JOURNAL OF COMPUTATIONAL PHYSICS 90, 161-175 (1990)

Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing

GUNTER DUECK AND TOBIAS SCHEUER

IBM Scientific Center, Heidelberg, West Germany

Received March 24, 1989; revised September 19, 1989

A new general purpose algorithm for the solution of combinatorial optimization problems is presented. The new threshold accepting method is even simpler structured than the well-known simulated annealing approach. The power of the new algorithm is demonstrated by computational results concerning the traveling salesman problem and the problem of the construction of error-correcting codes. Moreover, deterministic (!) versions of the new heuristic turn out to perform nearly equally well, consuming only a fraction of the computing time of the stochastic versions. As an example, the deterministic threshold accepting method yields very-near-to-optimum tours for the famous 442-cities traveling salesman problem of Grötschel within 1 to 2s of CPU time. © 1990 Academic Press, Inc.

INTRODUCTION

Simulated annealing (SA) is a stochastic optimization algorithm, which borrows deep ideas from statistical physics. It was invented in a pioneering paper by Kirkpatrick, Gelatt, and Vecchi [1] who applied it to VLSI layout and graph partitioning. Since then it has become a popular general purpose tool for a wide class of combinatorial optimization problems. In these problems one wants to find among many configurations the one which minimizes a certain "quality function." To accomplish this task Kirkpatrick *et al.* introduced the concept of "annealing" and combined it with the well-known Monte-Carlo algorithm by Metropolis *et al.* [2], which originally was used to numerically perform averages over large systems from statistical mechanics. The idea of SA runs as follows. First we choose an initial configuration. Then each step of the SA algorithm consists of a slight change of the old configuration into a new one. The "qualities" of the two configurations are compared. If the new configuration is better than it serves as the "old" configuration for the next step. If it is worse then it is accepted only with a certain probability as the current configuration for the next step. This probability depends on a time-dependent parameter called temperature and on the decrease in quality. The probability that a worse configuration is accepted is slowly lowered during the running time.

We make these explanations more precise:

SA ALGORITHM FOR MAXIMIZATION.

```
choose an initial configuration
choose an initial temperature $T>0$
*Opt*: choose a new configuration which is a stochastic small
  perturbation of the old configuration
  compute $\Delta E$ := quality(new configuration)-quality(old configuration)
  IF $\Delta E>0$
    THEN old configuration := new configuration
    ELSE with probability exp($\delta E/T$)
                old configuration := new configuration
  IF a long time no increase in quality or too many iterations
    THEN lower temperature $T$
  IF some time no change in quality anymore
    THEN stop
GOTO Opt
```

Note that if the temperature is high, very often worse configurations are accepted. It is a kind of art to choose a successful *annealing schedule*, that is, a rule for lowering the temperature in the algorithm. In most applications the success of the algorithm is very sensitive against the choice of the annealing schedule.

We briefly explain the parameters above within the framework of the traveling salesman problem (TSP). Here, the task is to find a minimum length tour through a given number of "cities." As an initial configuration, one chooses a random tour through all of the cities. A new configuration is obtained by small changes in the tour. The quality of a tour is given by its length.

The method called *threshold accepting* (TA) which we want to study here, is formally very similar to SA.

TA ALGORITHM FOR MAXIMIZATION.

```
choose an initial configuration
choose an initial THRESHOLD $T>0$
Opt: choose a new configuration which is a stochastic small
  perturbation of the old configuration
  compute $\Delta E$ := quality(new configuration)-quality(old configuration)
  IF $\Delta E> -T$
    THEN old configuration := new configuration
  IF a long time no increase in quality or too many iterations
    THEN lower THRESHOLD $T$
  IF some time no change in quality anymore
    THEN stop
GOTO Opt
```

The essential difference between SA and TA consists of the different acceptance rules. TA accepts *every* new configuration which is *not much worse* than the old one (SA accepts worse solutions only with rather small probabilities).

An apparent advantage of TA is its greater simplicity. It is not necessary to compute probabilities or to make random decisions. Moreover we claim:

> *TA yields better results than SA* (even in a considerable smaller amount of time respectively in a smaller amount of "new configuration choice steps.")

Until now we have studied TA for the TSP, for finding good error-correcting codes, and for the minimization of spin-glass Hamiltonians. Originally, we were working with Andreas Dress on spin-glass Hamiltonians during his stay in the IBM Scientific Center, Heidelberg. During this research, we found the new heuristic TA and observed its superiority to SA.

For a fair comparison of these methods, however, we looked for very elaborated results found by SA. A very challenging large problem is the 442-cities problem of Grötschel [3]. He found a good solution of tour length 51.45. In [4], Rossier *et al.* found a new record-breaking solution by a sophisticated SA algorithm with tour length 51.42. Mühlenbein *et al.* [5] used a 24-processor parallel computer to obtain in many hours of CPU time a solution with length 51.21. Independently, the problem was solved by Holland [6] in his dissertation using the methods of Padberg and Rinaldi [7]. The optimum tour length for the Grötschel problem is therefore known to be 50.80 (in real *8, 50.69 in integer distances).

Rossier's *et al.* paper does not only contain the results of their computations, but also the number of steps used in the SA approach. Thus, is was possible for us to compare SA and TA under really equal conditions. We used TA with the same number of steps as with SA in [4], and we can see the differences in the quality of the results. It turns out, that TA is overwhelmingly superior to SA. With the same number of steps we found that the TA solutions are *in the average much better* than the best SA solutions ever obtained. Especially, we found within 50 trials of our TA algorithm two tours below 51.00, namely 50.97 and 50.95, which are displayed below. We are disappointed that usually in the literature the authors publish only their best value obtained by an algorithm. This makes it nearly impossible to compare *methods*. However, for TA, the difference is so large, that we can demonstrate the superiority of TA in any case.

We report in the next section our results for different sizes of the TA method in detail (tour length of every run, mean, and variance).

A problem that is much harder than the TSP problem is that of finding good error-correcting codes. In [8] El Gamal *et al.* found excellent codes using the SA methods. They also give rough data concerning the running times of their algorithms. We wrote a quick application also of TA in this field and report our results here. We have enough results to demonstrate again the superiority of TA. We did not try to tune our algorithms in the best possible fashion here and we used only trivial lowering threshold schemes. Since in the case of finding codes the quality function has a very noncontinuous behavior (in contrast to the TSP), finding good codes needs more tedious work on the algorithm and, very probably, much more computational effort.

Originally, we gained our first experience with TA applied to the minimization of spin-glass Hamiltonians. In this case we also compared SA and TA and recognized the superiority of TA. However, we have no really fair comparable results obtained by SA. Hence, we report only our results on the TSP and for the construction of codes.

We turn now to another phenomenon which occurred during our experiments with Grötschel's TSP problem. We used TA with LIN-2-OPT configuration changes (definition below). For LIN-2-OPT one usually selects two cities at random and forms a new LIN-2-OPT configuration based on these two cities. After having written up our results on TA, we designed a *deterministic* version of TA. We did not select two cities *at random*, but we made a LIN-2-OPT exchange of pairs of cities in a prescribed order. The resulting TA method is completely deterministic. Originally we had no hope of obtaining good results because we shared the common opinion that deterministic algorithms do not behave well in comparison with stochastic methods.

Surprisingly, we found that the deterministic version yields equally good results, needs fewer steps, and is faster because there is no need to generate random numbers. We tried this algorithm for numerous random initial tours and obtained excellent results. As one could expect, the variance of the solutions is higher than in the stochastic case, but still it is not too large. We give the results in the next section. All programs with which we experimented consist of less than 150 lines of FORTRAN code. The running times were about 3 min for 4,000,000 steps in the stochastic TA down to about 4s for a 442,000 step deterministic run. Here, all computation times have been measured on an IBM 3090 Mod 200 VF.

Grötschel's 442-Problem: TA Algorithms

Grötschel's 442-cities problem is a Euclidean TSP. The coordinates of 442 cities are given in the Euclidean plane. A closed polygonal tour of minimum length joining all given points (cities) is asked for. If C is the set of cities, a tour can be regarded as a permutation $\pi : C \rightarrow C$. Given a permutation π, the corresponding tour starts in $\pi(1)$, goes to $\pi(2), \ldots$, goes to $\pi(N)$, and ends in $\pi(1)$, where N is the number of cities (here $N = 442$).

TA starts with a random permutation on C as an initial tour. As in [4], we choose the LIN-2-OPT exchange as a procedure to construct a new perturbation tour from an old one. This rule has been applied successfully to Euclidean TSPs (cf. [9]).

LIN-2-OPT.

```
choose $i, j \in C, i<j$
cut in the tour the connections between the cities
        $\pi(i)$ and $\pi(i+1)$   $(\pi(N+1) := \pi(1))$
  and   $\pi(j)$ and $\pi(j+1)$   $(\pi(N+1) := \pi(1))$
insert connections between
        $\pi(i)$ and $\pi(j)$
  and   $\pi(i+1)$ and $\pi(j+1)$   $(\pi(N+1) := \pi(1))$
```

Formally, the new permutation is given by

$$\tilde{\pi}(k) = \begin{cases} \pi(k) & \text{for } k \leqslant i \quad \text{or} \quad k > j \\ \pi(i+j+1-k) & \text{for } i < k \leqslant j. \end{cases}$$

Rossier *et al.* [4] used SA with this rule for the choice of the new configurations. The standard SA approach with the best annealing schedule is named A2N1 in [4]. In another series of experiments, Rossier *et al.* tried LIN-2-OPT exchanges only for those cities $\pi(i)$, $\pi(j)$ whose distance in the plane is rather small. This is a reasonable heuristic rule, because in the LIN-2-OPT the connection between $\pi(i)$ and $\pi(j)$ is inserted, and there is a poor chance of having a good step if this connection is of great length (this "distance" approach is called A2N4 in [4]). We designed TA algorithms for these two cases. Also we did not try to design TA algorithms for the other "neighbourhood relations," because we suspect that they are only good heuristics for this specific 442-problem. In [4], the best result is 51.42 with the so-called A2N3 method. However, our A2N4 analogue algorithm for TA already performs much better.

In [4], the reported results have been obtained by 2,000,000 LIN-2-OPT trials. We designed completely analogous TA algorithms for the pure approach (randomly select two cities and try LIN-2-OPT) and for the A2N4 approach (randomly select two neighboring cities and try LIN-2-OPT).

TA STANDARD ALGORITHM.

```
choose initial random tour
thresholds:
  0.13, 0.12, 0.11, 0.10, 0.095, 0.09, 0.085, 0.08, 0.075, 0.075, 0.075, 0.07, 0.07, 0.07,
0.065,
  0.065, 0.065, 0.065, 0.06, 0.06, 0.055, 0.055, 0.05, 0.05, 0.05, 0.04, 0.04, 0.03,
0.02, 0
FOR every threshold $T = 0.13, ..., 0$
DO $n$ times
  choose $i, j$ at random
  perform LIN-2-OPT
  IF length(new tour) < length(old tour) + $T$
    THEN old tour := new tour
```

In the next algorithm we consider only such pairs of cities with distance less than 0.45.

TA DISTANCE ALGORITHM.

```
choose initial random tour
thresholds:
  0.13, 0.12, 0.11, 0.10, 0.095, 0.09, 0.085, 0.08, 0.075, 0.075, 0.075, 0.07, 0.07, 0.07,
0.065,
  0.065, 0.065, 0.065, 0.06, 0.06, 0.055, 0.055, 0.05, 0.05, 0.05, 0.04, 0.04, 0.03,
0.02, 0
FOR every threshold $T = 0.13, \ldots, 0$
DO $n$ times
  choose $i, j$ at random such that
      $distance(\pi(i), \pi(j)) < 0.45$
  perform LIN-2-OPT
  IF length(new tour) < length(old tour) + $T$
    THEN old tour := new tour
```

In the SA algorithm it is very important to provide a good and carefully chosen annealing schedule. Therefore, it is obviously of interest to analyze the sensitivity of the TA algorithm against the choice of the threshold sequence. In the very first versions, we operated with the trivial sequence .15 .14 $\cdots$ .01 0. We have the feeling (really only the feeling, not, for instance, the impression) that the threshold sequence above is somewhat better. We made some experiments to find better threshold sequences; however, the trivial sequence is essentially best.

All these experiments give the hint that TA is rather insensitive in the threshold sequence.

DETERMINISTIC TA ALGORITHMS FOR THE TSP

For the 442-problem we designed a deterministic algorithm as follows. In a preprocessing run we compute for each city the next *nb* (we used *nb* = 10) neighbours (the *nb* cities with the least distance). Then the deterministic algorithm performs a LIN-2-OPT trial for each threshold, each city, and each of the nearest neighbours of that city in a prescribed order. More precisely, we perform

DETERMINISTIC TA ALGORITHM.

```
fix a positive integer $nb$
compute for every city the nearest $nb$ neighbour cities
choose initial random tour
thresholds:
  0.099, 0.098, 0.097, ..., 0.003, 0.002, 0.001, 0
```

```
FOR every threshold $T = 0.099, ..., 0$
FOR every city $i = 1, ..., 442$
FOR every city being one of the $nb$ nearest neighbours of
  city $i$
DO perform LIN-2-OPT
  IF length(new tour) < length(old tour) + $T$
    THEN old tour := new tour
```

We point out that we use a different threshold sequence for the deterministic version. The reason is that in this version there is some danger for the algorithm to circulate without any improvement if the algorithm is run a long time with the same threshold. Thus, we preferred to run the algorithm with a larger number of thresholds and with a smaller number of steps with the same threshold. Again, our experiments showed that TA is insensitive against the threshold sequence. Hence, we kept the trivial choice of the sequence in our program.

A complete run of this algorithm performs 442 times *nb* times 100 LIN-2-OPT trials. The running time depends essentially on the parameter *nb*.

In the preceding section we made a random choice of two cities, which we examined eventually, if the distance between the two cities is small enough, trying a LIN-2-OPT exchange. This procedure is very CPU time consuming, and we organized the algorithm only in this way, because we really wanted to have the same setup as was used in [4]. In general, the "nearest neighbour approach" seems more appropriate to us.

The computational results of the deterministic version are given below.

Grötschel's 442 Problem: Computational Results

We start with the computational results on Grötschel's TSP on 442 cities which were obtained by Rossier *et al.* [4] using the SA approach. In the first row, the best result of the pure LIN-2-OPT algorithm is given (make LIN-2-OPT improvements until no further progress is possible) (Table I).

TABLE I

Results for 442-TSP by SA

Variant	Total number of iterations	length of best tour
Lin 2-Opt	until improvement impossible	57.30
SA Standard	2.000.000	53.30
SA Distance	2.000.000	51.76

Note. These data are taken from [4].

Note that in [4] only the *best* results obtained by the different methods are given. It is not stated how many runs have been performed. Mühlenbein, Gorges-Schleuter, and Krämer obtained in [5] solutions of lengths 51.24 and 51.21 with evolution optimization. These results were obtained within 2h for *any* of the 24 processors of their parallel computer. Again it is not reported how many runs had been necessary.

Now we give our results for some different numbers of iterations. The runs with 2,000,000 steps are comparable to the results above (Table II).

We observe that TA standard (2,000,000) and TA distance (2,000,000) and even TA distance (1,500,000) are *in the average better* than the analogous best results of [4]. *Only a few of our results are worse than the best analogous results obtained by SA.*

Many of our runs give better results than the previous record-breaking values of Mühlenbein, Gorges-Schleuter, and Krämer [5]. Our best value (50.95) is very near to the optimum. Finally, we present graphics for our best two solutions. The first graphic displays the optimal tour of length 50.80 found by Holland [6] (Fig. 1).

The next two tours have been found by our TA algorithms (Figs. 2 and 3).

Observe that in both near-to-optimum tours there are parts of the tour which can

TABLE II

Results for 442-TSP by TA

Variant	Total number of iterations per run	Tour lengths	Mean	Variance
TA Standard	2.000.000	51.94 52.22 52.38 52.46 52.52 52.61 52.61 52.61 52.71 52.76 52.84 52.88 52.94 52.97 53.01 53.02 53.10 53.17 53.17 53.38 53.41 53.58 53.61 53.81 54.23	52.96	0.2598
TA Distance	2.000.000	50.97 51.07 51.16 51.27 51.29 51.32 51.32 51.38 51.42 51.48 51.53 51.53 51.53 51.53 51.57 51.58 51.62 51.63 51.63 51.68 51.69 51.80 51.86 51.89 51.96	51.51	0.0592
TA Distance	1.500.000	51.07 51.11 51.21 51.29 51.40 51.41 51.42 51.43 51.43 51.44 51.47 51.49 51.54 51.57 51.57 51.58 51.69 51.69 51.69 51.70 51.72 51.82 51.82 51.84 51.93	51.53	0.048
TA Distance	4.000.000	50.95 51.03 51.09 51.10 51.13 51.20 51.22 51.25 51.26 51.29 51.32 51.37 51.38 51.38 51.38 51.45 51.49 51.49 51.54 51.54 51.55 51.57 51.58 51.59 51.74	51.36	1.98

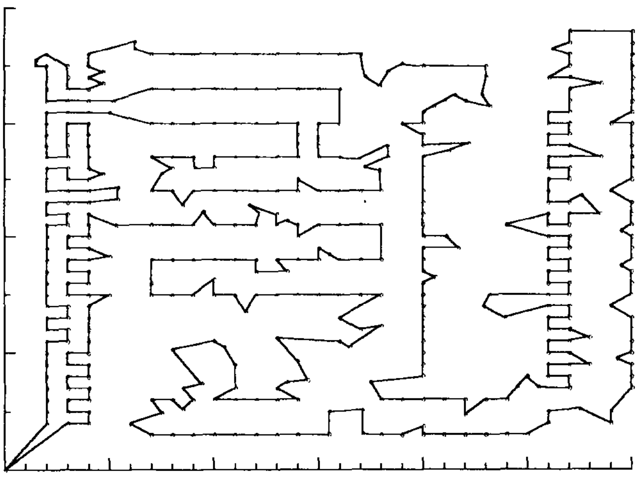

FIG. 1. Optimal tour for the 442-TSP by Holland. This tour has length 50.80.

easily be shortened "by hand." We never found any solution for the 442-cities problem which was not obviously bad in some small areas. For instance, the last solution can be polished by hand to become a tour of length 50.85. (The reader should find at least one obviously bad position in each of our near-to-optimum tours.) For tours with length ≈51.50, one can improve mostly by 0.30 or even more. Astonishingly, the best solution given in [4] of tour length 51.42 has no apparent weaknesses.

The running time for the various algorithms depends, of course, mainly on the total number of trials. The running times vary between one and three minutes of

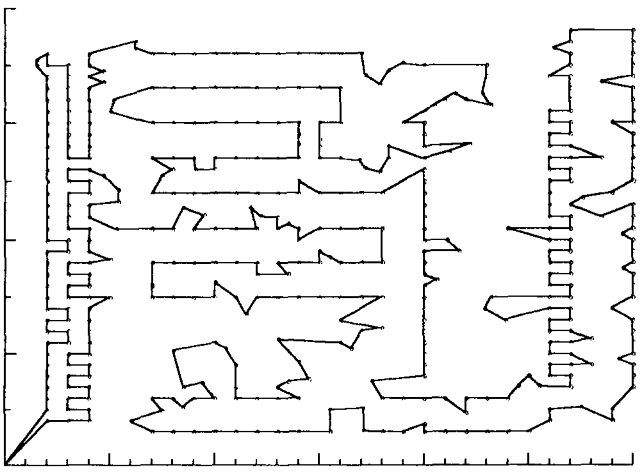

FIG. 2. A tour for the 442-TSP by TA distance (4,000,000). This tour has length 50.95.

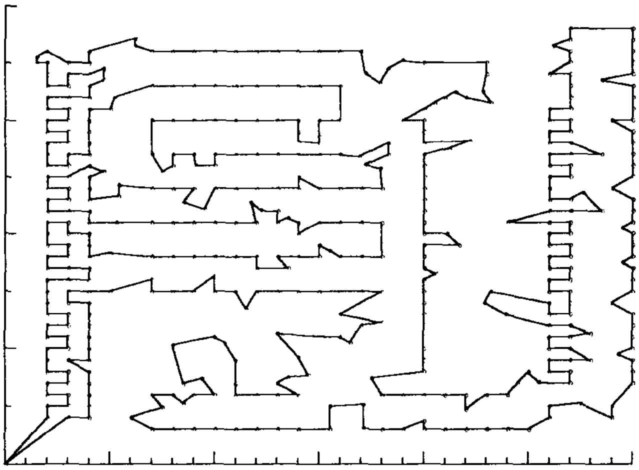

FIG. 3. A tour for the 442-TSP by TA distance (2,000,000). This tour has length 50.97.

TABLE III

Results for 442-TSP by Deterministic TA

Number of Runs	Parameter *nb*	Average Running Time in secs	Smallest Tour Length	Largest Tour Length	Number of Tours With Length Below 52.00
100	3	1.24	54.82	67.12	0
100	4	1.62	51.71	61.73	4
100	5	1.95	51.83	56.01	2
100	6	2.22	51.57	56.29	10
100	7	2.58	51.21	53.97	18
100	8	2.88	51.41	53.24	19
100	9	3.16	51.44	53.16	30
100	10	3.44	51.04	53.01	39
100	11	3.68	51.35	52.93	39
100	12	3.96	51.45	53.35	45
100	13	4 18	51.37	53.01	40
100	14	4.46	51.44	53.11	39
100	15	4.72	51.37	52.81	47
100	16	4.99	51 53	53.18	39
100	17	5.24	51.39	53.11	48
100	18	5.61	51.39	53.30	49
100	19	5.75	51.46	53.27	43
100	20	5.99	51.34	52.80	55

CPU time and are thus approximately one-half of the SA running times reported in [4]. Since we chose exactly the same number of trials for our *TA* method and since we have easier calculations in the TA approach (no probabilities, no exponential function), this is exactly what one would exspect.

The computational results of the **deterministic** algorithms are surprisingly excellent. Since the running times of these algorithms are only a few seconds we are abe to study these algorithms in detail. We give the results of 100 runs for each of the parameters $nb = 3, ..., 20$. We give the range of the tour lengths, the running times, and the number of resulting tour lengths below 52.00 (Table III).

A close look at Table III shows that for reasonable parameters *nb* one can achieve solutions below 52.00 within a very few seconds. Also, it is possible to generate very high quality solutions very quickly, if one takes "the best trial of *n* trials."

In a very recent paper, Mühlenbein [10] gives a statistic of the running time of his genetic algorithm. It is reported that a 16-processor machine (MEGAFRAME SUPERCLUSTER consisting of 16 transputers) is able to achieve solutions below 52.00 within 5000 s. 32 processors need 1500 s, 64 processors, 600 s. (We have used only one processor of a IBM 3090 with vector facility.) Furthermore, we easily achieve solutions below 51.50 in three to four minutes of time with our algorithm.

Some Recent Results on the 532-City TSP of Padberg and Rinaldi

In [7], Padberg and Rinaldi solved a 532-city TSP. The optimal tour length for this problem is shown to be 27,686. Recently, we got the coordinates of this problem and made a quick run of the deterministic TA algorithm. We used a trivial and not optimized threshold lowering schedule as in the program for the 442-city problem. We produced 100 different solutions with the parameter *nb* equal to 15.

The best tour has a length of 28,014; 61 tours have a length of less than 28,500; the length of the worst tour is 28,825. The average running time of a single run was 14.8 s. However, we are not aware of results with simulated annealing for this problem.

For their solution of the 532-city TSP Padberg and Rinaldi needed 50 good solutions for that problem. They used as an heuristic an adaptation of the well-known algorithm of Lin and Kernighan [12]. The solutions they obtained range between 28,150 and 29,143. It is reported, that the running time to obtain those 50 solutions was about 4 h on a VAX 11/780.

We conclude that the deterministic TA algorithm is of equal quality compared with the Lin–Kernighan algorithm for this problem, that it runs considerably faster, and, of course, that TA is the easiest heuristic to implement.

If one compares the results obtained by TA for the 442-problem and for the 532-problem, then it seems that TA yields better results for the 442-problem. The reason might be that the 442 is of a very regular structure compared with the 532-problem. It may be, that better results for the 532-problem necessitate a closer study of an appropriate threshold lowering schedule, which was not necessary for the 442-problem.

In [10], Mühlenbein reports results obtained with a parallel algorithm using concepts of genetics and learning. The best solution obtained by this algorithm is 27,702, which is very near to the optimum. On the other hand, the computation of this solution needed nearly 10,000 s of time for any of 64 transputers. It is not reported, how many trials were necessary for such a high quality solution. Also, it is reported that the algorithm needs about 1000 s on 64 transputers to achieve solutions of a length of 28,500 or less.

We conclude that Mühlenbein's algorithm generates better solutions for the 532-problem than TA does. For the 442-problem, however, TA gives better results. In any case, TA is orders of magnitude faster. Possibly, one could use deterministic TA to produce lots of good solutions as an input for genetic algorithms to drastically reduce their running time.

On the Generation of Error-Correcting Codes

In this section we describe the problem of finding constant weight codes. As well as in the preceding case, TA leads to good results.

A *binary* (c, n, d)*-code* is defined to be a set of c 0-1-sequences of length n called codewords, such that the Hamming distance between every pair of words is at least d; that is, every two words differ in at least d digits. Then, for each codeword we define the *weight* to be the number of 1's in this word. A *constant weight* (c, n, d, w)*-code* is a binary (c, n, d)-code consisting only of codewords with weight w. The problem is to find for given parameters n, d and w a constant weight code with as many codewords as possible.

For a TA algorithm we have to define

— a start configuration
— the form of a perturbation step to get new configurations
— a quality function.

For a start configuration, we randomly choose c codewords of the desired length n with the desired weight w. The TA algorithm will successively change this initial code, until (hopefully) all sequences have a mutual distance of at least d.

The simplest exchange step "old code" $\rightarrow$ "new code" is to alter one bit in a randomly chosen position in the code. However, if one changes one bit in a single codeword, its weight is changed, too. Therefore the canonical exchange step for this kind of problem is a *transposition of two positions in one of the c codewords.*

For the measurement of the quality we also used a canonical function:

$$\sum_{\substack{a,\, b \text{ codewords} \\ d(a,\, b) < d}} (d - d(a, b))^2.$$

Here, $d(—, —)$ denotes the Hamming distance between two codewords. We looked for an algorithm minimizing this function. A TA algorithm designed following these ideas yielded the code sizes, which we catalogue in Table IV. The best known lower bounds can be found in [8, 11], the upper bounds are taken from [8]. Our results are really a first trial, and we shall present more elaborated results in a further paper. The TA results are substantially better than the analogue ones for SA, except for the two cases where the best known lower bound was found by SA (here, the authors of [8] state that these codes were found only after "more computational effort"; we shall make a larger experiment with TA, too).

For the generation of codes with codewords of length $2n$ and weight n one can use a simple trick, which often leads to good results.

Suppose a is a codeword of length $2n$ and weight n. Let $\bar{a}$ be the complement of a, i.e., $\bar{a}$ is derived from a by altering all bits of a. Clearly, $d(a, \bar{a}) = 2n$. Observe that $\bar{a}$ has again weight n. Now use a TA algorithm with the quality function

$$\sum_{\substack{a,\, b \text{ codewords} \\ d(a,\, b) < d}} (d - d(a, b))^2 + \sum_{\substack{a,\, b \text{ codewords} \\ d(a,\, b) > 2n - d}} (d - d(a, b))^2,$$

which is to be minimized. If the algorithm stops with a function value of zero, then for any two codewords a, b in the final code we have

$$d \leqslant d(a, b) \leqslant 2n - d.$$

TABLE IV

Table of Constant Weight Codes

word length n	minimum distance d	weight w	best known lower bound	upper bound	lower bound by Simulated Annealing	lower bound by Threshold Accepting
21	10	9	26	41	19	19
22	10	9	31	57	23	29
22	10	10	38	74	-	32
23	10	7	18	23	18	17
23	10	8	28	50	28	22
23	10	9	39	87	24	35
23	10	11	53	135	39	46
24	10	8	33	68	33	31
24	10	9	49	119	24	42
24	10	11	75	223	57	62
24	10	12	80	247	60	65

Note. This table contains several lower bounds for comparison with the results obtained by TA.

TABLE V

The 21 Codewords and Their Complements

codewords	complements
0100100101010111010110	1011011010101000101001
0100011110010011011001	1011100001101100100110
0100011111100000101110	1011100000011111010001
1001011001011010011010	0110100110100101100101
0100010000101101011111	1011101111010010100000
0011011100000110010111	1100100011111001101000
1110111100001000110010	0001000011110111001101
1001110110100011010100	0110001001011100101011
0111110001110000010011	1000001110001111101100
1010100101100011001011	0101011010011100110100
0010101011001001011110	1101010100110110100001
1000100010010010111111	0111011101101101000000
1111000110010100001110	0000111001101011110001
0101101110101010000011	1010010001010101111100
0101000101001010111101	1010111010110101000010
0101110010000111101010	1010001101111000010101
0110110100111010001100	1001001011000101110011
0001110100011001100111	1110001011100110011000
0010000110111110110010	1101111001000001001101
1000110111001100011001	0111001000110011100110
0001101100110101111000	1110010011001010000111

Note. These 42 codewords form a (42, 22, 10, 11)-code.

This relation immediately implies that

$$d(a, \bar{b}) \geqslant d,$$

where $\bar{b}$ is the complement of b. (Note that $d(a, b) + d(a, \bar{b}) = 2n$ always.) Therefore, the new code consisting of the codewords found by the algorithm, together with all complements of these codewords, has minimum distance d.

The advantage of this approach lies in the fact that one can look for fewer codewords to get a large code. The disadvantage clearly is that the compound quality function is harder to minimize.

Using this method for the special case $n = 11$, $c = 21$, we found within very few seconds of CPU time the (21, 22, 10, 11)-code (see Table V). The distance matrix of this code is shown in Fig. 4.

We conclude that the 21 codewords in Table V form, together with their complements, a (42, 22, 10, 11)-code. Until now it was known only that there exist (38, 22, 10, 11)-codes, cf. [11].

	10	12	12	10	10	12	10	10	10	10	10	10	12	10	10	10	10	10	12	10
10		10	10	10	10	12	10	12	12	12	10	12	10	10	10	10	12	12	10	10
12	10		12	10	12	10	12	12	12	10	12	10	10	10	10	10	12	12	12	12
12	10	12		12	10	10	12	10	12	10	10	12	12	10	12	12	12	12	10	12
10	10	10	12		10	12	12	10	12	10	12	12	12	10	10	10	10	12	10	12
10	10	12	10	10		10	10	10	12	12	12	10	10	10	12	12	10	10	12	12
12	12	10	10	12	10		12	10	12	10	12	12	10	12	12	10	10	10	10	12
10	10	12	12	12	10	12		12	10	12	10	12	10	12	10	10	10	12	10	10
10	12	12	10	10	10	10	12		10	12	12	12	10	12	12	10	10	12	12	12
10	12	12	12	12	12	12	10	10		10	10	12	10	12	12	10	12	12	10	12
10	12	10	10	10	12	10	12	12	10		10	12	12	12	12	12	12	12	10	12
10	10	12	10	12	12	12	10	12	10	10		10	12	10	10	12	10	10	10	12
10	12	10	12	12	10	12	12	12	12	12	10		12	12	10	10	12	10	12	12
12	10	10	12	12	10	10	10	10	10	12	12	12		10	10	10	10	10	12	12
10	10	10	10	10	10	12	12	12	12	12	10	12	10		12	10	10	12	10	12
10	10	10	12	10	12	12	10	12	12	12	10	10	10	12		12	10	12	12	10
10	10	10	12	10	12	10	10	10	10	12	12	10	10	10	12		10	10	12	12
10	12	12	12	10	10	10	10	10	12	12	10	12	10	10	10	10		12	12	10
10	12	12	12	12	10	10	12	12	12	12	10	10	10	12	12	10	12		12	10
12	10	12	10	10	12	10	10	12	10	10	10	12	12	10	12	12	12	12		12
10	10	12	12	12	12	12	10	12	12	12	12	12	12	12	10	12	10	10	12	

FIG. 4. Distance matrix of the (21, 22, 10, 11)-code.

CONCLUDING REMARK

We have investigated TA only experimentally on an IBM 3090 VF. In a theoretical paper [13], Althöfer and Koschnick proved that TA has convergence properties which are similar to those of SA. They state in their very recent paper that "in a certain sense 'SA belongs to the convex hull of TA'."

REFERENCES

1. S. KIRKPATRICK, C. D. GELATT, AND M. P. VECCHI, *Science* **220**, 671 (1983).
2. N. METROPOLIS, A. ROSENBLUTH, M. ROSENBLUTH, A. TELLER, AND E. TELLER, *J. Chem. Phys.* **21**, 1087 (1953).
3. M. GRÖTSCHEL, Preprint No. 38, Universität Augsburg (unpublished).
4. Y. ROSSIER, M. TROYON, AND TH. M. LIEBLING, *OR Spektrum* **8**, 151, (1986).
5. H. MÜHLENBEIN, M. GORGES-SCHLEUTER, AND O. KRÄMER, *Parallel Comput.* **7**, 65, (1988).
6. O. A. HOLLAND, Dissertation, University of Bonn, 1987 (unpublished).
7. M. PADBERG AND G. RINALDI, *Oper. Res. Lett.* **6**, 1 (1987).
8. A. A. EL GAMAL, L. A. HEMACHANDRA, I. SHPERLING, AND V. K. WEI, *IEEE Trans. Inform. Theory* **IT-33**, 116 (1987).
9. S. LIN, *Bell Syst. Tech. J.* **44**, 2245 (1965).
10. H. MÜHLENBEIN, preprint (1988).
11. J. H. CONWAY AND N. J. A. SLOANE, *IEEE Trans. Inform. Theory* **IT-32**, 337 (1987).
12. S. LIN AND B. KERNIGHAN, *Oper. Res.* **21**, 498 (1973).
13. I. ALTHÖFER AND K.-U. KOSCHNICK, *SIAM J. Control Optim.* submitted.