ELSEVIER

European Journal of Operational Research 92 (1996) 402–416

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Theory and Methodology

Optimizing simulated annealing schedules with genetic programming

Andreas Bölte [a,*], Ulrich Wilhelm Thonemann [b]

[a] *Department of Production and Operations Management, University of Paderborn, Postfach 1621, 33095 Paderborn, Germany*
[b] *Department of Industrial Engineering, Stanford University, Stanford, CA 94305-4024, USA*

Received January 1994; revised October 1994

Abstract

Combinatorial optimization problems are encountered in many areas of science and engineering. Most of these problems are too difficult to be solved optimally, and hence heuristics are used to obtain "good" solutions in reasonable time. One heuristic that has been successfully applied to a variety of problems is Simulated Annealing. The performance of Simulated Annealing depends on the appropriate choice of a key parameter, the annealing schedule. Researchers usually experiment with some manually created annealing schedules and then use the one that performs best in their algorithms.

This work replaces this manual search by Genetic Programming, a method based on natural evolution. We demonstrate the potential of this new approach by optimizing the annealing schedule for a well-known combinatorial optimization problem, the Quadratic Assignment Problem. We introduce two new algorithms for solving the Quadratic Assignment Problem that perform extremely well and outperform existing Simulated Annealing algorithms.

Keywords: Genetic programming; Optimization; Simulated annealing; Quadratic assignment problem

1. Introduction

The Quadratic Assignment Problem (QAP) can be conveniently defined when we introduce two sets, R and T. Each of the sets contains N elements, indexed from 1 to N. The problem is to assign every element of R to exactly one element of T in order to achieve some objective. The QAP can be formulated as a binary program with a second-degree polynomial objective function and linear constraints:

(QAP1)

$$\min Z(X) = \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{N}\sum_{l=1}^{N} c_{ijkl}x_{ij}x_{kl}$$

$$\text{s.t.} \quad \sum_{j=1}^{N} x_{ij} = 1 \qquad i = 1, 2, \ldots, N,$$

$$\sum_{i=1}^{N} x_{ij} = 1 \qquad j = 1, 2, \ldots, N,$$

$$x_{ij} \in \{0, 1\} \qquad i, j = 1, 2, \ldots, N.$$

* Corresponding author.

SSDI 0377-2217(94)00350-5

The binary decision variables x_{ij} are 1 if element $i \in R$ is assigned to element $j \in T$ and 0 otherwise. The constraints ensure that the assignment is feasible. To determine the cost of an assignment we use the coefficients c_{ijkl}. They represent the cost of assigning $i \in R$ to $j \in T$ and, simultaneously, $k \in R$ to $l \in T$.

A feasible solution to the QAP can also be represented by a N-dimensional permutation vector $\boldsymbol{a} = (a_1, a_2, \ldots, a_N)$, where $a_i \in T$ denotes the element to which $i \in R$ is assigned. For example, $a_3 = 2$ means that element $3 \in R$ is assigned to element $2 \in T$. For our algorithms, this representation is computationally more efficient and will be used in the subsequent sections.

The QAP has a large number of applications in science and engineering. One well-known example is the problem of locating plants with material flow between them to minimize material transportation cost [23]. There, we can consider R to be the set of plants, and T to be the set of locations. The cost of locating the plants only depend on the amount of material shipped (b_{ik}) and the distance between the plants (d_{jl}), and we can replace the coefficients c_{ijkl} by

$$c_{ijkl} = b_{ik} d_{jl}. \tag{1}$$

Other applications include the layout of electronic components on a printed circuit board [33], scheduling jobs on a multiple-processor computer [11] and the design of keyboards [6]. See Burkard [5] for an extended list of applications.

This work is organized as follows. Section 2 reviews solution approaches to the QAP. Section 3 discusses Simulated Annealing. Section 4 presents Genetic Programming. Section 5 describes our algorithmic approach of optimizing annealing schedule with Genetic Programming. Section 6 presents an improved Simulated Annealing algorithm for the QAP. Section 7 completes the paper with a conclusion. For the readers convenience, all notation is summarized in the appendix.

2. Solution approaches

Although the QAP is relatively easy to formulate, it is very difficult to solve. Sahni and Gonzalez [30] showed that the QAP is NP-hard, and only implicit enumeration methods are known for solving it optimally. These approaches are based on branch-and-bound [12,14,25,29] or cutting-plane methods [3]. However, empirical studies suggest that the general QAP is too difficult to be solved optimally. A parallel implementation of Gilmore's [14] branch-and-bound algorithm ran 18 hours on a 32-node Intel IPSC/2 hypercube to solve a problem of size $N = 18$ [21]. Since most real-world problems are of size $N \gg 18$, heuristics are used to find "good" solutions within reasonable time. Heuristics for the QAP can be classified as construction, improvement or global-search heuristics.

Construction heuristics [26] begin with the basic problem data and build up a solution in an iterative fashion. Initially, all elements of R and T are unassigned. Empirical studies have shown that construction heuristics generate solutions of poor quality [4].

Improvement heuristics [2,18,19,29] and global-search heuristics on the other hand, start with an initial feasible solution and try to improve it. The most common improvement heuristics for the QAP are pairwise exchange improvement heuristics. They select two elements of R and compute the change in the objective function value when their counterparts in T are exchanged. The exchange is only realized when it improves the objective function value. This procedure is repeated until no further improvements can be found. The major disadvantage of improvement heuristics is that they usually get stuck in local optima and hence produce sub-optimal solutions.

Global-search heuristics try to escape these local optima by occasionally accepting exchanges that increase the objective function value. The best-known global-search heuristics are Tabu Search [15,16,32] and Simulated Annealing [8,21,22,38].

3. Simulated Annealing

Simulated Annealing originated in statistical mechanics. It is based on a Monte Carlo model

that was used by Metropolis et al. [28] to simulate a collection of atoms in equilibrium at a given temperature. Kirkpatrick et al. [22] were the first who applied Simulated Annealing to solve combinatorial optimization problems. The principle of the algorithm is simple: given an assignment $\boldsymbol{a}$, select two elements of R, say u and v, and compute the change in the objective function value ($\Delta Z_{uv}(\boldsymbol{a})$) when their counterparts in T are exchanged. If the objective function value is improved ($\Delta Z_{uv} < 0$), then accept the exchange and use the resulting configuration as the starting point for the next iteration. If $\Delta Z_{uv} \geq 0$, then accept the displacement with probability

$$P(\Delta Z) = e^{-\Delta z_{uv}/\nu}, \tag{2}$$

where ν is a parameter. Analogous to statistical mechanics, ν is called the temperature.

Simulated Annealing algorithms differ mainly with respect to the following four factors: neighborhood search, annealing schedule, equilibrium test and termination criteria.

3.1. Neighborhood search

The neighborhood $NB(r)$ is defined as all assignments that can be reached from a given assignment when r elements are exchanged. For pairwise exchange algorithms, the size of a neighborhood is $|NB(2)| = \frac{1}{2}N(N-1)$.

At each iteration of the Simulated Annealing algorithm two elements of R are selected and the cost of exchanging their counterparts in T is computed. Most authors (e.g., [22,27,17,38]) choose the next potential assignment at random from the neighbors of the current solution. Connolly [8] investigates a different selection mechanism, where the neighborhood is searched sequentially. First, all possible pairwise exchange partners (u, v) are ordered lexicographically, i.e.,

$$(1, 2), (1, 3), \ldots, (1, N), (2, 3), \ldots, (N-1, N), \tag{3}$$

and then they are investigated sequentially. Hence, we first investigate the pairwise exchange $(u, v) = (1, 2)$, next $(u, v) = (1, 3), \ldots$, and finally $(u, v) = (N-1, N)$. Then, we start all over again with $(u, v) = (1, 2)$. Connolly [8] and Thonemann [34] compare this neighborhood search with the more commonly used random search. They conclude that the sequential search outperforms the random search. Both authors obtain similar results when the list of potential pairwise exchanges is shuffled before being used. Hence, it is only important that the neighborhood is explored thoroughly and the precise sequence is irrelevant. We use the sequential neighborhood search (without shuffling) in our algorithms, since it is computationally more efficient.

3.2. Annealing schedule

The probability of accepting an assignment that increases the objective function value is given by (2) and depends on the temperature ν. Usually, ν is not a constant, but changes over time. When the initial value of the temperature is chosen "high enough" and decreased "slowly", then the Simulated Annealing algorithm will find the global optimum [1,13]. However, the only known annealing schedule that guarantees optimality is

$$\nu(t) = \text{Constant}/\log(t). \tag{4}$$

Implementing this schedule would result in run times that are too long for most applications. Hence, Simulated Annealing algorithms use a schedule that cools down faster. The schedules start with a relatively high temperature and decrease it either after each iteration or after some criterion is satisfied. The most commonly used annealing schedules are [22,38]

$$\nu(t) = \alpha \cdot \nu(t-1), \quad \alpha < 1, \tag{5}$$

and [27]

$$\nu(t) = \frac{\nu(t-1)}{1 + \beta\nu(t-1)}, \quad \beta \ll \nu(0). \tag{6}$$

3.3. Equilibrium test

Many Simulated Annealing algorithms use some kind of equilibrium test to determine if the temperature should be decreased. Wilhelm and Ward [38] check after a certain number of iterations the fluctuations in the objective function

value. If the fluctuations are small, thermal equilibrium is said to be reached, and the next temperature is chosen. Kirkpatrick et al. [22] use the number of accepted and rejected pairwise exchanges as an equilibrium criterion. Connolly's algorithm [8] does not use an equilibrium test, but determines the next temperature after each iteration.

3.4. Termination criterion

Some authors terminate their algorithms after it becomes unlikely that any further improvement can be found [38]. Those algorithms have a nondeterministic run time. Other authors prefer to fix the number of iterations a priori [8] to obtain algorithms with a deterministic run time.

One of the key issues in implementing a Simulated Annealing algorithm is the choice of the annealing schedule [10,22,38]. Most authors try several manually generated annealing schedules and choose the best of those in their algorithms. The initial temperature and the "cooling rate" are either fixed [38] or problem data dependent [27]. Davis and Ritter [10] use a genetic algorithm to optimize the annealing schedule. They assume that an optimal annealing schedule is of form (5) and use a genetic algorithm to adjust the parameters. This work takes a more general approach. As opposed to David and Ritter, we do not assume any special shape of the optimal annealing schedule. We use Genetic Programming [24] to simultaneously optimize the shape *and* the parameters of the annealing schedule.

4. Genetic Programming

Genetic Programming is based on Darwin's evolutionary theory about "survival of the fittest and natural selection". Holland [20] applied the basic principles of natural evolution to artificial systems. His Genetic Algorithms start with an initial generation of artificial individuals which is created randomly. Then, a fitness value is assigned to each individual to describe quantitatively how well the individual masters its task. Based on these fitness values, the next generation is created. Holland uses fixed length character strings to represent his individuals. We use a more flexible approach. Based on Koza's Genetic Programming [24] our individuals are represented as hierarchical computer programs that are modeled by LISP S-expressions.

4.1. First generation

In Genetic Programming, the first generation of individuals is generated randomly from a set of functions and terminals. Since we are interested in finding the optimal annealing schedule, our individuals are real-valued functions with one independent variable, the time, and one dependent variable, the temperature. We use the programming language LISP to represent the individuals. The reader unfamiliar with LISP is referred to Koza [24] or Wilenski [37]. Any LISP S-expression can be represented as a point labeled tree with ordered branches. The latter representation will prove useful in the subsequent discussion. For instance, the LISP S-expression of the function $0.9^t \cdot 10$ is `(* (^ 0.9 t) 10)`. Its representation as a tree is shown in Fig. 2.

To create the first generation of individuals we define two sets $\mathcal{F}$ and $\mathcal{T}$ that contain the functions and terminals, respectively:

$$\mathcal{F} = \{+, -, *, \%, \hat{}, \text{sign}, \cos, \sin\} \tag{7}$$

$$\mathcal{T} = \{t, \mathcal{R}\}. \tag{8}$$

The function set $\mathcal{F}$ contains a variety of real-valued functions to allow the algorithm the consideration of a large variety of annealing schedules. We used addition (+), subtraction (−), multiplication (*), division (%), power (^), sign (sign), cosine (cos) and sine (sin). The terminal set consists of the independent variable t and the terminal $\mathcal{R}$. $\mathcal{R}$ is used to generate real-valued constants. When it is selected, a randomly generated number is inserted at the corresponding location. We use the range $[-10.0, +10.0]$ for $\mathcal{R}$.

The initial population is produced by randomly creating an S-expression for each individual. First, an element of $\mathcal{F}$ is chosen as the root node. Then, an arc is attached for each of the required arguments. Fig. 1 shows this step for the multipli-

Fig. 1. Multiplication function as root.

cation function. Next, one element of $\mathcal{F} \cup \mathcal{T}$ is selected for each of the arcs. If a function is chosen, arcs are attached to the node, one for each argument. A terminal terminates the branch. Fig. 2 shows the tree that would be created if first the multiplication and power function are chosen, and subsequently the terminals 0.9, t and 10.

Since the root of the tree is always selected from the function set $\mathcal{F}$, it takes at least one argument, and hence the minimum depth of a tree is 2. We chose a maximum depth of MD = 6 for individuals in the first generation.

For the generation of the initial population we use "ramped half-and-half". This method was proposed by Koza [24] to produce a wide variety of trees of various sizes and shapes. Ramped half-and-half creates an equal number of trees for each depth between 2 and MD. In our implementation, 20% of the trees have depth $D = 2$, 20% have depth $D = 3, \ldots, 20\%$ have depth $D = 6$. Then, we generate for each depth 50% of the individuals via the full method and 50% of the trees via the "growth" method. The "full" method generates trees where every non backtracking path from the root to a terminal has exactly a specified length D. The "growth" method generates trees with a maximum depth of D. Duplicates are avoided in the initial population to ensure a larger diversity.

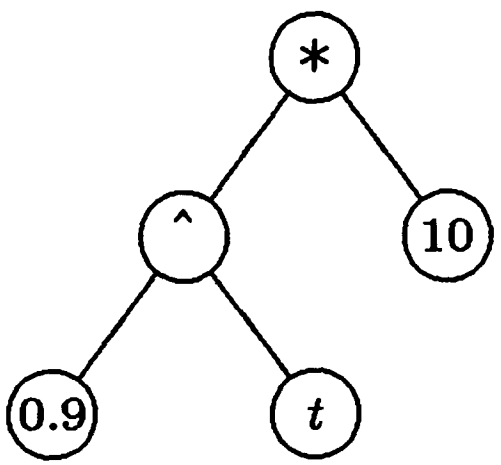

Fig. 2. S-expression `(* (^ 0.9 t) 10)` represented as a tree.

4.2. Fitness

Individuals are selected for genetic operations according to their fitness. To model the "survival of the fittest," good individuals, i.e., annealing schedules with which the Simulated Annealing algorithm performs well, are chosen with a higher probability. As suggested by Koza [24] we use the adjusted fitness $\mathrm{AF}(I)$ in our algorithm to determine this probability:

$$\mathrm{AF}(I) = \frac{1}{1 + \mathrm{SA}(I) - \mathrm{LB}}. \tag{9}$$

$\mathrm{SA}(I)$ is the objective function value of the Simulated Annealing algorithm when individual I is used as an annealing schedule. LB denotes a lower bound on the objective. It can be computed by the method described in Lawler [25]. Since $\mathrm{SA}(I) - \mathrm{LB}$ is non-negative, the adjusted fitness lies between 0 and 1. The adjusted fitness has the benefit of exaggerating the importance of small differences in the value of the standardized fitness as the standardized fitness approaches 0. Thus, as the population improves, greater emphasis is placed on the small differences that distinguish good individuals from very good ones [24].

We use fitness-proportionate selection in our algorithm and the normalized fitness $\mathrm{NF}(I)$ is used to determine the probability of involving an individual in reproduction and crossover:

$$\mathrm{NF}(I) = \frac{\mathrm{AF}(I)}{\sum_{I'} \mathrm{AF}(I')}. \tag{10}$$

4.3. Genetic operators

To produce the individuals of the next generation we apply genetic operators to the individuals of the current generation. We only use the basic genetic operations reproduction and crossover.

In conventional Genetic Algorithms operating on strings, mutation is used in addition to reproduction and crossover to ensure diversity of the population. In conventional Genetic Algorithms, it is common for a particular symbol to disappear at a particular position of a chromosome string at an early stage of the run when it is associated with inferior performance. However, due to the non-linear nature of the underlying problem the extinct symbol is often needed later in the run to optimize performance and the mutation operation is used to make the symbol occasionally available. In Genetic Programming, functions and terminals are not associated with fixed positions and only a small number of different functions and terminals is used. Hence, it is very unlikely that a symbol becomes extinct and thus the mutation operation is not needed to restore lost diversity. In conventional Genetic Algorithms, point mutation is used to introduce random changes in the population. In Genetic Programming, point mutation is an inherent part of the crossover operation as we will discuss at the end of this section and hence the mutation operator need not to be implemented as a separate operation. (See Koza [24], Section 6.5.1, for a comprehensive discussion of the relevance of mutation in Genetic Programming.)

The individuals that are involved in genetic operations are selected from the current population with a probability given by (10). A selected individual is not removed from the current population, i.e., the selection is performed with replacement, and hence a given individual can be selected more than once.

The *reproduction operation* for Genetic Programming ensures that good individuals remain in the population. It selects an individual from the current generation and copies it, without alteration, into the next population.

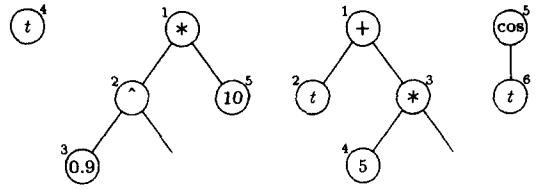

Fig. 4. Crossover fragments and remainders.

The *crossover operation* for Genetic Programming creates variation in the population by producing new offsprings that consist of parts of both parents. First, two individuals (the parents) are selected according to (10) and their nodes are labeled. Fig. 3 shows two sample trees. Then, a crossover point is selected for the first parent by generating a random number between 1 and the number of nodes. The corresponding node becomes the root of a subtree, which we will refer to as the crossover fragment. Then, we generate a second crossover fragment from the second tree. Fig. 4 shows the fragmented subtrees and the remainders when crossover points 4 and 5 are selected. To obtain two offsprings (individuals for the next generation), we attach the crossover fragment of the first tree to the remainder of the second tree and vice versa. Fig. 5 shows the resulting individuals.

Note that point mutation is an inherent part of the crossover operation. If two terminals are selected as crossover points, they are simply swapped from tree to tree and the crossover operation is in this case akin to a point mutation. Since the terminal set is small, most of its elements are present in the population and ran-

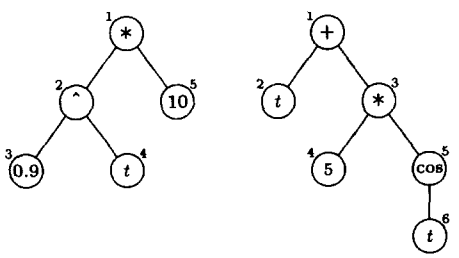

Fig. 3. Individuals selected for crossover.

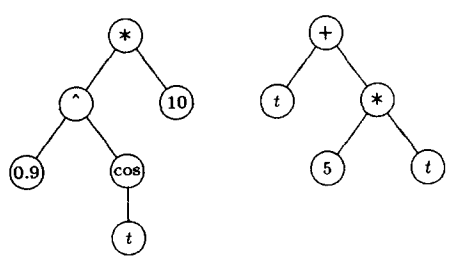

Fig. 5. Offsprings generated by crossover.

domly selecting a terminal from a randomly selected individual is similar to randomly generating a terminal [24].

To avoid that large amounts of computer time are used by some large individuals, we limit the maximum depth of the tree to 17. If an offspring would have a depth of more than 17, we would replace it by one of its parents.

Reproduction is performed on 10% of the population and 90% are created by crossover. If we would choose each node in the tree as a crossover point with the same probability, it would be rather likely that we would pick a terminal, since a typical tree consists of almost as many terminals as functions. Hence we select 20% of the crossover points out of $\mathcal{F} \cup \mathcal{T}$ and 80% out of $\mathcal{F}$.

The control parameter values used in the Genetic Programming algorithm were chosen according to Koza [24], who has successfully solved a large variety of test problems with these values.

5. TB1: Simulated Annealing algorithm with Genetic Programming as meta algorithm

The algorithm TB1 uses a Simulated Annealing algorithm that solves the QAP and a Genetic Programming algorithm that optimizes the annealing schedule. Each generation consists of 500 annealing schedules that are evaluated based on the objective function of the Simulated Annealing algorithm. The first generation is created randomly. Then, we use the genetic operators described in the previous section to produce the next generation of individuals. After 50 generations we report the best solution found. The flowchart of the Genetic Programming algorithm is shown in Fig. 6 and the flowchart of the Simulated Annealing algorithm is presented in Fig. 7.

Before the actual Simulated Annealing algorithm is applied, the problem data are normalized. This is done by dividing the b_{ik} (d_{jl}) by a normalizing constant, such that the largest b_{ik} (d_{jl}) has a value of 10. By normalizing the b_{ik} and d_{jl} we eliminate the dependency of the annealing schedule on the absolute values of the problem data.

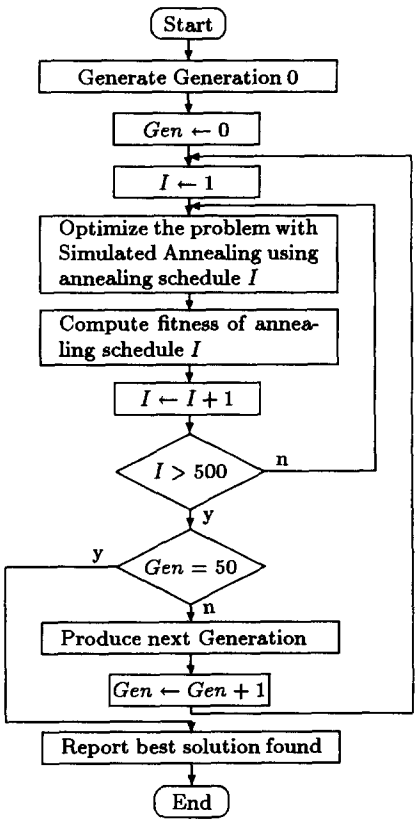

Fig. 6. Flowchart of Genetic Programming part of TB1.

As pointed out by several authors [4,8,34,35] the solution quality of a Simulated Annealing algorithm depends on the number of pairwise exchanges investigated. When more pairwise exchanges are investigated the solution quality increases. We fix the number of attempted pairwise exchanges (number of attempted pairwise exchanges = number of accepted pairwise exchanges + number of rejected pairwise exchanges) in TB1 at $SL(N) = C \cdot N(N-1)/2$, i.e., we investigate C neighborhoods. In our experiments we chose $C = 50$, an annealing schedule length often used in the literature (e.g., [8]).

To compare annealing schedules for problems of different sizes and to use one annealing schedule for the optimization of different problems, we fix the annealing schedule length at 500 and vary the number of attempted pairwise exchanges per temperature. We investigate a total of $SL(N)$

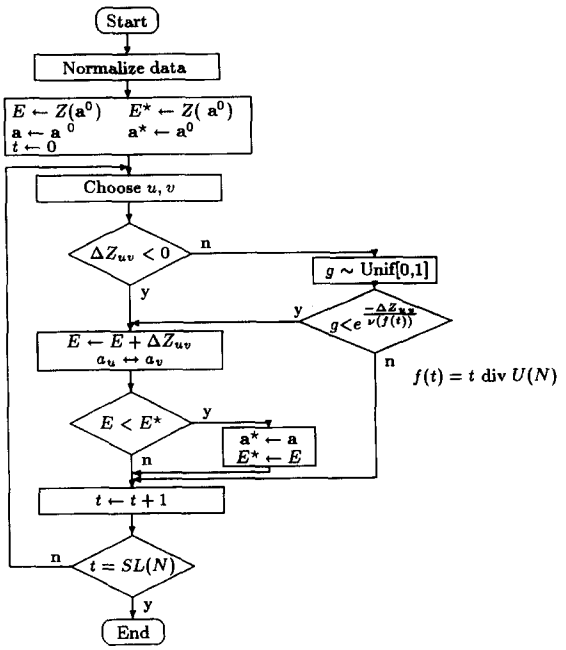

Fig. 7. Flowchart of Simulated Annealing part of TB1.

pairwise exchanges and hence have to use each temperature for $U(N) = \mathrm{SL}(N)/500$ iterations. The exchange partners are selected by sequential neighborhood search.

TB1 evaluates a total of 51 generations with 500 individuals each. Hence, 25,500 Simulated Annealing runs are executed for each run of TB1. This limits TB1's application to problems of moderate size. We applied the algorithm to five problems: H12, NVR20, NVR30, TB40 and WW50. The problems are described in Section 6 and our computational results are presented in Table 1. We report the percentage the solution is above the best known solution, i.e., the error, the run time in seconds on a Sun SPARC2 and the generation in which the best solution was found. The column "Annealing Schedule" refers to the figure that shows the annealing schedule which performed best. Oftentimes annealing schedules with a constant temperature performed well, but TB1 always found an annealing schedule with a non-constant temperature that performed at least as well. When an oscillating annealing schedule is used, the algorithm can escape local minima at high temperatures and will accept progressively fewer uphill moves as the temperature decreases. At low temperature it will finally get stuck in a local minimum. When the temperature increases, the algorithm can escape the local optimum again.

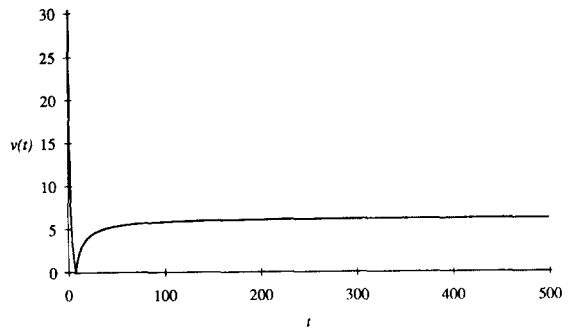

Fig. 8. Best annealing schedule found for NVR20.

TB1 found the best known solution for the four smaller problems and a solution that was only 0.02% above the best known solution for the largest problem. However, the implementation we presented consumed excessive amounts of CPU time which limits its application to problems of moderate size.

5.1. Properties of "good" annealing schedules

The annealing schedules presented in Figs. 8 to 11 are tailored to a given problem. When they were applied to other problems, they performed poorly. To obtain annealing schedules that gener-

Table 1
TB1: Experimental results

Problem	Error [%]	Time [s]	Generation	Annealing schedule
H12	0.00	4898	1	Constant 8.35
NVR20	0.00	15385	3	Fig. 8
NVR30	0.00	35250	4	Fig. 9
TB40	0.00	78737	7	Fig. 10
WW50	0.02	149626	26	Fig. 11

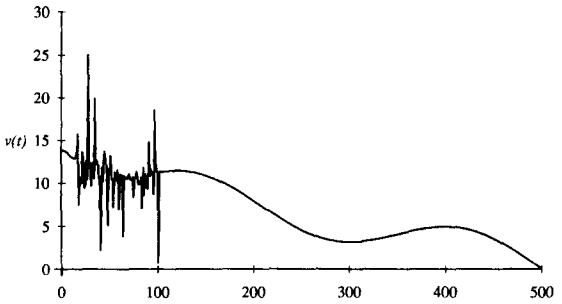

Fig. 9. Best annealing schedule found for NVR30.

alize well, we conducted another experiment. We used *one* annealing schedule to optimize H12, NVR20 and NVR30. Each QAP was solved with two different initial assignments and with each of these initial assignments it was solved twice, using different random number streams. Hence, each annealing schedule (individual) was used for $3 \cdot 2 \cdot 2 = 12$ different optimizations.

The average absolute deviation of the Simulated Annealing solution from the optimal solution increases when the problem size increases. Therefore, we divided the absolute deviation by the value of the best known solution to obtain an estimate for relative error. Then, we summed up these twelve estimates to obtain the standardized fitness. The annealing schedules that were generated by this approach had quite different shapes. However, good schedules had the following properties in common:

1. Most of the time the temperature was between 4 and 8.
2. The temperature was usually not constant or monotone decreasing, but oscillated between 4 and 8.

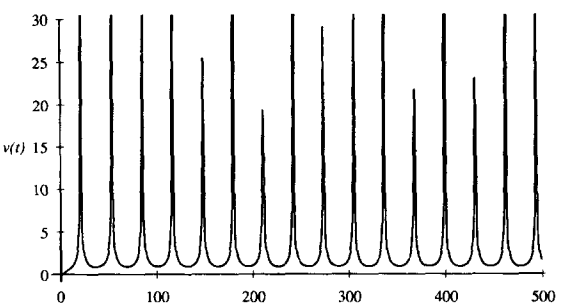

Fig. 10. Best annealing schedule found for WW50.

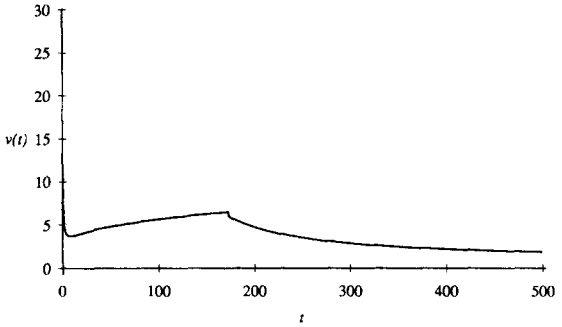

Fig. 11. Flowchart of Simulated Annealing algorithm TB2.

3. The period length of the oscillation was between 20 and 100 (based on a schedule length of 500).
4. The shape of the oscillation was not relevant. Good results were achieved with different shapes, like sine and rectangular.
5. The temperature at the beginning and at the end of the schedule was far above zero.

We use these properties of good annealing schedules in the algorithm described in the following section.

6. TB2: Simulated Annealing algorithm with improved annealing schedule

This section describes an improved Simulated Annealing algorithm for the QAP. The improvement could be achieved by implementing a modified annealing schedule. Fig. 12 shows the flowchart of the algorithm.

Before starting the actual Simulated Annealing algorithm we normalize the problem data as described earlier. Based on experiments with different starting temperatures, $v(0) = 10$ was chosen. Then, we decrease the temperature after each attempted pairwise exchange according to (6). We use an annealing schedule length of $SL(N) = C \cdot N(N-1)/2$ and search the neighborhood sequentially. The parameter β in (6) is determined by

$$\beta = \frac{10 - 2}{2 \cdot 10 \cdot SL(N)}. \tag{11}$$

With this value of β the temperature would be 2 after SL(N) attempted pairwise exchanges. However, this low temperature will usually not be reached. When SC(N) = $N(N-1)/4$ (i.e., half a neighborhood) consecutive pairwise exchanges are rejected, cooling is stopped. We denote this temperature by ν_c and the corresponding iteration t_c. ν_c is an indicator that a temperature has been reached where further improvements are unlikely. After cooling is stopped, we use

$$\nu(t) = \nu_c + \tfrac{1}{2}\nu_c \cos(\omega(t - t_c)) \tag{12}$$

to determine the temperature, where

$$\omega = \frac{16\pi}{25 \cdot N(N-1)}. \tag{13}$$

Eq. (12) generates an annealing schedule that oscillates around ν_c with an amplitude of $0.5\nu_c$. The value of ω ensures that the annealing schedule contains at most $C/6.25$ full periods. We chose $t - t_c$ instead of t as an argument for the cosine to warm the system up immediately after cooling is stopped. Thus, we can escape local optima faster. A sample schedule is presented in Fig. 13.

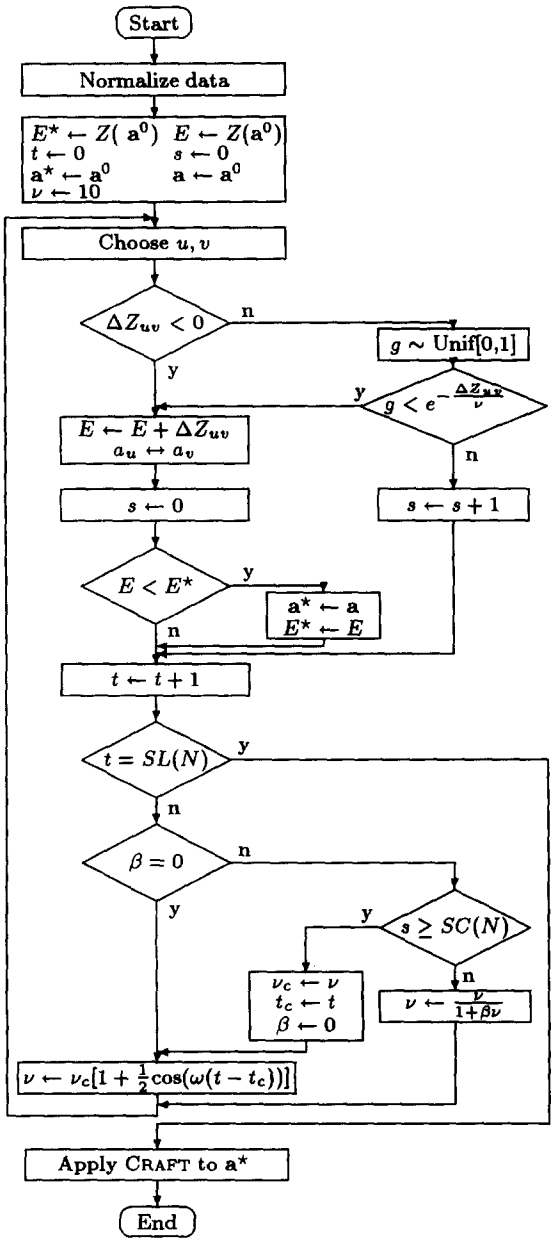

Fig. 12. Sample annealing schedule for TB2.

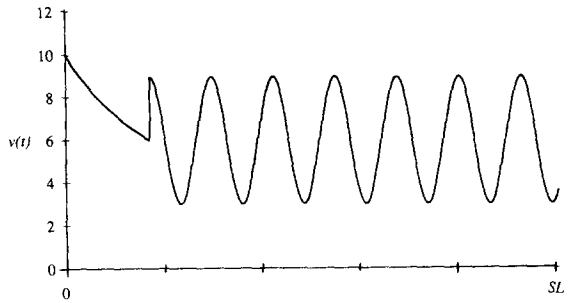

Fig. 13. Best annealing schedule found for TB40.

After the Simulated Annealing optimization is completed, we apply a downhill search on the best solution found. We use CRAFT [2] to perform this task.

6.1. Computational results

The performance evaluation of TB2 is based on twelve QAPs from the literature and three randomly generated QAPs. The problem H12 was introduced by Hillier [18], NVR20 and NVR30 were generated by Nugent et al. [29] and WW50 and WW100 were used by Wilhelm and Ward [38]. The problems with prefix SK were taken from Skorin-Kapov [32]. All problems have a structure that allows the decomposition accord-

Table 2
Properties of problem data

Problem	Size (N)	Max d_{jl}	Max b_{ik}	Flow dominance	Best known solution
H12	12	5	10	108%	289
NVR20	20	7	10	99%	1,285
NVR30	30	9	10	109%	3,062
TB30	30	11	250	134%	74,968
TB40	40	11	250	153%	120,258
SK42	42	11	10	108%	7,906
SK49	49	12	10	109%	11,693
WW50	50	13	9	53%	24,408
SK56	56	13	10	110%	17,229
SK64	64	14	10	108%	24,249
SK72	72	15	10	107%	33,128
SK81	81	16	10	107%	45,504
SK90	90	17	10	107%	57,767
WW100	100	18	9	63%	136,522
TB150	150	23	250	147%	4,067,028

Table 3
TB2: Computational results with $C = 50$

Problem	Error [%]	σ	$\text{opt}_{1\%}$ [%]	Time per run [s]
H12	0.30	1.5	84	0.16
NVR20	0.27	4.9	96	0.58
NVR30	0.40	8.9	100	1.63
TB30	0.95	451.4	64	1.61
TB40	1.02	525.7	40	3.52
SK42	0.41	12.8	100	3.98
SK49	0.44	23.2	100	6.29
WW50	0.20	19.8	100	6.76
SK56	0.58	39.6	100	9.36
SK64	0.28	37.4	100	13.72
SK72	0.61	90.2	90	19.53
SK81	0.47	89.8	100	27.31
SK90	0.46	97.6	100	38.08
WW100	0.33	84.5	100	52.42
TB150	0.63	12420.2	90	176.52

ing to (1). Vollmann and Buffa [36] suggest to use the flow dominance, the quotient of standard deviation and mean, of the b_{ik} as an indicator for the difficulty of a QAP. The properties of the QAPs are summarized in Table 2. Note that the newly generated QAPs (TB30, TB40, TB150) have a flow dominance that is considerably higher than the flow dominance of the problems from the literature. [1]

[1] The data of the new problems are available by contacting the authors.

The performance of TB2 was evaluated by comparing its solutions with those obtained by Connolly's [8] Simulated Annealing algorithm. Connolly's algorithm was chosen since it is the most powerful Simulated Annealing algorithm for the QAP [34]. We investigated the solution quality from a practitioners point of view who will apply a stochastic algorithm several times to the same problem and then choose the best result of several runs as the solution. In the experiments reported, we optimized each problem three times with the same initial assignment but different random number streams and then chose the best

Table 4
Connolly's Simulated Annealing algorithm: Computational results

Problem	Error [%]	σ	$opt_{1\%}$ [%]	Time per run [s]
H12	0.75	2.0	48	0.11
NVR20	0.37	5.1	88	0.44
NVR30	0.75	12.8	72	1.33
TB30	1.08	533.9	56	1.32
TB40	1.63	796.1	20	3.04
SK42	0.64	17.5	100	3.55
SK49	0.58	27.9	100	5.83
WW50	0.21	20.0	100	6.29
SK56	0.64	47.0	100	8.92
SK64	0.60	49.8	100	13.53
SK72	0.64	98.5	90	19.38
SK81	0.52	91.3	100	27.92
SK90	0.50	107.3	100	38.72
WW100	0.31	123.1	100	53.47
TB150	1.29	19715.3	30	179.04

out of the three runs as the solution. We define a run as a single application of the algorithm to a problem and a solution as the best result that was obtained in three runs.

The performance measures used are the average deviation from the best known solution (error), the standard deviation of the solution's objective function value and the percentage of solutions that are within 1% optimality. To compare TB2 with Connolly's algorithm we also conducted a sign test [31]. For problems of size $N \leq 30$ we generated 25 solutions (75 runs) and for problems of size $N > 30$ only 10 solutions (30 runs). The results are presented in Tables 3 and 4.

TB2 outperformed Connolly's algorithm with respect to all performance measures. The average deviation of the solution from the best known solution was less, the standard deviation of the solutions was smaller, and TB2 found more solutions that were within 1% optimality. The only exception was the average deviation from the best known solution for problem WW100, where Connolly's performed slightly better.

The computational results indicate that prob-

Table 5
TB2: Computational results with $C = 250$

Problem	Error [%]	σ	$opt_{1\%}$ [%]	Time per run [s]
H12	0.00	0.0	100	0.80
NVR20	0.03	1.5	100	2.93
NVR30	0.18	6.1	100	8.26
TB30	0.26	167.1	100	8.21
TB40	0.49	420.7	90	17.66
SK42	0.13	10.9	100	20.15
SK49	0.14	17.3	100	32.20
WW50	0.07	6.2	100	33.89
SK56	0.14	17.3	100	47.68
SK64	0.12	21.6	100	69.11
SK72	0.27	38.4	100	96.37
SK81	0.19	41.7	100	135.77
SK90	0.18	46.2	100	186.81
WW100	0.21	95.4	100	252.72
TB150	0.23	8883.14	100	827.60

Table 6
TB2: Computational results with $C = 1000$

Problem	Error [%]	σ	$opt_{1\%}$ [%]	Time per run [s]
H12	0.00	0.0	100	3.21
NVR20	0.01	0.9	100	11.72
NVR30	0.02	2.4	100	32.20
TB30	0.08	120.6	100	32.00
TB40	0.19	166.0	100	68.84
SK42	0.03	4.2	100	78.54
SK49	0.11	5.5	100	125.52
WW50	0.04	7.1	100	132.34
SK56	0.03	3.7	100	183.86
SK64	0.03	11.0	100	269.40
SK72	0.14	35.2	100	375.67
SK81	0.06	16.1	100	529.25
SK90	0.14	35.2	100	728.23
WW100	0.08	77.2	100	985.12
TB150	0.10	2464.6	100	3226.34

lems with a high flow dominance (TB30, TB40, TB150) are more difficult to solve. TB2 found for these problems solutions that were much better than those obtained by Connolly's algorithm. Problems with small flow dominance (WW50, WW100) are solved efficiently by both algorithms. We conclude that TB2 performs well on large variety of problems and finds solutions that are on average very close to the best known solutions.

The run times of the algorithms differed only slightly. Both algorithms use the same number of attempted pairwise exchanges. The run-time differences are due to the more complicated annealing schedules computations in TB2, the problem data normalization in TB2 and the post optimization, i.e., the application of CRAFT [2] to the solution found by the Simulated Annealing algorithm.

We also conducted a sign test to compare the two algorithms. The test was based on 210 solutions. The null hypothesis was that Connolly's algorithm produces better solutions than TB2, the alternative hypothesis was that TB2 produces better solutions than Connolly's algorithm. The null hypothesis was rejected on a confidence level of 99%. Hence, for a fixed number of attempted pairwise exchanges, TB2 clearly outperforms the Simulated Annealing algorithm by Connolly. This improvement is due to the modified annealing schedule.

We can improve the solution quality of TB2 by increasing the value of the parameter C, but at the cost of a longer run time. Tables 5 and 6 show the results for $C = 250$ and $C = 1000$.

The solution quality improves as C increases. For $C = 250$ ($C = 1000$) we obtained solutions with an error of less than 0.5% (0.2%). Only one (none) of the solutions was not within 1% optimality. As expected, the run time increased proportionally to C.

7. Conclusion

This work applied an "intelligent" search routine to optimize the annealing schedule for the QAP. The first algorithm presented (TB1) used Genetic Programming as a meta algorithm for Simulated Annealing. For a given problem TB1 optimizes the annealing schedule and finds solutions that are less than 0.02% above the best known solutions. However, TB1's run time limits its application to problems of moderate size. Based on our computational experience with TB1 we developed the Simulated Annealing algorithm TB2. The novelty of TB2 is an improved annealing schedule that is characterized by an oscillating component. Our computational experiments showed that TB2 outperforms the best known Simulated Annealing algorithms. TB2 found very

good solutions with moderate amounts of computation time.

To the best knowledge of the authors this is the first work that used an intelligent search algorithm to find the optimal shape of an annealing schedule. Further research has to investigate if the oscillating annealing schedules that we found are optimal. Our serial implementation of Genetic Programming required excessive amounts of computation time, and hence only a limited number of experiments could be conducted. A parallel implementation of the algorithm would result in a speedup that would allow a more thorough investigation.

Another interesting topic for future research is the optimization of annealing schedules for other combinatorial optimization problems, like the traveling salesman problem or the graph-partitioning problem.

Acknowledgments

The major part of this research was conducted at the University of Paderborn, Germany. The authors would like to thank Prof. Dr. Otto Rosenberg and Dipl.-Kfm. Arnd Mollemeier of the University of Paderborn for their support and their critical comments. They also would like to thank Prof. John R. Koza of Stanford University for his encouragement and the anonymous referees for their helpful comments.

References

[1] Anily, S., and Federgruen, A., "Simulated annealing methods with general acceptance probability", *Journal of Applied Probability* 24 (1987) 657–667.

[2] Armour, G.C., and Buffa, E.S., "A heuristic algorithm and simulation approach to the relative location of facilities", *Management Science* 9/2 (1963) 294–304.

[3] Bazaraa, M.S., and Sherali, M.D., "Bender's partitioning scheme applied to a new formulation of the Quadratic Assignment Problem", *Naval Research Logistics Quarterly* 27/1 (1980) 29–41.

[4] Bölte, A., *Modelle und Verfahren zur innerbetrieblichen Standortplanung*, Physica-Verlag, Heidelberg, 1994.

[5] Burkard, R.E., "Quadratic Assignment Problems", *European Journal of Operational Research* 15 (1984) 283–289.

[6] Burkard, R.E., and Offermann, J., "Entwurf von Schreibmaschinentastaturen mittels Quadratischer Zuordnungsprobleme", *Zeitschrift für Operations Research* 21 (1977) 121–132.

[7] Burkard, R.E., and Rendl, F., "A thermodynamically motivated simulation procedure for combinatorial optimization procedures". *European Journal of Operational Research* 17 (1984) 169–174.

[8] Connolly, D.T., "An improved annealing scheme for the QAP", *European Journal of Operational Research* 46 (1990) 93–100.

[9] Davis, L., *Genetic Algorithms and Simulated Annealing, Research Notes in Artificial Intelligence*, Pitman, 1987.

[10] Davis, L., and Ritter, F., "Schedule optimization with probabilistic search", *IEEE* (1987) 231–235.

[11] Dutta, A., and Koehler, G., "A Whinston: On optimal allocation in a distributed processing environment", *Management Science* 28 (1982) 839–853.

[12] Gavett, J.W., and Plyter, N.V., "The optimal assignment of facilities to locations by branch and bound", *Operations Research* 14/2 (1966) 210–232.

[13] Geman, S., and Geman, D., "Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images",*IEEE Transactions on Pattern Analysis and Machine Intelligence* 6/5 (1984) 721–741.

[14] Gilmore, P.C., "Optimal and suboptimal algorithms for the Quadratic Assignment Problem", *SIAM Journal* 10/2 (1962) 305–313.

[15] Glover, F., "Tabu Search, Part I", *ORSA Journal on Computing* 1 (1989) 190–206.

[16] Glover, F., "Tabu Search, Part II", *ORSA Journal on Computing* 2 (1990) 4–32.

[17] Golden, B.L., and Skiscim, C.C., "Using Simulated Annealing to solve routing and location problems", *Naval Research Logistics Quarterly* 33/2 (1986) 261–279.

[18] Hillier, F.S., "Quantitative tools for plant layout analysis", *Journal of Industrial Engineering* 14/1 (1963) 33–40.

[19] Hillier, F.S., and Connors, M.M., "Quadratic Assignment Problem algorithms and the location of indivisible facilities", *Management Science* 13 (1966) 42–57.

[20] Holland, J.H., *Adoption in Natural and Artificial Systems*, University of Michigan Press, 1975.

[21] Huntley, C.L., and Brown, D.E., "A parallel heuristic for Quadratic Assignment Problems", *Computers and Operations Research* 18/3 (1991) 275–289.

[22] Kirkpatrick, S., Gelatti, C.D., and Vecchi, M.P., "Optimization by Simulated Annealing", *Science* 220 (1983) 671–680.

[23] Koopmans, T.C., and Beckman, M.J., "Assignment problems and the location of economic activities", *Econometrica* 25/1 (1957) 53–76.

[24] Koza, J.R., *Genetic Programming: On the Programming of Computers by Means of Natural Selection and Genetics*, MIT Press, Cambridge, MA, 1993.

[25] Lawler, E.L., "The Quadratic Assignment Problem", *Management Science* 9/4 (1963) 586–599.

[26] Lee, R.C., and Moore, J.M., "CORELAP – Computerized Relationship Layout Planning", *Journal of Industrial Engineering* 18/3 (1967) 195–200.

[27] Lundy, M., and Mess, A., "Convergence of an annealing algorithm", *Mathematical Programming* 34 (1986) 111–124.

[28] Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E., "Equation of state calculation by fast computing machines", *Journal of Chemical Physics* 21 (1953) 1087–1092.

[29] Nugent, C.E., Vollmann, T.E., and Ruml, J., "An experimental comparison of techniques for the assignment of facilities to locations", *Operations Research* 16/1 (1968) 150–173.

[30] Sahni, S., and Gonzalez, T., "P-complete approximation problems", *Journal of the Association for Computing Machinery* 23 (1976) 555–565.

[31] Schwarze, J., *Grundlagen der Statistik: Wahrscheinlichkeitsrechnung und induktive Statistik*, Verlag Neue Wirtschafts-Briefe, 1986.

[32] Skorin-Kapov, J., "Tabu Search applied to the Quadratic Assignment Problem", *ORSA Journal on Computing* 2/1 (1990) 33–45.

[33] Steinberg, L., "The backboard wiring problem: A placement algorithm," *SIAM Review* 3 (1961) 37–50.

[34] Thonemann, U.W., *Verbesserung des Simulated Annealing unter Anwendung Genetischer Programmierung am Beispiel des Diskreten Quadratischen Layoutproblems*, Master's Thesis, Lehrstuhl für Betriebswirtschaftslehre, insbes. Produktionswirtschaft, Universität Paderborn, Germany, November 1992.

[35] Thonemann, U.W., "Finding improved annealing schedules with Genetic Programming," *Proceedings of The First IEEE Conference on Evolutionary Computation*, 1994, 391–395.

[36] Vollmann, T., and Buffa, E., "The facilities layout problem in perspective," *Management Science* 12 (1966) 450–468.

[37] Wilensky, R., *Common LISPcraft*, Norton, 1986.

[38] Wilhelm, M.R., and Ward, T.L., "Solving Quadratic Assignment Problems by Simulated Annealing," *IIE Transactions* 19/1 (1987) 107–119.