Pergamon

Int. Trans. Opl Res. Vol. 3, No. 1, pp. 1–21, 1996
Copyright © 1996 IFORS. Published by Elsevier Science Ltd.
Printed in Great Britain. All rights reserved
0969-6016/96 $15.00 + 0.00

S0969–6016(96)00004–4

Heuristics from Nature for Hard Combinatorial Optimization Problems*

A. COLORNI,[1] M. DORIGO,[1] F. MAFFIOLI,[1] V. MANIEZZO,[2] G. RIGHINI[3] and M. TRUBIAN[2]

[1]Politecnico di Milano, Italy, [2]Univeritàdi Bologna, Italy and [3]University di Milano, Italy

In this paper we try to describe the main characters of Heuristics 'derived' from Nature, a border area between Operations Research and Artificial Intelligence, with applications to graph optimization problems. These algorithms take inspiration from physics, biology, social sciences, and use a certain amount of repeated trials, given by one or more 'agents' operating with a mechanism of competition–cooperation. Two introductory sections, devoted respectively to a presentation of some general concepts and to a tentative classification of Heuristics from Nature open the work. The paper is then composed of six review sections: each of them concerns a heuristic and its application to an NP-hard combinatorial optimization problem. We consider the following topics: genetic algorithms with timetable problems, simulated annealing with dial-a-ride problems, sampling and clustering with communication spanning tree problems, tabu search with job-shop-scheduling problems, neural nets with location problems, ant system with travelling salesman and quadratic assignment problems. Copyright © 1996 IFORS. Published by Elsevier Science Ltd.

Key words: Metahewlistics comparison, NP-hard optimization, single and multiagent search.

1. INTRODUCTION

In recent years, one of the most important and promising research fields has been 'Heuristics from Nature', an area utilizing some analogies with natural or social systems (Schwefel and Männer, 1990; Männer and Manderick, 1992) to derive non-deterministic heuristic methods and to obtain very good results in NP-hard combinatorial optimization problems (Glover and Greenberg, 1989; Reeves, 1993). In this paper we try to describe the main characters of this sector, a border area between Operations Research and Artificial Intelligence, with some applications to combinatorial optimization problems. We propose a taxonomy which can be useful to better understand the direction of current research.

We start with a definition of the 'natural' metaphor that will be considered: systems are derived from physics, biology and social sciences. Heuristics are obtained

- using a certain number of repeated trials,
- employing one or more 'agents' (neurons, particles, chromosomes, ants, and so on),
- operating (in case of multiple agents) with a mechanism of competition–cooperation,
- embedding procedures of self-modification of the heuristic parameters or of the problem representation.

Nature has two big engines: *selection* that rewards the stronger individuals and penalizes the weaker ones, and *mutation* that introduces elements of randomness and permits the birth of new individuals. In Heuristics from Nature we have a similar situation: selection is the basic idea for optimization, mutation is the basic idea for non-deterministic search.

The characteristics of Heuristics from Nature can be summarized as follows:

1. they (loosely) model a phenomenon existing in nature,
2. they are non-deterministic,
3. they often present implicitly a parallel structure (multiple agents),
4. they are adaptive.†

This leads to the emergence of a 'reasonable behaviour' for the system (Bourgine and Varela, 1992),

* Partially supported by a Research Project 'MPI 40%' of the Italian Ministry of University and Scientific and Technological Research, and by Progretto Finalizzato 'Trasporti 2' of the National Research Council of Italy.

† With the term 'adaptive' we indicate the capacity of the system to use feedback information for modifying its parameters and its internal model: a well known example is neural nets (with the modification of link weights).

Correspondence: A. Colorni, Dipartimento di Elettrnica e Informazione, Politecnico di Milano, 20133 Milano, Italy, e-mail address: colorni@elet.polimi.it

that could be defined 'intelligent' [here we assume the definition of Minsky (1985) that 'intelligence is the ability of solving difficult problems']: in our case, it means the production of very good solutions for a combinatorial optimization problem.

A *combinatorial optimization problem* (COP) is specified by a class of problem instances. An *instance* is defined by specifying implicitly a pair (S,f), where S is the (finite) set of all the feasible solutions, called *solution space*, and the *cost function* f is a mapping $f: S \to \mathbb{R}$. The optimum value of f is*

$$f_o := \min\{f(i): i \in S\},$$

and the set of optima is

$$S_o := \{i \in S: f(i) = f_o\}.$$

Our task is to find some solution $i_o \in S_o$.

The way a COP has been defined corresponds to what is commonly called a *search problem.* A simpler kind of problem, the *decision problem*, takes the following general form: given a solution space S, a cost function f, a threshold θ, does there exist a feasible solution $i \in S$ s.t. $f(i) \leqslant \theta$?

It is obvious that being able to solve a search COP enables one to solve the corresponding decision version, whereas the opposite is not necessarily true.

Let n be the length (i.e. the number of binary digits) of the encoding of a COP instance. If the maximum amount of computing time needed to solve any instance of length n is bounded from above by a polynomial in n, for every n, we say that the COP under study is solvable in polynomial time and that the corresponding solution algorithm is polynomial. If k is the maximum exponent of such a polynomial we say that the COP is solvable in $O(n^k)$ time.†

$\mathscr{P}$ denotes the class of decision problems for which there exists an algorithm *determining* in polynomial time for every instance if the answer is 'yes' or 'no', whereas $\mathscr{NP}$ denotes the class of decision problems for which there exists an algorithm *verifying* in polynomial time for every instance if the answer 'yes' is correct. Obviously $\mathscr{P} \subseteq \mathscr{NP}$ and it is one of the most outstanding problems of today's Mathematics to find out if this inclusion is a proper one or not.

In the late 1960s it became more and more clear that most decision combinatorial problems are in $\mathscr{NP}$, whereas only a few of them, although practically very relevant, belong to $\mathscr{P}$: the reader is referred to the seminal work of Garey and Johnson (1979).

Some of the most important COPs are in fact *NP-hard*, that is as difficult to solve exactly as even the hardest problem in $\mathscr{NP}$. Approaches for solving such problems exactly are all based on implicit enumeration of the feasible solutions and hence require in the worst-case an exponential number of steps: on the other hand, when the dimensionality n of the instance becomes larger and larger, as in most real-world applications, no exponential algorithm could be of practical utility.**

Therefore, complexity studies have strongly motivated the development of *heuristics* for hard COPs. This survey addresses some of the most successful recently proposed *randomized* heuristics for COPs. The use of random steps is motivated as a tool for exploring the solution space and hence obtaining a better confidence into the optimality of the found solution.

The paper begins with a general part, devoted to an introduction and a brief analysis of Heuristics from Nature, with a tentative classification of them. The paper is then composed of six sections: each of them is concerned with a class of Heuristics from Nature and its application to an NP-hard combinatorial optimization problem.

2. HEURISTICS FROM NATURE: A TENTATIVE CLASSIFICATION

Let us consider the most celebrated combinatorial optimization problem, the *Travelling Salesman Problem* (TSP), and let us suppose to have an 'agent' that randomly moves in the search space (an n-dimension graph): the only condition we impose is that the agent should be able to memorize in a list the cities already visited, to avoid repetitions. In this way after n moves the agent returns to the

*Without loss of generality let us consider minimization problems, unless otherwise said.

†A function $f(n)$ is $O(g(n))$ if and only if there exists a constant $c > 0$ s.t. $f(n) \leqslant cg(n)$ for all n.

**This picture is probably too pessimistic: on *average* in fact even exact implicit enumeration (like e.g. Branch-and-Bound) algorithms behave much better than in the *worst-case*.

starting city and obtains a solution. This random search procedures can be repeated, starting from different cities, for obtaining the classical 'multistart' algorithm (MS), our reference method.

You can obtain different heuristics as follows:

1. use a greedy technique for choosing each move of the agent,
2. use a local search technique (exchanging node positions) for improving a solution,
3. use a random local search technique and accept only improving exchanges,
4. use *m* agents starting from different cities,
5. use a population of agents with non-deterministic recruitment,
6. use a clustering technique for partitioning space and/or agents,
7. use a non-deterministic accepting rule for non-improving exchanges,
8. use information on the (last) moves for implementing a system memory.

The above ideas (introduced in a non-formal way) are basic steps for constructing a set of more and more sophisticated algorithms that are the object of this work. In particular, we will treat the following heuristics.

1. GA – Genetic algorithms (Holland, 1975; Goldberg, 1989), the general heuristic for discrete optimization, applied to timetable problems. A similar technique, not illustrated here for the sake of brevity, is the Evolution Strategies (ES): the reader is referred to (Rechenberg, 1973; Schwefel, 1981).
2. SA – Simulated annealing (Van Laarhoven, and Aarts, 1987), the well known method proposed by Metropolis *et al.* (1953) and revised for optimization by Kirkpatrick *et al.* (1983), applied to dial-a-ride problems.
3. SC – Sampling and clustering (Boender *et al.*, 1982), a method proposed for discrete optimization problems in (Camerini *et al.*, 1986), applied to communication spanning tree problems.
4. TS – Tabu search (Glover, 1989, 1990), the general technique proposed by Glover, applied to job-shop scheduling problems. In this section some ideas of the Greedy random search method (GR) will also be presented.
5. NN – Neural nets (Hopfield and Tank, 1985), the well known paradigm proposed first in (McCulloch and Pitts, 1943) and revised by Hopfield for TSP (1984), presented here in a more recent form, and applied to location problems. A technique which is a sort of bridge between SA and NN is the so called Boltzmann Machine (BM) (Aarts and Korst, 1989).
6. AS – Ant system (Colorni *et al.*, 1991), a method proposed in 1991 by some of the authors of this paper, applied to travelling salesman and to quadratic assignment problems.

Many classification criteria are possible. Here we suggest the use of the following four characteristics:

(a) constructive (a1) vs improving algorithms (a2),
(b) non-structured (b1) vs structured space (b2),
(c) single solution (c1) vs population of solutions (c2),
(d) memoryless (d1) vs memorizing algorithms (d2).

Point (b), structure of the search space, refers to the definition of concepts such as distance, metrics, neighborhood, that we will introduce case by case: in general, classical heuristics as greedy (constructive), add-and-drop or Lin–Kernigan (1973) (improving), utilize the Hamming distance or the minimum number of swaps, and define neighbor solutions as those obtained with 'legal moves' in the search space.

Considering only the first three characteristics, we obtain Table 1.

Two main features have to be balanced in constructing heuristic algorithms:

- the degree of *exploitation*, that is the amount of effort directed to local search in the present region of the search space (if a region is promising, search more thoroughly);
- the degree of *exploration*, that is the amount of effort spent to search in distant regions of the space (sometimes choose a solution in a far region and/or accept a worsening one, to gain the possibility of discovering new better solutions).

These two possibilities are conflicting: a good trade-off between them is very important and must be carefully tuned in each algorithm.

Another trade-off that must be considered is between effort (viz. number of iterations) and efficacy

Table 1. A tentative classification of Heuristics from Nature

	b1–c1	b1–c2	b2–c1	b2–c2
a1	NN	MS	GR	AS
a2	BM	GA, ES	SA,TS	SC

(viz. value of the final solution): in some sense, the design of a good heuristic method is a multiattribute problem with two conflicting objectives, effort and efficacy.

One of the main open problems in this area is the study of the asymptotic properties of the algorithms (Aarts and Korst, 1989). On this subject, let us mention some general ideas which will be considered again later in this paper:

- in some algorithms there is a parameter (indicated as control, or learning, or equilibrium parameter) that varies slowly with the aim of avoiding local optima and permitting the exploration of the space;
- the more slowly this parameter varies the higher is the probability that the final solution obtained is a global optimum;
- it is possible to design 'metaheuristics' containing, as a feature, specific local searches for the optimum value of this control parameter (see, for instance, Glover, 1989).

3. GENETIC ALGORITHMS

3.1. Features of the method

Genetic Algorithms (GA) were first introduced by Holland (1975) as a highly robust search algorithm. Later on, especially in the work of Goldberg (1989), they were mainly used as optimization devices: we present here a standard GA applied to a maximization problem.

GA use the population genetics metaphor. In the GA community, an optimization problem is translated into the problem of finding the most fit *individual* (sometimes also called *chromosome*) within a *population*. Fitness is measured by a *fitness function*, which is functionally related to the objective function of the problem to be solved. Individuals are the equivalent of solutions and a population is a set of N individuals. Each individual consists of a collection (usually a string) of atomic elements called *genes*. Each gene can take values on a predefined set. The GA operate on the population modifying its components. Modifications occur according to genetic rules, implemented by the *genetic operators* explained in the following. Figure 1 shows the Pascal-like version of a standard GA.

The *Reproduction* operator obtains a new individual (NewPop$_i(t+1)$) taking into account the fitness Fit$_j$ of each individual Pop$_j$(t) in the current population **Pop**(t).

Recombine population is an operator which takes as input the population **NewPop** and returns a new population **CrossPop**. The recombination is performed by the *crossover operator*, which randomly extracts two individuals (parents), chooses with uniformly distributed probability a crossing point in the chromosomes representing the two parents, and then exchanges the values to the right of the crossing point, recombining them to generate two sons. The crossover operator is applied with probability p_c, independent of the specific individuals on which it is applied.

Mutate population is an operator which takes as input the population **CrossPop** and returns a new population **MutPop** in which some individuals underwent mutation. Each individual can be selected, with probability p_m, and selected individuals are mutated. For instance, if the individuals are coded as strings on $\{0,1\}$, mutation changes a 0 into a 1 and vice-versa. The *mutation operator* introduces basic variations in the population, guaranteeing the possibility of exploring the whole search space, independently from the specific initial population.

3.2. Application to timetable problems

GA have been successfully applied to a wide variety of theoretical and practical problems. Here we

```
t: = 0; initialize PoP(t) with N chromosomes $Pop_i$(t);
while not (terminating condition) do
  for i: = 1 to N do $Fit_i$: = fitness ($Pop_i(t)$);
  for i: = 1 to N
    $NewPop_i(t+1)$: = randomly choose $Pop_j(t) \in \mathbf{Pop}(t)$
                    with probability $p_j = \frac{Fit_j}{\sum_{k=1}^{N} Fit_k}$;
    CrossPop(t + 1): = recombine population(NewPop(t + 1));
    MutPop(t + 1): = mutate population(Cross Pop(t + 1));
    Pop(t + 1): = MutPop(t + 1);
    t: = t + 1
end while
```

Fig. 1. Genetic Algorithsm.

propose an application to timetable problems (Chahal and De Werra, 1989), a specific case of which is presented in Colorni *et al.* (1990).

The specific problem we faced regards the construction of the lesson timetable for an Italian high school. The choice of this particular problem instance gave us the opportunity of testing the system and obtaining a reliable validation of the output provided by the implemented model.

The problem is described by:

- a list of m teachers;
- a list of p classes involved;
- a list of n weekly teaching hours for each class;
- the curriculum of each class, that is the list of the frequencies of the teachers working in the class;
- some external conditions (for example the hours during which some teachers are involved in other sections or activities).

We want to compute

$$\min f(s,D,W,P)$$

where s is the number of unfeasibilities, as defined in the following; D is the set of didactic costs (e.g., having the hours of the same subject clustered in a few days of the week); W is a set of organizational costs (e.g., having no teacher available for possible temporary teaching posts); P is a set of personal costs (e.g., having the day-off in an undesired day of the week).

Every solution (timetable) generated by our algorithm is feasible if it satisfies the following constraints:

- every teacher and every class must be present in the timetable in a predefined number of hours;
- there may not be more than one teacher in the same class in the same hour;
- no teacher can be in two classes in the same hour;
- there can be no 'uncovered hours' (that is, hours when no teacher has been assigned to a class).

To apply a GA to this problem, it is necessary to identify:

1. a meaningful *representation* for the candidate solutions;
2. a *fitness function* to assess different solutions;
3. a set of useful *genetic operators*, that can efficiently recombine and mutate the candidate solutions.

These points have been tackled as follows.

Representation. This aspect has been solved by separately scheduling groups of p (usually 10) classes, characterized by a list of m (usually 20–25) shared teachers, over a period of n (usually 30) weekly teaching hours. This allows us to represent the problem as a matrix R (an $m \times n$ matrix of elements r_{ij}) where each row corresponds to a teacher and each column to an hour. The alphabet chosen is a set A ($r_{ij} \in A$) of the jobs that teachers have to perform: its elements are the lessons to be taught and other specific activities. Every element r_{ij} of the matrix R is a gene: its allelic value may vary on a subset of A specific to the teacher corresponding to the row containing the gene. Each individual of the GA population is coded by a matrix R.

Fitness function. This aspect was solved after several interactions with school teachers: we came out with a generalized cost, hierarchically structured, which represents the distance existing between the current timetable and the ideal needs of the school. The cost is defined through a set of weights interactively chosen by the user. The hierarchical structure has been chosen to acknowledge the different relevance of the several groups of problem conditions: the differences are reflected in the weights, which have different orders of magnitudes. Specifically, we identified three levels of weights: at *level 1* (feasibility conditions) we penalize possible superpositions of teachers (two or more teachers during the same hour in the same class) and 'uncovered hours' for the classes (hours when in a class there is no teacher); at *level 2* we considered didactical and organizational requirements (e.g. not the same teacher every day at the last hour, a uniform distribution of the hours of the same subject over the week, and so on); at *level 3* we consider the distances between the current timetable and the preferences expressed by each teacher for his/her specific timetable. We also defined a function connecting generalized cost (to be minimized) with fitness function (to be maximized).

Genetic operators. This points has required the adaptation of the standard mutation and crossover operators. Moreover, a local search algorithm has been introduced as a further operator which has two beneficial effects: the first is to speed-up search [a possibility advocated by several authors, see for instance (Mühlenbein, 1989)], the second is the opportunity to recover feasible solutions from unfeasible ones, when a specific fitness function is used (Colorni *et al.*, 1990).

4. SIMULATED ANNEALING

4.1. Features of the method

To apply any local search procedures to a COP, we must define a *neighborhood* $\mathcal{N}(i)$ of a feasible solution $i \in S$. We believe that this is best done in an operational way saying that $\mathcal{N}(i)$ is a subset of S *reachable* from i by a specified *neighborhood exploration algorithm:* we call such a procedure, generating all neighbors $j \in \mathcal{N}(i)$ in a certain order, GEN(i,j). The output of a local search procedure, using GEN(i,j) is called a local minimum with respect to $\mathcal{N}(i)$, since for all $j \in \mathcal{N}(i)$ we have $f(i) \leqslant f(j)$.

There are several ways of enhancing the quality of the solution. *Simulated Annealing* (SA) is one of them, which introduces a more sophisticated way of moving from the current solution to one of its neighbors, accepting with a certain probability to move also when the quality of the new solution is worse than the previous one.

The origin of SA goes back to 1953 when it was used to simulate on a computer the annealing process of crystals. The idea to apply this methodology to COPs came much later (Kirkpatrick *et al.*, 1983). It is assumed that procedure GEN(i,j) randomly generates $j \in \mathcal{N}(i)$ with a uniform distribution over all members of $\mathcal{N}(i)$. Let Δ be the difference $f(j) - f(i)$: the probability that the algorithm accepts j as the new current feasible solution is given by

$$\text{Prob}\{j \text{ after } i\} = \begin{cases} 1 & \text{if } \Delta < 0 \\ e^{-\Delta/t} & \text{if } \Delta \geqslant 0. \end{cases} \tag{1}$$

In the classical SA algorithm this process should continue in principle until equilibrium is reached. Then the control parameter t (often called 'temperature', remembering the origin of SA) is decreased and another sequence of iterations begins. The whole procedures stops when t is so small that (in practice) j is accepted only if $\Delta < 0$, in which case SA is not different from the local search procedure. A Pascal-like code for SA is reported in Fig. 2.

In this code $g(.)$ is the updating rule for parameter t, often simply $t := at$ with some constant $a < 1$. The logical flag λ remains TRUE only if no movement has been accepted during a full sequence of iterations at the same temperature t. The most 'mysterious' part of Fig. 2. is the subroutine EQUILIBRIUM utilized to decide when t has to be updated. The verification of a true equilibrium condition is indeed often quite difficult. A simple way out is to introduce another parameter H, which

```
repeat
  $\lambda$: = TRUE; $l$: = $i$;
  repeat
    GEN($i$,$j$);
    if $f(j) < f(l)$ then $l$: = $j$;
    $\Delta$: = $f(j) - f(i)$;
    if $\Delta < 0$ then set $i$: = $j$ and $\lambda$: = FALSE else do
      generate $q$ uniformly random in [0,1);
      if $q < e^{-\Delta/t}$ then set $i$: = $j$ and $\lambda$: = FALSE
  until EQUILIBRIUM;
  $t$: = $g(t)$
until $\lambda$;
output $l$
```

Fig. 2. Simulated Annealing.

fixes the maximum number of iterations to be performed at constant t (likely depending on the number of variables of the COP considered).

It has been recognized long ago that the behaviour of SA can be modelled as a succession of *Markov chains*, one for each value of t. Although theoretically very appealing, the conditions for convergence of SA to a global optimum obtained in this way tell very little about the way the algorithm will behave in practice. In fact in order to get probability 1 of converging to a global optimum we should stick to Markov chains with an infinite number of transitions and to a function $g(t)$ for updating the temperature (a part of the so-called *cooling strategy*) going to zero no more rapidly than $O(1/\log n)$ (Van Laarhoven and Aarts, 1987). These conditions are clearly impossible to satisfy in practice. What would help is not a study of asymptotic convergence, as it has been done in the recent past, but rather a deeper insight into the *rapidity* of convergence to the stationary distribution. This is much more difficult and only very preliminary results have yet appeared (Diaconis and Hanlon, 1992).

Cooling strategies. Given the situation previously summarized as to the theoretical understanding of SA, we are bound to consider this method (as already pointed out in the introduction to this section) as nothing but a *randomized local search*, whose control parameters have to be chosen every time one wishes to apply it to an instance of a COP. This set of parameters is called a *cooling strategy* and contains the following items: the initial (high) value of the 'temperature' t_0; the rule $g(t)$ for updating t; the length H of each Markov chain. A fourth item is often specified, when the procedures of Fig. 2 is simplified in its stopping criterion, and uses, instead of the logic variable λ, the simple fact that the temperature is low enough: in this case we need to specify also the final (low) value of the temperature, t_f.

Extensive experimentations (Van Laarhoven and Aarts, 1987) have pointed out that the choice of the cooling strategy is crucial for the success of the SA algorithm, which may arrive at very good quality solutions, at the expense of quite large computing times. Van Laarhoven and Aarts suggest a polynomial-time cooling schedule and specify how to obtain the values of the parameters once the problem and its neighbourhood have been decided.

4.2. *Application to dial-a-ride problems*

A large number of SA applications have been produced in many different fields (Aarts and Korst, 1989). From these, we discuss here the dial-a-ride (DaR) problem for two reasons: because it is one of the more complex models in vehicle routing and because our research group developed real applications in weak demand zones and in urban areas of Italy as non-conventional public transport systems (Colorni *et al.*, 1990). In (Van Laarhoven *et al.*, 1988) one can find other applications of SA to problems considered in this paper.

A DaR problem is defined by three types of data (Psaraftis, 1983):

(a) a network R of r nodes, with minimum distances d_{ij} between every pair of nodes ($i,j = 1,\ldots,r$) *and* *different speeds* $v(\lambda)$ that permit the computation of different running times $\tau_{ij}(\lambda)$ in various day periods ($\lambda = 1,\ldots,\Lambda$; usually at least two periods can be considered, the normal and the congested situation, but for sake of simplicity in the following we will consider only the normal one);

(b) a set N of n customers, such that for each customer $k(k = 1,\ldots,n)$ we define the node i_k of pick-up, the node j_k of delivery, the earliest time $\tau 1_k$ of pickup, the latest time $\tau 2_k$ of delivery, eventually the numbers η_k of passengers;

(c) a service fleet M of m vehicles, with capacity q_l and depot node $i_l(l = 1,\ldots,m)$.

A DaR service can be studied with a static or a dynamic model: in the static model the requests $(i_k,j_k,\tau 1_k,\tau 2_k,\eta_k)$ of all the customers are known in advance, in the dynamic one the calls of the customers can arrive during the service (hence every call could be the last of the period and it is not possible to forecast the future pattern of the requests).

In a static problem four different objective functions are possible:

- f_1 (to be maximized) = the number of customers accepted by the service;
- f_2 (to be maximized) = the quality of the service, measured by the average of the quality levels for the accepted customers;
- f_3 (to be minimized) = the number of vehicles actually used for the service;
- f_4 (to be minimized) = the total length (or the total time) of travel for the vehicles of the fleet.

Although the function f_4 is the most studied, in real applications a mix of f_1,f_2,f_3 is the actual objective of the planner.

In the dynamic model with a real-time management of the service, usually m is fixed (so the value of f_3 is a constant), the total length is not so important (so the function f_4 can be disregarded), the number n of customer is unknown (so the function f_1 is considered only as a general target), the function f_2 (level of quality) is the real objective of the problem.

A reasonable measure of the service level of quality for the customer k is a ratio between the sum of his waiting and riding times and the theoretical time $\tau_{i_kj_k}$ of the travel. Notice that with this definition the *level of service* for customer k, here indicated as los_k, must be minimized. The values of los_k is constrained as follows

$$1 \leqslant \mathrm{los}_k \leqslant \mathrm{GLOS} \qquad k = 1,\ldots,n \tag{2}$$

GLOS being the *guaranteed* level of service, a parameter that permits a tradeoff between acceptance of many customers and quality of the offered service: the customer k is rejected if no vehicle is able to serve him with a los_k not greater than GLOS; hence a small value of GLOS supplies a high quality service for few customers (with many rejected ones), whereas a large value of GLOS supplies a poor quality service for several customers.

Problem constraints are the following:

- pick-up node i_k must precede delivery node j_k for every customer k;
- customer k is accepted if there is a vehicle l that permits $\mathrm{los}_k \leqslant \mathrm{GLOS}$;
- for vehicle l, times τ_{i_k} and τ_{j_k} (transits to nodes i_k and j_k) are such that

$$\tau 1_k \leqslant \tau_{i_k} \leqslant \tau_{j_k} - \tau_{i_kj_k} \leqslant \tau 2_k - \tau_{i_kj_k}$$

- the capacity constraint of vehicle l is always satisfied;
- total service time during a day is considered.

The DaR problem is a particular, highly constrained case of multi-vehicle routing, with time windows and priorities. The problem is NP-hard (Desrochers *et al.*, 1990; Laporte, 1992): its difficulty implies the mandatory choice of heuristic methods for solving it in real cases.

Our attention is now devoted to the use of SA techniques for the solution of the dynamic case: our objective function is then to minimize the average los_k

$$\overline{\mathrm{LOS}} = \frac{1}{h}\sum_{k=1}^{h} \mathrm{los}_k,$$

considering that the number of calls h is not '*a priori*' known, so that the mean must be computed again at every new call in the following way.

At the arrival of a new call k the solution method must define:

(a) the best assignment of the call to a vehicle (*assignment* phase)
(b) the new routing of the selected vehicle (*routing* phase).

The second phase (routing) is the core of the solution method. The new call k (with its nodes i_k and j_k) is randomly inserted in the schedule of vehicle l, obtaining an initial solution: then a search is made in the permutation space of the nodes corresponding to pick-ups and deliveries of all the customers (the already accepted customers and the new one). A metric in this search space is defined so that the distance between two solutions is the minimum number of node swaps.

The search for a final solution in this space is made according to the SA rules explained in section 4.1. When a permutation j is not feasible the objective function $\overline{\text{LOS}} = f(j)$ *is forced to a big value* (*for instance*, 1000). *The generation is uniform on the solutions that are adjacent* (*distance* = 1) to the current one. The accepting rule has a standard cooling strategy, with temperature decreasing as $t_{n+1} = \alpha t_n$ (with $\alpha = 0.95$) and maximum number of iteration $H = 10$ (this value has been chosen taking into account an average number of pick-up delivery nodes for small DaR cases). An improvement with respect to the random initialization step has been obtained considering as initial solution, for the new call k, the best solution found for the previously accepted $k - 1$ customers.

The DaR problem with a SA method for the routing phase was studied in two different real situations: a rural area with weak demand in Val Nure (a valley of Northern Italy) in which a DaR service has successfully substituted for the public transport system since 1989, and a district of the city of Milan for which the DaR service is presently simulated as a reinforcement of the regular service.

In the case of Val Nure (Colorni *et al.*, 1990) the network has approx. $r = 90$ nodes, the number of vehicles was $m = 5$ with $q_l = 17$ seats, the average number of calls per day was approx. $n = 50$ (afterwards $n = 100$) gathered in the first hours of the morning; the result was a service with $\overline{\text{LOS}} < 2$ (considered a very good performance by the population); calls were processed in real time by software running on a PC.

5. SAMPLING AND CLUSTERING

5.1. *Features of the method*

The well-known heuristic called *multistart* (MS) provides perhaps the simplest example of a local search approach randomized to solve hard COPs. This procedures is characterized by a *global phase* which randomly generates a fixed number K of new starting points in the solution space S, then by a *local phase* consisting of a simple deterministic local search applied to each point. The local phase gives rise to a set of new locally optimal solutions, to be added to the current set of local optima. The whole process is repeatedly applied until a *test phase* stops the algorithm whenever the set of local optima is not changed during the last set of local searches (or after a fixed number of iterations).

In the *Sampling and Clustering* (SC) method (Boender *et al.*, 1982; Rinnooy Kan and Timmer, 1987), the *global phase* is rather sophisticated, whereas the other two phases remain simple.

There exist a number of settings in which a pure local search algorithm repeated starting from several starting points, randomly generated uniformly in S, too often yields the same local optimum. The basic idea behind the SC method is to try to identify *clusters* of initial points which would lead to the same local optimum, thus reducing the computational effort: a successful clustering should eliminate the repetition of the local phase for many points without worsening the result.

A SC method for a COP (Camerini *et al.*, 1986) uses the following disjoint sets:

- a set Y of points from which a search for a local optimum can start;
- a set Y^* of local optima;
- a set $\bar{Y}$ of points such that a search starting from them leads to a previously found point of Y^*.

The elements of Y are called *candidate points*, and those of $Y^* \cup \bar{Y}$ are called *seed points*. The sets $Y, Y^*, \bar{Y}$ are initialized to Φ. In each iteration of the main loop of SC the *global phase* consists of the following steps:

1. randomly generate K new points from S and add them to Y;
2. eliminate from Y a fraction γ of points which have the worst values of the objective function;

3. cluster (according to a suitable *clustering rule*) as many points of Y as possible around the seed points.

If some points of Y remain unclustered, the *local phase* begins selecting the best candidate among these points and executing a local search until a local optimum is found. If this optimum is a new one it is added to Y^*, otherwise the starting point is added to $\bar{Y}$. This step is then repeated until all candidate points have been clustered. The *test phase* stops the algorithm whenever, having succeeded in clustering all points of Y, no new local optimum has been found, i.e. the set Y^* has not changed.

Several clustering rules are possible, and all of them are based on the idea of *distance* between the points in S. For many COPs it is possible to define a one-to-one correspondence between solutions in S and m-dimensional boolean vectors: if this is the case, it is quite natural to measure the distance between two solutions ($s_1,s_2 \in S$) by the *Hamming distance* $\delta(y_1,y_2)$ between the corresponding boolean vectors y_1,y_2 in $\{0,1\}^m$. For examples of clustering rules see (Boender *et al.*, 1982) and (Camerini *et al.*, 1986).

SC algorithms strongly depend on the assumption that the solutions can be randomly generated uniformly in S. For many COPs it is easy to randomly generate solutions uniformly, but not randomly generate *feasible* solutions uniformly. In these cases we have to decide if the method has to waste time in checking feasibility and penalizing the unfeasible solutions or if it is preferable to relax the uniformity assumption.

Quite naturally, the SC method fits problems better for which the cost of computing a local optimum from a given starting solution is (on average) higher than the cost of clustering a given solution.

5.2. *Application to communication spanning tree problems*

The communication spanning tree (CST) problem (Camerini *et al.*, 1979) is an NP-hard network design problem which can be stated as follows. Given a graph $G = (N,A)$, with two cost functions $C: A \rightarrow \mathbb{R}$ and $F: A \rightarrow \mathbb{R}$, and a weight function $D: N \rightarrow \mathbb{R}^+$, find a spanning tree $G^* = (N,A^*)$ minimizing

$$\sum_{a \in A^*} [F(a) + C(a) \sum_{(x,y) \in S_a} D(x)D(y)] \tag{3}$$

where S_a is the set of all pairs $(x,y) \in N \times N$ such that arc a belongs to the path joining x and y in G^*.

To apply a SC algorithm to a CST problem we need only to define a local search procedure for it, i.e. a *neighborhood structure*. Given a solution $i \in S$ we define $\mathcal{N}(i)$ in the following way

$$\mathcal{N}(i) := \{j \mid \delta(i,j) \leqslant h\} \tag{4}$$

where $\delta(x, y)$ denote the Hamming distance between two boolean vectors x and y, each representing a tree, and h is a parameter. With this structure we say that a point p is h – *optimal* when it yields the best value of the objective function among all points in the solution space at a distance $\leqslant h$ from it. Let us observe that since h is fixed, the number of points at a distance $\leqslant h$ from any given point grows polynomially with the problem size, and therefore h – *optimality* can be checked in polynomial time.

To compare the *multistart* approach and SC, the two methods have been applied to a set of 18 instances of CST problem (Guastalla and Villa, 1985). A first kind of instances ha been obtained by randomly selecting n nodes in a 10×10 grid-type structure: n could assume the values 10, 20, 30 and 40. In a second type of instances the nodes of the graph were the 20 administrative centres of Italian regions. Results show that SC guarantees solutions of the same quality as *multistart* within smaller amount of time (the saving in computing time is around 30%): this is obviously not a very promising experimental behaviour. Nevertheless we believe that some of the key features of SC deserve to be reconsidered for future developments of metaheuristics for COPs.

6. TABU SEARCH

6.1. Features of the method

Tabu Search (TS) (Glover, 1989, 1990) is a metaheuristic which is concerned with imposing restrictions to guide a search process. These restrictions operate in several forms, both by direct exclusion of search alternatives, classed as 'tabu', and by modifying evaluations and probabilities of selection of such alternatives. Here we will characterize TS as a *local search* optimization method, but we have to stress that constructive procedures may be guided by this approach.

Let S denote the set of feasible solutions of an instance of a COP, and let $\mathcal{N}: S \rightarrow 2^S$ and $f: S \rightarrow \mathbb{R}$ denote a *neighborhood* and a *cost* function on S, respectively. In the context of TS the function $\mathcal{N}$ is given by defining a set of modifications (*moves*) of a feasible solution which lead to other feasible solutions, i.e.

$$\mathcal{N}(i) := \{j \mid j \in S, \text{ there exists a move from } i \text{ to j}\}. \qquad (5)$$

The fundamental element underlying TS is the use of *flexible memory*. Taking into account the history of the search process by taking note of *recency*, *frequency*, and *quality* of the moves applied up to the current iteration, the memory structures operate by *modifying* the neighborhood $\mathcal{N}(i)$ of the current solution i and the cost $f(i)$ associated to each element $j \in \mathcal{N}(i)$: the optimization approach operates by selecting at each iteration the *best* evaluated solution in the *modified* neighborhood.

A chief mechanism for exploiting memory in TS is to classify a subset of the moves defining the neighborhood $\mathcal{N}(i)$ of a current solution i as forbidden (*tabu*): the method forbids moves with certain *attributes*, with the goals of preventing cycling and guiding the search towards promising or unexplored regions of S.

A principal notion of recency-based TS memory is the tabu tenure of an attribute, which identifies the number of iterations the attribute remains tabu-active (where a move is classified tabu if a specified number of subset of its attributes are tabu-active). The tabu tenure can vary according to the role the attribute plays in the move. For example, attributes that strongly restrict the available moves when treated as tabu-active are customarily given a shorter tenure than those that weakly restrict the available moves. Tabu tenure also often is allowed to vary about a 'preferred central value' (which depends on the type of attribute considered) by allowing a small deviation from this value, deterministically or randomly generated when the attribute becomes tabu-active.

Memory structures to record the current tabu tenure of an attribute need only specify the largest iteration that the attribute will be tabu-active, since comparing this value to the current iteration immediately discloses the current tabu-active status. Corresponding arrays can be used to maintain various forms of frequency information associated with the attributes for use in longer term memory. Where many attributes exist, so that a separate tenure memory for each may be expensive to maintain, a memory structure called a tabu list is sometimes used. Such a list contains precisely the attributes that are currently tabu-active, together with their associated 'final tabu-active iteration'. The information about iterations can be dropped if all attributes receive the same tenure, since then the attributes can be added and dropped from the list in a simple revolving fashion.

Tabu restrictions can be violated under certain circumstances. For example when a tabu move would result in a solution better than any visited so far, its tabu classification may be overridden. Similar conditions are called *aspiration criteria.*

The combination of recency based memory with frequency based memory adds a component that operates over a longer horizon. One can memorize for each attribute its relative frequency, i.e. how often an attribute belongs to applied moves: in all occasions where no admissible improving moves exist, the frequency information will modify the evaluation $f(j)$ of the elements $j \in \mathcal{N}(i)$ by adding to $f(j)$ a penalty which depends on how often the move from i to j has been already applied. This is a way to implement a *diversification* strategy, to drive the search towards unexplored regions.

A different idea is to combine frequencies of attributes and quality of associated solutions: when no admissible improving moves are available one can prefer those moves containing attributes with greater frequency counts and higher evaluations. This is a way to implement an *intensification* strategy, to let the search converge towards more promising regions.

A hybrid approach that combines ideas from SA and TS has been proposed by Faigle and Kern (1989), which combines frequency information with the random sampling and temperature-based acceptance criterion of SA. The use of random sampling in SA may be contrasted with the use of probabilities in the TS variant called probabilistic tabu search (PTS). This approach retains the strategy of examining elements from a candidate list (instead of randomly picking elements and accepting or rejecting them without explicit comparison to other elements), and in addition strongly biases probabilities to favor choices that receive best evaluations. Thus, PTS makes no reference to temperature and it takes account of TS memory by translating tabu status into penalties that reduce the evaluations of moves considered. Aspiration criteria are allowed to override the probabilities, yielding a deterministic choice when the 'winning move' is sufficiently attractive. Recently PTS methods have been found effective in solving 0–1 mixed IP problems, multidimensional knapsack problems and telecommunication problems.

Hybrids of SA and TS have also been proposed that expand the SA basis for move evaluations (Kassou, 1992), or allow the temperature to be manipulated strategically rather than monotonically (Osman, 1993). This strategic (non-monotonic) manipulation of temperature is an instance of a basic tabu search approach called strategic oscillation, which more generally imposes control that can drive solutions towards and away from boundaries such as feasibility and infeasibility, or that can prescribe oscillations among multiple choice rules, neighborhood types, and so forth.

A simple hybrid approach which combines ideas from GA and TS is obtained in this way: first apply tabu search in a parallel framework to a set of starting solutions in order to generate a set of good quality solutions; then apply GA to recombine the elements so generated. The whole process is repeatedly applied using the recombined solutions as starting ones (Norman and Moscato, 1991). A second kind of hybrid is given by observing that *alleles* of GA may be compared with attributes in tabu search: a hybrid method can be created introducing frequency based memory in GA to mark the history of alleles over populations (see Glover, 1994 for details).

Hybrid approaches combining TS and NN can be found in De Werra and Hertz (1989) and Beyer and Ogier (1991).

Tabu search has been successfully applied in a lot of different areas: scheduling, transportation, layout and circuit design, telecommunications, graphs, expert systems, and many others (see Glover and Laguna, 1993 for a survey).

To illustrate how some aspects of tabu search can be adapted to a particular problem, we briefly describe a randomized tabu search algorithm for solving the job-shop scheduling problem, as presented in (Dell'Amico and Trubian, 1993); for an application of *frequency based memory* to the same problem see Taillard (1994); for an algorithm that exploits the intensification strategy of recovering elite solutions (together with their associated TS memory), see Norwicki and Smutnicki (1993).

6.2. Application to job-shop scheduling problems

The job-shop scheduling is an NP-hard COP that can be stated as follows. A set M of *machines* and a set J of *jobs* are given. Each job consists of an ordered sequence (chain) of *operations* from set $O = \{1,\ldots,N\}$. Each operation $i \in O$ belongs to a job J_i and has to be processed on machine M_i for d_i consecutive time instants. The problem is to assign the operations to time intervals in such a way that no jobs are pre-empted, no two jobs are processed at the same time on the same machine, and the maximum of the completion times $C_i (i = 1,\ldots,N)$ of all operations (*makespan*) is minimized.

For a survey of local search algorithms applied to the job-shop scheduling see (Vaessens *et al.*, 1994).

Neighborhood structures. A move consists of moving a candidate operation i from the current position in its machine to a different position. For example, given the sequences of operations (a,b,c,d,i,e,f) the candidate operation i will be moved before b generating the sequence (a,i,b,c,d,e,f). Observing that this kind of move can be decomposed in the sequence of swaps: (d,i), (c,i), (b,i), we can memorize in the tabu list the reversal of performed swaps: (i,d), (i,c) and (i,b). A candidate move is tabu if a component swap of it belongs to the tabu list.

Tabu list. As data structure we use a square matrix T for each machine, with dimensions equal to the number of operations processed by that machine. The element T_{ij} contains the count of the iteration in which the pair of operations (i,j) had been reverted last time. We forbid a swap of operations (i, j) if the value of T_{ji} plus the length of the tabu list is greater than the count of current iteration.

The length of the tabu list varies according to rules which depend on the fact whether the current solution is better than the previous one or not.

Starting solution. To generate a feasible starting solution we applied a list scheduling heuristic combined with a greedy random search method (GR), as given in (Hart and Shogan, 1987). A GR is an alternative to a greedy heuristic in the sense that, instead of selecting at each stage the decision with highest associated payoff, the algorithm randomly selects a decision among a subset Q of the set of currently feasible decisions. The core of the method is the way in which one can build up the set Q. Let us suppose that it is possible to assign a value to each decision and that the priority of each decision depends on the assigned value. The set Q is then given in two different ways: it contains the decisions with an assigned value within p % of the highest assigned value, or it contains the c decisions with highest assigned value. Note that setting $p = 0$ or $c = 1$ transforms this random heuristic in to a greedy one.

We decided to choose the randomized approach in the job-shop scheduling problem and we have experimentally observed that it gives, on average, better solutions than the corresponding deterministic version: at least one of the solutions generated applying less than 10 times the randomized algorithm is better than the corresponding deterministic solution. However this kind of randomized procedure does not guarantee producing a local optimum with respect to even simple neighborhood structures.

Computational results. The algorithm was tested on a set of 53 standard benchmark problem instances (Dell'Amico and Trubian, 1993). This TS algorithm finds an optimal solution for 33 of 46 problems for which the optimal solution is know. For the 13 remaining ones the approximation error with respect to the optimum is smaller than or equal to 1%. The computing times are generally small and increase almost linearly with the number of operations.

7. NEURAL NETS

7.1. Features of the method

A *Neural Net* (NN) is a set of elements or *neurons*, connected by *links*, weighted by a real number w_{ij} and oriented (in general $w_{ij} \neq w_{ji}$). In the *threshold nets* every neuron i computes the following two quantities: the *activation level* $u_i = \sum_j w_{ij}x_j - \theta_i$ (which is the weighted sum of the inputs coming from the other neurons decreased by a *threshold* value θ_i) and the *output* x_i, which depends on the activation level u_i through a bounded, non-decreasing, non-linear function. The net evolves by successive iterations according to a *transition rule.* The computation of the state is done by each neuron locally and independently. When the state of the net does not change, the net is in a *stable state.* Though being a fundamental feature of neural models, *parallelism* is not the most important one; as a matter of fact the diffusion of neural models happens in spite of the need to simulate them by sequential algorithms. One main feature of NN consists of the following property: from the 'simple' behaviour of local decision-makers (the neurons) and from the structure of the connections between them (the topology of the net) a 'complex' and in some sense 'intelligent' behaviour emerges.

In COPs neural nets allow approximately solving problems, starting from their description in terms of variables space, constraints and objective-function, without requiring any explicitely coded procedure; on the other hand, the solutions are seldom comparable with those found by *ad hoc* algorithms, both in terms of approximation and in terms of computation time. This is also true for other general purpose heuristics such as SA and GA. The application of neural nets to COPs was opened by Hopfield and Tank (1985). A review of neural models for COPs can be found in Righini (1992). It is possible to distinguish two different families of neural nets for optimization, the *spin glass models* and the *deformable models.*

Spin glass models. This class of models contains nets derived from the Hopfield model (1982, 1984) and the Boltzmann machine (Aarts and Korst, 1989); Hopfield nets and Boltzmann machines are nothing but the deterministic and the probabilistic version of the same neural algorithm, as was pointed out in successive developments. In such NN the data of the problem instance are used to *build* the network, which then evolves up to a stable state. Spin glass models are inspired by the analogy with physical systems known as spin glasses (Barahona, 1982): techniques from statistical mechanics are therefore appropriate to study their properties and behaviour. The convergence of a net is guaranteed if there exists a *Lyapunov function* of the state, decreasing at each iteration and bounded from below. In the physical analogy this function corresponds to the *energy* of the spin glass. Since a net spontaneously tends to minimize its Lyapunov function, the idea is to map the objective function of a COP on it. The evolution of the net represents then a search in the solutions space, which converges to a *local optimum.* About the convergence of NN, the reader is also referred to Goles and Olivos (1981). The dynamics of the net can be serial, parallel or mixed (Bertoni and Campadelli, 1990). In the serial mode only one unit can change its state at each iteration, in the parallel mode all units can change. The state of every unit is a deterministic or probabilistic function of the state of all units linked with it and of the connections.

Deformable models. The second family of neural nets for COPs contains models derived from Kohonen's self-organizing maps (Kohonen, 1984); in such models the data of the problem instance are used as *inputs* to the net. In its evolution the net modifies the values of the links weights. Such models are inspired by the capability of some cerebral areas to learn the topological relations of external stimuli and to replicate them in their interneuronal connections. Nets of this class are made of two layers fully connected with each other and with no feedbacks. The state of the network is determined by the weights of links instead of the state of neurons as in spin glass models. Deformable models are suitable for COPs with topological constraints, that are hard to represent in spin glass models: the most interesting results of application of deformable models to COPs were obtained for the TSP: we cite the *elastic net* (Durbin and Willshaw, 1987), the basic model of Kohonen and those directly derived from it (Angeniol *et al.*, 1988; Fort, 1988).

Yuille (1990) and Simic (1990) showed that it is possible to obtain both elastic net and Hopfield-Tank net as particular cases of a more general model, named *Generalized Deformable Template.*

The computational complexity of neural computation has been studied by Wiedermann (1990). Bruck and Goodman (1990) proved that the computational power of a NN is not superior to that of a Turing Machine: every task which can be performed by a NN can also be performed by a classical sequential algorithm. It is also worth noticing that neural nets do not guarantee any approximation bound (for instance ε-approximation) nor any bound on the number of iterations needed to converge to a stable state (Yao, 1992).

7.2. *Application to location/allocation problems*

A class of problems that suggest a very straightforward implementation onto a NN is that of assignment problems, including among others linear assignment, quadratic assignment and location/allocation problems. The application described here concerns two location/allocation problems: the euclidean -median and euclidean P-barycenter problems.

N points, representing users, are given in R^2 and P others ($P < N$), representing facilities, must be located optimally. We indicate by a_i the position of user $i (i = 1,\ldots,N)$ and by x_j the position of facility $j (j = 1,\ldots,P)$. The objective functions to be minimized are functions of the distances between every user and its nearest facility:

$$\min f = \sum_{j=1}^{P} \sum_{i=1}^{N} w_{ij} d_{ij} \text{ (medians)} \tag{6a}$$

$$\min g = \sum_{j=1}^{P} \sum_{i=1}^{N} w_{ij} d_{ij}^2 \text{ (barycenters)} \tag{6b}$$

$$\sum_{j=1}^{P} w_{ij} = 1 \qquad \forall i = 1,\ldots,N \tag{7}$$

$$w_{ij} \in \{0,1\}.$$

Facility j is allocated to user i if and only if $w_{ij} = 1$. The relaxation of the integrality constraints over the w_{ij} allows the straightforward application of the Mean Field Annealing neural model (Peterson

Table 2. Comparison between cluster-and-locate algorithm and mean field annealing (NN) algorithm on 10 instances of P-barycenter problem with $P = 5$ and $N = 20$

Problem	Cluster-and-locate			Mean field annealing		
instance	g	t (s)	n iter.	g	t (s)	n iter.
1	0.654	1.54	3	0.577	16.58	18
2	0.591	1.60	3	0.347	23.29	25
3	0.528	2.03	4	0.489	18.57	20
4	0.529	1.59	3	0.547	21.53	23
5	0.493	1.54	3	0.301	23.45	25
6	0.336	2.09	4	0.285	22.41	24
7	0.566	1.09	2	0.273	19.60	21
8	0.452	2.09	4	0.461	17.63	19
9	0.662	1.60	3	0.577	21.59	23
10	0.684	1.09	2	0.458	16.58	18
Avg. values	0.5495	1.63	3.1	0.4315	20.12	21.6

and Soederberg, 1989): one neuron is defined for each variable w_{ij}; binary values for all w_{ij} variables are achieved only at the end the of computation. Every w_{ij} variable is initialized around the value $1/P$, with a small random noise, needed to break the symmetry of the initial unstable equilibrium state. The updating rule, that every neuron uses at each iteration, is the following:

$$w_{ij} = \frac{\exp\left(\frac{-1}{t}\frac{\alpha E}{\alpha W_{ij}}\right)}{\sum_{l=1}^{P}\exp\left(\frac{-1}{t}\frac{\alpha E}{\alpha W_{il}}\right)} \tag{8}$$

where E coincides with f or g in (6). Such a rule ensures that the values of the w_{ij} are normalized, so that constraint (7) is satisfied after each iteration. The parameter t slowly decreases, and in the limit $t \to 0$ all neurons assume binary values.

The optimal locations of a median and of a barycenter of N weighted points are given by the following formulae, respectively:

$$x_j = \frac{\sum_i \frac{w_{ij}}{d_{ij}} a_i}{\sum_i \frac{w_{ij}}{d_{ij}}} \text{(medians)} \tag{9a}$$

$$x_j = \frac{\sum_i w_{ij} a_i}{\sum_i w_{ij}} \text{(barycenters)} \tag{9b}$$

From them it is possible to obtain the expressions of the partial derivatives, which are not reported here. All the $P \times N$ partial derivatives can be computed in parallel; moreover every neuron provides a feedback onto all the others.

Experimental results and comparisons. Classical approaches to multi-facility location problems in the plane are the iterative cluster-and-locate algorithm and heuristics based on successive point insertions. A classical cluster-and-locate algorithm has been compared with a Mean Field algorithm on 10 instances of a P-barycenter problem with $N = 20$ and $P = 5$. As shown in Table 2, the Mean Field Annealing algorithm found solutions of much better quality. The annealing algorithm can be very time-consuming, of course, but a suitable trade-off between solution quality and computing time can be achieved by tuning annealing parameters (such as the initial temperature and the annealing rate).

```
1. $\eta_{ij} = 1/d_{ij}$; $\tau_{ij} := \bar{\tau}$;
2. for each ant $k$ ($k := 1$ to $M$) do
     $TL_k :=$ (starting town of ant $k$)
     while $|TL_k| < N$ do
       select the town $j$ to go to;
       add $j$ to the ordered list $TL_k$;
     $L_k :=$ length of the tour described by $TL_k$;
     $\Delta\tau_{ij}^k := \begin{cases} \frac{Q}{L_k} & \text{if arc}(i,j) \text{ belongs to the tour described by } TL_k \\ 0 & \text{otherwise} \end{cases}$
     $\Delta\tau_{ij} := \Delta\tau_{ij} + \Delta\tau_{ij}^k$;
3. $\tau_{ij} := \rho\tau_{ij} + \Delta\tau_{ij}$;
4. if not (terminating condition) then go to 2;
```

Fig. 3. Ant System.

8. ANT SYSTEM

8.1. Features of the method

In this section we illustrate a natural heuristic first introduced in Colorni *et al.* (1991, 1992): the *Ant System* (AS). The basic idea underlying this heuristic is that of simulating the behaviour of a set of agents that cooperate to solve an optimization problem by means of very simple communications. The natural metaphor we employ is the *ant colony* metaphor. To illustrate the approach we use the *travelling salesman problem* (TSP), but the formulation can then be used for the solution of any COP requiring the search of a node permutation in a graph.

Let us start with the following observation: real ants seem to be able to find their way (from the nest to a food source and back, or around an obstacle) with relative ease, although they are almost blind (Denebourg *et al.*, 1983). Ethological studies discovered that this capacity is the result of the interplay via chemical communication between ants (through a substance called *pheromone*) and an emergent phenomenon caused by the simultaneous presence of many ants. In our work we tried to reproduce this mechanism in an artificial ant colony, which we call *Ant System*. More about the natural analogy can be found in Dorigo *et al.* (1996). We now introduce the algorithm and try to give a rationale for its functioning. Consider a TSP problem defined on a graph $G = (N,A)$ of $|N|$ nodes and $|A|$ arcs, where arc (i,j) connects nodes n_i and n_j and has associated real value $d_{ij} = d(n_i,n_j)$. We must find a permutation π of the nodes that minimizes the quantity $\sum_{i=1}^{|N|} d(n_{\pi(i)},n_{\pi(i+1)})$, where $\pi(|N|+1) \equiv \pi(1)$.

We define an *ant* to be an agent with the following properties:

- the ant remembers already visited towns using an ordered list called tabu list (TL) for this purpose (caution, TL here is different from that of Tabu Search),
- at every step the ant chooses, using a probabilistic rule, a town to move to among those not in the TL,
- after a tour has been completed the ant lays a trail τ_{ij} on each arc (i,j) used (trail is the analog of pheromone) and clears its tabu list.

Figure 3 shows the Pascal-like version of AS.

The probabilistic rule used as *select* function in step 2 is a Monte Carlo procedures where the probability of choosing an unvisited town j as a destination is given by:

$$\text{Prob}(j) = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \notin TL} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}} \qquad (10)$$

where τ_{ij} is the amount of *trail* on arc (i,j) and η_{ij} is called *visibility* of node j from node i, defined as

$\eta_{ij} = 1/d_{ij}$ in the case of TSP. In Fig. 3 the parameter $\rho(0 \leqslant \rho < 1)$ is such that $(1 - \rho)$ can be interpreted as the evaporation of trail; the parameter $\bar{\tau}$ is such that initially the ants are free to move almost randomly; parameters α and β allow the user to control the relative impact of the two criteria (visibility and trail intensity) used in (10), Q is an arbitrary positive constant.

If we consider M ants moving on the graph G concurrently, then τ_{ij} is the way by which ants communicate. In fact, as stated by the *select* function, ants choose with higher probability edges with higher amounts of trail. Therefore, an ant laying trail on a set of edges makes that set more 'desirable' for other ants.

A way to interpret how the algorithm works is to consider the superposition of effects: each ant, if it did not consider trail effects (i.e., if $\alpha = 0$), would move with a local, probabilistic greedy rule. This greedy rule will probably lead to bad final results. The reason the greedy rule does not work is that an ant is constrained to make a closed tour and therefore final steps are constrained by early steps. So the tour followed by a single ant, according to a greedy policy, is composed by some (initial) parts that are very good and some (final) others that are not. If we now consider the effect of the simultaneous presence of many ants, then each one contributes to the trail distribution. Good sets of arcs will be followed by many ants and therefore will receive a great amount of trail; bad sets of arcs (chosen only in order to satisfy constraints) will be chosen only by few ants and therefore receive a small amount of trail.

Experimental testing of the effectiveness of the AS could not be carried out on big TSP instances. Current research on this problem deals with instances with more than 100,000 towns and we cannot use so much memory on our machines; moreover, the computational complexity of one cycle of the AS is $O(N^3)$, therefore the time needed to face very big instances is prohibitive.

We point out however that we never made an assumption of symmetry of the graph: we devoted therefore our attention to the asymmetric problem (ATSP), for which specialized heuristics deal with instances with a few hundred towns. ATSP provides good instances to test different versions of AS. We used a standard problem, called RY48P (Fischetti and Toth, 1992), having an optimal value of 14,422 obtained by a branch-and-bound procedure but with a high computational time: the AS obtains a value of 14537 (0.8% worst) with 2000 cycles.

Several extensions of the basic algorithm have been tested. The first one consists of a *constructive procedure* that starts with a small number of towns and adds the others one by one, allowing the ants to identify a good trail distribution for each subproblem, to be used as a basis for the new subproblem with one added town. The rationale was that search in the small initial space and successive adaptations of partial solutions could be more effective that search in the huge complete problem space.

The second extension consists of a modification of the trail levels when a *stagnation* is detected. A stagnant behaviour is detected when the trail distribution is such that all the ants follow the same tour. Since stagnation prevents further search, partial trail reinitialization* makes the system restart from a point still in the region of the search space that was previously identified as a promising one.

The third extension assigns specific α and β values to each ant and let it evolve by a *genetic algorithm*, with a fitness function inversely related to the length of the tour proposed by the ant. The parameters are coded as bit strings, the system runs for a fixed number of iterations during which the best tour and the average tour length of each ant are computed. This data is the basis for the fitness function assessment. Standard GA operators can then be used.

Table 3 presents a comparison of the results obtained with the different versions of the AS (values are averages over five runs of each algorithm).

8.2. Application to Quadratic Assignment problems

The computational approach described in Section 8.1 has been tested on other COPs including the Quadratic Assignment Problem and the Job-Shop Scheduling Problem.

A Quadratic Assignment Problem (QAP) of order n is the formalization of the problem that arises when trying to assign n facilities to n locations, where both the terms (facilities and locations) are considered in the broadest sense of their meaning. Formally the problem can be defined by three

*Trail modifications are obtained with a change, for a small number of iterations, of the parameters α and β, so that ants follow paths different from those previously used.

Table 3. Ant algorithms on RY48P instance

	Tour length	Error	Iterations
Basic AS	14,899	3.3%	1788
Constructive	14,878	3.1%	5995
Trial modification	14,537	0.8%	1969
Generic α and β	14,997	4.0%	1954

$n \times n$ matrices:

$D = \{d_{ij}\}$ is the distance between location i and location j;
$F = \{f_{hk}\}$ is the flow (of information, products or whatever) between facilities h and k;
$C = \{c_{ih}\}$ is the cost of assigning facility h to location i.

A permutation π can be interpreted as an assignment of facility $h = \pi(i)$ to location i, $(i = 1,\ldots,n)$. The problem is then to identify a permutation π of both row and column indexes of the matrix F that minimizes the total cost (Edwards, 1977):

$$\min z = \sum_{i,j=1}^{n} d_{ij} f_{\pi(i)\pi(j)} + \sum_{i=1}^{n} c_{i\pi(i)} \tag{11}$$

Usually D and F are integer-valued symmetric matrices and the matrix C is not considered. In our application of the AS to QAP, the visibility η_{ih} represents an estimate of the goodness of assigning facility h to location i and is expressed by a combination of the *potentials* of the distance matrix and flow matrix: these potentials d and f are defined as row sums of each matrix. For example:

$$D = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 4 & 5 \\ 2 & 4 & 0 & 6 \\ 3 & 5 & 6 & 0 \end{bmatrix} \rightarrow d = \begin{bmatrix} 6 \\ 10 \\ 12 \\ 14 \end{bmatrix}; \; F = \begin{bmatrix} 0 & 60 & 50 & 10 \\ 60 & 0 & 30 & 20 \\ 50 & 30 & 0 & 40 \\ 10 & 20 & 40 & 0 \end{bmatrix} \rightarrow f = \begin{bmatrix} 120 \\ 110 \\ 120 \\ 0 \end{bmatrix}.$$

From the two vectors d and f, a third matrix $S = d \cdot f^{T}$ is obtained, where each element $s_{ih} = d_i \cdot f_h$:

$$S = \begin{bmatrix} 720 & 660 & 720 & 420 \\ 1200 & 1100 & 1200 & 700 \\ 1440 & 840 & 1680 & 15400 \\ 1680 & 980 & & \end{bmatrix}.$$

The ants choose the node to move to on the basis of (10) in which $\eta_{ih} = 1/s_{ih}$, interpreting each element η_{ih} as the attractiveness of the choice of assigning facility h to location (node) i. For example, 1/420 on the first row is linked to the choice of assigning function 4 to node 1; it is considered more attractive than the choice of assigning function 3 to node 1 ($\eta_{13} = 1/720$).

In the first step each ant chooses via a Monte Carlo procedure on the elements of the first row of S the node to start from; the generic kth step, consists in choosing with a similar procedure which element of the kth row to move to, excluding those of the columns already visited. The usual trail updating procedure is applied on another matrix, interpreting its elements as trail levels on corresponding graph edges.

This algorithm has been included in a system, called ALGODESK, designed to compare the efficiency of different heuristics (Maniezzo *et al.*, 1995): it has been found that the AS, coupled with a greedy procedure that optimizes locally the permutation proposed by each ant, is one of the most effective among the heuristics tested on a large set of QAP instances of dimension up to 30 of the QAPLIB library (Burkard *et al.*, 1991).

9. CONCLUSIONS

In this paper we have shown the main characters of a set of non-deterministic algorithms derived by natural systems, applying them to the solution of some well known NP-hard COPs. In other papers,

Table 4. A summary of research perspectives in Heuristics from Nature

	SA	TS	NN	GA	SC	AS
[1] results	XXX	///	///	///	///	...
[2] theory	XXX	///	///	///	...	...
[3] packages	///	///	///	...	...	
[4] complexity	///	///	...	...		

we also tested natural heuristics on different specific problems [for example QAP (Maniezzo *et al.*, 1995)], with the aim of designing a real comparison environment. Here we try to describe the connections and the similarities of different methods: on this subject, the reader can also see (Aarts and Korst, 1989; Reeves, 1993).

The basic idea of Heuristics from Nature is that it is possible to simulate many aspects of natural systems, in particular the two natural engines of *selection* (corresponding to optimization) and *mutation* (corresponding to random search).

In Table 4 we propose our understanding of the current situation of research on Heuristics from Nature. Columns represent the methods and rows describe four progressive steps (which are very common in developing research sectors). They are:

[1] the presence of practical results;
[2] the definition of a theoretical framework;
[3] the availability of commercial packages;
[4] the study of computational complexity and related properties.

In the table, the symbol XXX means a well defined situation (with low level of likely future developments), the symbol /// means a dynamic situation (with a large possibilities of novel research), finally the symbol ... means a birth situation (with a very big amount of work still to be done).

The table we present here is largely subjective. The aim of the table is only a synthetic representation of the state of the art from the point of view of our research group, given the current lack of any wider agreement on this point. We present it as a basis for discussion on this point. Future works will be the design of 'hybrid' algorithms that cross strong ideas of the methods presented here, the study of new applications to NP-hard problems, the construction of packages in which the parallelism of the methods could be implemented.

Many questions remain and many problems are open: as a final remark we propose a brief catalogue:

1. The *learning procedure*: under which conditions is self-organization of the system possible?
2. The *evolutionary mechanism*: what is the role of cooperation and competition mechanisms?
3. The *search strategy*: what is the trade-off between exploitation and exploration phases?
4. The *role of experience*: how do you operate in parameter tuning, for balancing convergence speed and good results?
5. The *running conditions*: how does the topology of the problem influence the process and the final solution?
6. The *data collection*: you have a very big amount of information, what data do you collect and which results do you present for a good observation of the system?

Acknowledgements – The authors are indebted to the anonymous referees for some useful suggestions.

REFERENCES

Aarts, E. & Korst, J. (1989). *Simulated Annealing and Boltzmann Machines.* Wiley.

Angeniol, B., DeLaCroix-Vaubois, G. and LeTexier, J. (1988). Self-organizing feature maps and the TSP. *Neural Networks.* Vol. 1, pp. 289–293.

Barahona, F. (1982). On the computational complexity of using spin glass models. *J. Phis. A: Math. Gen.*, Vol. 15, pp. 3241–3253.

Bertoni, A. & Campadelli, P. (1990) Analysis of parallel and sequential Boltzmann machines. *International Neural Networks Conference.*

Beyer, D. & Ogier, R. (1991). Tabu learning: a neural network search method for solving nonconvex optimization problems. *Proceedings of the International Joint Conference on Neural networks,* IEEE and INNS, Singapore.

Boender, C.G.E., Rinnooy Kan, A.H.G. & Timmer, G.T. (1982). A stochastic method for global optimization. *Math. Programming,* Vol. 12, pp. 125–140.

Bourgine, P. & Varela, F.J. (Eds.) (1992). *Towards a Practice of Autonomous Systems.* Cambridge: The MIT Press.

Bruck, J. & Goodman, J. (1990). On the power of neural networks for solving hard problems. *Journal of Complexity,* Vol. 6, pp. 129–135.

Burkard, R.E., Karish, S. & Rendl, F. (1991). QAPLIB – a quadratic assignment problem library. *European Journal of Operational Research,* Vol. 55, pp. 115–119.

Camerini, P.M., Colorni, A. & Maffioli, F. (1986). Some experience in applying a stochastic method to location problems. *Math. Programming Study,* Vol. 26, pp. 229–232.

Camerini, P.M., Fratta, L. & Maffioli, F. (1979). Some results on the design of tree-structured communication networks. *ITC-9,* Torremolinos.

Chahal, N. & De Werra, D. (1989). An interactive system for constructing Timetables. *European Journal of Operational Research,* Vol. 40, pp. 32–37.

Colorni, A., Dorigo, M. & Maniezzo, V. (1991). Distributed optimization by ant colonies. *Proceedings of ECAL-91 – European Conference on Artificial Life,* Paris, France, F. Varela & P. Bourgine (Eds) (pp. 134–142). Elsevier.

Colorni, A., Doriogo, M. & Maniezzo, V. (1992). An investigation of some properties of an ant algorithm. *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN* 92), Brussels, Belgium, R. Männer, B. Manderick (Eds.), (pp. 509–520). Elsevier.

Colorni, A., Dorigo, M. & Maniezzo, V. (1992). Genetic algorithms: a new approach to the time-table problem. *NATO ASI Series,* Vol. F82, Combinatorial Optimization, M. Akgil *et al.* (Eds.) Springer.

Colorni, A., Laniado, E. & Vittadini, M.R. (1990). Il progetto prontobus. *Terra,* Vol. 9, pp. 35–39. (in Italian).

Dell'Amico, M. & Trubian, M. (1993). Applying Tabu-search to the job-shop scheduling problem. *Annals of Operations Research,* Vol. 41, pp. 231–252.

Denebourg, J.L., Pasteels, J.M. & Vehaeghe, J.C. (1983). Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical Biology,* Vol. 105.

Desrochers, M., Lenstra, J.K. & Savelsbergh, M.W. (1990). A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research,* Vol. 46, pp. 322–332.

De Werra, D. & Hertz, A. (1989). Tabu search techniques: a tutorial and an application to neural networks. *OR Spectrum,* Vol. 11, pp. 131–141.

Diaconis, P. & Hanlon, P. (1992). Eigen analysis for some examples of the metropolis algorithm. Res. Rep., Dept. of Math., Harvard Univ.

Dorigo, M., Maniezzo, V. & Colorni, A. (1996). The ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics – Part B.* Vol. 26, pp. 29–41.

Durbin, R. & Willshaw, D. (1987). An analogue approach to the TSP using an elastic net method. *Nature,* Vol. 326, pp. 689–691.

Edwards, C.S. (1977). The derivation of a greedy approximator for the Koopmans–Beckmann quadratic assignment problem. *Math. Programming Study,* Vol. 13.

Faigle, U. & Kern, W. (1989). Some convergence results for probabilistic Tabu search. Internal Report 822, Faculty of Applied Math., University of Twente.

Fischetti, M. & Toth, P. (1992). An additive bounding procedure for the asymmetric travelling salesman problem. *Math. Programming,* Vol. 53, pp. 173–197.

Fort, J.C. (1988) Solving a combinatorial problem via self-organizing process: an application of the Kohonen algorithm to the travelling salesman problem. *Biological Cybernetics,* Vol. 59, pp. 33–40.

Garey, M.R. & Johnson, D.S. (1979). *Computers and Intractability: a Guide to the Theory of NP-Completeness.* W.H. Freeman and Co.

Glover, F. (1989). Tabu search, Part I. *ORSA Journal on Computing,* Vol. 1, No. 3, pp. 190–206.

Glover, F. (1990). Tabu search, Part II. *ORSA Journal on Computing,* Vol. 2, pp. 4–32.

Glover, F. (1994). Tabu search for nonlinear and parametric optimization (with links to genetic algorithms). *Discrete Applied Mathematics,* Vol. 49, pp. 231–256.

Glover, F. & Greenberg, H.J. (1989). New approaches for heuristic search: a bilateral linkage with artificial intelligence. *European Journal of Operational Research,* Vol. 39, pp. 119–130.

Glover, F. & Laguna, M. (1993). Tabu search. In Colin R. Reeves (Ed.) *Modern Heuristic Techniques for Combinatorial Problems.* Oxford, UK: Blackwell Scientific Publications.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley.

Goles, E. & Olivos, J. (1981). The convergence of symmetric threshold automata. *Information and Control,* Vol. 51, pp. 98–104.

Guastalla, L. & Villa, F. (1985). Metodi stocastici per il progetto delle reti di communicazione ad albero. Degree Thesis, Dep. of Electronic Engineering, Politecnico di Milano.

Hart, J.P. & Shogan, A.W. (1987). Semi-greedy heuristics: an empirical study. *Operations Research Letters,* Vol. 6, pp. 107–114.

Hertz, A. & de Werra, D. (1987). Using tabu search techniques for graph coloring. *Computing,* Vol. 29, pp. 345–351.

Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press.

Hopfield, J. (1982). Neural networks and physical systems with emergent Collective computational abilities. *Proc. Natl. Acad. Sci. USA,* Vol. 81, pp. 3088–3092.

Kassou, I. (1992). Amelioration d'Ordonnancements par les methodes de voisinage. Doctoral thesis, INSA, Rouen.

Kirkpatrick, S., Gelatt Jr., C.D. & Vecchi, M.P. (1983). Optimization by simulated annealing. *Science,* Vol. 220, pp. 671–1680.

Kohonen, T. (1984). *Self-organization and Associative Memory.* Springer.

Laporte, G. (1992). The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operational Research,* Vol. 59, pp. 345–358.

Lin. S. & Kernighan, B.W. (1993). An effective heuristics algorithm for the travelling salesman problem. *Operations Research,* Vol. 21, pp. 498–516.

Maniezzo, V., Dorigo, M. & Colorni, A. (1995). ALGODESK: an experimental comparison of eight evolutionary heuristics applied to the QAP problem. *European Journal of Operational Research,* Vol. 8, pp. 188–204.

Männer, R. & Manderick, B. (Eds) *Parallel Problem Solving from Nature*, 2. North-Holland.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas imminent in nervous activity. *Bull. of Math. and Biophysics*, Vol. 5, pp. 115–133.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A. & Teller, E. (1953). Equation of state calculation by fast computing machines. *J. of Chem. Physics*, Vol. 21, pp. 1087–1092.

Minsky, M. (1985). *The Society of Mind*. Touchstone Books.

Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization, In Schaffer (Ed.) *Proc. of the 3rd Int. Conf. on Genetic Algorithms*, Morgan Kaufmann.

Norman, M. G. & Moscato, P. (1991). A competitive-cooperative approach to complex combinatorial search. *Proceedings of the 20th Joint Conference on Informatics and O.R.* (*20th JAIIO*), Buenos Aires, Argentina, pp. 315–329.

Nowicki, E. & Smutnicki, C. (1995). A fast taboo search algorithm for the job shop problem. *Management Science*, to appear.

Osman, I. H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, Vol. 41.

Peterson, C. & Soederberg, B. (1989). A new method for mapping optimization problems onto neural networks. *Int. J. of Nueral Systems*, Vol. 1, pp. 3–22.

Psaraftis, H. N. (1983). An exact algorithm for the single-vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, Vol. 7, pp. 351–357.

Rechenberg, I. (1973). *Evolutionsstrategie*. Germany: Stuttgart, Fromman-Holtzbog.

Reeves, C. R. (1993). (Ed.) *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell.

Righini, G. (1992). Modelli di reti neurali per ottimizzazione combinatoria. *Ricerca Operativa*, Vol. 62, pp. 29–67 (in Italian).

Rinnooy Kan, A. H. G. & Timmer, G. T. (1987). Stochastic global optimization methods I, II. *Math. Programming*, Vol. 39, pp. 27–78.

Schwefel, H. P. (1981). *Neumetical Optimization of Computer Models*, New York: Wiley.

Schwefel, H. P. & Männer, R. (1990). Parallel problem solving from nature. *Proc. of PPSN-I*.

Simic, P. (1990). Statistical mechanics as the underlying theory of 'elastic' and'neural' optimization. *Network: Comp. Neural Systems*, Vol. 1, pp. 89–103.

Taillard, E. (1989). Parallel taboo search technique for the jobshop scheduling problem. *ORSA Journal on Computing*, Vol. 6, pp. 108–117.

Vaessens, R. J. M., Aarts, E. H. L. & Lenstra, J. K. (1994). Job shop scheduling by local search, Preprint, Dept. of Math. and Comp. Sc., Eindhoven Univ. of Techn.

Van Laarhoven, P. J. M. & Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*. D. Reidel Publ. Co.

Van Laarhoven, P. J. M., Aarts, E. H. L. & Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, Vol. 40, pp. 113–125.

Wiedermann, J. (1990). Complexity issues in neurocomputing, Res. Rep. A 10/90, Universitaet des Saarslanden, Saarbruecken.

Yao, X. (1992). Finding approximate solutions to NP-hard problems by neural networks is hard. *Information Processing Letters*, Vol. 41, pp. 93–98.

Yuille, A. (1990). Generalized deformable templates, statistical physics and matching problems. *Neural Computation*, Vol. 2, pp. 1–24.