-

uations per iteration. These strategies use rules to find candidate solutions (or moves) that are more likely than average to (result in) good solutions.

2.3 Long-term memory

Long-term memory is used for *diversification* and *intensification* of the search process.

Diversification strategies are used to force the search into previously unexplored regions of the solution space. An example is *frequency-based* memory. This type of memory penalises attributes depending on the span of time they have belonged to solutions visited by the search.

Intensification strategies are used to encourage move combinations that have worked well in the past, or to return the search to attractive regions that have been insufficiently explored. An example is an *elite selection strategy*. Different variants have been proposed, one of these is to keep a list of elite solutions and—when the rate of finding new best solutions falls below a certain threshold—resume the tabu search from one of the elite solutions while erasing short-term memory.

3 Robustness and flexibility

3.1 Definitions

Robustness We have defined robustness as *the insensitivity of a solution with respect to changes in the environment in which this solution is implemented*.

In many real-life combinatorial optimisation problems, robustness is just as important an issue as is optimality, and several methods have been developed to create solutions to optimisation problems that perform well under a changing environment. Robustness refers to the insensitivity of a solution with respect to the input data. We distinguish two types of robustness depending on whether the solution is robust in the solution space (*solution robustness*) or in the objective value space (*quality robustness*).

Quality robustness A solution is called *quality robust* if it is robust in the objective value space, i.e. if its solution quality remains high if changes in the input data occur. This type of robustness is important in problems in which a frequent re-optimisation of the problem is infeasible, for example in warehouse location problems. A typical warehouse location problem searches for the ideal placement of warehouses in the plane or on a graph, given a

number of customers with their respective demands. Once the location of the warehouses is determined and the warehouses have been built, this cannot be undone without extremely high costs. Therefore, it is imperative that the solution initially found remains as close to optimal as possible under changing circumstances. In general, quality robustness is needed in situations where volatile input data is used to base inflexible decisions on.

Solution robustness In other problems a new solution has to be found when changes in input data occur and—as a result—the original solution does not satisfy the constraints of the new problem. To this end the problem is re-optimised. In these cases we might require the new solution to be as close to the original one as possible. A solution is called *solution robust* if it is robust in the solution space, i.e. if the new solution changes only slightly with changes in input data. In vehicle routing, for example, most changes in input data (e.g. a new customer has to be serviced, a certain route is closed, etc.) will require new routes to be calculated. In practice however, there is a preference for the routes to stay largely the same. Several reasons can be found for this: drivers may get confused if their routes change every other day, errors may be made by the planners, etc. Also, the company may require that the same customers are visited by the same drivers as much as possible in order to build better relationships with these customers (see e.g. Ribeiro and Lorenço (2001)). For these and other reasons, vehicle routing becomes more of a tactical decision. It is therefore desirable that a vehicle routing algorithm generates solutions that do not differ too much when changes in input data occur. Contrary to quality robustness, solution robustness is a not only a property of the solution, but also of the solution algorithm.

The main difference between quality robustness and solution robustness is that in the former case, it is the quality of the solution that is not allowed to change. In the latter case, it is the solution itself that is not allowed to change. The robust tabu search procedure discussed in this paper is intended to produce solutions that are quality robust.

Flexibility A notion closely relation to robustness is *flexibility*. A solution can be called flexible if it can be changed to adapt to a changing environment easily and without a large decrease in solution quality. We refer to flexibility when there is some "intervention" of the decision maker on the solution, other than using the algorithm that was used to find the solution. We call this intervention a *repair procedure* as in most cases, the goal of changing the solution will be to create a new solution that satisfies the constraints of the problem. Whether flexibility or robustness is required depends on

Table 1: Robustness and flexibility taxonomy

	Solution quality remains high	**Solution remains similar**
No repair procedure applied	Quality robustness	Solution robustness
Repair procedure applied	Quality-robust flexibility	Solution-robust flexibility

whether the decision maker can change the solution after it was found by the algorithm. When a machine breakdown occurs in a scheduling environment, for example, it is sometimes possible to easily and without a large increase in the objective value (e.g. tardiness) find a new solution that satisfies all constraints. Such a solution can be called flexible. In other cases, it might be impossible or undesirable to change the solution by other means than the solution algorithm.

Solution-robust versus quality-robust flexibility We can call a solution *quality-robust flexible* if it can be changed without incurring a decrease in the objective function value. A solution can be called *solution-robust flexible* if the solution found after applying the repair procedure is similar to the original solution.

Summary A summary of the results presented here can be seen in table 1. In this paper, we only investigate the aspect of quality robustness.

3.2 Robustness versus optimality

Robustness and optimality are often conflicting objectives in that an increase in one of them will often incur a decrease in the other. In most cases, however, the relationship between the two objectives is rather unclear. Finding a good tradeoff between robustness and optimality is a matter of judgment on the part of the decision maker. Mathematical procedures can help this person to take a responsible decision. However, they cannot make the decisions in his/her place.

Models of real-world systems are always subject to incomplete, noisy, and erroneous data. This is a problem to operations researchers that attempt to exploit these models and apply their mathematical solutions in practice. In contrast, most techniques for solving such models assume deterministic data. In general, models are formulated by performing *worst-case* or *mean-value* analysis. The latter has been shown to have very large error bounds,

whereas the former is known to produce very conservative and sometimes very expensive solutions.

4 Robust evaluation functions

Robust tabu search modifies the function evaluation of a tabu search procedure. If effectively replaces the solution evaluation function $f(x)$ with a new one, $f_r(x)$. This new evaluation function can take on some different forms, but always follows the following basic principles.

- **Principle 1** The new evaluation function adds some *noise* or *perturbations* to the current solution before evaluating it. So, the robust tabu search procedure evaluates $x^* = x + \delta$ instead of x. We will call x^* a *derived* solution. Derived solutions may or may not belong to the original solution space. The noise added is dependent on the expected noise in the input data and should reflect the expected changes in input data.
- **Principle 2** The new evaluation function does not evaluate a single point, but evaluates several derived solutions and combines these into a single function value. This new function will be called the *robust evaluation function* $f_r(x)$.

A general form of a robust evaluation function is

$$f_r(x) = \frac{1}{n} \sum_{i=1}^{n} w_i f(x + \delta_i) \tag{1}$$

In equation 1, n is the number of derived solutions that are evaluated. The function value $f(x + \delta_i)$ of the i-th derived solution $x + \delta_i$ is weighed by a factor w_i. Many forms of this function are possible, including some very complex ones. These can be divided in *deterministic* and *stochastic* ones, depending on the nature of the noise that is added. We'll see examples of both cases later.

One of the main advantages of robust tabu search is that it only modifies the evaluation function. It does not alter any tabu-search-specific features of the procedure. This makes it extremely easy to implement.

The value of n is an indicator of the importance that is attached to robustness, as opposed to solution quality. If n is larger, more derived solutions will be evaluated, resulting in a more robust solution.

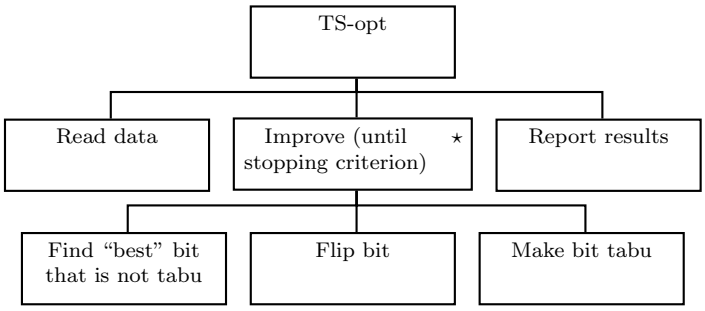

Figure 1: Tabu search procedure for function optimisation[1]

5 Robust solutions of a function on a finite domain

In this section, two applications of the robust tabu search technique will be investigated. The applications are of the "educational" type, meaning that they are too simple to be useful in practical cases. Their only aim is to efficiently show the principles of robust tabu search without worrying about implementational details. We hope to convey the message that the basic principles are valid regardless of the specific tabu search procedure used.

5.1 Tabu search procedure

In this paragraph, a simple procedure for maximising or minimising a continuous function f of a single variable x on a finite domain $[c, d]$ is developed. We call x_{bin} the binary representation of a solution x. The value of x can be obtained from its bit-string representation by the formula $x = c+(d-c)\times\frac{x_{\text{bin}}}{2^l}$, where l is the number of bits in x_{bin}, which we assume to be constant, and therefore 2^l is the number of values that x can take. We assume that the properties of the function are such that—given a sufficiently large value of l—the values of x that correspond to the optima of $f(x)$ can be approximated.

The tabu search procedure moves from a solution x to a solution x' by a *1-flip* move. At each iteration, exactly one bit of x_{bin} is flipped to yield x'_{bin}. Each bit is tested and the bit that is flipped is the one that yields the x' with the highest function value. The tabu list contains the indices of the bits that were most recently flipped. This tabu search procedure is schematically shown in figure 5.1.

[1]This *program structure diagram* is a schematic representation of the algorithm. The horizontal structure indicates precedence, i.e. the components are executed from left to right. The vertical structure indicates hierarchy (e.g. the component *Improve (until stopping criterion)* consists of an iteration (indicated by the asterisk) of the three components

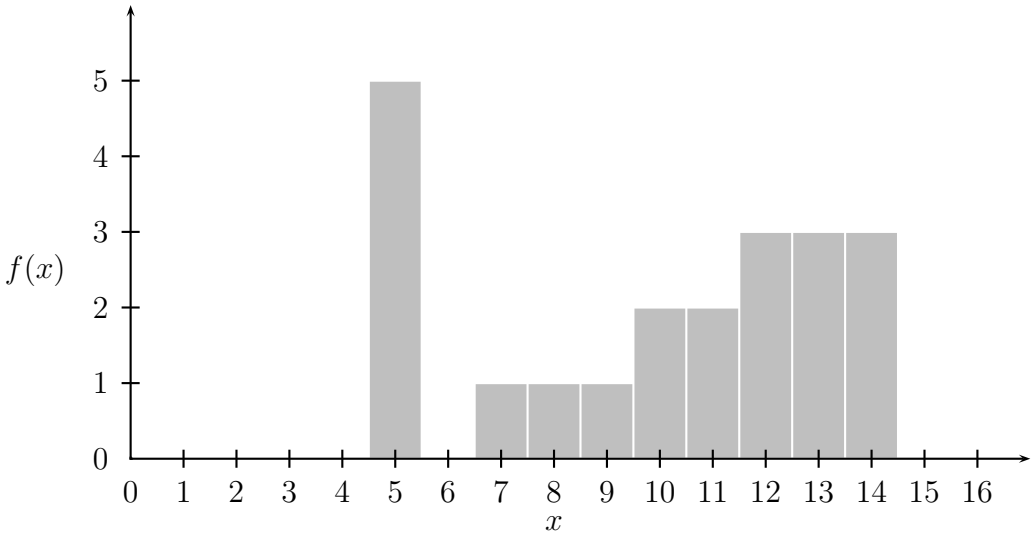

Figure 2: A discrete function

Although much too simple to be of practical use, this procedure can be used to show the effectiveness of robust tabu search.

5.2 Example 1 (discrete function)

To show the principles of the procedure proposed in section 4, suppose we wish to find a robust maximum of the function in figure 2.

A solution x of this problem can be encoded as a binary vector of 4 bits. The proposed tabu search procedure (with a tabu tenure of 2 and without a robust optimisation function) will within a few iterations find the maximum: $x = 5$ and $f(x) = 5$. This solution is optimal but not robust: any deviation of x from the value of 5 will result in an immediate and drastic decline of $f(x)$.

To find a more robust solution, the evaluation function is changed to

$$f_r(x) = \frac{1}{3} \sum_{i=-1}^{+1} f(x+i) \tag{2}$$

The new evaluation function calculates the average of the function value of x and its two neighbouring points $x+1$ and $x-1$. The same tabu search procedure, using the new evaluation function will find a much more robust

below. See e.g. Jackson (1975) for a detailed explanation of the syntax of program structure diagrams.

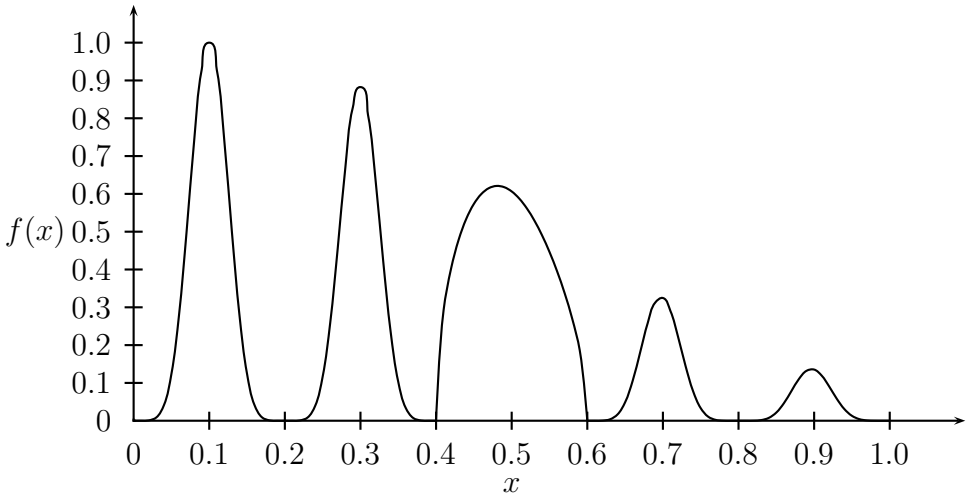

Figure 3: A continuous function

solution: $x = 13$. For this solution, the value of $f_r(x)$ is 3, whereas for $x = 5$, $f_r(x) = \frac{5}{3}$. Depending on the function to optimise, the robust evaluation function $f_r(x)$ will have a different form. In this example the noise added to the solution is completely deterministic. In the next paragraphs, we will examine the effect of adding random noise.

5.3 Example 2 (continuous function)

We wish to maximise the following function on the interval $x \in [0, 1]$:

$$f(x) = \begin{cases} e^{-2\left(\frac{x-0.1}{0.8}\right)}\sqrt{|\sin(5\pi x)|} & 0.4 < x \leq 0.6 \\ e^{-2\left(\frac{x-0.1}{0.8}\right)}\sin^6(5\pi x) & \text{otherwise} \end{cases} \tag{3}$$

This function is depicted in figure 3. The function has five unequal peaks, four of which are very narrow. The global maximum is at $x = 0.1$. On the other hand, the function has a much more stable maximum at $x = 0.486$.

The proposed tabu search procedure with robust evaluation function

$$f_r(x) = \frac{1}{21}\sum_{i=-10}^{10} f(x + \frac{i}{100}) \tag{4}$$

has no trouble finding the broad peak. However, there are several disadvantages of using a deterministic robust evaluation function. First of all,

determining a deterministic robust evaluation function is only possible in very simple cases such as this one. In a more complex context, the noise will necessarily have to be of a stochastic nature. Secondly, stochastic noise is certainly more realistic than deterministic noise in most cases. One of the main reasons why we look for robust solutions is exactly the stochastic nature of the environment in which most solutions are implemented. This is why the noise in essence reflects the possible changes in the real-world implementation of the environment.

5.4 Stochastic robust evaluation function

We can define a stochastic robust evaluation function for this problem as

$$f_r(x) = \frac{1}{n}\sum_{i=1}^{n} f(x+\delta_i), \tag{5}$$

where $\delta_i \sim \mathcal{N}(0,\sigma)$ is a stochastic noise. In the following, we will assume that $\delta_i (i = 1,\ldots,n)$ is a sequence of i.i.d. normally distributed variates. Following (Tsutsui and Ghosh, 1997), the size of the standard deviation σ of the normal distribution from which δ_i is drawn, can be calculated when the maximum allowable width of a peak to be considered as a sharp peak, is given. This standard deviation is determined based on a "reduction factor", i.e. the fraction by which the height of a narrow peak is reduced when noise is added. We can use the same method to determine σ.

Once the standard deviation of the normal distribution is calculated, the only parameter in the algorithm is the number of evaluations that are needed to obtain a robust optimum. This question is addressed in the next section.

6 Estimating the number of evaluations

6.1 "Population effect" in evolutionary algorithms

When using evolutionary algorithms to obtain robust solutions, there is a clear effect of the fact that these algorithms are population-based. As Branke (1998) reports, a standard evolutionary algorithm will "favour hills over peaks". The reasons for this are that the probability that an individual in the initial population will be in the basin of attraction of a hill instead of a peak is higher. Also, the average fitness of individuals on a hill is higher. Local search algorithms that are not population-based do not posses this quality. In fact, Tsutsui (1999) reports that adding a single perturbation to a phenotype is preferable over adding many perturbations and taking the

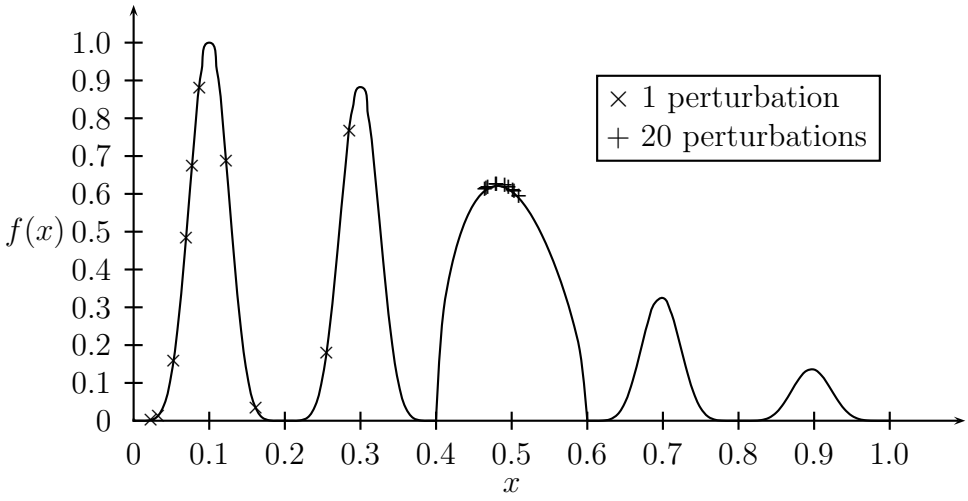

Figure 4: Solutions for 10 runs of the tabu search procedure with 1 and 20 perturbations per function evaluation

mean of the function evaluations. Clearly, adding a single perturbation to a solution before evaluating it will in the case of local search lead to choosing a random solution instead of a robust one. In most cases, this approach will choose a solution that is bad, but is close to (within range of the random perturbation) of a peak. This is clearly not a desirable characteristic.

As an experiment, we performed ten runs of the proposed tabu search and used the stochastic robust evaluation function with one perturbation added per function evaluation. The size of the perturbation was chosen as a normally distributed variates with mean 0 and standard deviation 0.0625. So, $f_r(x) = f(x + \delta)$ with $\delta \sim \mathcal{N}(0, 0.625)$. For the calculation of this standard deviation, we refer to Tsutsui and Ghosh (1997).

In figure 4, the solutions are graphically shown of 10 runs of the tabu search procedure with one perturbation per function evaluation (indicated by $\times$). It can be clearly seen that the solutions found by adding one single perturbation are randomly scattered around the peak at $x = 0.1$. The robust evaluation for each of the ten runs shown in figure 4 was greater than 0.99. This indicates that for each of the perturbations, $x + \delta_i \approx 0.1$.

By taking several function evaluations of perturbed solutions, the search for robust solutions can be greatly improved. In figure 4, a second experiment of 10 runs of the algorithm is shown. This time the robust evaluation function is $f_r(x) = \sum_{i=1}^{20} f(x+\delta_i)$. Again δ_i is a normally distributed noise with mean 0 and standard deviation 0.0625. This time, the solutions found by the tabu

search procedure (indicated by +) are all very close to the robust hill of the function.

6.2 An approximate bound on the number of evaluations

To estimate the number of evaluations n that are required to find the robust solutions with a certain confidence, we make the following assumptions:

1. The tabu search algorithm enumerates all possible solutions exactly once in the search process. The number of solutions is 2^l where l is the length of the bitstring. Recent results by Glover and Hanafi (1999) show that such tabu search procedures can be created in practice.

2. The width of the sharp peak of the solution is equal to $2w$. The sharp peak is located at point a on the x-axis and extends from $a - w$ to $a + w$.

3. The sharp peak is found when there is an x for which each of the n perturbations $x + \delta_i$ are between the boundaries of the sharp peak, i.e. when $a - w \leq x + \delta_i \leq a + w \quad \forall i (i = 1, \ldots, n)$. In all other cases, a broad peak is found.

4. The values of δ_i are i.i.d. normally distributed variates with mean 0 and standard deviation σ.

If the solution currently found by the algorithm is x, then the probability of the perturbed solution $x + \delta_i$ with $\delta_i \sim \mathcal{N}(0, \sigma)$ being in the interval $[a - w, a + w]$ is given by

$$P[a - w \leq x + \delta_i \leq a + w] = F\left[\frac{x - (a - w)}{\sigma}\right] - F\left[\frac{x - (a + w)}{\sigma}\right]$$

where $F(x)$ is the cumulative standard normal distribution given by

$$F(x) = \int_{-\infty}^{x} \frac{e^{-\frac{y^2}{2}}}{\sqrt{2\pi}} dy$$

This can be seen in figure 5 for $x = 0.4$, $a = 0.55$, $2w = 0.1$ and $\delta_i \sim \mathcal{N}(0, 0.15)$. The probability is 0.161.

When a certain solution is evaluated n times, the probability of all n evaluations of a certain value x being in the interval $[a - w, a + w]$ is the product of the individual probabilities.

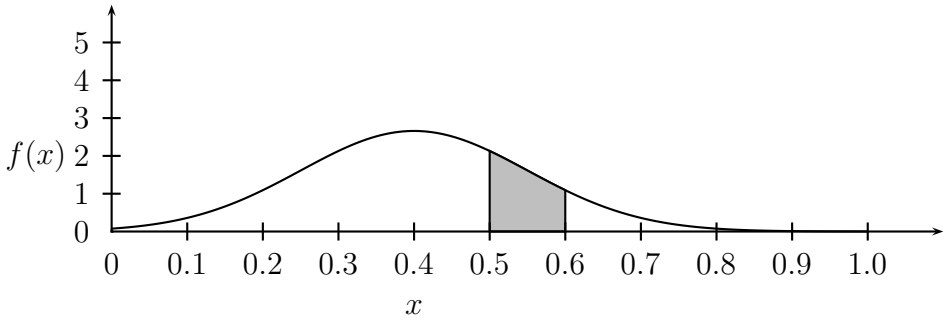

Figure 5: $P[a-w \leq x+\delta_i \leq a+w]$ is given by the surface of the coloured piece of the normal distribution ($x = 0.4$, $a = 0.55$, $2w = 0.1$ and $\delta_i \sim \mathcal{N}(0, 0.15)$)

$$P[\forall i \in [1,n] : a-w \leq x+\delta_i \leq a+w] = \left[F\left[\frac{x-(a-w)}{\sigma}\right] - F\left[\frac{x-(a+w)}{\sigma}\right]\right]^n$$

If we assume that a bitstring of length l is used, the number of solutions evaluated by the tabu search procedure is 2^l. To estimate the probability that at least one of the solutions has all of its derived solutions inside the interval $[a-w, a+w]$, the value of $P[\forall i \in [1,n] : a-w \leq x_j+\delta_i \leq a+w]$ is calculated for each $x_j (j \in [1, 2^l])$.

Let A_j denote the event that n perturbed solutions of $x_j (j \in [1, 2^l])$ are in the interval $[a-w, a+w]$. The probability that at least one solution x will have all of its perturbed solutions in the interval $[a-w, a+w]$ is equal to $P[A_1 \cup A_2 \cup \cdots \cup A_{2^l}]$. To calculate this measure we can apply the inclusion–exclusion formula.

$$\begin{aligned} P\left[\bigcup_{i=1}^{n} A_i\right] = & \sum_i P[A_i] \\ & - \sum_{i<j} P[A_i \cap A_j] \\ & + \sum_{i<j<k} P[A_i \cap A_j \cap A_k] \\ & - \ldots \\ & + (-1)^{(n+1)} P[A_1 \cap A_2 \cap \cdots \cap A_n] \end{aligned}$$

Unfortunately, calculating the value of this formula requires an exponential amount of computing time in the number of solutions evaluated. We therefore employ an algorithmic approximation formula developed by Hunter (1976) that uses only the first- and second-order marginal probabilities.

Let $A_j (j = 1, \ldots, 2^l)$ label the nodes of a graph G. Let the intersection $A_{jk} = A_j \cap A_k (j = 1, \ldots, 2^l, k = j+1, \ldots, 2^l)$ denote the edge of G connecting nodes A_j and A_k. Furthermore, let τ be a spanning tree of the graph G containing $2^l - 1$ edges so that each of the nodes is connected by at least one edge. The spanning tree is a sub-graph of G connecting each node and having no cycles. The following relationship is a bound on the joint probability of the events A_1 to A_{2^l}.

$$P\left[\bigcup_{j=1}^{2^l} A_j\right] \leq \sum_{j=1}^{2^l} P[A_j] - \max_{\tau} \sum_{A_{jk} \in \tau} P[A_{jk}]$$

In our case, since A_j and A_k are independent events, $P[A_{jk}] = P[A_j].P[A_k]$. Calculating Hunter's approximation requires solving a maximum spanning tree problem. This can be easily done with e.g. Prim's or Kruskal's algorithm (see e.g. Sedgewick (1988)).

6.3 Calculating the bound

Figure 6 shows some calculations of the probability that each of the n perturbed evaluations will be in $[a-w, a+w]$. In all cases, the interval on which the function is defined is $[0, 1]$ and $a = 0.5$, i.e. the peak is located in the center of the interval.

In figure 6a, a bitstring length of 8 is used, resulting in $2^l = 256$ different possible solutions. In this figure, $\frac{w}{\sigma}$ is kept constant and equal to 0.5. The number of robust evaluations needed to find a robust solution is slightly higher when w and σ are larger, but at 10 evaluations, a robust solution is found in all cases.

In figure 6b, the effect of representing the solutions using a smaller bitstring is investigated. The results in this figure are obtained with a bitstring of length $l = 6$. In this case, the number of solutions evaluated by the tabu search algorithm is smaller, and consequently, the probability of a narrow peak accidentally being chosen is smaller too. Of course, the reverse is also true: if the number of bits in the solution vector increases, so should the number of evaluations.

In figure 6c, the width of the sharp peak is kept constant and the size of σ is varied. If σ is small, the derived solutions will have a high probability of being in a narrow interval around the original solution. This will increase

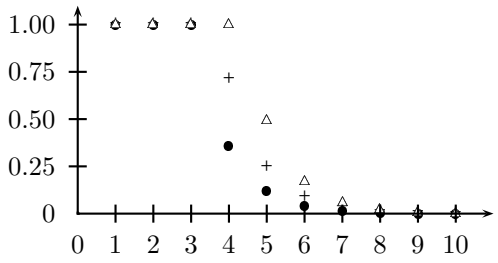

Figure 6a

	$l = 8$	$2^l = 256$
•	$2w = 0.05$	$\sigma = 0.05$
+	$2w = 0.1$	$\sigma = 0.1$
△	$2w = 0.2$	$\sigma = 0.2$

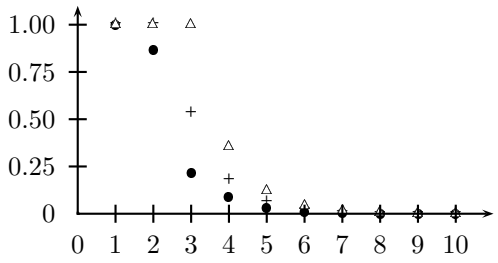

Figure 6b

	$l = 6$	$2^l = 64$
•	$2w = 0.05$	$\sigma = 0.05$
+	$2w = 0.1$	$\sigma = 0.1$
△	$2w = 0.2$	$\sigma = 0.2$

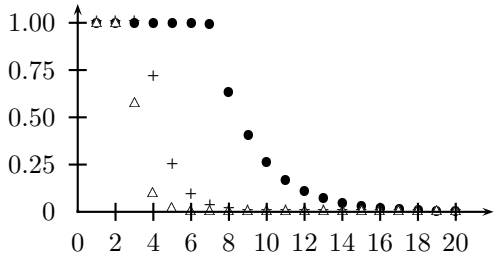

Figure 6c

	$l = 8$	$2^l = 256$
•	$2w = 0.1$	$\sigma = 0.05$
+	$2w = 0.1$	$\sigma = 0.1$
△	$2w = 0.1$	$\sigma = 0.2$

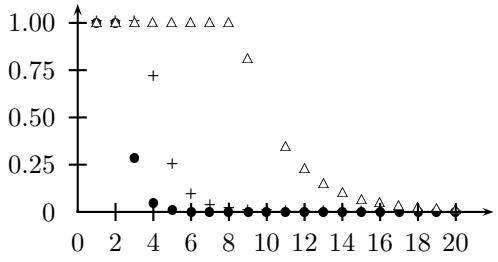

Figure 6d

	$l = 8$	$2^l = 256$
•	$2w = 0.05$	$\sigma = 0.1$
+	$2w = 0.1$	$\sigma = 0.1$
△	$2w = 0.2$	$\sigma = 0.1$

Figure 6: Number of evaluations n to reach a certain probability of finding a sharp peak

the probability of accidentally finding a sharp peak. Therefore, the number of evaluations needed to find a robust solution increases when the standard deviation of the normally distributed noise is smaller. The same results are found in figure 6d, where w is varied and σ is kept constant. If σ is small relative to the width of the sharp peak, the number of evaluations should be higher.

6.4 Interpretation of the approximation

In most cases, the approximation will be an underestimation of the number of evaluations required to reliably find the broad hills of the function. The reasons for this are the following.

- Although tabu search algorithms can be designed that visit each possible solution exactly once, standard tabu search algorithms will not do this. Instead, they will focus on regions that contain solutions with good quality, including narrow peaks. Solutions in these regions will in general be evaluated several times. Since solutions close to narrow peaks have a higher probability of all their derived solutions being in the peak interval, visiting these solutions more often will increase the probability of such a solution being chosen wrongly.
- The assumption that all derived solutions must fall inside the narrow peak interval is not very realistic. When the number of evaluations rises, not all of the derived solutions of a certain solution need to be inside the peak interval for this solution to obtain a higher robust evaluation than all other solutions. Therefore, the number of evaluations needed will be higher than the one given by the approximation.

In general, it is therefore advisable to use the approximation to obtain a rough estimate of the number of evaluations needed, but use a higher number when actually using the algorithm. Using the approximation also provides insight into the effects of the parameters of the algorithm (such as the number of bits l in the vector, the width $2w$ of broad peaks and the standard deviation of the Gaussian noise σ). A value of $n = 20$ seems to provide a very high probability of finding a broad hill in every realistic case.

7 Conclusions and future work

By evaluating several derived solutions and combining these evaluations into a robust evaluation function, a tabu search procedure can be altered so that

it searches for solutions that are relatively insensitive to perturbations in the input data. The design of robust evaluation functions and the application of the proposed technique to realistic problems remains a topic for future research.

Bibliography

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

Beale, E. (1955). On minimizing a convex function subject to linear inequalities, *Journal of the Royal Statistical Society* **17**: 173–184.

Box, G. and Draper, N. (1987). *Empirical Model-Building and Response Surfaces*, John Wiley & Sons, New York.

Box, G. and Wilson, K. (1951). On the experimental attainment of optimum conditions, *Journal of the Royal Statistical Society* **13**: 1–45.

Branke, J. (1998). Creating robust solutions by means of evolutionary algorithms, *Parallel Problem Solving from Nature V*, LNCS vol. 1498, Springer Verlag, pp. 119–128.

Dantzig, G. (1955). Linear programming under uncertainty, *Management Science* **1**: 197–206.

Dupačová, J. (1987). Stochastic programming with incomplete information: A survey of results on postoptimization and sensitivity analysis, *Optimization* **18**: 507–532.

Dupačová, J. (1990). Stability and sensitivity analysis for stochastic programming, *Annals of Operations Research* **27**: 115–142.

Giovannitti-Jensen, A. and Myers, R. (1989). Graphical assessment of the prediction capability of response surface designs, *Technometrics* **31**(2): 159–171.

Glover, F. (1989). Tabu search – part I, *ORSA Journal on Computing* **1**: 190–206.

Glover, F. (1990). Tabu search – part II, *ORSA Journal on Computing* **2**: 4–32.

Glover, F. and Hanafi, S. (1999). Tabu search and finite convergence, *Technical Report HCES-04-99*, Hearin Centre for Entreprise Research.

Glover, F. and Laguna, M. (1999). *Tabu Search*, Kluwer, Boston.

Hunter, D. (1976). An upper bound for the probability of a union, *Journal of Applied Probability* **13**: 597–603.

Jackson, M. (1975). *Principles of Program Design*, Academic Press, London.

Kackar, R. (1985). Off-line quality control, parameter design and the Taguchi method, *Journal of Quality Technology* **17**(Oct): 176–209.

Kall, P. and Wallace, S. (1994). *Stochastic Programming*, John Wiley & Sons, New York.

Krottmaier, J. (1993). *Optimizing engineering designs*, McGraw-Hill, London.

Mulvey, J., Vanderbei, R. and Zenios, S. (1995). Robust optimization of large-scale systems, *Operations Research* **43**(2): 264–281.

Myers, R. H., Khuri, A. I. and Carter, W. H. (1989). Response surface methodology: 1966–1988, *Technometrics* **31**(2): 137–157.

Ribeiro, R. and Lorenço, H. (2001). A multi-objective model for a multi-period distribution management problem, *in* J. P. de Sousa (ed.), *Proc. 4th Metaheuristics International Conference MIC'01*, Porto.

Sedgewick, R. (1988). *Algorithms*, Addison–Wesley, New York.

Taillard, E. (1990). Robust taboo search for the quadratic assignment problem, *Parallel Computing* **17**: 443–455.

Tsutsui, S. (1999). A comparative study on the effects of adding perturbations to phenotypic parameters in genetic algorithms with a robust solution searching scheme, *Proceedings of the 1999 IEEE Systems, Man, and Cybernetics Conference (SMC'99 Tokyo)*, pp. III–585–591.

Tsutsui, S. and Ghosh, A. (1997). Genetic algorithms with a robust solution searching scheme, *IEEE Transactions on Evolutionary Computation* **1**(3): 201–208.

Tsutsui, S., Ghosh, A. and Fujimoto, Y. (1996). A robust solution searching scheme in genetic search, *in* H.-M. Voigt, W. Ebeling, I. Rechenberg and H.-P. Schwefel (eds), *Parallel problem solving from nature - PPSN IV*, Vol. 10, Springer, Berlin, pp. 543–552.

Tsutsui, S. and Jain, J. (1998). Properties of robust solution searching in multi-dimensional space with genetic algorithms, *Proc. 2nd Int. Conf. on Knowledge-Based Electronic Systems (KES-98)*.

Wets, R.-J. B. (1974). Stochastic problems with fixed resources: the equivalent deterministic problem, *SIAM Review* **16**: 309–339.