ELSEVIER European Journal of Operational Research 125 (2000) 266–277

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Application of the noising method to the travelling salesman problem

Irène Charon, Olivier Hudry [*]

École nationale supérieure des télécommunications, 46 rue Barrault, 75634 Paris Cedex 13, France

Received 1 September 1998; accepted 1 March 1999

Abstract

In this paper, we study the application of the noising method, a recent combinatorial optimization metaheuristic, to the Travelling Salesman Problem (TSP). We first detail the features of the noising method in order to adapt it to the TSP. Then we "experimentally" compare it to the simulated annealing method, and we study its sensitiveness to different parameters involved in its design. Two types of TSPs have been considered: the randomly weighted TSP, for which the weights of the edges are randomly chosen, and the Euclidean TSP, for which the vertices belong to the Euclidean plane and where the weights of the edges are given by the Euclidean distances between the vertices. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Combinatorial optimization; Noising method; Simulated annealing; Metaheuristics; Travelling salesman problem

1. Introduction

The noising method is a recent combinatorial optimization heuristic: its first results have been published in 1993 in [5]. It is not designed to solve only one special type of problem (see [6] for references upon other applications), but, as other algorithms sometimes called "metaheuristics" (see for instance [2] or [13]), to be applicable to various kinds of combinatorial optimization problems. Such a problem may be described as follows: given a finite set S and a function f defined on S, find the minimum of f over S and an element of S minimizing f. To minimize f, the noising method does not take the genuine values of f into account but considers that they are perturbed in some way by *noises*. During the running of the method, the noises decrease down to 0 so that, at the end, we deal with the genuine function f.

The aim of this paper is to analyse the sensitiveness of the noising method to the tuning of its parameters when applied to the Travelling Salesman Problem (TSP). So, here, S is the set of all Hamiltonian cycles (HC) in a given weighted non-oriented graph G and f gives the weight of such a

[*] Corresponding author. Tel.: +33-1-45-81-77-77/63; fax: +33-1-45-81-31-19.

E-mail address: hudry@inf.enst.fr (O. Hudry).

cycle; the TSP consists in finding an HC minimizing f for any given graph G. In the following, G is assumed to be complete; n denotes the number of vertices of G and the vertices of G will be denoted by $x_1, x_2, \dots, x_n$ (where $x_{n+i} = x_i$); as an HC can be considered as defined by a permutation σ on $\{x_1, x_2, \dots, x_n\}$, such an HC will be noted $x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(n)}$. It is well-known that the TSP is NP-hard and remains so if the weights are Euclidean distances (for further references on the TSP, see for instance [8,9,11]).

The paper is organized as follows. In Section 2, we give a short presentation of the noising method and we show how to adapt it to the TSP. Section 3 is devoted to the results provided by the application of the noising method to the TSP for which the valuations are randomly chosen (they are not necessarily distances); we especially study the efficiency of the method with respect to the number of iterations and more generally with respect to the parameters on which the noising method depends. In Section 4, we do the same for the Euclidean TSP, for which the vertices are (randomly or not) spread over the plane and the distances are the Euclidean ones. Conclusions take place in Section 5.

2. Principle of the noising method applied to the TSP

In this section, we briefly present the main principles of the noising method in order to apply it to the TSP. More precisely, we detail two possible schemes of the noising method applied to the TSP, though other schemes could have been designed (the interested reader can find a review about the different schemes and the other applications of the noising method in [6]).

Many heuristics applied to combinatorial optimization problems are based on *elementary transformations*, that is, operations changing one feature of the current solution (here, an HC) without changing its global structure. For the TSP, the well-known 2-opt transformation proposed by Lin [12] gives such an elementary transformation: if the current HC C is $x_{\sigma(1)}, x_{\sigma(2)}, \dots, x_{\sigma(n)}$ for some permutation σ, Lin's 2-opt consists in removing from C two edges $\{x_{\sigma(i)},\ x_{\sigma(i+1)}\}$ and $\{x_{\sigma(j)},\ x_{\sigma(j+1)}\}$ for some indices i and j with $n \geqslant j > i + 1 \geqslant 2$ and $j \neq n$ if $i = 1$, and replacing them by the two edges $\{x_{\sigma(i)},\ x_{\sigma(j)}\}$ and $\{x_{\sigma(i+1)},\ x_{\sigma(j+1)}\}$ to get a new HC. The application of this elementary transformation to the current HC is utterly specified by the two values of i and j with $n \geqslant j > i + 1 \geqslant 2$ and $(i, j) \neq (1, n)$.

Thanks to this transformation, we may define the *neighbourhood* $N(C)$ of any HC C: $N(C)$ is the set of the HCs that we get by applying Lin's 2-opt to C for any values of i and j with $n \geqslant j > i + 1 \geqslant 2$ and $(i, j) \neq (1, n)$. An element of $N(C)$ is called a *neighbour* of C; any HC has $NS = n(n-3)/2$ neighbours; this quantity NS will be called the *neighbourhood size*. Now, we may design an *iterative improvement method* (also called a *descent*) for the TSP: from the current HC C, we apply the 2-opt transformation so that we get a new HC $C' \in N(C)$. Let $\Delta f(C, C')$ denote the variation of f: if we have $\Delta f(C, C') < 0$, then C' becomes the new current HC (C' is a solution better than C with respect to f), otherwise we keep C as the current HC. Then we do it again with the current HC, until it is impossible to find a neighbour of the final HC C_{final} which is better than $C_{\text{final}} :\ \forall C \in N(C_{\text{final}}),\ f(C) \geqslant f(C_{\text{final}})$.

Notice that there are several ways of choosing C' from C. As C' is the result of a 2-opt transformation applied to C, C' can be characterized by the two parameters i and j of this 2-opt. So, we may get C' by choosing i and j randomly (with $n \geqslant j > i + 1 \geqslant 2$ and $(i, j) \neq (1, n)$), as it is done in a classical simulated annealing method (it is the case for instance in [4]; see also [10]). We can also try the pairs (i, j) in a systematic way: starting with a random pair (i_0, j_0), we try $(i_0, j_0 + 1)$ after (i_0, j_0), then $(i_0, j_0 + 2)$ and so on until (if necessary) the last possible pair. We call this strategy *a systematic exploration of the neighbourhood.* This strategy avoids to consider a same neighbour twice and usually allows to save CPU time.

Like several other metaheuristics, the *noising method* is based on a descent. But, instead of computing the genuine values of f, we add a perturbation that we call a *noise* to f. In a more general context (see [6]), we may design several ways of perturbing the values of f. In this paper, we consider only one kind of noising: noises are added to the variation Δf of f when an elementary transformation has been applied to the current solution in order to get one of its neighbours. More precisely, instead of the genuine variation Δf, we consider a "noised" variation $\Delta f_{\text{noised}} = \Delta f + \rho$, where ρ (the noise) is a random real drawn with a uniform distribution into the interval $[-r, +r]$ (other distributions also can be tried; see [6]) and where r is the *noise-rate*. The initial value $r_{\max}$ of r depends on the maximum weight of G. In order to converge towards f, the noise-rate r decreases from time to time, when a given number of elementary transformations are tried. We will call "iteration" all the operations performed when one elementary transformation is tried. The number of elementary transformations tried, or rather the number of iterations performed between two decreases of r, will be noted *nb_it_at_fixed_rate*; it will be convenient to express it with respect to the neighbourhood size *NS*. The principle of the noising method for our application can be summarized as in Fig. 1. The whole process stops when a given number *total_nb_it* of iterations are performed. For the choice of the neighbour C' of C, we apply a systematic exploration.

In order to get back the genuine function f at the end of the method, we may think to make r decrease down to 0. But in fact, when r is low enough, it appears from our experiments that the solution does not evolve any longer. Thus it is useless to make r decrease down to 0, which consumes CPU time without improving the solution. Consequently, the noise-rate is bounded by two extreme values $r_{\max}$ and $r_{\min}$ (if we want to be sure to reach a local minimum while $r_{\min}$ is not equal to 0, it is always possible to add a final "unnoised" descent after the repeat-loop in the scheme of Fig. 1). It is also necessary to precise the way of decreasing r; we choose an arithmetical decreasing, as in [5] (see [6] for references on other choices). It is easy to compute the rate μ of this arithmetical decreasing from the other parameters:

$$\mu = \frac{r_{\max} - r_{\min}}{\left(total\_nb\_it / nb\_it\_at\_fixed\_rate\right) - 1}.$$

Of course, the best solution (with respect to the genuine values of f) *best_sol* computed during the method is memorized and, as shown by Fig. 1, the method returns *best_sol* at the end.

Then it appears that this version of the noising method depends on four parameters: $r_{\max}$, $r_{\min}$, *total_nb_it* and *nb_it_at_fixed_rate* (the initial solution does not play an important role for the method: we may initialize the process with a random solution, or a solution found by another heuristic...). Some variants can be designed, with extra parameters or not (see [6]). We describe two of them below.

- *First variant*: The first variant consists in alternating a certain number of "noised" iterations with "unnoised" descents. More precisely, in order to stay closer to f, we apply a descent with respect to f (that is, with a noise equal to 0) when a fixed number of "noised" iterations are performed. This number will be chosen as a multiple of *NS*. This variant allows to check a good number of local minima (with respect to f) which could provide good solutions.
- *Second variant*: The second variant consists of coming back to the best computed solution periodically. Indeed, because of the noises added to the variations Δf, it may happen that we leave an interesting area of the space of solutions for a less interesting one. So one possible strategy is to periodically restart the current solution with the best solution found since the beginning. Of course, it is not useful to restart the current solution too often. To define the frequency of the restart, we intro-

```
• Compute an initial solution $C$
• $best\_sol \leftarrow C$
• $r \leftarrow r_{max}$
• $nb\_it \leftarrow 0$
• repeat the following steps:
    * $nb\_it \leftarrow nb\_it + 1$
    * let $C'$ be a neighbour of $C$: $C' \in N(C)$
    * draw a random number $\rho$ in $[-r, r]$ uniformly
    * compute $\Delta f(C, C')$ and $\Delta f_{noised}(C, C') = \Delta f(C, C') + \rho$
    * if $\Delta f_{noised}(C, C') < 0$, then $C \leftarrow C'$
    * if $f(C) < f(best\_sol)$, then $best\_sol \leftarrow C$
    * if $nb\_it \equiv 0$ (mod $nb\_it\_at\_fixed\_rate$), then decrease $r$
• until $nb\_it = total\_nb\_it$
• return $best\_sol$.
```

Fig. 1. A scheme of the noising method.

duce a new parameter that we call *nb_restarts*, which gives the number of restarts applied during the whole process. Thus, when *total_nb_it* iterations are performed during the whole process, there are about *total_nb_it*/*nb_restarts* iterations performed between two restarts.

With these different principles, we study the application of two noising methods to the TSP, called Noising1 and Noising2 in the following. Noising1 is a basic noising method following the scheme of Fig. 1 without the above variants, while Noising2 includes them; so Noising2 needs the tuning of two extra parameters: the number αNS of "noised" iterations performed between two "unnoised" descents (first variant) and the number of restarts *nb_restarts* (second variant). Noising2 is illustrated by Fig. 2.

The reason for which we add a noise to the variations of f is to avoid being trapped by a local minimum. It means that the maximum value $r_{\max}$ of the noise-rate should be chosen in such a way that, at least at the beginning of the process, a *bad transformation* (that is, a transformation yielding an increase of f), may be accepted, as it is also the case in simulated annealing for instance. As the added noises are chosen in an interval centered on 0, we may also reject a *good transformation* (that is, a transformation yielding a decrease of f), contrarily to what happens with simulated annealing...

```
• Compute an initial solution $C$
• $best\_sol \leftarrow C$
• $r \leftarrow r_{max}$
• $nb\_it \leftarrow 0$
• repeat the following steps:
    * $nb\_it \leftarrow nb\_it + 1$
    * let $C'$ be a neighbour of $C$: $C' \in N(C)$
    * draw a random number $\rho$ in $[-r, r]$ uniformly
    * compute $\Delta f(C, C')$ and $\Delta f_{noised}(C, C') = \Delta f(C, C') + \rho$
    * if $\Delta f_{noised}(C, C') < 0$, then $C \leftarrow C'$
    * if $f(C) < f(best\_sol)$, then $best\_sol \leftarrow C$
    * if $nb\_it \equiv 0$ (mod $nb\_it\_at\_fixed\_rate$), then decrease $r$
    * if $nb\_it \equiv 0$ (mod $\alpha NS$), then
        - apply an "unnoised" descent to $C$ (thus, we get a new $C$)
        - if $f(C) < f(best\_sol)$, then $best\_sol \leftarrow C$
    * if $nb\_it \equiv 0$ (mod $nb\_restarts$), then $C \leftarrow best\_sol$
• until $nb\_it = total\_nb\_it$
• return $best\_sol$.
```

Fig. 2. The scheme of Noising2.

3. Analysis of the noising methods applied to randomly weighted TSPs

In this section, we deal with a complete graph G of which the weights are not necessarily distances (they do not necessarily satisfy the triangular inequalities). Among our experiments, we only detail here those results that we got on eight graphs, but the conclusions for the other studied graphs (randomly generated or coming from the public library TSPLIB maintained by Reinelt [14]) are qualitatively the same; for this reason, we do not report the results got for them. The first two graphs, called "rand100" and "rand200" in the following, have respectively 100 and 200 vertices; their weights were randomly generated with a uniform distribution on $\{1, 2, \ldots, 100\}$ for rand100 and on $\{1, 2, \ldots, 1000\}$ for rand200. Their optimum values are unknown; the best value that we computed for rand100 is 286 (to be more specific, the average value provided by a simple descent is about 470); the best value that we computed for rand200 is 4516 (and the average value provided by a simple descent is about 7200). The other six graphs come from the library TSPLIB ([14]). Their names are gr120, KroA100, Pr152, KroB200, Lin318, pcb442; the numbers appearing in the names give their number of vertices; they are in fact graphs of which the vertices are points in the plane and the weights are the Euclidean distances but, in this section, we do not take advantage of this property and we do as if the weights were not distances. The minimum weights of an HC are known for these graphs: they are equal to 6942 for gr120, 21282 for KroA100, 73682 for Pr152, 29437 for KroB200, 42090 for Lin318 and 50778 for pcb442. We compare the results of Noising1 and Noising2 to those ones which have been provided by a classical simulated annealing method as in [4] or [10] (other simulated annealing schemes, for the TSP or for other problems as well, may be found in [1]).

One difficulty to compare these results comes from the fact that we may compare them according to two main criteria: the quality of the solutions (the values of f) and the consumed resources (for instance, the CPU time). In order to keep only one criterion, we give the same resources to each method. The choice of the CPU time does not seem fair, because for example simulated annealing may consume more or less time to compute exponentials, depending on the presence or not of a mathematical preprocessor speeding computations up. To measure the resources, we define an *evaluation* as the operation done when a value of f is computed. Then we give to each method the same amount of evaluations to find a good solution; as an elementary transformation involves also the computation of one value of f, and as conversely a value of f is computed only when we apply an elementary transformation (except the first time, when we compute the value of the starting solution), the number of evaluations given to a method is also the number of elementary transformations tried during the running of the method. This choice has also the advantage to lead to a notion of resources that are computer-free, programming language-free and, in some extent, less sensitive to a clumsy programming. In order not to deal with too big numbers, the number of evaluations given to each method, as well as the number of iterations, are supposed to be multiples of $NS = n(n-3)/2$.

For Noising2, we perform $4NS$ "noised" iterations between two "unnoised" descents: $\alpha = 4$. Still for Noising2, the number of restarts is chosen as about the square root of the total number of iterations divided by NS:

$$nb\_restarts \approx \sqrt{\frac{total\_nb\_it}{NS}}.$$

It involves that the number of iterations performed between two starts is approximately equal to $\sqrt{total\_nb\_it \times NS}$. To be more specific, one descent typically requires $7NS$ evaluations for rand100 and about NS for gr120.

The values of the parameters are chosen as the best ones found during the experiments done in order to tune them. For instance, for rand100, they are:

- For Noising1 and Noising2: $r_{\max} = 10$, $r_{\min} = 5$;
- For the simulated annealing method (SA in what follows): initial temperature = 8, number of temperature decreases = 35, geometric decreasing with a ratio equal to 0.925.

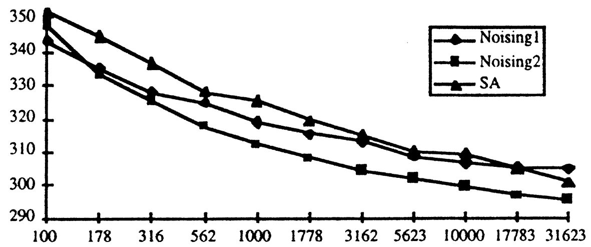
While they are, for gr120:

- For Noising1 and Noising2: $r_{\max} = 80$, $r_{\min} = 20$;
- For SA: initial temperature = 80, number of temperature decreases = 35, geometric decreasing with a ratio equal to 0.925.

Fig. 3 shows, for rand100 and rand200, the evolution of the average of f over 100 tests according to the number of evaluations *total_nb_it* devoted to the three methods Noising1, Noising2 and SA. Notice that the scale is logarithmic for the number of evaluations.

Figs. 4–6 show, for the TSPLIB graphs, the evolution of the average of f over 100 tests according to the number of evaluations (still with a logarithmic scale for the number of evaluations).

From these experiments, it appears that Noising2 seems better than SA independently from the number of evaluations. On the other hand, Noising1, which is usually the best for a small number of evaluations, loses its leadership quickly with respect to Noising2, more progressively with respect to SA, to become less efficient than SA when the resources (the number of evaluations) are important. These conclusions are the same for the



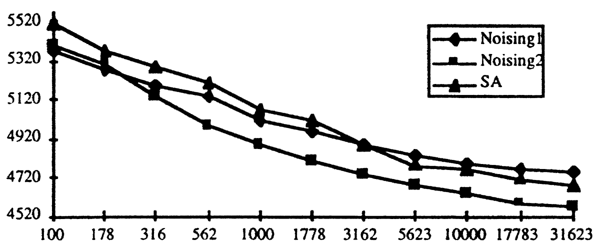


Fig. 3. Evolution of f in function of *total_nb_it*/*NS* for rand100 (left) and rand200 (right).

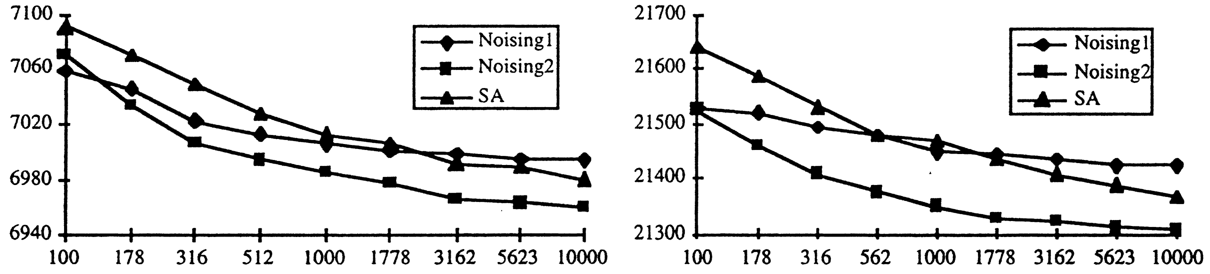


Fig. 4. Evolution of f in function of *total_nb_it/NS* for gr120 (left) and Kroa100 (right).

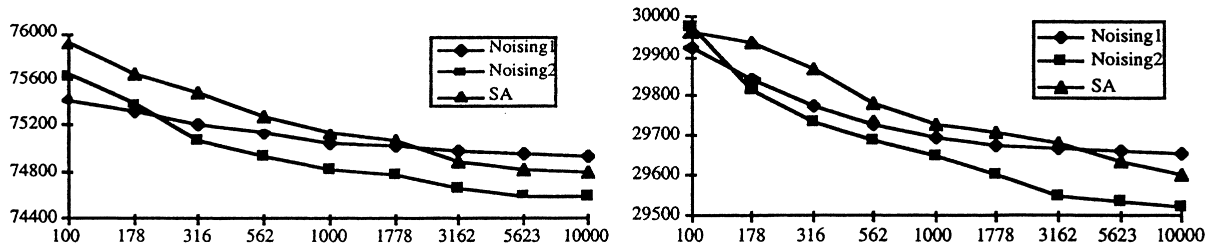


Fig. 5. Evolution of f in function of *total_nb_it/NS* for Pr152 (left) and KroB200 (right).

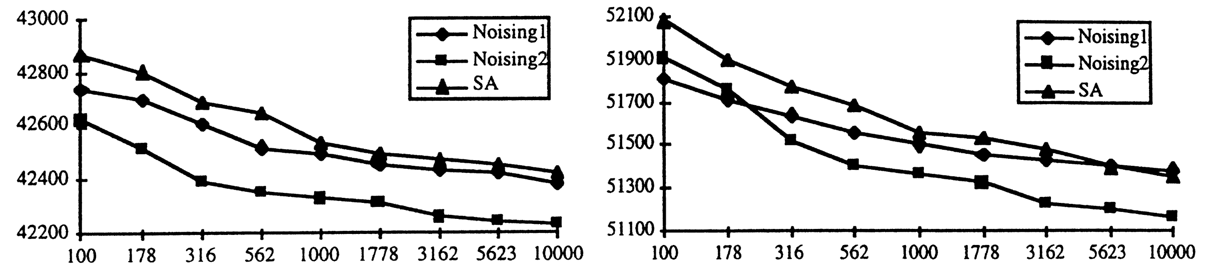


Fig. 6. Evolution of f in function of *total_nb_it/NS* for Lin318 (left) and pcb442 (right).

other graphs (randomly weighted or coming from the TSPLIB) that we tested and for which we do not report the detailed results, similar to the previous ones.

About the CPU time, for the same number of evaluations, our SA (we did not try the schemes described in [1], though they may be more powerful than the one adopted here) is approximately three times longer than the noising methods. So, if the resources would have been the CPU time, the gap between SA and Noising2 would be still more important and the progression of Noising1 towards SA still slower. This larger amount of time for SA can be explained by different reasons: the random exploration of the neighbourhood, the fact that SA accepts more transformations than Noising1 or Noising2 (and such an accepted transformation involves an update of the current solution, which is done in $O(n)$ while the rest of an iteration is in $O(1)$), the use of mathematical functions which consume time (the exponential), a greater use of random numbers, which also consumes time... If we decide to choose the CPU time as the resources, then the noisings are clearly better than SA. If we consider the number of evaluations *total_nb_it* as the resources, Noising2 is better than SA for any value of *total_nb_it* while Noising1 is better than SA when *total_nb_it* is less than about 3000*NS*.

We studied also the sensitiveness of the noisings with respect to their parameters. Except the total number of iterations, which is obviously important but for which it is useless to do an experimental study (the greater this number, the better the algorithm), these parameters are $r_{\max}$ and $r_{\min}$ for Noising1 and Noising2, and also α and *nb_restarts* for Noising2. Our aim is to know whether they must be tuned precisely or not in order to get an efficient scheme for the noisings.

Here again, we report only a part of our experiments, because the results stated below are qualitatively the same for the other experiments. The following study of the sensitiveness was done on rand100 and on Noising2 with 1000*NS* evaluations (in fact, we tried other quantities for the number of evaluations: it is remarkable that we got almost the same tunings for $r_{\max}$ and $r_{\min}$; this conclusion holds also for Noising1 and for the other graphs). First, we give to $r_{\min}$ the best value that we found, that is 5, and we vary $r_{\max}$ from 6 to 16. Fig. 7 shows the average value of f over 100 trials, for each value of $r_{\max}$. Then, we give to $r_{\max}$ the best value that we found, that is 10, and we vary $r_{\min}$ from 0 to 10. Fig. 8 shows the average value of f over 100 trials, for each value of $r_{\min}$.

From these figures, it appears that a too small value for $r_{\max}$ does not allow to get good results: then the behaviour of the method gets closer to the one of a descent (but it still remains quite better than a descent), because the noises do not change enough the values taken by f. On the other hand, giving a too great value to $r_{\max}$ consumes resources uselessly at the beginning of the process: it is like a blind walk in the space of solutions. Then, when the values taken by $r_{\max}$ become more interesting, that is, smaller, the remaining resources are no longer enough to find a solution as good as the one found when $r_{\max}$ is about 10. Nonetheless, we see that the quality of the solution is almost the same for $r_{\max}$ belonging to {9, 10, 11, 12}, and even for $r_{\max}$ belonging to {8, 13, 14} the solution is not far the one provided by $r_{\max} = 10$. This means that it is not crucial to find the optimal value of $r_{\max}$ very precisely: finding it at 10% or even 20% can be satisfying.

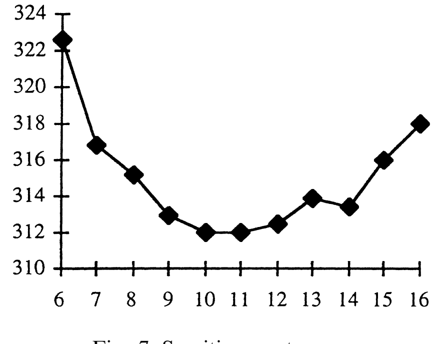


Fig. 7. Sensitiveness to $r_{\max}$.

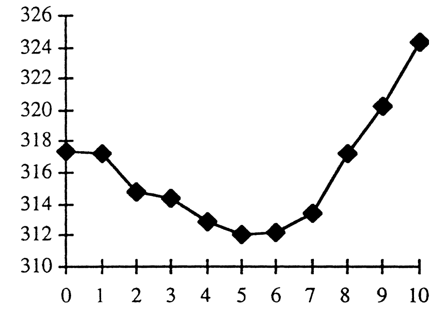


Fig. 8. Sensitiveness to $r_{\min}$.

For $r_{\min}$, a too great value, near the one of $r_{\max}$, is obviously a bad choice: we never consider the genuine function to optimize and so, if the value of $r_{\max}$ is properly chosen in order to be efficient, the process will optimize a "noised" function which can be far from the genuine one. On the other hand, if we consider the values of $r_{\min}$ near 0, we see that the result is less good than with $r_{\min} = 5$. A more detailed analysis on what happens at each iteration shows that the last iterations are then useless: the noises are not high enough to change the variations of the function significantly, and so we consume resources uselessly at the end of the process. Anyway, this phenomenon disappears when we increase the resources. If we allow more evaluations, the solution found with $r_{\min} = 0$ is the same as the one found with the optimal value of $r_{\min}$. Thus, if the user has the possibility to spend time for some useless iterations, it can be a good

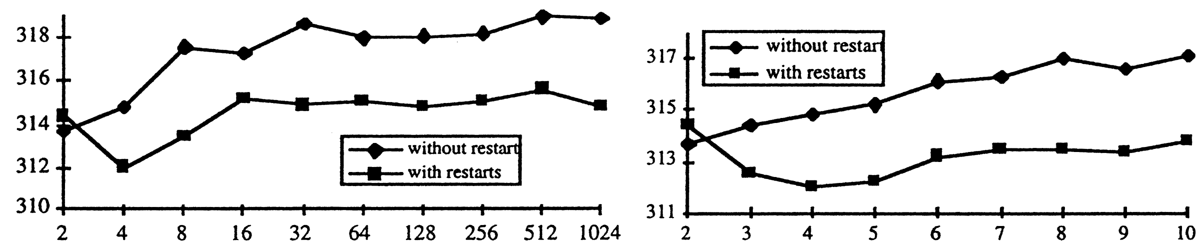


Fig. 9. Sensitiveness to α.

deal for him or her to increase the number of evaluations a little bit and to fix $r_{\min}$ to 0, rather than spending time to look for a better value of $r_{\min}$. We may also observe that, as for $r_{\max}$, it does not seem very important to find the best value of $r_{\min}$: here, choosing $r_{\min}$ in $\{4, 5, 6, 7\}$ or even choosing $r_{\min} \leqslant 7$ leads to rather good results. Other experiments on other problems (see [6] for references) confirm these conclusions.

For the study of α and *nb_restarts*, we consider rand100 once again, still with 1000*NS* evaluations (but, once again, we did many other experiments with the same qualitative conclusions). First, we vary α with $nb\_restarts = 32$ (since $\sqrt{1000} \approx 31.6$) or without this variant ($nb\_restarts = 0$). Fig. 9 shows the evolution of the average value of f over 100 trials for $\alpha = 2^{\beta}$ with $1 \leqslant \beta \leqslant 10$ (for $\beta = 10$, we get $\alpha = 1024$: so there is no "unnoised" descent, since the total number of evaluations is equal to 1000*NS*) and for $2 \leqslant \alpha \leqslant 10$. From these results, we see that the optimal value of α is rather precisely equal to 4 (though 3 and 5 give almost the same results) if the restart variant is applied simultaneously. But, if we never come back to the best solution found since the beginning of the process, then applying unnoised descents more frequently seems to be better; in fact, it is a kind of compensation: the possible wandering due to the addition of noises can be balanced by a periodic return to the best solution found since the beginning or by applying a descent with respect to the genuine function; both are ways to come back to a solution better than the current one, what could be a good thing in case of wandering.

We see also from Fig. 9 that combining the two variants gives better results than applying only one of them. This is confirmed by the results of Fig. 10 in which we vary the number v of iterations between two restarts divided by *NS* (still with $total\_nb\_it = 1000NS$):

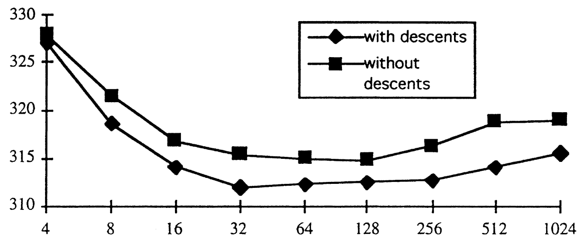


Fig. 10. Sensitiveness to $v = 1000/nb\_restarts$.

$$v = \frac{total\_nb\_it}{nb\_restarts \times NS} = \frac{1000}{nb\_restarts},$$

with $\alpha = 4$ or without applying "unnoised" descents (the scale for v is still logarithmic). Many values for *nb_restarts* seem to be correct, except the too important ones (associated with the small values of v). For $v < 16$, coming back to the best solution is too frequent and limits the diversity of the search too much. From these experiments and others done on other problems (see [6]), it seems that taking $nb\_restarts \approx \sqrt{total\_nb\_it/NS}$ (what gives $nb\_restarts = 32$ here) is generally a good choice.

4. Analysis of the noising methods applied to Euclidean TSPs

In this part, the vertices of G are points in the plane and the valuation of an edge $\{x_i, x_j\}$ is the

Euclidean distance between these two points x_i and x_j.

We may take advantage of the triangular inequalities satisfied in the Euclidean case: there will be no crossing edges in an optimal HC. One consequence of this is that, to get a rather good initial solution, we may do the following (as in [4]): first, we cluster the vertices into k regions; then the vertices in the same region are linked together by a Hamiltonian sub-chain; finally, the k Hamiltonian sub-chains are linked into a unique Hamiltonian chain by using a given tour of the regions. To improve this initial solution, we apply the Lin's 2-opt once again, restricting its application to edges of which the vertices belong to a same region or to two adjacent regions. Thus the neighbourhood size NS is quite less than in the general case, that allows to save CPU time by reducing the number of tried elementary transformations.

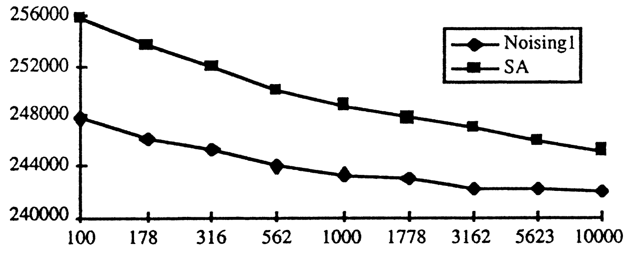
We detail here the results obtained only for four graphs; but, as for Section 3, the conclusions are the same with other graphs (randomly generated or coming from the TSPLIB) on which we tested the methods. The two first graphs come from the library TSPLIB [14], their names are vm1084 and d1291, there are 1084 vertices for vm1084 and 1291 for d1291; they are given by their coordinates in the plane and the weight of an edge $\{x_i, x_j\}$ is given by the round value of the Euclidean distance between x_i and x_j. The weights of the optimal solutions are known: they are equal to 239297 for vm1084 and to 50801 for d1291. The third graph, named Euc1000 below, has 1000 vertices belonging to the unit square; their locations are chosen randomly with a uniform distribution on the unit square; the minimum weight is not known: the best one we found is around 23.161. The fourth one is called Grid2500 below; its $n = 2500$ vertices are located on the crossings of a regular grid in the unit square: their coordinates are $(0.01 + 0.02i, 0.01 + 0.02j)$ with $0 \leqslant i \leqslant 49$ and $0 \leqslant j \leqslant 49$; it is easy to see that the minimum weight of an HC is $\sqrt{n} = 50$: an optimal solution goes only through "horizontal" or "vertical" edges.

This study is quite similar to the previous one. In order to apply the above restriction to the Lin's 2-opt, the plane containing the vertices is divided into 400 (resp. 324 and 784) little squares for vm1084 and d1291 (resp. for Euc1000 and Grid2500), so that each square contains only few vertices (as in [4]). The average neighbourhood size NS is about 26,400 for vm1084, 37,500 for d1291, 27,800 for Euc1000 and 71,700 for Grid2500. We only compare Noising1 and SA (we shall see latter why we do not consider Noising2). The values of the parameters are chosen as the best ones that we found; they are tuned as follows:

- for Noising1: $r_{\min} = 0$ for the four graphs; $r_{\max} = 700$ for vm1084, 200 for d1291, 0.04 for Euc1000, 0.03 for Grid2500;
- for SA: initial temperature = 400 for vm1084, 90 for d1291, 0.02 for Euc1000, 0.03 for Grid2500; number of temperature decreases = 60 for vm1084 and d1291, 30 for Euc1000 and for Grid2500; geometric decreasing with a ratio equal to 0.925 for the four graphs.

With these values, we get the results displayed by Figs. 11 and 12: they show the evolution of the average values taken by f over 100 tests when the number of evaluations devoted to Noising1 or to SA varies (with a logarithmic scale) from $100NS$ to $10{,}000NS$.

We see that Noising1 is more efficient than our SA, whatever the number of evaluations is. Notice



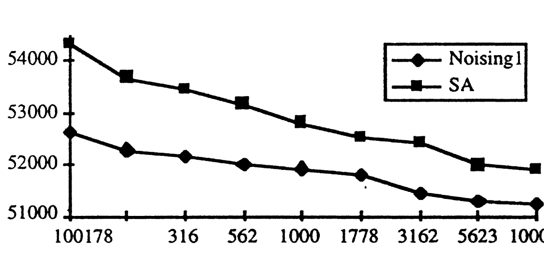
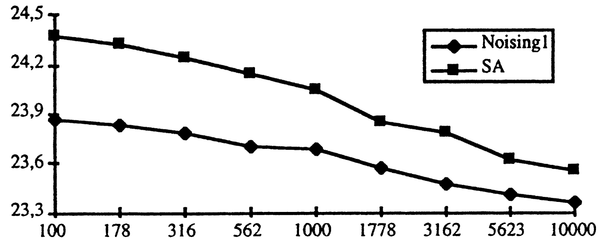


Fig. 11. Evolution of f in function of $total\_nb\_it/NS$ for vm1084 (left) and d1291 (right).



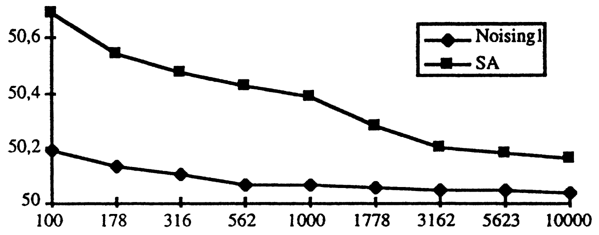
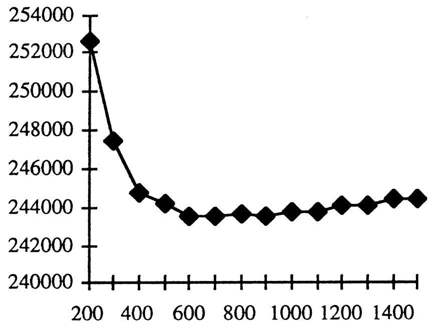


Fig. 12. Evolution of f in function of *total_nb_it*/*NS* for Eucl1000 (left) and Grid2500 (right).

that, for a same number of evaluations, the CPU time needed by SA is about three times the one consumed by Noising1, for the same reasons as before.

Fig. 13 shows the sensitiveness of Noising1 with respect to $r_{\max}$ (left) and to $r_{\min}$ (right) when applied to vm1084 (we get the same kind of results for the other graphs). For this, we fix the number of evaluations to 1000*NS* (here also, other choices for the number of evaluations lead to similar tunings); $r_{\min}$ is equal to 0 when $r_{\max}$ varies, and $r_{\max}$ is equal to 400 when $r_{\min}$ varies. As for the randomly weighted TSPs, we observe that it is better not to choose $r_{\max}$ too small; here, the values belonging to [400, 1500] give satisfying results with an optimum around 600. For $r_{\min}$, it is clearly better to choose it equal to 0, so that Noising1 finishes like an "unnoised" descent.

We also studied the effect of the variants leading to Noising2 (introduction of "unnoised" descents and periodic restarts from the best solution computed since the beginning). The results are given by Fig. 14. The left part of Fig. 14 shows the evolution of f in function of $\alpha = 2^{\beta}$ with $1 \leqslant \beta \leqslant 10$



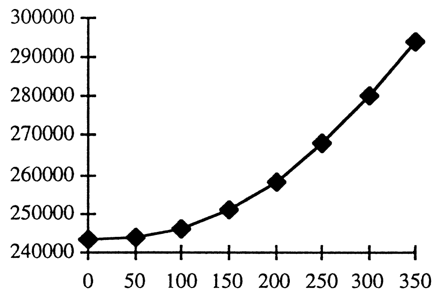
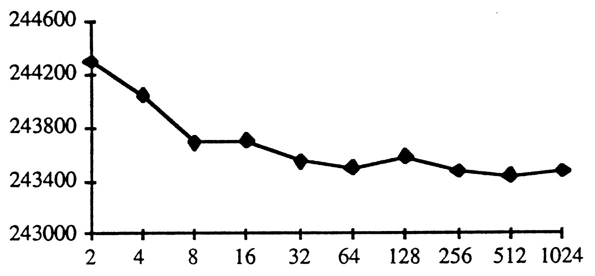


Fig. 13. Sensitiveness to $r_{\max}$ (left) and to $r_{\min}$ (right).



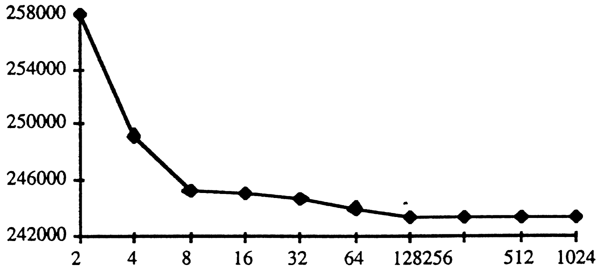


Fig. 14. Sensitiveness to α (left) and to $v = 1000/nb\_restarts$ (right).

($\beta = 10$ involves that there is no "unnoised" descent) while there is no restart. The right part of Fig. 14 shows the evolution of f in function of

$$v = \frac{total\_nb\_it}{nb\_restarts \times NS} = \frac{1000}{nb\_restarts}$$

($v = 1024$ involves that there is no restart) while there is no "unnoised" descent. It clearly appears that, for this type of TSP, these variants are not desirable, their effects are rather negative. Thus it seems better not to apply them and it is the reason why we have detailed here the results only for Noising1 and not for Noising2 (nevertheless, the results provided by Noising2 are better than those of SA).

As said above, Euc1000 belongs to the family of the Euclidean TSPs for which the vertices are randomly drawn inside the unit square with a uniform distribution of probability. In such a case, Beardwood et al. [3] proved that the ratio of the optimal value f^* of f to $\sqrt{n}$ tends to a constant limit λ when n tends to infinity:

$$\lim_{n \to +\infty} \frac{f^*}{\sqrt{n}} = \lambda.$$

They estimated the value of λ and found $\lambda \approx 0.749$. According to Stein [15], the value of the limit would be rather $\lambda \approx 0.765$. Johnson and McGeoch [8] think that these two values are overestimated and propose $\lambda \approx 0.7124$. In our experiments with Euclidean TSPs on 1000 vertices, we get ratios equal to 0.739 for Noising1 and equal to 0.745 for SA (for the same value of n, the ratio provided by SA in [4] is equal to 0.749).

5. Conclusions

The study described in this paper is done with a fixed number of evaluations. It is the criterion which is less favourable to the noising method. Considering for instance the CPU time would clearly reinforce the conclusions: for the same amount of evaluations, our SA is about three times longer than Noising1 or Noising2. Another relevant criterion would be the number of 2-opt transformations performed during the process. We observe that, still for the same amount of evaluations, our SA makes about eight times more transformations: the conclusion would not be different, but the gap between the methods would be more important.

The noising method is quite simple to tune. For instance, Noising1 gives pretty good results with $r_{\min}$ equal to 0. Then, it is sufficient to tune $r_{\max}$ and to choose the number of evaluations with respect to the time that the user can spend on his or her problem. As the optimum value of $r_{\max}$ depends very little on the number of evaluations, it is easy to find a quick and good tuning of $r_{\max}$ with rather few evaluations (but not too few!) and to keep it for more evaluations. Moreover, we see that the sensitiveness of this parameter is not very important; so it is not necessary to tune it very precisely.

For the scheme of Noising2, even if we set the coefficients α and *nb_restarts* as parameters in order to study the effect of the corresponding variants, we advise the user to fix their values a priori to 4 for α and to $\sqrt{total\_nb\_it/NS}$ for *nb_restarts*, as we do above. This strategy avoids the tuning of two parameters and gives usually good results. Indeed, we tried these variants on other problems (see [6] for references) and we got almost always the same tuning.

Of course, the user may also adopt these values in a first phase and try to improve them later. Another possibility, that we are studying now ([7]), consists in designing a self-tuned version of the noising method that the user could apply to a broad range of combinatorial optimization problems with only one parameter: the total amount of CPU time (or another resource) that he or she would like to spend to solve his or her problem. About the comparison between Noising1 and Noising2, we observed that usually Noising2 is more efficient than Noising1, though it is not always the case, as shown for instance by the Euclidean TSP...

We think that the noising method deserves interest because it provides, as well as (and sometimes better than) other methods like simulated annealing, a way of solving hard problems like the TSP with a usually "good" solution within a "reasonable" CPU time. We hope that other researchers will try to apply the noising method, alone or hybridized with other heuristics, to other problems; our aim will be reached if this paper can stimulate them to do so and can help them in adapting the noising method to their problems.

References

[1] E.H.L. Aarts, J.H.M. Korst, P.J.M. van Laarhoven, Simulated annealing, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, New York, 1997, pp. 91–120.

[2] E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, New York, 1997.

[3] J. Beardwood, J.H. Halton, J.M. Hammersley, The shortest path through many points, in: Proceedings of the Cambridge Philosophical Society, vol. 55, 1959, pp. 299–327.

[4] E. Bonomi, J.-L. Lutton, The *N*-city travelling salesman problem: Statistical mechanics and the Metropolis algorithm, SIAM Review 26 (1984) 551–568.

[5] I. Charon, O. Hudry, The noising method: A new combinatorial optimization method, Operations Research Letters 14 (1993) 133–137.

[6] I. Charon, O. Hudry, The noising method: A generalization of some metaheuristics, submitted for publication.

[7] I. Charon, O. Hudry, A totally self-tuned noising method, in preparation.

[8] D.S. Johnson, L.A. McGeoch, The traveling salesman problem: A case study, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, New York, 1997, pp. 215–310.

[9] M. Jünger, G. Reinelt, G. Rinaldi, The traveling salesman problem, in: M.O. Ball et al. (Eds.), Handbooks in OR and MS, vol. 7, Elsevier, Amsterdam, 1995, pp. 225–330.

[10] P.J.M. van Laarhoven, E.H.L. Aarts, Simulated Annealing: Theory and Applications, Kluwer, Dordrecht, 1987.

[11] E.L. Lawler, A. Lenstra, G. Rinnooy Kan, The Travelling Salesman Problem, Wiley, New York, 1985.

[12] S. Lin, Computer solutions for the travelling salesman problem, Bell System Technical Journal 44 (1965) 2245–2269.

[13] C. Reeves (Ed.), Modern Heuristic Techniques for Combinatorial Problems, McGraw-Hill, London, 1995.

[14] G. Reinelt, TSPLIB – A traveling salesman problem library, ORSA Journal on Computing 3, 4 (1991) 376–384.

[15] D. Stein, Scheduling dial-a-ride transportation systems: An asymptotic approach, Ph.D. thesis, Harvard University, Cambridge, MA, 1977.