European Journal of Operational Research 81 (1995) 188–204
North-Holland

Theory and Methodology

Algodesk: An experimental comparison of eight evolutionary heuristics applied to the Quadratic Assignment Problem

Vittorio Maniezzo, Marco Dorigo

Politecnico di Milano Artificial Intelligence and Robotics Project, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milan, Italy

Alberto Colorni

Centro di Teoria dei Sistemi del CNR, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 220133 Milan, Italy

Received July 1992; revised September 1992

Abstract: This work compares the effectiveness of eight evolutionary heuristic algorithms applied to the Quadratic Assignment Problem (QAP), reputedly one of the most difficult combinatorial optimization problems. QAP is merely used as a carrier for the comparison: we do not attempt to compare any heuristics with solving algorithms specific for it. Results are given, both with respect to the best result achieved by each algorithm in a limited time span and to its speed of convergence to that result.

Keywords: Optimization; Heuristics; QAP

1. Introduction

The theory of NP-completeness (Garey and Johnson, 1979) tells us that, unless P = NP, many problems cannot be solved optimally in a reasonable amount of time. However, real-world problems have to be faced, hence there is a necessity to develop heuristic algorithms that yield a solution not too far from the optimal one, using limited computing time and storage space.

It is not surprising that, in parallel with the increasing awareness of the intractability of so many problems, a flourishing of different heuristics characterized the last decades. Some of them are very problem-specific, while others get easily stuck in local optima; we are interested in heuristic algorithms for combinatorial optimization that are both *robust* (applicable to a wide variety of problems with minimal, if any, modification of their basic structure) and *effective* as global optimization tools.

We concentrated on evolutionary heuristics, i.e. algorithms that start with an initial randomly chosen (population of) solution(s) and that update it iteratively until a terminating condition is met. This class of

Correspondence to: Dr. A. Colorni, Centro di Teoria dei Sistemi del CNR, Dipartimento di Elettronica e Informazione, Politecnico di Milano, Piazza Leonardo da Vinci 32, 220133 Milan, Italy.

SSDI 0377-2217(93)E0128-K

algorithms has experimentally proved effective with respect to both previously stated criteria. We were particularly interested in those algorithms that work with a population of solutions for two reasons. The first is their appealing possibility of utilizing multiple solutions for determining the future evolution (thus using more information than their single solution counterparts); the second reason is their greater ease of parallelization which promises a much more effective exploitation of the possibilities offered by parallel computers.

We therefore studied several population-based evolutionary heuristics: genetic algorithms, evolution strategies, sampling and clustering, Boltzmann machines and immune networks. We compared their effectiveness both among themselves and with the two most robust and effective single-solution algorithms so far proposed: simulated annealing and tabu search. Moreover, as a further benchmark, we also studied the performance of a straightforward multistart greedy algorithm (multigreedy). The performance of the multigreedy algorithm allows us to assess whether the parallel processing of information is computationally worthwhile.

The comparison has been carried out over a tough NP-hard problem that up to now has resisted all attempts to solve it optimally without an almost complete enumeration of solutions: the Quadratic Assignment Problem. This problem was chosen not only because of its difficulty, but also because (i) it is a generalization of many other combinatorial optimization problems (Finke and Medova Dempster, 1989); (ii) it has a great number of real-world interpretations; (iii) most of the mentioned algorithms have been applied to it and the results have been reported in the literature, thus relieving us from the burden to run extensive field tests in order to assess the optimal parameter settings for each of them.

The paper is organized as follows: in Section 2 we describe the Quadratic Assignment Problem; in Section 3 we present the algorithms: for each of them we propose a very brief introduction and the basic algorithm, referring the reader to specific publications for a deeper understanding. In Section 4 we describe the package which allowed tests and comparisons of the algorithms and the results of their application; in Section 5 we present conclusions and future work.

2. The Quadratic Assignment 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. It was first formulated by Koopmans and Beckmann (1957) and since then it has been used as a suitable model for many different real-world problems: backboard wiring, campus planning, typewriter keyboard design, hospital layout, ranking of archeological data, ordering of interrelated data on a magnetic tape, minimizing average job completion in machine scheduling, and others (Burkard, 1984).

Formally the problem can be defined by three $n \times n$ mactrices:
$\boldsymbol{D} = \{d_{ij}\}$ = The distance between location i and location j.
$\boldsymbol{F} = \{f_{hk}\}$ = The flow (of information, products or some other quantity) between facilities h and k.
$\boldsymbol{C} = \{c_{ih}\}$ = The cost of assigning facility h to location i.

Usually $\boldsymbol{D}$ and $\boldsymbol{F}$ are integer-valued symmetric matrices and the matrix $\boldsymbol{C}$ of the assignment costs is not considered. A permutation π can be interpreted as an assignment of facility $h = \pi(i)$ to location i, for each $i = 1, \ldots, n$. The problem is then to identify a permutation π of both row and column indexes of the matrix $\boldsymbol{F}$ that minimizes the total cost [Edwards, 1980]:

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

The problem can be also formulated in a way that makes the quadratic nature of the objective function more explicit. The permutation π can in fact be expressed by an $n \times n$ permutation matrix $\boldsymbol{X}$,

whose elements x_{hi} are 1 if facility h is to be assigned to location i, and 0 otherwise. The objective function becomes

$$\min z = \sum_{i,j,h,k=1}^{n} d_{ih} f_{jk} x_{ij} x_{hk} + \sum_{i,j=1}^{n} c_{ij} x_{ij} \tag{2}$$

subject to the constraints

$$\sum_{i=1}^{n} x_{ij} = 1, \quad j = 1, \ldots, n$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad i = 1, \ldots, n$$

$$x_{ij} \in \{0,1\}, \quad i, j = 1, \ldots, n.$$

Since the multiplication $\boldsymbol{FX}$ of matrix $\boldsymbol{F}$ with a permutation matrix $\boldsymbol{X}$ results in a corresponding permutation of the row indexes of $\boldsymbol{F}$ and since the multiplication $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{F}$ of the transposed of the same permutation matrix $\boldsymbol{X}$ and $\boldsymbol{F}$ results in a corresponding permutation of the column indexes of $\boldsymbol{F}$, it can easily be verified that QAP can also be expressed in trace form as

$$\min z = \mathrm{tr}(\boldsymbol{DX}^{\mathrm{T}}\boldsymbol{FX} + \boldsymbol{CX}) \tag{3}$$

where the elements of the permutation matrix $\boldsymbol{X}$ are subject to the same constraints as those of formula (2).

QAP is a generalization of several other problems (Finke and Medova Dempster, 1989), including the Travelling Salesman Problem, the Triangulation Problem and the Matching Problem. Since QAP is a generalization of TSP, it is an NP-hard problem. It has been shown (Sahni and Gonzales, 1976) that even finding an ε-approximate solution for any problem instance is NP-hard (an ε-approximation of the optimal value z^* computed for a solution x is a value $z(x)$ such that $|z^* - z(x)|/z^* < \varepsilon$). This restriction does not hold for the TSP, and in fact the QAP seems in general tougher than the TSP: limited enumeration approaches can yield the optimal solution for TSP instances of several thousands towns and for QAP instances of order 15–20 (that is, QAP can be solved optimally only through a more or less complete enumeration: given the current computer technology 15–20 is in fact the limit that can be reached for a complete analysis of all possible permutations in a reasonable time).

An interesting asymptotic property proved in Rhee (1988) is that the difference between the worst and the best solution of a QAP instance becomes smaller when the size of the problem becomes larger if d_{ij} and f_{hk} are mutually independent sequences of independent uniformly distributed random variables; already for $n \geq 50$ the relative difference is very small. This shows that tests on randomly generated problems must be considered very carefully before being accepted as significant: if many local optima of similar quality exist, a local search will discover a relatively good solution with high probability. This effect has been in fact observed in our experiments, as it is pointed out in the conclusions of the paper.

To get a feeling of the complexity of the problem, we plotted in Fig. 1 the values taken by the objective function of a Nugent problem instance of size 12 (Nugent, Vollmann and Ruml, 1968): we plotted the values corresponding to the permutations having every possible combination of values in the first two positions. It is evident that the function is highly multimodal, suggesting a hard to solve problem.

3. Algodesk

In order to compare the relative effectiveness of the different heuristics, we implemented a unified software system called '*Algodesk*', that will be more thoroughly presented in Section 4. Algodesk allows

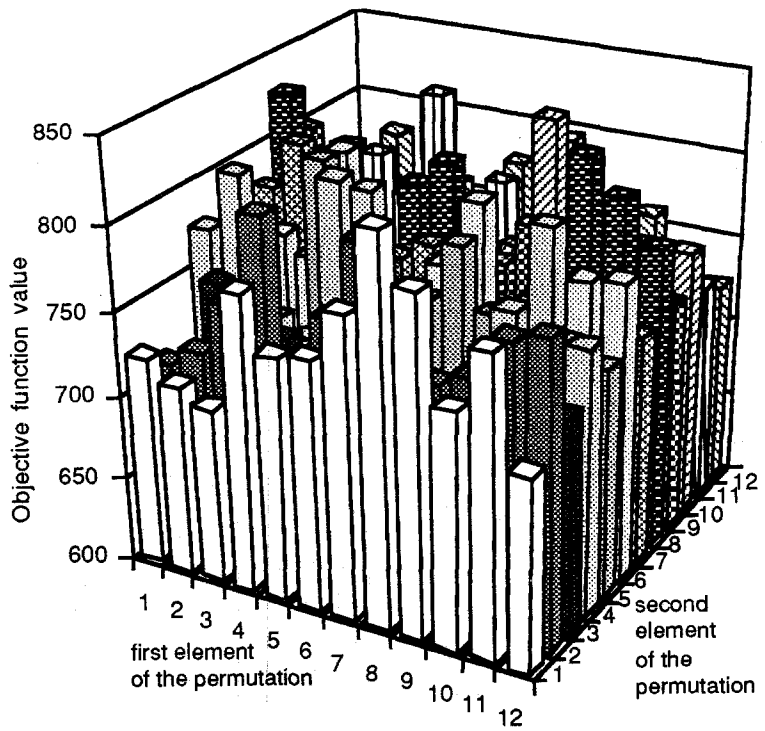

Figure 1. Fitness landscape for the Nugent 12 problem

to apply any of the algorithms it contains to any of the QAP problem instances available (Maniezzo, 1991). Each algorithm that we tested codifies a different evolutionary heuristic. Most of the algorithms are in some way inspired by nature, and update populations of solutions. However, we also included some approaches which, although not inspired by nature, have proven very effective and robust.

The list of the currently available algorithms includes: *Boltzmann Machine* (BM), *Evolution Strategy* (ES), *Genetic Algorithm* (GA), *MultiGreedy* (MG), *Sampling and Clustering* (SC), *Simulated Annealing* (SA), *Tabu Search* (TS), *Immune Networks* (IN). Each of these is detailed in the following subsections. They can be classified as nature-inspired or non-nature-inspired algorithms and – alternatively – as algorithms that consider a single solution at each cycle and algorithms that update a whole population of solutions at each cycle. The proposed classification is presented in Figure 2.

MultiGreedy has been classified as both a single-solution and as a population-based algorithm because of the different interpretations than can be given to the way it works; multigreedy is discussed in Section 3.1. A different classification, more related to the metaphors which inspired the algorithms, divides them among thermodynamically motivated algorithms (SA, BM), evolutionarily motivated algorithms, with a biological acception of the term 'evolutionary' (GA, ES, IN), and algorithms without a natural interpretation (MG, TS, SC).

	single solution	population of solutions
non nature-inspired	(MG), TS	(MG), SC
nature-inspired	SA, BM	GA, ES, IN

Figure 2. Classification of Algodesk heuristics

The algorithms were compared with respect to the best solution found in a fixed amount of time, equal for all of them. We therefore included in each of them an exit test, called END_TEST, that compared the time elapsed since the start of the search with the user-specified maximum time allowed. This is the only termination criterion for some algorithms (GA, ES, TS, MG, SC, IN), while others (SA, BM) already have their own terminating condition. END_TEST has however been included in all the algorithms to allow comparison.

All the algorithms that involve local search (MG, SA, TS, SC) use the same routine to explore the *neighborhood* of a current solution, utilizing the formulae reported in Burkard and Rendl (1984) and Taillard (1990) to efficiently compute the variation in the objective function due to a swap of the elements in positions r and s of the permutation φ that leads to a new permutation π.

3.1. MultiGreedy

The multigreedy algorithm (MG) was included as a testbed against which to compare the other algorithms. It implements a multistart greedy approach, in which several randomly chosen solutions are independently carried to their local optima by exploring at each step the complete neighborhood of the current solution and then moving according to the maximum gradient. We introduced a slight modification of this basic approach using a population of solutions, each of which is carried to the optimum. At each cycle a new random population is generated and all its elements are carried to their optima. This is the same as generating a large population from the beginning, carrying all its elements to their optima and then stopping, but the notion of population that we introduce by our iterative version is useful in the comparison, as it provides results about the performance of populations of totally non cooperative individuals.

Experiments have also been performed with an algorithm that changed the current solution as soon as a better one was identified (without a complete exploration of the neighborhood), but this approach proved less effective.

The multigreedy algorithm is the following.

Multigreedy Algorithm

```
repeat
   randomly generate M solutions (S(i), i = 1,..., M)
   for each solution S(i) compute the corresponding objective function z(S(i))
   set BEST_SOLUTION to solution S(i) such that z(S(i)) is minimal
   for i := 1 to M do
lab: explore the whole neighborhood of solution S(i) and store the best neighbor S'(i)
     if z(S(i)) > z(S'(i))
        then set S(i) := S'(i)
             goto lab
        else if z(S(i)) < z(BEST_SOLUTION)                    {BEST_SOLUTION is the best
                                                               permutation identified so far}
                then set BEST_SOLUTION := S(i)
   endfor
until END_TEST
```

The only parameter of this algorithm is M, the number of solutions composing the population.

3.2. Simulated Annealing

Simulated annealing (SA) is an effective single-solution randomized heuristic, based on an algorithm originally presented in Metropolis et al. (1953) and proposed as a combinatorial optimization tool in Kirkpatrick, Gelatt and Vecchi (1983). Examples of its application to QAP are presented in Burkard and

Rendl (1984) and in Connolly (1990). It updates a single solution, accepting in probability also modifications that involve a worsening of the objective function.

The algorithm that we implemented is the following.

Simulated Annealing Algorithm
choose the initial solution S randomly
set TEMPERATURE := INIT_TEMP
set $t := 0$
repeat
  set $t := t + 1$
  for $i := 1$ to N_ITERATIONS do
    generate S' randomly in the neighborhood of S
    set $\delta := z(S') - z(S)$
    if ($\delta < 0$) or (Random(0, 1) < exp($-\delta$/TEMPERATURE))
      then set $S := S'$
  endfor
  set TEMPERATURE := Anneal(TEMPERATURE, t)
until TEMPERATURE < LOW or END_TEST or (no changes have been accepted for S)

The algorithm uses the function Anneal that transforms the current temperature level into a lower one. Three different annealing schedules have been implemented and the user is free to choose among them. The alternative schedules are:

$$T(t+1) = \alpha T(t), \qquad T(t+1) = \frac{T(t)}{1+\alpha T(t)}, \qquad T(t+1) = \frac{T(0)}{1+\alpha t},$$

where α ($0 < \alpha < 1$) is a user-defined parameter, called Cooling rate. In our experiments we always used the first function.

The function Random generates a uniformly distributed pseudorandom variable in the specified interval.

The parameters of this algorithm are

INIT_TEMP:	The initial temperature level.
N_ITERATIONS:	The number of iterations to be performed between two temperature changes.
LOW:	Since all the annealing schedules tend asymptotically to 0, this parameter can be used to shorten the final stage of a run.
α:	The cooling rate of the Anneal function.

3.3. Tabu Search

Tabu Search (TS) is another evolutionary heuristic that updates a single solution. It was originally proposed in Glover (1989, 1990); specific applications to QAP are presented in Skorin-Kapov (1990) and Taillard (1990). The idea behind it is to start from a random solution and successively swap pairs of its elements. Each time a swap (a *move*) has been chosen, the reverse one is linked at the beginning of a fixed-length list of inhibited moves, the *tabu list*. The new candidate swap brings the solution to its best neighbor: if the swap is present in the tabu list, it is accepted only if it decreases the objective function value below the minimal level so far achieved (*aspiration level*).

The algorithm implemented includes a variable-sized tabu list as proposed by Taillard: a minimal and a maximal length are specified and during the search the actual length is randomly changed.

The algorithm is the following.

Tabu Search Algorithm
choose the initial solution S randomly, set its tabu list $T = \emptyset$
set BEST_SOLUTION := S
set $i := 0$ {i is the iteration counter}
repeat
  set $i := i + 1$
  identify S', the best neighbor of S
  set SWAP := Move(S, S') {SWAP holds the move transforming S into S'}
  if SWAP $\notin T$
    then Update(SWAP, T, LENGTH_TABU_LIST)
      set $S := S'$
      if $z(S) < z$(BEST_SOLUTION) then set BEST_SOLUTION := S
    else if $z(S') < z$(BEST_SOLUTION) {use of aspiration level}
      then Update(SWAP, T, LENGTH_TABU_LIST)
        set BEST_SOLUTION := S'
        set $S := S'$
  if (i mod 2 * MAX_TABU_LIST) = 0
    then set LENGTH_TABU_LIST := Random(MIN_TABU_LIST, MAX_TABU_LIST)
until END_TEST

The algorithm uses the following functions.
Update: Inserts SWAP as the first element of T and removes the last element of T if the tabu list was full.
Move: Returns the swap that transforms a solution into a second one.
The parameters of the algorithm are
MIN_TABU_LIST, MAX_TABU_LIST: They specify the minimal and maximal allowed length of the tabu list, respectively.

3.4. Genetic Algorithm

Genetic Algorithms (GA) are a computational device, originally proposed by Holland (1975), inspired by population genetics. Their effectiveness as a combinatorial optimization tool has been recently advocated (Peterson, 1990); specific applications to QAP have been described in Mühlenbein (1989) and Brown, Huntley and Spillane (1989).

Their strength is essentially due to their updating a whole population of possible solutions: this allows an intrinsically parallel exploration of the search space (Holland, 1975); (Bertoni and Dorigo, 1993).

In order to apply GA, which work best in unconstrained optimization, to QAP, we had to adapt the basic algorithm in order to generate only feasible solutions. Moreover GA are able to cope only with maximization problems, while QAP is a minimization problem: we had therefore to map the QAP objective function into a *fitness function* to be maximized.

The algorithm that we implemented is the following.

Genetic Algorithm
initialize $S := (S(i), i = 1, \ldots, M)$ and compute their fitnesses {initialization of a population of solutions}
set BEST_SOLUTION to the solution in S with the maximal fitness value
repeat
  for $i := 1$ to M do
    ff(i) := Fitness($z(S(i))$) {computation of the fitness}
    if $z(S(i)) < z$(BEST_SOLUTION) then BEST_SOLUTION := $S(i)$

```
  for i := 1 to M do p(i) := ff(i)/Σff(i)        {p(i) is the reproduction probability}
    S' := Reproduction(S)                        {reproduction builds population S' from S}
    randomly generate ½M pairs of solutions in S'
    S'' := Crossover(S')
    S''' := Mutate(S'')
    S := S'''
until END_TEST
```

This algorithm utilizes the following functions.

Fitness: Transforms the objective function, to be minimized, into a fitness function to be maximized, so that the GA can be applied; it implements a monotonically decreasing function, in our experiments a linear function (see Grefenstette and Baker, 1989).

Reproduction: Selects one by one the members of the new population S' from those of S via a Monte Carlo procedure based on the reproduction probabilities previously computed.

Crossover: Is an adaptation of the PMX crossover operator proposed in Goldberg and Lingle (1985). It is applied with probability p_c on each pair of solutions; it chooses two cutting points in the parents' strings, p_1 and p_2, and copies the middle segment of each parent in the corresponding position of the alternate offspring, respectively o_1 and o_2. The first and third segment of each offspring are copied from the corresponding parent (p_1 for o_1 and p_2 for o_2), except for the elements already present in the middle position. The empty positions are filled with the elements still not present, taken from the corresponding parent in the order in which they are present in it.

Mutate: Randomly chooses two positions in the permutation and swaps their contents.

The parameters of the algorithm are:

M: Size of the population.

p_m: Mutation probability.

p_c: Crossover probability (utilized within the function Crossover).

3.5. Evolution Strategies; discrete and continuous versions

Evolution Strategies (ES) are a computational model developed independently but with a structure similar to genetic algorithms (Rechenberg, 1973; Schwefel, 1975). The general framework is identical for ES and GA: both rely on populations of individuals modified by operators of selection, mutation and recombination (called crossover in GA terms). Nevertheless the details of their realization differ significantly.

ES were initially conceived for the optimization of real parameters. To apply them to combinatorial optimization problems, specifically to QAP, we had to modify their basic paradigm. This has been done in two ways: in the first (discrete space evolution strategies, ED) the elements of the population are permutations, and mutation and recombination operators have been largely borrowed from GA. In the second (continuous space evolution strategies, EC) the basic algorithm has not been modified, but the individuals of the population are interpreted as a permutation, utilizing their rank order as a permutation index (Rudolph, 1990).

Both the algorithms start with a population S_μ of solutions and apply some operators on it, obtaining a population S_λ. The discrete space algorithm proposed implements the so-called (μ, λ) reproduction strategy, in which the individuals of S_μ chosen at the end of the reproduction step are the best of S_λ. An alternative policy, called the $(\mu + \lambda)$ strategy, consists in choosing the individuals of the next S_μ among the best individuals of $S_\lambda \cup S_\mu$. For completeness of presentation (μ, λ) selection will be shown in ED, and $(\mu + \lambda)$ selection will be shown in EC, but both selection strategies are possible for both algorithms.

The discrete space algorithm, ED with (μ, λ) selection, is the following.

Evolution Strategy Algorithm (discrete version)
initialize $S_\mu := (S(i),\ i = 1, \ldots, \mu)$ and compute their fitnesses {initialization of a population of solutions}
set BEST_SOLUTION to the solution in S_μ with maximal fitness value
repeat
  for $i := 1$ to λ do {build a new population S_λ out of S_μ, $\lambda \geq \mu$}
    $S_\lambda(i) :=$ Recombination (S_μ)
    $S_\lambda(i) :=$ Mutate $(S_\lambda(i))$
    if $z(S_\lambda(i) < z$ (BEST_SOLUTION) then BEST_SOLUTION $:= S_\lambda(i)$
  endfor
  set S_μ to the best μ individuals of S_λ {this is the (μ, λ) strategy}
until END_TEST

The functions used in this algorithm are the following.

Recombination: Can take two forms: discrete recombination and global discrete recombination. The first takes two parents and builds an offspring by randomly choosing its elements between the corresponding ones of the parents. The second chooses a random number of parents and then builds the offspring randomly choosing its elements among the corresponding ones of the parents. Details of their implementation can be found in Bäck, Hoffmeister and Schwefel (1991). The feasibility of the offspring is obtained by constraining the possible element choices to be compatible with the assignments already made in the previous part of the string.

Mutate: Randomly chooses two positions in the permutation and swaps their contents.

The continuous space version of Evolution Strategies is based on a mapping from real-coded individuals to permutations, thus allowing to utilize the original ES algorithm. This mapping is based on a ranking of the values of each element of the individual. For example, suppose that an individual is codified by the real-valued string (0.123 1.324 3.427 −7.298 2.375). The corresponding ranking of its elements consists of the integer string (2 3 5 1 4), which is actually a permutation that can be interpreted as a solution of a combinatorial optimization problem.

The continuous space algorithm, EC with $(\mu + \lambda)$ selection, is the following.

Evolution Strategy Algorithm (continuous version)
initialize $S_\mu := (S(i),\ i = 1, \ldots, \mu)$ and compute their fitnesses {initialization of a population of solutions}
set BEST_SOLUTION to the solution in S_μ with maximal fitness value
repeat
  for $i := 1$ to λ do
    $S_\lambda(i) :=$ Recombination (S_μ)
    $S_\lambda(i) :=$ Mutate $(S_\lambda(i))$
    permutation$(i) :=$ Map$(S_\lambda(i))$ {permutation(i) is the permutation corresponding to $S_\lambda(i)$}
    if z(permutation$(i)) < z$(BEST_SOLUTION) then BEST_SOLUTION $:=$ permutation(i)
  endfor
  set S_μ to the best μ individuals of $S_\lambda \cup S_\mu$ {this is the $(\mu + \lambda)$ strategy}
until END_TEST

The functions used in this algorithm are the following.

Mutate: Modifies each gene of the individual. Each individual is codified by a vector $S(i) = (x_i,\ s_i)$, where $x_i = (x_i^1, \ldots, x_i^n)$ codifies a problem solution and $s_i = \sigma_i^1, \ldots, \sigma_i^n$ contains

control parameters. The mutated solution is obtained modifying each gene according to

$$x_i' = x_i + N(0, \sigma_i')$$

where $N(0, \sigma_i')$ is a normally distributed random variable with expectation 0 and variance

$$\sigma_i' = \sigma_i * e^{N(0, t_1)} * e^{N(0, t_0)}$$

where $t_0 = 1/(2n)$ and $t_1 = 1/\sqrt{4n}$. See Schwefel (1975) for more details.

Recombination: Implements four kinds of recombinations: discrete and intermediate, each either between two parents or global; discrete recombination is the same as in ESc, intermediate recombination averages the values of the corresponding elements of the parents.

Map: Returns the permutation obtained substituting each gene's value with its corresponding rank order (see the example above).

The parameters are the same for both ESc and ESd and include

μ: Size of the parent population.

λ: Size of the enlarged population in the recombination step.

3.6. Sampling and Clustering

Sampling and Clustering (SC) is a heuristic which works on a variable-sized population of tentative solutions. It was originally proposed in Boender, Rinnooy Kan, Timmer and Stougie (1982) and in Camerini, Colorni and Maffioli (1986). The idea behind it is to generate a population of uniformly distributed tentative solutions, to consider only the best of them and to cluster all the solutions around seed points that will be taken to their local optima.

The algorithm is the following.

Sampling & Clustering Algorithm
set $S := \emptyset$, **Loc_opt** $:= \emptyset$, **Seeds** $:= \emptyset$
repeat
  randomly generate M uniformly distributed solutions and add them to S
  eliminate from S a percentage γ of the worst solutions
  if **Loc_opt** $\neq \emptyset$ then Cluster(S, **Loc_opt**)
  if **Seeds** $\neq \emptyset$ then Cluster(S, **Seeds**)
  if $S \neq \emptyset$ then
    choose the best solution $S(i)$ in S and carry it to its local optimum $S^*(i)$
    if $S^*(i) \in$ **Loc_opt**
      then add $S(i)$ to **Seeds**
      else add $S^*(i)$ to **Loc_opt**
until END_TEST or no new local optimum has been identified

The function used in the algorithm is the following.

Cluster(A, B): Clusters as many points of A as possible around those of B. It works considering one by one the points of B and clustering around them all the points of A that would *probably* lead to the same local optimum. When a point of A is clustered, it is removed from A. The routine first clusters around the current point of B all the points of A that are obtainable from it with a single swap, if there were any, then it clusters those obtainable with two swaps, then with three swaps and so on. As soon as a distance is found (measured in minimal number of swaps) at which no point of A lies from the seed, the clustering is terminated and a new point of B is considered. The routine returns A, containing all and only the points that could not be clustered.

The parameters of this algorithm are
M: Number of new solutions generated at each cycle.
γ: Percentage of solutions discarded at each cycle.

3.7. Boltzmann Machine

The Boltzmann Machine (BM) (Hinton and Seinowski, 1986) is a connectionist model closely related to SA that seems well suited for solving combinatorial optimization problems (Aarts and Korst, 1988). We tested the application to QAP of the basic BM algorithm that has been described in Chakrapani and Skorin-Kapov (1990).

The algorithm implemented is the following.

Boltzmann Machine Algorithm

```
initialize the neurons activation matrix N          {this is a permutation matrix}
initialize the weight matrix W
set TEMPERATURE := INIT_TEMP
repeat
   for i := 1 to N_ITERATIONS do
      randomly choose an element N(i, j) and change its state
      compute the variation of Consensus, ΔC_ij
      if (ΔC_ij < 0) or (Random(0, 1) < exp(−ΔC_ij/TEMPERATURE))
         then accept the new state for N(i, j)
   endfor
   set TEMPERATURE := Anneal(TEMPERATURE)
until TEMPERATURE < LOW OR END_TEST
```

The functions used in this algorithm are the following.

Anneal: Transforms the current temperature level into a lower one. The functions used to define the temperature annealing schedules are the same functions used for SA.

Consensus: Computes the variation of consensus following the variation of the state of an element. The consensus of a configuration is computed as $C = \sum_{ij} w_{ij} u_i u_j$, where w_{ij} is the weight of the connection between node i, which is in the state u_i, and node j, which is in the state u_j.

The parameters of this algorithm are the same of SA, namely:

INIT_TEMP: The initial temperature level.

N_ITERATIONS: The number of iterations to be performed in the inner cycle.

LOW: Since all the annealing schedules tend asymptotically to 0, this parameter can be used to shorten the final stage of a run.

α: The cooling rate.

3.8. Immune Networks

Immune Networks (IN) (Bersini and Varela, 1991) are a computational model inspired by the main functional principles of the immune system. They try to capture the essential traits of the dynamic process that allows the recruitment of new problem-specific lymphocite species from the huge pool produced by the bone-marrow.

This approach has been proposed to solve both continuous and discrete optimization problems.

The algorithm we implemented is the following.

Immune Network Algorithm
generate randomly M initial solutions ($S(i)$, $i = 1, \ldots, M$)
repeat
  Recruited := $\emptyset$
  Candidates := Gen_cand(S)
  while |**Recruited**| < N_RECRUIT do
    randomly choose a solution $C \in$ **Candidates**
    set **Candidates** := **Candidates** − {C}
    choose N_AFFINITY solutions $S(1), \ldots, S(\text{N\_AFFINITY})$

$$\text{if} \frac{\sum_{j=1}^{\text{N\_AFFINITY}} z(S(j))\ \text{Affinity}(S(j), C)}{\sum_{j=1}^{\text{N\_AFFINITY}} \text{Affinity}(S(j), C)} \geq \text{Threshold}(S, \text{N\_THRESHOLD})$$

    then **Recruited** := **Recruited** ∪ {C}
    if **Candidates** = $\emptyset$ then **Candidates** := Gen_cand(S)
  endwhile
  set **Eliminate** := the worst N_RECRUIT solutions in S
  S := (S − **Eliminate**) ∪ **Recruited**
until END_TEST

The functions used in this algorithm are the following.

Gen_cand: Generates a new set of candidates from S by applying crossover and mutation (as in genetic algorithms).

Affinity: Measures the similarity of two solutions; we used the following affinity function: Affinity($S(i)$, $S(j)$) = $1 - x/n$, where x is the number of different allocations proposed by the two permutations $S(i)$ and $S(j)$. The solutions against which candidate C is compared are chosen among those of S that have a distance from C less than AFFINITY_RADIUS.

Threshold: Is given by the following formula:

$$\text{Threshold}(S, \text{N\_THRESHOLD}) = \sum_{i=1}^{\text{N\_THRESHOLD}} z(S(i))/\text{N\_THRESHOLD}.$$

The parameters of this algorithm are:

N_RECRUIT: The number of individuals to be recruited at each cycle.

N_AFFINITY: The number of individuals in S with which a new candidate solution is compared to decide about its recruitment.

AFFINITY_RADIUS: Maximal distance from the candidate of the solutions against which the candidate is compared.

N_THRESHOLD: The number of individuals in S used to set the recruitment threshold.

4. Experimental results

Experiments have been carried out over some well-known problem instances: the Nugent problems (Nugent, Vollmann and Ruml, 1968), sizes 5 to 30, the Elshafei problem (Elshafei, 1977), size 19, and one Krarup problem of size 30 (Krarup and Pruzan, 1978). All the algorithms have been implemented in a unified environment on an IBM-PC. The resulting system, called Algodesk, allows the application of any heuristic to any QAP problem instance written in a compatible format. It presents output data of the

Table 1
Best solution found in 1 hour of search

	Nugent 5	Nugent 7	Nugent 12	Nugent 15	Nugent 20	Nugent 30	Elshafei 19	Krarup 30
MG	50	148	578	1150	2570	6146	17212548	90090
SA	50	148	578	1150	2570	6128	17937024	89800
TS	50	148	578	1150	2570	6124	17212548	90090
GA	50	148	588	1160	2688	6784	17640584	108830
ED	50	148	598	1168	2654	6308	19600212	97880
EC	50	148	596	1186	2692	6850	21342612	98860
SC	50	148	590	1150	2638	6182	22941664	92900
BM	50	150	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
IN	50	148	586	1158	2644	6468	18316048	95960
Best known	50	148	578	1150	2570	6124	17212548	88900

resulting evolution both graphically and in a form that eases the analysis of maxima, averages, standard deviations, etc., both with respect to the number of cycles and to the time elapsed.

The comparison has been made by letting every algorithm solve a problem on the same machine. In Table 1 we give the best results achieved over five 1-hour-long runs: [1] we set in fact the END_TEST of the algorithms (MG, TS, GA, ES, SC) to 1 hour and set the cooling rate of SA and BM so that their runs as well lasted approximately 1 hour. The n.a. – not available – symbol in Tables 1 and 2 indicates that the algorithm could not be applied to the corresponding problem instance on our machine. BM were in fact a priori unable to cope with higher-order problem instances on the machines used, due to memory constraints.

GA and ES (both in the continuous and in the discrete space versions) were surprisingly ineffective. This is due to the limited time-span of the runs and to the limited computational power of the computers used, which cut their search when substantial progress was still possible. Moreover, the power of these algorithms is best exploited by parallel computers.

In Figure 3 we present typical evolutions of each algorithm on the Nugent 15 problem (optimal value: 1150). The x-axis represents the duration of the run in seconds, while the y-axis shows the best solution found. At the beginning of the search, when no solution has yet been suggested, an arbitrary high value is given.

[1] The parameters used were: MG: $M = 30$; SA: $\alpha = 0.99$, INIT_TEMP problem dependent, LOW = 0.5, N_ITERATIONS = such to have 1-hour-long runs; TS: MIN_TABU_LIST = 20, MAX_TABU_LIST = 40; GA: $M = 30$, $p_c = 0.6$, $p_m = 1/$problem dimension; EC, ED: $\mu = 15$, $\lambda = 100$; SC: $M = 30$, $\gamma = 40\%$; BM: as SA; IN: N_RECRUIT = 5, N_AFFINITY = 2, AFFINITY_RADIUS problem dependent, N_THRESHOLD = 2.

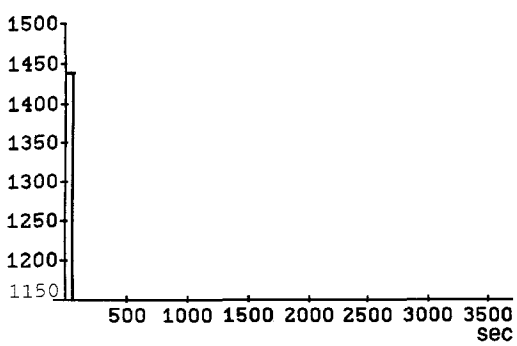

Figure 3.1. Multigreedy

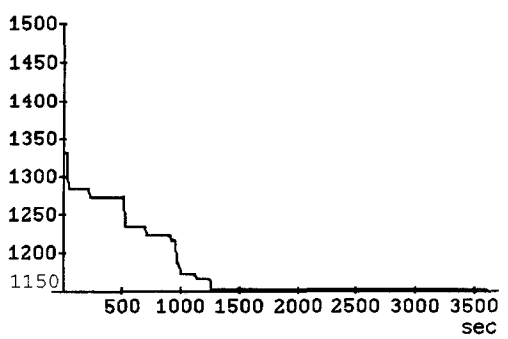

Figure 3.2. Simulated Annealing

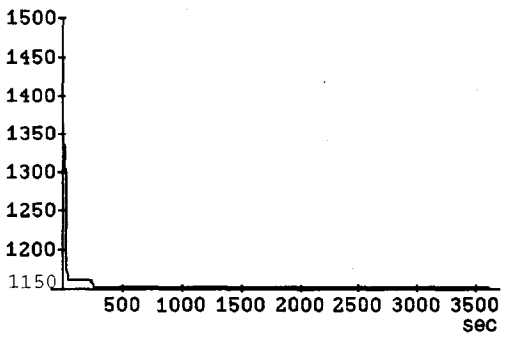

Figure 3.3. Tabu Search

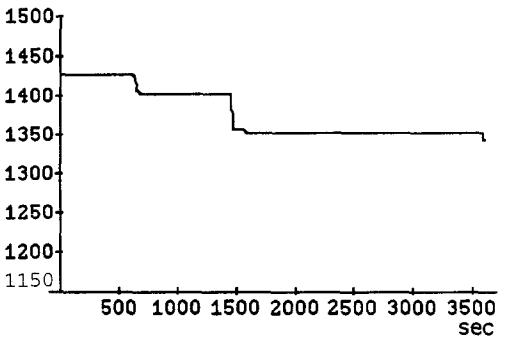

Figure 3.4. Genetic Algorithm

Figure 3.5. Discrete space evol. strategy

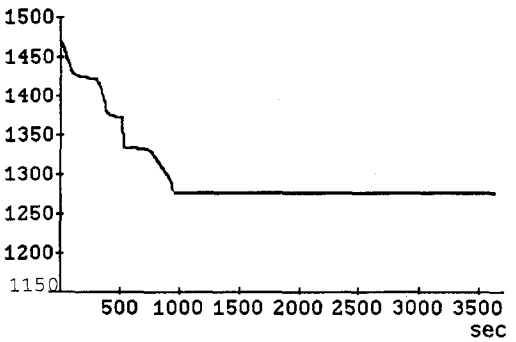

Figure 3.6. Continuous space evol. strategy

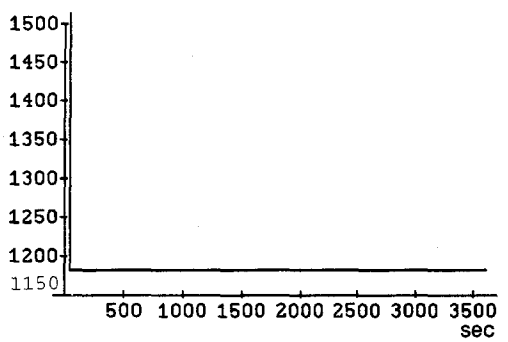

Figure 3.7. Sampling and clustering

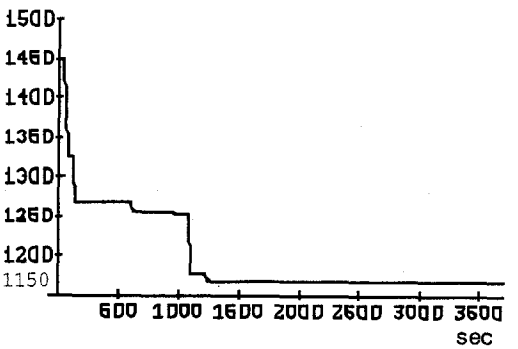

Figure 3.8. Immune network

Given these data, it is possible to compute an index R_t, which quantifies the performance of each heuristic in the solution of each problem. For instance,

$$R_t = \sum_{t=1}^{T} h(t)/(T * \text{BestKnown}) \tag{4}$$

(where $h(t)$ is the result of the heuristics being considered at time t and time t is an integer valued variable taking values between 1 and T) quantifies how much and for how long the heuristic under consideration has produced a result worse than the best known one, as expressed in BestKnown.

Table 2
The R_t index after 1 hour of search

	Nugent 5	Nugent 7	Nugent 12	Nugent 15	Nugent 20	Nugent 30	Elshafei 19	Krarup 30
MG	1.000	1.000	1.000	1.000	1.002	1.008	1.001	1.040
SA	1.000	1.000	1.000	1.006	1.013	1.031	1.042	1.035
TS	1.000	1.000	1.000	1.000	1.002	1.013	1.007	1.030
GA	1.000	1.000	1.026	1.025	1.063	1.134	1.057	1.266
ED	1.000	1.001	1.047	1.029	1.049	1.105	1.146	1.478
EC	1.000	1.005	1.123	1.042	1.094	1.156	1.148	1.472
SC	1.000	1.000	1.001	1.002	1.004	1.005	1.003	1.040
BM	1.000	1.093	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
IN	1.000	1.000	1.036	1.028	1.047	1.118	1.098	1.216

Applying formula (4) to the problems presented in Table 1, we obtain the results presented in Table 2, which are relative to 1-hour searches.

Part of the data contained in Table 2 is also graphically presented in Figure 4, where the different performances of the tested algorithms appear more plainly (in Figure 4 we reported results only on some of the test problems in order to obtain a more easy-to-read graphic).

5. Conclusions

The paper presents a comparison of 8 evolutionary heuristic algorithms applied, as a testbed, to the Quadratic Assignment Problem. For each heuristic, the specific algorithm that has been implemented is given.

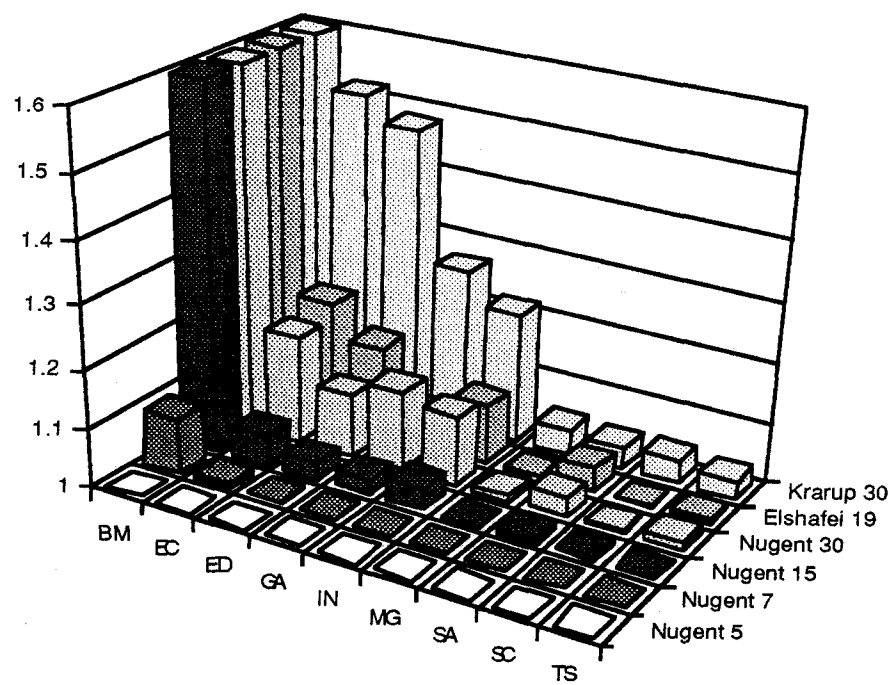

Figure 4. R_t index after 1 hour of search

We developed a software system, Algodesk, which allows to apply any of the heuristics to any of the problem instances and to analyze the resulting output in a unified context.

The comparison is not aimed at the determination of the best result achievable (these data have already been provided in the literature for most of the algorithms), but at the assessment of their efficiency in yielding good solutions. For this reason, the raw data consists of the best solution produced by each algorithm in 1-hour-long runs on identical IBM-PC machines.

A synoptic table is presented, showing the results achieved by each algorithm on each instance. Moreover, an index is proposed that quantifies the effectiveness of a search, both with respect to the result achieved and to the speed with which it has been attained. Particularly interesting (and somewhat surprising) is:

(i) the effectiveness of the multigreedy approach, thus of the local search operator;
(ii) in general, the superiority of single-solution approaches to population-based ones when these last are run on a single-processor hardware and are not coupled with local search operators;
(iii) the inefficiency of Boltzmann machines on searches of limited duration.

Point (i) reflects a QAP property which was not yet put into evidence in the literature: any approach based on local search is bound to be very effective as an heuristic for QAP.

Point (ii) testifies the need to pursue studies on population-based heuristics. While it is in fact intuitive that an exchange of information between different solution could ease the search process, it appears that current communication operators are not worth their computational cost.

Future work includes the extension of the set of algorithms tested (including, for example, the Ant-system, Colorni, Dorigo and Maniezzo, 1991, 1992), of problem instances used, and of terminating conditions (a comparison on an equal number of function evaluations could be interesting). Eventually we want to achieve the definition of a new heuristic, encompassing the strong points of the better performing algorithms.

References

Aarts, E.H.L., and Korst, J.H.M. (1988), *Simulated Annealing and Boltzmann Machines*, Wiley, New York.

Bäck, T., Hoffmeister, F., and Schwefel, H.P. (1991), "A survey of evolution strategies", in: *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA.

Bersini, H., and Varela, F.J. (1991), "The immune recruitment mechanism: A selective evolutionary strategy", in: *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA.

Bertoni, A., and Dorigo, M. (1993) "Implicit parallelism in genetic algorithms", *Artificial Intelligence* 61/2, 307–314.

Boender, C.G.E., Rinnooy Kan, A.H.G., Timmer, G.T. and Stougie, L. (1982), "A stochastic method for global optimization", *Mathematical Programming* 22.

Brown, D.E., Huntley, C.L., and Spillane, A.R. (1989), "A parallel genetic heuristic for the Quadratic Assignment Problem", in: *Proc. of the Third Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA.

Burkard, R.E. (1984), "Quadratic Assignment Problems", *European Journal of Operational Research* 15.

Burkard, R.E., and Rendl, F. (1984), "A thermodynamically motivated simulation procedure for combinatorial optimization problems", *European Journal of Operational Research* 17.

Camerini, P.M., Colorni, A., and Maffioli, F. (1986), "Some experience in applying a stochastic method to location problems", *Mathematical Programming Study* 26.

Carraresi, P., and Malucelli, F. (1988), "Quadratic Assignment Problems: A review", *Ricerca Operativa* 47.

Chakrapani, J., and Skorin-Kapov, J. (1990), "Connectionist approaches to the Quadratic Assignment Problem", Report HAR-90-08, W.A. Harriman School for Management and Policy, SUNY at Stony Brook.

Colorni, A., Dorigo, M., and Maniezzo, V. (1991), "Distributed optimization by ant colonies", in: *Proc. of the First European Conference on Artificial Life*, Paris, MIT Press, Cambridge, CA.

Colorni, A., Dorigo, M., and Maniezzo, V. (1992), "An investigation of some properties of an ant algorithm", in: R. Männer and B. Manderick (eds.), *Parallel Problem Solving from Nature – 2*, North-Holland, Amsterdam.

Connolly, D.T. (1990), "An improved annealing scheme for the QAP", *European Journal of Operational Research* 46.

Edwards, C.S. (1977), "The derivation of a greedy approximator for the Koopmans–Beckmann Quadratic Assignment Problem", *Mathematical Programming Study* 13.

Elshafei, A.E. (1977), "Hospital layout as a Quadratic Assignment Problem", *OR Quarterly* 28.

Finke, G., Burkard, R.E., and Rendl, F. (1987), "Quadratic Assignment Problems", *Annals of Discrete Mathematics* 31.

Finke, G., Burkard, R.E., and Rendl, F. (1987), "Quadratic Assignment Problems", *Annals of Discrete Mathematics* 31.

Finke, G., and Medova Dempster, E. (1989), "Combinatorial optimization problems in trace forms", *Ricerca Operativa* 52.

Garey, M.R., and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA.

Glover, F. (1989), "Tabu Search – Part I", *ORSA Journal on Computing* 1.

Glover, F. (1990), "Tabu Search – Part II", *ORSA Journal on Computing* 2.

Goldberg, D., and Lingle, J.R. (1985), "Alleles, loci and the Traveling Salesman Problem", in: *Proc. of an Int. Conf. on Genetic Algorithms and their Applications*, Pittsburgh, Lawrence Erlbaum, London.

Grefenstette, J.J., and Baker, J.E. (1989), "How genetic algorithms work: A critical look at implicit parallelism", in: Schaffer (ed.), *Proc. of the Third Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA.

Hinton, G.E., and Sejnowski, T.J. (1986), "Learning and relearning in Boltzmann machines", in: D.E. Rumelhart and J.L. McLelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* MIT Press, Cambridge, MA.

Holland, J.H. (1975), *Adaptation in Natural and Artificial Systems*, University of Michigan Press, MI.

Kirkpatrick, S., Gelatt, C.D., and Vecchi, M.P. (1983), "Optimization by Simulated Annealing", *Science* 220.

Koopmans, T.C., and Beckmann, M.J. (1957), "Assignment problems and the location of economic activities", *Econometrica* 25.

Krarup, J., and Pruzan, P.M. (1978), "Computer-aided layout design", *Mathematical Programming Study* 9.

Maniezzo, V. (1991), "The rudes and the shrewds", Technical Report 91-042, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953), "Equation of state calculations by fast computing machines", *Journal of Chemical Physics* 21.

Mühlenbein, H. (1989), "Parallel genetic algorithms, population genetics and combinatorial optimization", in: Schaffer (ed.), *Proc. of the Third Int. Conf. on Genetic Algorithms*, Morgan Kaufmann, Los Altos, CA.

Nugent, C.E., Vollmann, T.E., and Ruml, J. (1968), "An experimental comparison of techniques for the assignment of facilities of locations", *Operations Research* 16.

Peterson, C. (1990), "Parallel distributed approaches to combinatorial optimization: Benchmark studies on Travelling Salesman Problem", *Neural Computation* 2.

Rechenberg, I. (1973), *Evolutionsstrategie*, Fromman-Holzbog.

Rhee, W.T. (1988), "A note on asymptotic properties of the Quadratic Assignment Problem", *Operations Research Letters* 7.

Rudolph, G. (1990), "Optimization of combinatorial problems by means of parallel evolutionary algorithms", ESPRIT PCA Meeting, Ispra.

Sahni, S., and Gonzalez, T. (1976), "P-complete approximation problems", *Journal of the ACM*, 23.

Schwefel, H.-P. (1975), "Evolutionsstrategie und numerische Optimierung", Ph.D. Thesis, Technische Universität Berlin. Also available as *Numerical Optimization of Computer Models*, Wiley, New York, 1981.

Skorin-Kapov, J. (1990), "Tabu Search applied to the Quadratic Assignment Problem", *ORSA Journal on Computing* 2.

Taillard, E. (1990), "Robust Taboo Search for the Quadratic Assignment Problem", Report ORPW 90/10, DMA, Swiss Federal Institute of Technology of Lausanne.