JOURNAL OF COMPUTATIONAL PHYSICS 104, 86–92 (1993)

New Optimization Heuristics
The Great Deluge Algorithm and the Record-to-Record Travel

GUNTER DUECK

IBM Germany, Heidelberg Scientific Center, Tiergartenstrasse 15, D-6900 Heidelberg

Received December 4, 1990; revised October 29, 1991

In a former paper we introduced a very effective new general purpose optimization principle. We compared this method, which we called *threshold accepting* (TA), with the well-known simulated annealing (SA) method for discrete optimization. The empirical results demonstrated the superiority of the TA algorithm. In further experiments with the TA principle we discovered two new powerful optimization heuristics: The *great deluge algorithm* (GDA) and the *record-to-record travel* (RRT). These algorithms resemble in their structure the formerly studied TA and the SA method. The differences lie in their acceptance rules for worse intermediate solutions. Both, the GDA and the RRT, are essentially one-parameter algorithms; i.e., for the achievement of best possible performance, a good choice is necessary only for a *single parameter*. This is in contrast for instance to the classical SA algorithm, where it is necessary to choose carefully a certain sequence of parameters, the so-called annealing schedule. The quality of the computational results obtained so far by RRT and GDA shows that the new algorithms behave equally well as TA and thus a fortiori better than SA.

INTRODUCTION

In [1] we introduced the threshold accepting (TA) principle which yielded computational results superior to the well-known simulated annealing (SA) method. We demonstrated in tests on traveling salesman problems that TA

- yields much better results than SA
- needs considerably less computing time
- is more insensitive in its parameters.

During tests of TA on other problems, especially on knapsack problems (a detailed report will follow), we found that one can simplify even the TA principle. This resulted in two new optimization heuristics which we called the *great deluge algorithm* (GDA) and the *Record-to-Record Travel* (RRT). We give a description of all these algorithms in discussion here.

In many typical optimization problems one wants to find among many configurations one configuration which maximizes or minimizes a certain "quality-function." In the familiar traveling salesman problem one wants to find a circuit of travel through a given number of cities with minimal tour length. The algorithms we discuss here work as follows. First an initial configuration is chosen. Then each step of the four algorithms consists of a slight change of the old configuration into a new one. The "qualities" of the two configurations are compared. Then a decision is made if the new configuration is "acceptable." If the new configuration is acceptable it serves as the "old" configuration for the next step. If it is not acceptable, the algorithm proceeds with a new change of the old configuration.

The four algorithms differ in their decision rule to determine whether a configuration is acceptable or not. Kirkpatrick *et al.* [2] introduced the concept of "annealing" and combined it with the well-known Monte-Carlo algorithm by Metropolis *et al.* [3] which originally was used to numerically perform averages over large systems from statistical mechanics. The idea of SA runs as follows. The "qualities" of the two configurations (the old one and the new one) are compared. If the new configuration is better then 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 pricise:

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

0021-9991/93 $5.00

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 explain briefly 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 studied in [1], 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.

TA accepts *every* new configuration which is *not much worse* than the old one (SA accepts worse solutions only with rather small probabilities).

The Great Deluge Algorithm for Maximization.

Choose an initial configuration
choose the "rain speed" UP > 0
choose the initial WATER-LEVEL > 0
Opt: choose a new configuration which is a stochastic small
perturbation of the old configuration
compute $E :=$ quality (new configuration)
IF $E >$ WATER-LEVEL
THEN old configuration := new configuration
WATER-LEVEL := WATER-LEVEL + UP
IF a long time no increase in quality or too many iterations
THEN stop
GOTO Opt

Imagine, the GDA is to find the maximum point on a certain surface, for instance, the highest point in a fictitious empty country. Then, "we let it rain without end" in this country. The algorithm "walks around" in this country, but it never makes a step beyond the ever increasing water level. And it rains and rains $\cdots$. Our idea is that in the end the GDA "gets wet feet" when it has reached one of the very heighest points in the country so that is has found a point close to the optimum.

Originally, we did not really believe in this procedure, because our intuition mentioned: If you start this algorithm on an isle with no mountains then you will obtain very poor results. If the rain decomposes the country very quickly into different continents then you cannot hope to always reach very high points, etc. Nevertheless, we shall see that it works, and it works extremely well.

The Record-to-Record Travel for Maximization

Choose an initial configuration
choose an allowed DEVIATION > 0
set RECORD := quality (initial config.)
Opt: choose a new configuration which is a stochastic small perturbation of the old configuration
compute $E :=$ quality (new configuration)
IF $E >$ RECORD-DEVIATION
THEN old cnfiguration := new configuration
IF $E >$ RECORD
THEN RECORD := E
IF a long time no increase in quality or too many iterations
THEN stop
GOTO Opt

During a run of RT any configuration is accepted the quality of which is not much worse than the best value RECORD obtained so far. RRT is in some sense a variant of GDA. "The water level" in the RRT is the value of RECORD-DEVIATION.

The new optimization heuristics have the advantage that they depend only on one single parameter. For the GDA it is necessary to choose the rain speed UP only. The value UP is the only parameter which is responsible for the computation speed and for the quality of the results. If the rain speed is high, the algorithm is very fast and produces results of only minor quality. If UP is chosen to be very small then the algorithm produces an excellent result after a long computation time.

In the RRT algorithm, a very similar behaviour can be observed when varying the parameter DEVIATION. A small value gives a quick poor result, a large one excellent results after a long time.

In our experiments with the GDA we found a reasonable rule for the choice of the parameter UP: the best UP should be somewhat smaller than 1 % of the average gap between the quality of the current configuration and the water level. This easy rule revealed to us very quickly a very good choice for UP in all cases that we tried to attack an optimization problem with GDA.

We see that GDA and RRT are extremely easy to implement, and they depend only on a single parameter which is not hard to choose in a satisfactory manner.

It is left now to report the computational results. We have studied TA until now for the TSP, for finding good error-correcting codes, for the minimization of spin-glass Hamiltonians, for 0–1 linear programming, 0–1 quadratic programming, quadratic assignment problems, etc. In a forthcoming report we shall present our results on integer programming, together with a report on a large customer project. In the present paper we give a comparison of the new heuristics only for two typical traveling salesman problems, the 442-cities problem of Grötschel [4] and the 532-cities problem of Padberg and Rinaldi [8]. For these two problems, many papers have been written with computational results. Furthermore, the optimum tour lengths have been recently determined.

GRÖTSCHEL'S 442-PROBLEM: THE ALGORITHMS

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

All our algorithms start with a random permutation on C as an initial tour. As in [5], 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

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

Rossier *et al.* [5] used SA with this rule for the choice of the new configurations. A comparison of SA and TA results for the 442-problem is given in [1]. In [1], we found out that deterministic versions of TA are much faster and nearly equally well. In this paper we consider therefore only those algorithms. In [1], we considered the following algorithm for the 422-problem.

In a preprocessing run we compute for each city the next *nb* (we use mostly $nb = 10$) neighbors (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 *nb* neighbors of that city in a prescribed order. In tests we had seen that it makes no sense to try LIN-2-OPT exchanges with two cities which are rather far away from each other. Thus we try to narrow the reach of the LIN-2-OPT trials. The parameter *nb* measures "how local" exchanges can be. For small *nb*, only exchanges in the very neighborhood are permitted. If one looks at pictures of optimal TSP-solutions, one can observe that most of the cities are connected with cities in the very neighborhood. The computational results will show that it is sufficient to try only neighbored LIN-2-OPT changes. This is a very important observation, because "one round" through the following algorithm needs only linearly many trials rather than a quadraticly growing number.

DETERMINISTIC TA ALGORITHM (ORIGINAL FORM)

fix a positive integer *nb*
compute for every city the nearest *nb* neighbor 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 neighbors of city i
DO perform LIN-2-OPT
IF length (new tour) < length (old tour) + T
THEN old tour := new tour

A complete run of this algorithm performs 442 **nb** 100 LIN-2-OPT trials. Thus, the running time depends essentially on the parameter *nb*. In further experiments with the deterministic TA algorithms, it turned out that it is much

better to exchange the loops in the above algorithm. We use in the sequel the

DETERMINISTIC TA ALGORITHM (IMPROVED FORM). Here, the loop over the cities is the outer loop, the loop over the thresholds the inner one: The improvement is quite significant. It seems to be important to try a LIN-2-OPT exchange first with cities being far from another and then with cities being close to another.

For the 442-cities problem, the GDA is of the following form.

DETERMINISTIC GREAT DELUGE ALGORITHM.

fix a positive integer *nb*
compute for every city the nearest nb neighbor cities
choose initial random tour
initial (water) LEVEL: length of the initial tour
Nextround:
****
FOR every $j = nb, ..., 1$
FOR every city $i = 1, ..., 442$
perform LIN-2-OPT with city i and its jth nearest neighbor
IF L := length (new tour) < LEVEL
THEN old tour := new tour
LEVEL 1 := LEVEL − 0.01
LEVEL 2 := LEVEL − (LEVEL − L)/500.
LEVEL := minimum (LEVEL 1, LEVEL 2)
****
IF there has been a decrease in LEVEL in this round
THEN GOTO Nextround
ELSE stop

Note that this algorithm is not of the pure form we gave in the Introduction. We lower the level here by the difference between the level and the length of the current tour divided by 500, but at least by 0.01. One can run the algorithm also by setting UP := 0.01 and using the pure form. The qualities of the results are equally good. However, the form given above runs considerably faster. The reason is simply that in the first part of the algorithm "the rain speed" can be higher than in the last fraction of time without any effect on the result quality.

In a similar way, the *deterministic record-to-record travel* is designed for the particular TSP case.

GRÖTSCHEL'S 442 PROBLEM: COMPUTATIONAL RESULTS

We recall some of the computational results on Grötschel's TSP on 442 cities (cf. [1] for more details). Rossier *et al.* [5] report that they performed the pure LIN-2-OPT exchange algorithm many times (accept LIN-2-OPT *improvements* until no further improvement is possible). The best result was 57.30.

TABLE I

Results for 442-TSP by Deterministic TA

Number of runs	Parameter *nb*	Average running time in seconds	Smallest tour length	Largest tour length	Number of tours with length below 52.00
100	4	1.62	51.71	61.73	4
100	6	2.22	51.57	56.29	10
100	8	2.88	51.41	53.24	19
100	10	3.44	51.04	53.01	39
100	12	3.96	51.45	53.35	45
100	14	4.46	51.44	53.11	39
100	16	4.99	51.53	53.18	39
100	18	5.61	51.39	53.30	49
100	20	5.99	51.34	52.80	55

Using the SA approach with neighborhood conditions similar to the neighborhood conditions we use in this paper, Rossier *et al.* [5] achieved as their best result 51.76. With more sophisticated neighborhood conditions which seem to be suited for the 442 problem (however, not for general TSPs), they could compute a solution with tour length 51.42. At this time, this was the best known solution for this problem.

Mühlenbein *et al.* obtained in [6] solutions of length 51.24 and 51.21 with evolution algorithms which consume a great deal of CPU time (many hours). Holland in [7] obtained an optimal solution. Its tour length is 50.80 (in real* 8; Holland originally gave an "integer solution" of tour length 50.69).

In [1] we reported the computational results for the TA algorithm and the deterministic TA algorithm. Here, we give the results of the improved TA algorithm, the GDA,

TABLE II

Results for 442-TSP by Deterministic TA (Improved Form)

Number of runs	Parameter *nb*	Average running time in seconds	Smallest tour length	Largest tour length	Number of tours with length below 52.00
100	4	1.53	50.98	54.05	24
100	6	2.26	51.22	53.79	55
100	8	3.01	51.24	52.93	64
100	10	3.69	50.90	52.81	67
100	12	4.49	51.23	52.86	80
100	14	5.17	51.21	52.79	75
100	16	5.84	50.98	52.88	70
100	18	6.56	51.17	52.72	76
100	20	7.07	51.27	52.86	74

TABLE III

Results for 442-TSP by Deterministic Record-to-Record Travel

Number of runs	Parameter *nb*	Average running time in seconds	Smallest tour length	Largest tour length	Number of tours with length below 52.00
100	4	3.30	51.22	54.55	37
100	6	4.62	51.21	52.92	69
100	8	6.04	51.12	52.77	72
100	10	7.31	51.15	52.34	91
100	12	8.72	51.21	52.45	80
100	14	10.19	51.21	52.81	90
100	16	11.57	51.03	52.36	87
100	18	12.87	50.95	52.23	94
100	20	14.19	51.15	52.38	89

and the RRT. It is very interesting to compare the TA algorithm and its improved form. For this purpose, we recall the table with the computational results for the TA from [1]. We give the range of the tour lengths, the running times, and the number of resulting tour lengths below 52.00 (Table I).

In Table II we report the results of the improved TA version. Only the order of the loops has been exchanged. This simple trick results in a tremendous improvement in the tour lengths. We see that the new version gives excellent results. A large fraction of the solutions have a tour length less than 52.00. In a 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 achieve with our algorithm easily solutions below 51.20 in 3–4 min of time.

The Table III shows the results of the RRT. For the 442-problem, it turned out that some more rounds are necessary than in the TA algorithm, where we ran 100 rounds. After a very few runs of the RRT it was clear that 200 are sufficient, and the following results have been obtained within 200 rounds. (Afterwards, we observed that nearly the same quality of results is achievable in about 160 runs.) Hence, the computation times are longer using RRT (approximately two times).

We observe that if *nb* is greater or equal to 10, then RRT gives nearly equally good results. The same is true for the GDA. We hoped that we could possibly achieve better results for large *nb*. Table IV for the GDA results shows that also here small neighborhood parameters are completely sufficient.

From these data we see that all these three algorithms give very good results, and it would be hard to decide wether one of them is superior to the others. However, GDA and RRT are much easier to implement, because they are only single-parameter dependent. The tables above give only the best and worst tour lengths out of series of 100 program runs. In order to give an overview of the full result we now show a complete list of the 100 tour lengths obtained by RRT with parameter *nb* equal to 10 (see Table V).

TABLE IV

Results for 442-TSP by Deterministic Great Deluge Algorithm

Number of runs	Parameter *nb*	Average running time in seconds	Smallest tour length	Largest tour length	Number of tours with length below 52.00
100	5	2.52	51.44	55.00	32
100	15	6.50	51.18	52.92	62
100	25	10.83	51.14	52.63	73
100	35	14.22	51.17	52.90	71
100	45	19.10	51.03	52.69	77
100	55	25.14	51.01	52.74	87

TABLE V

Number of Neighbors = 10

51.15218	51.50959	51.69645	51.88964
51.19908	51.51166	51.69684	51.90404
51.20750	51.51596	51.70575	51.91516
51.21960	51.51639	51.71298	51.92648
51.25765	51.52743	51.72124	51.92985
51.28157	51.54187	51.72490	51.94372
51.30762	51.54449	51.72714	51.96005
51.32246	51.57118	51.73302	51.96446
51.33484	51.57459	51.73346	51.97546
51.34033	51.58344	51.74033	51.99759
51.36479	51.58921	51.74247	52.01247
51.36934	51.59735	51.74995	52.02172
51.38676	51.59795	51.74970	52.06296
51.39428	51.60586	51.75121	52.06343
51.40242	51.60654	51.76048	52.06737
51.41333	51.60880	51.77002	52.12821
51.41458	51.60816	51.77520	52.16198
51.42940	51.61629	51.80050	52.28577
51.43057	51.63180	51.80341	52.34396
51.44337	51.63486	51.81393	
51.44392	51.63512	51.84111	
51.45236	51.64210	51.85109	
51.45799	51.65594	51.85835	
51.46776	51.66754	51.85979	
51.47778	51.67024	51.87803	
51.49124	51.68682	51.87907	
51.49804	51.68936	51.88855	

Note. Average time = 7.31 s.

RESULTS ON THE 532-CITY TSP OF PADBERG AND RINALDI

In [8], Padberg and Rinaldi solved a 532-city TSP. The optimal tour length for this problem is shown to be 27,686.

We recall briefly here some results and remarks from [1]. For their solution of the 532-city TSP Padberg and Rinaldi needed 50 good solutions for that problem. They used as an heuristic an adaption of the well-known algorithm of Lin and Kernighan [11]. 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 shall see that the deterministic TA algorithm, the GDA, and the RRT are of equal quality compared with the Lin–Kernighan algorithm for this problem, that they run considerably faster, and, of course, that they are much easier to implement. If one compares the results obtained by TA for the 442 problem and for the 532 problem, then one can hardly observe a different behavior.

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 close 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. It is mentioned, however, 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 our algorithms do. For the 442 problem, however, we obtained considerably better results. In any case, our algorithms are orders of magnitude faster. We give in the sequel the computational results for the 532 problem within the same framework as for the 442 problem (Table VI).

Again we see from the following results for the GDA and the RRT that the achievable tour lengths are in quite the same range. Table VII shows the results of the RRT (200 rounds).

As for the 442 problem we observe that it is sufficient to run the algorithms with a rather small neighborhood parameter.

The GDA times for the 532 problem are faster than the computation times for RRT and TA. On the average, GDA ran about 130 rounds. GDA and RRT give better results than TA; GDA here is the most successful algorithm.

TABLE VII

Results for 532-TSP by Deterministic Record-to-Record Travel

Number of runs	Parameter *nb*	Average running time in seconds	Smallest tour length	Largest tour length	Number of tours with length below 28,000	Number of tours with length below 28,250	Number of tours with length below 28,500
100	4	3.82	28059	31647	0	4	8
100	6	5.21	28001	29074	0	37	84
100	8	6.41	27921	28903	3	47	90
100	10	7.72	27880	28729	2	42	92
100	12	8.91	27900	28784	5	45	94
100	14	10.04	27893	28.629	5	53	94
100	16	11.53	27879	28635	5	47	95
100	18	12.66	27898	28543	6	60	95
100	20	13.74	27875	28745	11	60	96

TABLE VI

Results for 532-TSP by Deterministic TA (Improved Form)

Number of runs	Parameter *nb*	Average running time in seconds	Smallest tour length	Largest tour length	Number of tours with length below 28,000	Number of tours with length below 28,250	Number of tours with length below 28,500
100	4	4.84	29722	38632	0	0	0
100	6	6.74	28202	30669	0	6	23
100	8	8.56	27934	29678	1	18	63
100	10	10.42	28014	29476	0	28	78
100	12	12.17	27890	29006	4	36	82
100	14	13.83	27858	28722	4	51	97
100	16	15.62	27972	28748	1	51	90
100	18	17.49	27990	28646	1	42	89
100	20	19.24	27947	28657	2	46	89

TABLE VIII

Results for 532-TSP by Deterministic Great Deluge Algorithm

Number of runs	Parameter *nb*	Average running time in seconds	Smallest tour length	Largest tour length	Number of tours with length below 28,000	Number of tours with length below 28,250	Number of tours with length below 28,500
100	4	6.22	27962	31928	1	17	37
100	6	7.14	27953	29306	6	52	93
100	8	8.18	27938	28582	10	60	98
100	10	9.03	27854	28626	4	53	96
100	12	10.29	27920	28559	6	56	97
100	14	11.54	27873	28522	7	65	98
100	16	12.87	27888	28507	11	59	99
100	18	14.47	27931	28540	7	64	98
100	20	15.71	27863	28561	8	65	97

CONCLUDING REMARKS

We demonstrated the power of the new heuristics for the solution of well-known TSPs. In the IBM plant in Sindelfingen, Germany, there are many thousands of puncher patterns of the kind of the 442 problem. The machines do not run according to the Euclidean metric. It is necessary to implement the maximum metric or the Manhattan metric (and in these cases not really the distance is important, but the *time* needed for this distance). The optimized tours obtained by GDA look very good in all patterns: There are sparse, dense, regular, clustered patterns. However, we do not know the true optima in these cases, so we cannot report analogous results for such patterns.

Meanwhile we tried the much harder problem class of chip placement with these algorithms. For two IBM internal reference chips we obtained much better results (10% less wire length) than any other previously known ones (including, for instance, SA results). For publication of such results the problem will occur that the chip data are not public domain. In a forthcoming report we shall quantify our results. We plan to generate an artificial chip data set for the research community.

Studies are ready also for transportation/distribution problems as well as studies for large production scheduling cases and for food mixing (integer programming). In all these studies we were able to achieve better results than concurrent methods. We could achieve excellent solutions in large problems where the classical method or program packages have size problems (CPU time or storage).

REFERENCES

1. G. Dueck and T. Scheuer, *J. Comput. Phys.* **90**, 161 (1990).
2. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, *Science* **220**, 671 (1983).
3. N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller, *J. Chem. Phys.* **21**, 1087 (1953).
4. M. Grötschel, Preprint No. 38, Universität Augsburg (unpublished).
5. Y. Rossier, M. Troyon, and Th. M. Liebling, *OR Spektrum* **8**, 151 (1986).
6. H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer, *Parallel Comput.* **7**, 65 (1988).
7. O. A. Holland, Dissertation, University of Bonn (1987).
8. M. Padberg and G. Rinaldi, *Oper. Res. Lett.* **6**, 1 (1987).
9. S. Lin, *Bell Syst. Tech. J.* **44**, 2245 (1965).
10. H. Mühlenbein, preprint (1988).
11. S. Lin and B. Kernighan, *Oper. Res.* **21**, 498 (1973).