Solving the Traveling Salesman Problem With Annealing-Based Heuristics: A Computational Study

Joshua W. Pepper, Bruce L. Golden, and Edward A. Wasil

***Abstract*—Recently, several general optimization algorithms based on the demon algorithm from statistical physics have been developed and tested on a few traveling salesman problems with encouraging results. In this paper, we conduct an extensive computational study of 11 annealing-based heuristics for the traveling salesman problem. We code versions of simulated annealing, threshold accepting, record-to-record travel and eight heuristics based on the demon algorithm. We apply each heuristic to 29 traveling salesman problems taken from a well-known online library, compare the results with respect to accuracy and running time and provide insights and suggestions for future work.**

***Index Terms*—Demon algorithm, heuristics, traveling saleman problem.**

I. Introduction

THE Traveling Salesman Problem (TSP) is a classic problem in combinatorial optimization. Given a complete graph, $G = \{N, E\}$, where N is the set of nodes, E is the set of edges and distances are Euclidean, we try to find the shortest tour that visits every node in N exactly once.

Since the 1950s, many heuristics for the TSP have been developed that find good solutions, but not necessarily optimal solutions (see Jünger *et al.* [1] for details on construction and improvement heuristics for the TSP). In the 1980s, focus shifted to applying general-purpose metaheuristics, including simulated annealing, deterministic annealing, genetic algorithms, neural networks, and tabu search, to the TSP (see the volume by Reeves [2] and the paper by Osman and Kelly [3] for more information on metaheuristics). In 1998, several general optimization algorithms based on the demon algorithm from statistical physics (see Creutz [4]) were developed and tested on the TSP (see the papers by Wood and Downs [5], [6]). All of the metaheuristics try to escape from or avoid poor local minima (e.g., by accepting changes that increase the length of the tour).

In this paper, we conduct an extensive computational study of 11 annealing-based heuristics for the TSP. We code versions of simulated annealing (SA), threshold accepting (TA), record-to-record travel (RRT) and eight heuristics based on the demon algorithm (DA). Four of the DA heuristics are new. We apply each heuristic to 29 problems taken from TSPLIB and compare the performances of the heuristics with respect to accuracy and running time.

Our purpose is not to argue that demon algorithms are the best procedures for solving TSPs or even Euclidean TSPs. (Many of the best procedures are discussed by Johnson and McGeoch [7].) Rather, we believe that demon algorithms are interesting and clever new metaheuristics that can be applied with ease to a wide variety of combinatorial optimization problems. In this paper, we perform extensive computational tests on Euclidean TSPs and conclude that some of the demon algorithm variants are competitive with simulated annealing.

In Section II, we provide an overview of SA, TA, RRT and DA that includes a step-by-step summary of the standard version of each algorithm. We describe four standard versions and four new variants of DA. In Section III, we present the details of our computational study. We describe the test problems and implementation details (including neighbor lists and the greedy start procedure). In Section IV, we give the results generated by 11 algorithms. We compare the results based on accuracy and running time and provide insights into the performances of different versions of the demon algorithms. In Section V, we present our conclusions.

II. Local Search Heuristics

Over the last 40 years, many heuristic methods have been developed for solving the traveling salesman problem. Many heuristic methods improve solutions to the TSP iteratively, where each new tour is shorter than the previous tour. In the 1990s, research focused on applying general-purpose metaheuristics to the TSP. The metaheuristics select new solutions that sometimes increase the length of the tour and thereby avoid becoming trapped in poor local optima. In this section, we describe 11 metaheuristics for the TSP: simulated annealing, two types of deterministic annealing and eight variants of the demon algorithm.

A. Simulated Annealing

SA simulates a collection of atoms in equilibrium at a set temperature. SA starts with an initial state. A random change is made to this state and the change in energy, ΔE, is calculated. If the new state has a lower energy than the previous state, i.e., $\Delta E \leq 0$, the new state is carried into the next iteration. However, if the new state has a higher energy, it is accepted with

Manuscript received January 11, 2001; revised January 14, 2002. This paper was recommended by Associate Editor M. Embrechts.

J. W. Pepper is with the The MITRE Corporation, McLean, VA 22102 USA.

B. L. Golden is with the Robert H. Smith School of Business, University of Maryland, College Park, MD 20742 USA (e-mail: bgolden@rhsmith.umd.edu).

E. A. Wasil is with the Kogod School of Business, American University, Washington, DC 20016 USA.

Publisher Item Identifier S 1083-4427(02)02692-9.

probability $P(\Delta E) = \exp(-\Delta E/k_B T)$, where T is the temperature and k_B is Boltzmann's constant. This acceptance of an uphill move (a move with $\Delta E \geq 0$) is analogous to a higher energy molecule knocking loose a molecule trapped in a state of excess energy [8]. The SA algorithm is given in Table I (for background details on SA see [9]).

Determining the cooling (annealing) schedule in step 3(f) of SA is crucial to the success of the algorithm. The chance of accepting an energy-increasing move is inversely proportional to ΔE and proportional to T, which decreases with time. If the user sets the annealing schedule intelligently, the algorithm can escape shallow local minima early on and explore deeper minima more fully as it progresses.

B. Deterministic Annealing

In two deterministic versions of simulated annealing called threshold accepting and record-to-record travel (see [10] and [11]), an uphill move is accepted if the new tour length is less than some amount.

1) Threshold Accepting: In threshold accepting, a threshold, T, is specified (see Table I). This threshold is the bound on the increase in tour length from one iteration to the next. In step 3(c) of TA, a new tour is accepted only if ΔE, the change in energy, is less than T. Thus, tours of increased length can be accepted. In step 3(e) of TA, the threshold is annealed according to a schedule.

2) Record-to-Record Travel: In record-to-record travel (see Table I), $RECORD$ is the length of the best tour encountered. The algorithm can only explore tours with length shorter than the deviation, D, plus the $RECORD$ (see step 3(b) of RRT). Thus, RRT allows uphill moves, but the value $RECORD + D$ serves as a bound on the length of tours that can be accepted. As the $RECORD$ improves, the bound decreases and the algorithm restricts itself to tours of shorter length.

C. Demon Algorithm

The demon algorithm is a variant of simulated annealing (see [5], [6]). The demon algorithm replaces the probability of accepting a tour of increased length with the concept of a creditor, known as the demon. When the algorithm accepts a tour of shorter length, the decrease in length is credited to the demon. Similarly, if the algorithm accepts a tour of increased length, the increase in length is debited from the demon. A tour of increased length will be accepted only if the demon has enough credit to pay for the increase in tour length. The basic demon algorithm (DA) is given in Table I.

Wood and Downs [5], [6] modified DA and developed four standard demon algorithms for optimization: bounded demon algorithm (BD), randomized bounded demon algorithm (RBD), annealed demon algorithm (AD) and randomized annealed demon algorithm (RAD). The modifications to the basic demon algorithm for BD, RBD, AD and RAD are given in Table II.

In its basic form, the demon algorithm is not a minimization technique. Wood and Downs [5], [6] proposed two modifications that gradually remove energy from the demon in order to encourage minimization. One modification imposes a specified upper bound on the demon value that may restrict the demon value after energy decreasing moves. The upper bound prevents the demon from receiving all of the credit from accepting tours of shorter length. This modification is incorporated into the bounded demon algorithm (the upper bound is given by D_0 under BD in Table II).

The second modification reduces the demon value according to a specified schedule in much the same way that Kirkpatrick *et al.* [9] annealed the temperature in SA. This modification is incorporated into the annealed demon algorithm (AD).

With both modifications, an algorithm decreases the credit given to the demon. An algorithm will have more difficulty accepting tours of increased length after accepting tours of decreased length, forcing it to seek tours of shorter length.

Wood and Downs [5], [6] tried to improve BD and AD by introducing a randomized component in which the demon value is replaced with the demon mean value and the demon mean value performs the role of the creditor. The demon's value is a Gaussian (normal) random variable centered around the demon mean value (D_M) with a specified standard deviation (D_{sd}). The two randomized versions (RBD and RAD) of the bounded and annealed demon algorithms are obtained by replacing D with D_M (see Table II) and adding a step (3.d.i in Table II) to

TABLE I
SIMULATED ANNEALING, THRESHOLD ACCEPTING, RECORD-TO-RECORD TRAVEL AND DEMON ALGORITHMS

SA
1. Choose an initial tour S
2. Choose a temperature $T = T_0 > 0$
3. Repeat:
   a. Choose a new tour S'
   b. Let $\Delta E = E(S') - E(S)$, where $E(S)$ is the energy (length) of tour S
   c. If $\Delta E \leq 0$, accept new tour, i.e., $S = S'$
   d. Else if exp $(-\Delta E/T) \geq$ rand (0, 1), accept new tour
   e. Else reject new tour
   f. Reduce temperature according to annealing schedule
4. Until termination conditions are met

TA
1. Choose an initial tour S
2. Choose a threshold $T = T_0 > 0$
3. Repeat:
   a. Choose a new tour S'
   b. Let $\Delta E = E(S') - E(S)$, where $E(S)$ is the energy (length) of tour S
   c. If $\Delta E < T$, accept new tour, i.e., $S = S'$
   d. Else reject new tour
   e. Reduce threshold according to annealing schedule
4. Until termination conditions are met

RRT
1. Choose an initial tour S and set $RECORD = E(S)$
2. Choose a deviation $D > 0$
3. Repeat:
   a. Choose a new tour S'
   b. If $E(S') < RECORD + D$
      i. Let $S = S'$
      ii. If $E(S') < RECORD$, let $RECORD = E(S')$
4. Until termination conditions are met

DA
1. Choose an initial tour S
2. Choose a demon value $D > 0$
3. Repeat:
   a. Choose a new tour S'
   b. Let $\Delta E = E(S') - E(S)$, where $E(S)$ is the energy of tour S
   c. If $\Delta E < D$, accept new tour: $S = S'$ and $D = D - \Delta E$
4. Until termination conditions are met

TABLE II
MODIFICATIONS TO PRODUCE FOUR DEMON ALGORITHMS

BD
2. Choose a demon value $D = D_0 > 0$
3.d If $D > D_0$, set $D = D_0$
RBD
2. Choose a demon mean value $D_M = D_{M0} > 0$
3.c If $\Delta E < 0$, accept new tour
3.d Else
3.d.i $D = D_M +$ *Gaussian noise* value
3.d.ii If $\Delta E < D$, accept new tour: $S = S'$ and $D_M = D_M - \Delta E$
3.e If $D_M > D_{M0}$, set $D_M = D_{M0}$
AD
3.d Reduce D according to annealing schedule
RAD
2. Choose a demon mean value $D_M > 0$
3.c If $\Delta E < 0$, accept new tour
3.d Else
3.d.i $D = D_M +$ *Gaussian noise* value
3.d.ii If $\Delta E < D$, accept new tour: $S = S'$ and $D_M = D_M - \Delta E$
3.e Reduce D_M according to annealing schedule

TABLE III
DEMON ALGORITHM VARIANTS

ABD
2. Choose a demon value $D = D_0 > 0$
3.d If $D > D_0$, set $D = D_0$
3.e Reduce D_0 according to annealing schedule
RABD
2. Choose a demon mean value $D_M = D_{M0} > 0$
3.c If $\Delta E < 0$, accept new tour
3.d Else
3.d.i $D = D_M +$ *Gaussian noise* value
3.d.ii If $\Delta E < D$, accept new tour: $S = S'$ and $D_M = D_M - \Delta E$
3.e Reduce D_{M0} according to annealing schedule
3.f If $D_M > D_{M0}$, Set $D_M = D_{M0}$
ADH
2. Choose a demon mean value $D_M > 0$
3.c. If $\Delta E < 0$, accept new tour
3.d. Else
3.d.i. $D = D_M +$ *Gaussian noise* value
3.d.ii. If $\Delta E < D$, accept new tour: $S = S'$ and $D_M = D_M - \Delta E$
3.e. Reduce D_M according to annealing schedule
3.f. Reduce D_{sd} according to annealing schedule
ABDH
2. Choose a demon mean value $D_M = D_{M0} > 0$
3.c. If $\Delta E < 0$, accept new tour
3.d. Else
3.d.i. $D = D_M +$ *Gaussian noise* value
3.d.ii. If $\Delta E < D$, accept new tour: $S = S'$ and $D_M = D_M - \Delta E$
3.e. Reduce D_{M0} according to annealing schedule
3.f. Reduce D_{sd} according to annealing schedule
3.e. If $D_M > D_{M0}$, set $D_M = D_{M0}$

generate a demon value from a distribution with mean $D_M +$ *Gaussian noise* value. By using the randomized component, Wood and Downs allowed rare large increases in energy that were not permitted by the deterministic algorithms.

We point out that Wood and Downs [5], [6] tested the four algorithms (BD, RBD, AD and RAD) on two TSPs.

1) Variants of the Demon Algorithm: Building on the four standard demon algorithms that have appeared in the literature, we propose four new variants: the annealed bounded demon algorithm (ABD), randomized annealed bounded demon algorithm (RABD), hybrid annealed demon algorithm (ADH) and hybrid annealed bounded demon algorithm (ABDH). The modifications to the basic demon algorithm for the four variants are given in Table III.

In ABD and RABD, we apply a bound to the demon value and anneal that bound in the hope that this will force the demon value to decrease smoothly over the long term. This should lead to higher-quality tours.

In ADH, we anneal the standard deviation of the demon value over time in addition to annealing the demon mean value. In ABDH, we anneal the standard deviation of the demon value over time in addition to annealing the demon bound. There are different annealing schedules for the standard deviation, demon mean value and the demon bound. Over time, the randomized components are reduced so that both hybrids tend toward the deterministic counterparts.

III. COMPUTATIONAL STUDY

In this section, we describe the test problems and implementation details of the algorithms that we use in our computational study.

A. Test Problems

We select 29 problems from the TSPLIB online library (see Reinelt [12] and the Web site at http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/). The problems range in size from 105 to 1432 nodes and were used in previous computational studies (see Coy *et al.* [13]–[15]). The identifier to each problem is given in Table IV. Each entry gives the number of nodes in the problem (e.g., lin105 has 105 nodes).

B. Implementation Details

Each algorithm is written in C++ and all computational work is carried out on several identical SPARC ULTRA 5 workstations. We use the two-exchange procedure of Lin [16] to generate a new solution in each iteration of each algorithm (the two-exchange procedure alters the existing tour by exchanging two arcs that are in the tour for two arcs that are not in the tour). To make the two-exchange procedure run quickly, we store the nearest neighbors of each node in a list and only look at exchanges with the nearest neighbors. We build neighbor lists with 20 members for each node (the size of the list is based on Dueck's testing [10]). We construct a starting tour using the randomized greedy heuristic recommended by Johnson and McGeoch [7]. (To generate a randomized greedy tour, we start with unconnected nodes and sorted arc lengths, insert the arc with the shortest length as long as a subtour is not created and continue to insert arcs until a tour is formed. At each insertion, the shortest arc that can be added to a partial tour without creating a subtour has a 2/3 probability of being selected.) If the two-exchange procedure cycles 25 times through the neighbor lists without improving the best solution found, termination occurs.

Eight of the algorithms require an annealing schedule and we use a geometric schedule (i.e., value = value $\times \alpha$ is applied every fixed number of iterations) based on Johnson *et al.* [17]

TABLE IV
PROBLEM INFORMATION

Small Problems	Medium Problems	Large Problems
lin105	rd400	pr1002
pr107	pr439	pcb1173
pr124	pcb442	rl1304
pr136	d493	nrw1379
pr144	u574	u1432
pr152	rat575	
u159	p654	
rat195	d657	
d198	u724	
pr226	rat783	
gil262		
pr264		
pr299		
lin318		

TABLE V
ORIGINAL PARAMETERS AND RECHARACTERIZED PARAMETERS

Algorithm	Original Parameter	Original Parameter	Recharacterized Parameter
SA, TA	T_0, α	T_0, D, D_0, D_{M0}	$= if \times E_{init}$
RRT	D	D_{sd}	$= sdf \times E_{init}$
BD	D_0	α	α
RBD	D_{M0}, D_{sd}	β	β
AD, ABD	D_0, α	E_{init} is the length of the initial tour	
RAD, RABD	D_{M0}, D_{sd}, α	α, β are multipliers for annealing schedules	
ADH	D_0, D_{sd}, α, β		
ABDH	D_{M0}, D_{sd}, α, β		

TABLE VI
FINAL VALUES OF PARAMETERS

Algorithm	*if*	*sdf*	α	β
SA	0.0035		0.9646	
TA	0.0722		0.9515	
RRT	0.0055			
BD	0.0027			
RBD	0.0030	0.0072		
AD	0.0745		0.5268	
RAD	0.0585	0.0005	0.4698	
ABD	0.0457		0.9716	
RABD	0.0407	0.0008	0.9418	
ADH	0.0489	0.0282	0.4977	0.8460
ABDH	0.0375	0.0303	0.9720	0.8585

in work on the graph partitioning problem. We apply annealing after each complete exploration of all neighbor lists.

As recommended in the literature (see Barr *et al.* [18]), we determine the values of parameters in each of the 11 algorithms in a systematic way and use a single set of values to solve all problems (we do not fine-tune values for a single problem). The key idea is that if we can select one set of parameters for an algorithm, which works well on problems that we haven't seen yet, then we can feel confident in the robustness of the algorithm. In order to find values that are effective over different instances, we recharacterize the parameters. The original and recharacterized parameters are given in Table V. These recharacterizations are an attempt to standardize the parameters across the 11 algorithms.

We use the two-stage process of Golden *et al.* [19] to set the values of the new parameters. In Stage 1, we select three problems from TSPLIB that have different size, scale and structure (problems d198, pcb442, and u724) and use a genetic algorithm to generate a unique parameter vector for each problem that optimizes the performance of each algorithm. This leaves us with three parameter vectors. In Stage 2, we combine the three parameter vectors to produce a single vector of values that works well for all three problems, using a genetic-algorithm-based finishing heuristic. This finishing heuristic, essentially, searches over the region defined by the convex hull of the three parameter vectors. In this way, we generate a single vector of parameter values for each algorithm. The values of the parameters that we use in our computational experiments are given in Table VI. The complete details of our two-stage process are given in Pepper [20].

Each algorithm is run 25 times on each of the 29 TSPLIB problems using the single set of parameter values obtained from the genetic-algorithm-based finishing heuristic. More specifically, the seed of the random number generator changes from one run to the next. For the solution generated by each run, we compute the percent above the optimal solution in order to compare the 11 algorithms against one another.

IV. COMPUTATIONAL RESULTS

In this section, we present and discuss the results generated by 11 algorithms on the problems from TSPLIB. In our experiments, we compute the average percent above the optimal solution (average of 25 runs) for each algorithm on each problem. Next, we average over the problems. In Table VII, we display four different averages and record the total running time (in hours for all 25 runs on all 29 problems).

In Table VII, over all 29 problems, we see that SA was the most accurate algorithm (see Average row). On average, SA generated solutions that were 3.09% above the optimal solutions. Three demon algorithms performed nearly as well as SA. On average, RABD, ABDH and ABD generated solutions that were 3.24%, 3.76% and 3.81% above the optimal solutions, respectively. RBD and RRT performed poorly with solutions that averaged 7.66% and 6.78% above the optimal solutions, respectively.

Average accuracies in the range of 3.1% to 3.8% for SA, RABD, ABDH, and ABD are comparable to results on the same 29 problems reported by Jünger, *et al.* [1] for three-opt (one three-opt run from a nearest neighbor starting solution for each problem generated solutions that were, on average, 3.80% above the optimal solutions) and Coy *et al.* [15] for their sequential smoothing algorithm (one sequential smoothing run from a greedy starting solution for each problem generated solutions that were, on average, 3.46% above the optimal solutions).

Over all 29 instances, RBD had the shortest total running time (24.67 hours), followed by RABD (29.47 hours) and SA (33.33 hours). ADH had the longest total running time (61.75 hours).

TABLE VII
AVERAGE PERCENT ABOVE THE OPTIMAL SOLUTION (AVERAGE OF 25 RUNS FOR EACH PROBLEM) AND TOTAL RUNNING TIME (IN HOURS)

Performance Measure	SA	TA	RRT	BD	RBD	AD	RAD	ABD	RABD	ADH	ABDH
SAverage[1]	2.76	5.37	4.22	5.26	4.33	3.24	2.82	2.65	2.63	2.97	2.69
MAverage[2]	3.25	4.18	6.79	4.44	9.38	3.27	4.38	2.77	3.64	2.95	2.89
LAverage[3]	3.70	9.95	13.96	7.73	13.59	10.40	10.94	9.15	4.13	9.19	8.52
Average[4]	3.09	5.75	6.78	5.40	7.66	4.49	4.76	3.81	3.24	4.03	3.76
Running Time (hrs)	33.33	41.86	33.81	53.83	24.67	58.11	47.81	35.83	29.47	61.75	36.61

[1] Average on small-size problems (lin105 to lin318)
[2] Average on medium-size problems (rd400 to rat783)
[3] Average on large-size problems (pr1002 to u1432)
[4] Average on all 29 problems

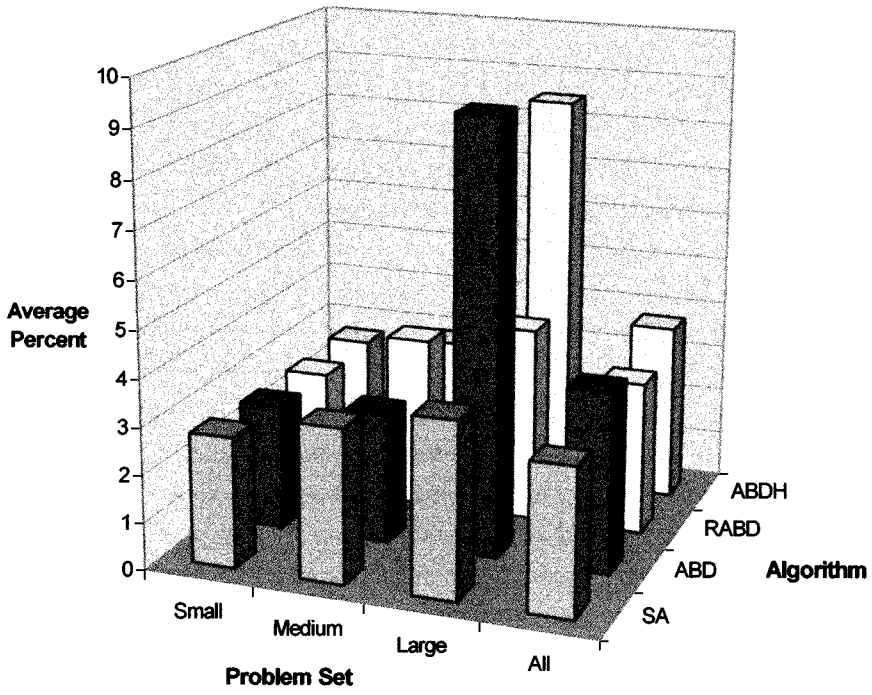

Fig. 1. Average percent above optimality for simulated annealing and three demon algorithms.

When we split the set of problems into three sets—small-size problems that have 105 to 318 nodes, medium-size problems that have 400 to 783 nodes and large-size problems that have 1002 to 1432 nodes—some interesting observations emerge (see Table VII and Fig. 1). Over the 14 small-size problems (see SAverage row), we see that RABD was the most accurate algorithm. On average, RABD generated solutions that were 2.63% above the optimal solutions. ABD, ABDH, and SA were close behind with averages of 2.65%, 2.69%, and 2.76%, respectively.

Over the ten medium-size problems (see MAverage row), we see that ABD was the most accurate algorithm. On average, ABD generated solutions that were 2.77% above the optimal solutions. ABDH was close behind with an average of 2.89%.

Over the five large-size problems (see LAverage row), we see that SA was the most accurate algorithm. On average, SA generated solutions that were 3.70% above the optimal solutions. RABD was close behind with an average of 4.13%. The other algorithms performed poorly due to the fact that they had difficulty with the two largest problems. The explanation for this is unclear and should be investigated further in future work.

V. CONCLUSIONS

Based on our extensive computational results and analyzes, SA emerged as the best of the non-demon algorithms with respect to overall average accuracy and running time. Three demon algorithms were close competitors to simulated annealing: ABD, RABD, and ABDH. In particular, RABD performs nearly as well as SA and is more than 10% faster.

If the performance of these three demon algorithms could be improved on large-size problems, they might outperform SA. This issue merits further study. A fourth demon algorithm, ADH, might also be worthy of further consideration. Unfortunately, its running time was very long (more than double RABD's time).

For demon algorithms, it is clear that our proposed variants (see Table III) outperform the earlier variants due to Wood and

Downs (see Table II). It also seems that annealing the demon value or demon mean, as well as the standard deviation of the demon value, when possible, is key to producing high-quality results that are comparable to results produced by simulated annealing and other annealing-based heuristics.

In this paper, we focused entirely on Euclidean TSPs. Future work should investigate the behavior of demon algorithms on TSPs where distances are random, rather than Euclidean. Although there are still many unexplored issues, we have shown, for now, that demon algorithms can solve Euclidean TSPs reasonably well and relatively quickly. They are easy to implement and have few parameters to fine tune. The most complicated variant requires no more than a few dozen lines of computer code.

REFERENCES

[1] M. Jünger, G. Reinelt, and G. Rinaldi, "The traveling salesman problem," in *Network Models*, M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, Eds. Amsterdam, The Netherlands: North-Holland, 1995, pp. 225–330.

[2] C. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*. New York: Halsted, 1993.

[3] I. Osman and J. Kelly, "Meta-heuristics: An overview," in *Meta-Heuristics: Theory and Applications*, I. Osman and J. Kelly, Eds. Boston, MA: Kluwer, 1996, pp. 1–21.

[4] M. Creutz, "Microcanonical Monte Carlo simulation," *Phys. Rev. Lett.*, vol. 50, pp. 1411–1414, 1983.

[5] I. Wood and T. Downs, "Fast optimization by demon algorithms," in *ACNN '98: 9th Australian Conf. Neural Networks*, 1998.

[6] ——, "Demon algorithms and their application to optimization problems," in *IEEE World Congr. Computational Intelligence*, 1998, pp. 1661–1666.

[7] D. Johnson and L. McGeoch, "The traveling salesman problem: A case study," in *Local Search in Combinatorial Optimization*, E. Aarts and J. K. Lenstra, Eds. London, U.K.: Wiley, 1997, pp. 215–310.

[8] S. Carlson, "Algorithm of the gods," *Sci. Amer.*, pp. 121–123, March 1997.

[9] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[10] G. Dueck, "New optimization heuristics: The great deluge algorithm and the record-to-record travel," *J. Computat. Phys.*, vol. 104, pp. 86–92, 1993.

[11] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *J. Computat. Phys.*, vol. 90, pp. 161–175, 1990.

[12] G. Reinelt, "TSPLIB—A traveling salesman problem library," *ORSA J. Comput.*, vol. 3, pp. 376–384, 1991.

[13] S. Coy, B. Golden, and E. Wasil, "A computational study of smoothing heuristics for the traveling salesman problem," *Eur. J. Oper. Res.*, vol. 124, pp. 15–27, 2000.

[14] S. Coy, B. Golden, G. Runger, and E. Wasil, "See the forest before the trees: Fine-tuned learning and its application to the traveling salesman problem," *IEEE Trans. Syst., Man, Cybern. A*, vol. 28, pp. 454–464, 1998.

[15] S. Coy, B. Golden, E. Wasil, and G. Runger, "Solving the TSP with sequential smoothing," in *Proc. 2nd Int. Conf. Computational Intelligence and Neuroscience*, 1997, pp. 280–283.

[16] S. Lin, "Computer solutions of the traveling salesman problem," *Bell Syst. Tech. J.*, vol. 44, pp. 2245–2269, 1965.

[17] D. Johnson, C. Aragon, L. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; Part I, Graph partitioning," *Oper. Res.*, vol. 37, pp. 865–892, 1989.

[18] R. Barr, B. Golden, J. Kelly, M. Resende, and W. Stewart, "Designing and reporting on computational experiments with heuristic methods," *J. Heuristics*, vol. 1, pp. 9–32, 1995.

[19] B. Golden, J. Pepper, and T. Vossen, "Using genetic algorithms for setting parameter values in heuristic search," in *Intelligent Engineering Systems Through Artificial Neural Networks*. New York: ASME, 1998, vol. 8, pp. 239–245.

[20] J. Pepper, "A Computational study of annealing-based heuristics for the traveling salesman problem," M.A. thesis, Univ. Maryland, College Park, 1999.

Joshua W. Pepper received the B.A. degree in mathematics from Randolph-Macon College, Ashland, VA, and the M.A. degree in applied mathematics (operations research) from the University of Maryland, College Park.

He is now with The MITRE Corporation, McLean, VA. His research focuses on solving combinatorial problems with heuristic approaches.

Bruce L. Golden is the France-Merrick Chair in Management Science at the Robert H. Smith School of Business, University of Maryland, College Park. He is Editor-in-Chief of *Networks* and previously served as Editor-in-Chief of the *INFORMS Journal on Computing*.

Dr. Golden has received numerous awards while at the University of Maryland, including the Distinguished Scholar-Teacher Award.

Edward A. Wasil is a Professor of Management Science in the Kogod School of Business at American University, Washington, DC, where he has taught courses in managerial statistics, production and operations management and operations research for sixteen years. He serves as the Feature Article Editor of the *INFORMS Journal on Computing* and conducts research in applied data mining and network optimization.