J Heuristics
DOI 10.1007/s10732-009-9104-8

The effective application of a new approach to the generalized orienteering problem

John Silberholz · Bruce Golden

Received: 1 January 2008 / Revised: 8 January 2009 / Accepted: 11 February 2009

Abstract The Orienteering Problem (OP) is an important problem in network optimization in which each city in a network is assigned a score and a maximum-score path from a designated start city to a designated end city is sought that is shorter than a pre-specified length limit. The Generalized Orienteering Problem (GOP) is a generalized version of the OP in which each city is assigned a number of scores for different attributes and the overall function to optimize is a function of these attribute scores. In this paper, the function used was a non-linear combination of attribute scores, making the problem difficult to solve. The GOP has a number of applications, largely in the field of routing. We designed a two-parameter iterative algorithm for the GOP, and computational experiments suggest that this algorithm performs as well as or better than other heuristics for the GOP in terms of solution quality while running faster. Further computational experiments suggest that our algorithm also outperforms the leading algorithm for solving the OP in terms of solution quality while maintaining a comparable solution speed.

Keywords Generalized orienteering problem · Heuristics

1 Introduction

The orienteering problem (OP) is a well established problem in combinatorial optimization. In this problem, there is a set of n nodes or cities, V, and each node i has an associated non-negative score $S(i)$. If a city is visited on a route, then its score is gathered (but visiting a city more than once does not yield additional scoring). Hence, the score associated with a path visiting a set of nodes N is $S_N = \sum_{i \in N} S(i)$. Algorithms for the OP seek the path from a defined source node (*init*) to a defined

J. Silberholz · B. Golden (✉)
R.H. Smith School of Business, University of Maryland, College Park, MD 20742, USA
e-mail: bgolden@rhsmith.umd.edu

destination node (*end*) that yields the highest score while not exceeding a pre-defined distance limit, d_{lim}.

The generalized orienteering problem (GOP) differs from the OP in the way in which total score is calculated. For the GOP, each city i is assigned m attribute scores, $S_1(i), S_2(i), \dots, S_m(i)$. Any function of these attribute scores can then be used to determine a final score for a path. Hence, the GOP is more flexible than the OP. Though, of course, any function to calculate the score of a path containing a set of nodes N would be acceptable in the generalized version of the OP, we chose to use the function presented in Wang et al. (2008) for computational tests. This function inputs a weight W_i for each attribute i, such that $\sum_{i=1}^{m} W_i = 1$. For a group of nodes N, the score of a path visiting these nodes is defined as $S_N = \sum_{i=1}^{m} W_i [\sum_{j \in N} \{S_i(j)^k\}]^{1/k}$ for some non-negative exponent k. As k approaches infinity, the value of this function approaches the sum of the maximum scores attained by members of N for each of the attributes. When $k = 1$ and $m = 1$, we have the OP.

The function chosen for analysis is an instance of the submodular orienteering problem (SOP), a problem for which each subset of nodes in a graph is assigned a score based on a function f. f is considered a monotone submodular function if whenever A and B are subsets of the nodes and $A \subseteq B$, then $f(A \cup \{v\}) - f(A) \geq f(B \cup \{v\}) - f(B)$ for any node v and $f(A) \leq f(B)$. In Chekuri and Pál (2005), an algorithm is presented to solve the SOP and theoretical results are proven about this algorithm. Though the function chosen for this paper is an instance of the SOP, it is important to note that not all GOP functions will be SOP functions.

The GOP has many applications in the field of routing. There have been a wide range of applications established for the OP in this field, and many of these applications are actually better suited for the GOP due to the latter's generalized nature. For instance, in Golden et al. (1987), the authors describe an application of the OP to the delivery of home heating fuel. In this application, utility managers would assign each customer a score based on their urgency of need for home heating fuel and would select a subset of customers to serve based on need and geography while adhering to supply limitations. Urgency would take into account each customer's tank size as well as historical and seasonal rates of usage. Further, a company might consider how long a household has been a customer—more loyal heating fuel users should gain preference. By combining these factors into a single objective function based on its preferences and then using the GOP, the heating fuel company could make a better decision about which customers to serve.

There have been several heuristic approaches proposed for the generalized orienteering problem. The first is a four-phase heuristic proposed in Ramesh and Brown (1991). In this approach, the authors took a four-phase approach of vertex insertion, cost improvement, vertex deletion, and maximal insertions. In Wang et al. (1996), the authors took a different approach, solving the GOP using an Artificial Neural Network (ANN) and testing on a dataset representing 27 cities in China. Wang et al. (2008) and Geem et al. (2005) presented a genetic algorithm and a harmony search procedure, respectively, to solve the GOP, and each limited testing instances to the dataset representing Chinese cities.

There have been a large number of heuristics proposed for the OP. One of the first was a stochastic algorithm due to Tsiligirides (1984). Particularly effective have been a heuristic presented in Chao et al. (1996) that focused on record-to-record improvement and a tabu search procedure presented in Gendreau et al. (1998). The former outperformed most other leading OP heuristics on instances containing up to 66 nodes. The latter performed well on larger instances, reporting near-optimal solutions on instances with as many as 300 nodes on graphs where the distance limit was small compared to the optimal Hamiltonian tour length and with up to 100 nodes on graphs where the distance limit was large compared to the optimal Hamiltonian tour length.

While there have been no effective optimal solutions published for the GOP, much work has been done in formulating quick optimal solutions for the OP. Though a number of approaches have been published, the approach that has solved the largest problems in reasonable runtimes is the branch-and-cut procedure presented in Fischetti et al. (1998). This paper defined a number of classes of instances—including many that were based on benchmark problems so others could compare results to optimal values—and solved problems with up to 500 nodes.

In this paper, we present a new approach to the GOP. In Sect. 2, we provide the details of this new heuristic. In Sect. 3, we compare our results against the most effective heuristics in the literature for the both the GOP and the OP. We close with conclusions and future directions for research in Sect. 4.

2 A two-parameter iterative algorithm

In this section, we present a two-parameter iterative algorithm approach to the GOP. This heuristic maintains a single GOP solution, iteratively applying a series of procedures to the current solution. Pseudocode for the algorithm can be found in Appendix B.

A Process P is the basis for the 2-parameter iterative algorithm. This process maintains a single solution and performs operations upon it. First, this solution is initialized as described in Sect. 2.1. Then, the solution undergoes iterative modification, as described in Sect. 2.3, until it has not undergone improvement for t iterations (t is a parameter).

This Process P is run repeatedly until a returned solution is worse than the previous solution that was returned by the process. At that point, the best solution yet encountered by the heuristic is returned.

The following sections describe the heuristic in detail.

2.1 Initialization

The current solution is initialized using a technique of iteratively appending nodes to the end of the path. Initially, the partial path contains only the starting node. Each iteration, i nodes (i is a parameter) not in the current solution are randomly selected, with repeats allowed. The destination node is not allowed in this selection. Of these selected nodes, the one that minimizes the sum of the distance between itself and the current end of the path and the distance between itself and the destination node is the one selected. This node is added to the end of the current path. The process is

Journal ID: 10732, Article ID: 9104, Date: 2009-02-28, Proof No: 1

continued either until all nodes have been added to the path or until the length of the path would exceed the distance limit if the destination node were added to the end. If the latter occurs, the destination node replaces the last node of the path, resulting in a feasible path. Otherwise, the destination node is added to the end of the path.

After this initialization, 2-opt is applied to the solution. The method of 2-opt reverses a subpath of a solution if that reversal will reduce the overall length of the solution. This method is repeatedly applied until no more 2-opt moves are available for the new solution. Finally, path tightening as described in Sect. 2.2 is applied to the new solution.

2.2 Path tightening

Path tightening is a local-search method that adds nodes to a solution when its length is less than the length limit, increasing that solution's score as much as possible. First, the score of the path with each exterior node added is calculated, and these modified scores are sorted, with nodes producing the highest-score paths at the front of the list. This list is then iterated from the front, with each node being added if it can be included without violating the length limit. Each node is added at the interior position of the solution that will result in the shortest total path length. List iteration continues until no more nodes can be added to the solution.

2.3 Iterative modification

Each iteration, the current solution is modified. First, i unique nodes are removed from the interior of the solution. Then, a modified version of path tightening, as described in Sect. 2.2, is used. In this modified version, the nodes that were just removed are given the lowest priority in the reinsertions by tightening, regardless of the score of the path that would be obtained by adding these nodes. In this way, we force the insertion of new nodes into the solution, helping combat convergence to local maxima.

After this procedure, repeated 2-opt is performed on the solution, as described in Sect. 2.1. Finally, unmodified path tightening, as described in Sect. 2.2, is performed.

3 Computational experiments

3.1 Parameters

Two parameters are used to control the two-parameter iterative algorithm's performance. The first, t, the number of iterations in Process P without improvement before termination, was set to the value of 4500. This value was one that seemed reasonable based on preliminary computational experiments. The parameter i, the number of nodes to choose from each iteration of path initialization and the number of nodes removed each iterative change, was set to be 4, a value that worked well in computational experiments.

3.2 Computational tests

In computational testing, a Systemax Venture H524 computer with 512 MB RAM and a 3.06 GHz processor was used. All source code was programmed in C. For each instance considered, the 2-parameter heuristic described in Sect. 2 was run once, with its final runtime and solution being reported.

3.3 Comparisons to GOP heuristics

For the GOP, we compared our approach with other heuristics on the dataset that has been the standard for comparison thus far—a 27-city problem in China for which each of the cities has been rated in terms of its natural beauty, historical interest, cultural value, and business opportunities. The specifics of this instance can be found in, for instance, Wang et al. (1996, 2008), Geem et al. (2005). For this dataset, we considered 5 values of k—1, 3, 4, 5, and 10. For each of these exponent values, we considered 5 different weight vectors. Four of these gave all the weight to one of the attributes, and the last gave equal (25%) weight to each attribute. Last, in accordance with the literature, we set the distance limit to 5,000 kilometers.

Table 1 provides summary results for these computational tests; complete results are found in Table 6. Each row represents the 5 instances associated with the listed k value. The columns represent the three algorithms encountered in the literature that also tested this dataset—the ANN described in Wang et al. (1996), the GA described in Wang et al. (2008), and the harmony search described in Geem et al. (2005). They are abbreviated as ANN, WGW-GA, and HS, respectively. Each cell in the table is split. The first entry is the number of instances with the listed k value for which the two-parameter iterative algorithm outperformed the algorithm listed in the column heading. The second number is the number of instances with the listed k value for which the algorithm listed in the column heading outperformed this paper's algorithm. The maximum sum of values for any cell is 5, as there were only 5 instances associated with each row of the table. Any sum less than 5 indicates that the algorithms returned identical scores for at least one of the instances. The harmony search was only tested on instances with $k = 5$, which is why most entries under its column heading are missing.

Detailed results for these computational tests can be found in Appendix A. It is interesting to note that this paper's heuristic was never outperformed by any of the other heuristics. This suggests that it is an effective approach for the GOP. However, further testing should be done on instances with more nodes to determine the effects of larger instances on the runtimes and solution qualities of the algorithms. Also, more testing might be done on the harmony search procedure so there are more points of comparison.

At the same time, the two-parameter iterative algorithm maintained fast runtimes—it averaged 0.4 seconds of runtime per instance. The attribute of instances that had the largest effect on runtime was the weight array—this paper's algorithm averaged 0.6 seconds of runtime per instance on problems with even weight distribution but only 0.4 seconds per instance on the other instances. Table 2 provides a comparison of the runtimes of the algorithms considered for the GOP. The HS is not included

Table 1 Comparison of heuristics over 27-node, 4-attribute problem (25 instances). The first entry in each cell is the number of instances with the exponent k listed in the row for which the two-parameter iterative algorithm outperformed the heuristic listed on the column heading. The second entry in each cell is the number of instances with the exponent k listed in the row for which the heuristic listed in the column heading outperformed the two-parameter iterative algorithm

k	WGW-GA	ANN	HS
1	0/0	0/0	–
3	2/0	2/0	–
4	4/0	2/0	–
5	4/0	3/0	2/0
10	0/0	4/0	–
Total	10/0	11/0	2/0

Table 2 Comparison of heuristic runtimes over 27-node, 4-attribute problem (25 instances). The first number in each cell is the runtime in seconds normalized to the hardware discussed in Sect. 3.2. The other number is the runtime in seconds on the original hardware used for testing

k	2-P IA	WGW-GA	ANN
1	0.2 (0.2)	3.3 (33.2)	5.5 (54.6)
3	0.4 (0.4)	2.8 (27.5)	6.3 (62.8)
4	0.4 (0.4)	2.3 (23.4)	5.2 (52.3)
5	0.4 (0.4)	2.5 (25.5)	5.7 (56.8)
10	0.9 (0.9)	2.4 (24.2)	6.3 (63.1)

because its paper contains no runtimes. Because the WGW-GA and ANN were both tested on an older computer than the one used to test this paper's algorithm, direct comparison of runtimes is not meaningful. However, based on the results in Dongarra (2008), it seems that conservatively assuming a factor of 10 between the speeds of the computers will allow an approximate comparison between the runtimes. This factor is used to normalize the results in Table 2.

Based on the results of Table 2, it appears that even when correcting for hardware differences, this paper's two-parameter iterative algorithm is faster than the other approaches considered for the GOP. However, it is interesting to note that the algorithm ran slowest when the value of the exponent k was the highest. This was likely caused because when the exponent is high, a disproportionate number of solutions have very similar values due to the nature of the function being considered. In general, the 2-P IA will run slower if many solutions have very similar values in a solution space.

3.4 Comparisons to OP heuristics

While comparison of the two-parameter iterative algorithm to other GOP heuristics is interesting because it is a comparison of heuristics designed for the same problem, these comparisons are not as interesting as they might have been because the dataset tested is small. As a result, we chose to compare our algorithm to OP heuristics on larger instances to gauge its flexibility and effectiveness as the number of nodes increases.

3.4.1 Comparison on small OP instances

A much-considered set of test problems for the OP was published in Tsiligirides (1984). This source describes three complete graphs, with 21, 32, and 33 nodes. As these graphs are quite small, two additional graphs, a square-shaped graph of size 66 nodes and a diamond-shaped graph of size 64 nodes were also created in Chao (1993). For each of the graphs, a number of distance limits are tested. In total, 89 instances were considered. In Chao et al. (1996), results over these instances were provided for a record-to-record improvement heuristic presented in that paper, as well as for a stochastic algorithm presented in Tsiligirides (1984) and recoded so it could be used in comparisons. By testing the two-parameter iterative algorithm's performance on these instances, we can directly compare our heuristic's performance to the performance of those heuristics.

Table 3 shows summary results over these instances; full results are found in Table 7. TA represents the stochastic algorithm from Tsiligirides (1984) and CR represents the record-to-record improvement heuristic from Chao et al. (1996). The format of this table is very similar to the format of Table 1. Each row represents a specific graph, listed based on n, the number of nodes, and *ins*, the number of distance limits tested (meaning, essentially, the number of instances represented by the row). Each cell in the table is split into two values—the number of instances in that row for which the two-parameter iterative algorithm outperformed the heuristic in the column heading followed by the number of instances for which the heuristic in the column heading outperformed this paper's algorithm on the instances. If the two numbers in a cell do not add up to the *ins* value for a row, that implies that the heuristics returned the same result for some of the instances.

Based on the results, it appears that the two-parameter iterative algorithm outperformed both the record-to-record improvement heuristic (CR) due to Chao et al.

Table 3 Comparison of heuristics over 89 OP instances based on 5 graphs. The first entry in each cell is the number of instances based on the graph listed in the row for which the two-parameter iterative algorithm outperformed the heuristic listed in the column heading. The second entry in each cell is the number of instances based on the graph listed in the row for which the heuristic listed in the column heading outperformed this paper's algorithm

Graph data		Heuristics	
n	*ins*	TA	CR
32	18	11/0	0/0
21	11	7/0	0/0
33	20	20/0	0/0
66	26	13/1	7/1
64	14	13/1	4/1
Total	89	64/2	11/2

Table 4 Comparison of heuristic runtimes over 89 OP instances based on 5 graphs. The first number in each cell is the runtime in seconds normalized to the hardware discussed in Sect. 3.2. The other number is the runtime in seconds on the original hardware used for testing

Graph Data		Heuristics		
n	*ins*	2-P IA	TA	CR
32	18	0.22 (0.22)	–	0.04 (19.46)
21	11	0.13 (0.13)	–	0.01 (5.07)
33	20	0.24 (0.24)	–	0.04 (18.39)
66	26	0.77 (0.77)	0.95 (474.75)	0.32 (158.64)
64	14	0.73 (0.73)	0.77 (383.59)	0.23 (117.02)

(1996) and the stochastic algorithm (TA) due to Tsiligirides (1984) in terms of solution quality.

The two-parameter iterative algorithm is able to produce good solutions in reasonable runtimes for these instances, as well. It averaged 0.21 seconds of runtime per instance on problems generated from the smallest three graphs and 0.75 seconds of runtime per instance on instances generated from the largest two graphs. Table 4 provides a comparison of the runtimes of the three algorithms considered. Because the record-to-record improvement heuristic (CR) and the stochastic algorithm (TA) were both tested on a Sun 4/370, an older computer than the one used to test this paper's algorithm, direct comparison of runtimes is not meaningful. However, based on the results in Dongarra (2008), it seems assuming a factor of 500 between the speeds of the computers will allow an approximate comparison between the runtimes. This factor is used to normalize the results in Table 4.

Results for some instances for the stochastic algorithm (TA) due to Tsiligirides (1984) are not provided, as they are not published for the tests on the Sun 4/370 found in Chao et al. (1996). As can be seen in the table, this paper's algorithm (2-P IA in the table) and the TA algorithm have similar normalized runtimes.

However, the normalized runtime of the record-to-record improvement heuristic (CR) due to Chao et al. (1996) is quicker than the runtime of the 2-P IA. While this is the case, the runtime of the CR seems to be increasing more quickly as problem instance size increases. On the smallest problem instances (with $n = 21$), the CR ran roughly 13 times faster than the 2-P IA. On the problem instances with $n = 32$ and $n = 33$, the CR ran roughly 6 times faster than the 2-P IA. Finally, on the problem instances with $n = 64$ and $n = 66$, the CR ran roughly 2.5 times faster than the 2-P IA. If this trend continues on larger problem instances, the 2-P IA should perform in similar or quicker runtimes than the CR on larger instances.

3.4.2 *Comparison on large TSPLib-based instances*

We also tested the two-parameter iterative algorithm on much larger instances described in Fischetti et al. (1998). In this paper, the authors described a method of converting TSPLib instances to OP instances. They used the distances from the TSPLib instances, as described in Reinelt (1991), as the distances in the OP instance and

« HEUR 10732 layout: Small Extended v.1.2 reference style: mathphys file: heur9104.tex (GIT) aid: 9104 doctopic: OriginalPaper class: spr-small-v1.1 v.2009/02/23 Prn:27/02/2009; 8:17 p. 9/23»

assigned a score to each node according to three rules. In the first rule, called Generation 1, they assigned 1 point to each node, including node 1, which is the start and finish node in each problem. The second generation technique, called Generation 2, provides pseudorandom node scoring by assigning $1 + ((7141 * (i - 1) + 73) \bmod 100)$ points to node i, assuming nodes are numbered from 1 to n. The last generation technique, called Generation 3, assigns $1 + \lfloor \frac{99 * dist_{1i}}{M} \rfloor$ points to node i, if $dist_{1i}$ is the distance from the depot (node 1) to the node i and M is the maximum distance of any node from the depot. In this scoring mechanism, designed to value nodes far from the depot, no score is associated with the depot. For each instance, the distance limit was selected as $\lfloor \frac{Opt(Pbm.)}{2} \rfloor$, where $Opt(Pbm.)$ is the shortest Hamiltonian tour for that problem.

Fischetti et al. (1998) considered all TSPLib instances ranging in size from 48 nodes to 400 nodes, creating an instance with each score generation technique for each TSPLib instance. For most instances, the branch-and-cut technique described in that paper returned an optimal solution within the allotted 5-hour runtime maximum. As there were 42 TSPLib instances considered and 3 score generation techniques for each instance, we considered a total of 126 instances of this type.

To date, the best results published on large instances have been those described in Gendreau et al. (1998), so we chose to compare our results to theirs. Using the C code tested in that paper, we were able to compare solution qualities and runtimes on the same computational platform (the one mentioned in Sect. 3.2). Detailed results of the computational tests for both heuristics can be found in Appendix A.

In Table 5, we compare percentages below optimal of each heuristic on different ranges of problem sizes. In that table, we report the results of both algorithms on all the large TSPLib-based instances. For some instances, the branch-and-cut technique used in Fischetti et al. (1998) did not return an optimal solution within a 5-hour time limit, so the authors instead listed the best result encountered after 5 hours of computation. The numbers in the table for the two-parameter iterative algorithm and tabu search represent the average percentage below the optimal solution or best solution found within 5 hours, whichever was provided, of that heuristic's results.

We note that, in general, the two-parameter iterative algorithm performed well in terms of solution quality. This can be seen in the results for larger instances. On instances with 131–200 nodes, the algorithm's error was more than 1.5% smaller that of the tabu search (TS) presented in Gendreau et al. (1998). For problems with more than 200 nodes, this error gap exceeded 3.3%.

Table 5 Comparison of the average errors from best known solution or optimal. Gendreau et al.'s tabu search (TS) and the two-parameter iterative algorithm (2-P IA) are compared over 126 instances. Instances are split into ranges based on number of nodes. In the table, n denotes the number of instances in each size range

Range	n	TS err.	TS sec.	2-P IA err.	2-P IA sec.1
≤90	24	0.45%	1.36	0.19%	0.72
91–130	42	2.14%	2.99	1.71%	2.44
131–200	33	5.13%	5.68	3.61%	6.01
201–400	27	9.94%	19.53	6.62%	21.28

At the same time, the runtimes of the two algorithms were comparable, even for the largest instances. The 2-parameter iterative algorithm executed in slightly shorter runtimes for small instances, while the TS was slightly quicker for larger datasets. However, the difference in average runtime per instance was less than 2 seconds even for the largest instances tested. We can make this direct comparison of the runtimes because the algorithms considered were coded in the same language, compiled by the same compiler with the same compiler flags, and run on the same computer.

Of the different score generation attributes of the TSPLib-based instances considered, the two-parameter iterative algorithm performed the best on instances created using Generation 2 (the random score generation) and worst on instances created using Generation 1 (where each city is assigned score 1). The algorithm averaged 3.85% error on Generation 1 instances, 2.15% error on Generation 2 instances, and 2.92% error on Generation 3 instances.

The relatively weak performance on the Generation 1 instances makes sense in the context of the heuristic, however, as graphs in which each node's addition would be equally advantageous in terms of score are pathological for the two-parameter iterative algorithm. In the tightening phase of the algorithm, as described in Sect. 2.2, nodes that would add the most to the score of the current solution are greedily added to the current solution. However, in graphs with score distributions created using Generation 1, every node not in the current solution is equally likely to be selected, even though the closer ones would clearly be more advantageous to add than more distant ones. Thus, the path tightening local search has difficulty converging to locally optimal solutions for these types of graphs, explaining the comparatively poor results.

In the general sense, the two-parameter algorithm performs best on graphs for which nearby nodes vary in score, as it strengthens the decisions made by the tightening phase of the algorithm. The two-parameter algorithm performs worst on graphs for which nearby nodes vary little in score, as was the case in Generation 1 graphs.

3.5 Variability to seed

Due to the greedy nature of a number of the mechanisms in the 2-parameter iterative algorithm, the algorithm shows a large variability to seed. To test this variability, the algorithm was run five times on each of the large-scale TSPLib-based instances with different seeds, and the best and worst solutions of those five runs were collected. The results of these executions are presented in Table 9. Over the four ranges of problem sizes (small problems with less than or equal to 90 nodes, medium problems with 91 to 130 nodes, large problems with 131 to 200 nodes, and very large problems with more than 200 nodes), the variability to seed was directly affected by the problem size. On the small problems, the best of the five solutions averaged a 0.14% error, while the worst solution averaged a 0.66% error.

However, on larger problems, there were larger ranges between the best-of-five and worst-of-five errors. On the medium problems, the best of the five solutions averaged a 0.49% error, while the worst averaged 3.01% above the best-known solution. On the large and very large problems, the ranges were 2.65% to 5.65% and 3.61% to 7.96%, respectively.

The downside of this variability to seed is that a single run of the algorithm could span a range of error values, making it more difficult to predict the error of the output of a single algorithm execution. In an extreme example, on the problem pr124 with score generation 3, one of the five executions of the algorithm yielded a solution with an error of 1.1%, while another execution yielded a solution with an error of 30.2%.

Because the two-parameter iterative algorithm executes quickly (in less than a minute for nearly all problem instances considered), this large variability to seed implies that running the algorithm a number of times with different seeds and taking the best result is an effective technique for improving solutions. For the very large problems considered, if the algorithm had been run 5 times with different seeds and the best result had been returned, the average error of the 2-parameter iterative algorithm would have been decreased from 6.62% to 3.61%, a sizeable improvement. Using this technique, a new best solution was found for one of the problem instances tested. For the problem pr226 with score generation 2, one of the executions of the 2-parameter iterative algorithm returned a solution of 6641, better than the solution of 6615 the branch-and-cut algorithm presented in Fischetti et al. (1998) returned after five hours of computation.

Therefore, while the two-parameter iterative algorithm's variability to seed is a liability if the algorithm is run one time for each problem instance, it can be helpful if the algorithm is run more than once and the best solution is taken.

4 Conclusions

We presented an effective algorithm for the GOP and tested it on a number of test problems. We found the heuristic to be effective on small-scale GOP and OP problems, outperforming existing approaches in a small fraction of their runtime and, therefore, demonstrating both the flexibility and effectiveness of the new approach. In computational tests on larger instances, ranging up to 400 nodes in size, we found our heuristic was effective, producing higher quality solutions than the current best algorithm for the OP in comparable runtimes.

Much work remains to be done on the GOP. Heuristics for this problem are generally only being tested on a single small dataset, so it is difficult to gauge the effectiveness of GOP heuristics as problem size increases. Additionally, the literature has focused on a single nonlinear function for optimization, but other functions should be tested on the published heuristics.

Acknowledgements We thank Dr. Frédéric Semet and Dr. Michel Gendreau for providing us with the code used in Gendreau et al. (1998) for comparison purposes.

Appendix A: Detailed computational results

In the appendix, we provide detailed results of the computational tests performed on the two-parameter iterative algorithm so that others may compare results with those presented in this paper. We first detail the testing of the 27-node GOP dataset in Sect. 1.1. Next we describe the testing of the instances presented in Tsiligirides (1984) and Chao et al. (1996) in Sect. 1.2. After, we discuss the results of testing on the TSPLib-based instances in Sect. 1.3. Last, we detail the results of variability to seed testing for this paper's algorithm on the TSPLib-based instances in Sect. 1.4.

« HEUR 10732 layout: Small Extended v.1.2 reference style: mathphys file: heur9104.tex (GIT) aid: 9104 doctopic: OriginalPaper class: spr-small-v1.1 v.2009/02/23 Prn:27/02/2009; 8:17 p. 12/23»

1.1 Detailed results for GOP testing

For the GOP dataset with 27 nodes and 4 attributes, we tested 5 values of the exponent—1, 3, 4, 5, and 10, denoted k in Table 6. For each k, five weights are tested (denoted *Wt.* in the table). The first, denoted as 0 in the table, is an even weight where each attribute is given a 25% weight. In weight $i, i \neq 0$, attribute i is given all the weight. Each instance was tested with distance limit 5,000. For the first three algorithms—the two-parameter iterative algorithm from this paper (denoted *2-P IA*

Table 6 Detailed results for 27-node, 4-attribute GOP dataset. k represents the exponent used and *Wt.* is the attribute weighing scheme used. *Sln.* represents the solutions of the heuristics for the instances and *Sec.* represents the runtimes of the heuristics in seconds

Instance		2-P IA		WGW-GA		ANN		HS
k	Wt.	Avg.	Sec.	Sln.	Sec.	Sln.	Sec.	Sln.
1	0	99.50	0.4	99.50	32.5	99.50	61.2	–
1	1	105.00	0.2	105.00	37.7	105.00	36.0	–
1	2	97.00	0.2	97.00	24.8	97.00	34.8	–
1	3	102.00	0.2	102.00	34.2	102.00	40.8	–
1	4	96.00	0.2	96.00	36.9	96.00	100.2	–
3	0	16.76	0.7	16.58	21.2	16.76	100.8	–
3	1	17.95	0.3	17.95	38.2	17.95	51.0	–
3	2	17.04	0.3	17.04	24.1	16.87	51.0	–
3	3	17.45	0.3	17.45	32.8	17.45	30.0	–
3	4	16.78	0.3	16.67	21.2	16.67	81.0	–
4	0	13.71	0.7	13.66	23.4	13.71	70.2	–
4	1	14.69	0.3	14.60	24.1	14.69	51.0	–
4	2	13.99	0.3	13.96	24.5	13.87	34.8	–
4	3	14.29	0.3	14.29	20.7	14.29	34.8	–
4	4	13.84	0.3	13.78	24.4	13.78	70.8	–
5	0	12.38	0.6	12.28	32.4	12.38	61.2	12.38
5	1	13.10	0.3	13.08	21.9	13.05	46.2	13.08
5	2	12.56	0.3	12.51	22.1	12.51	40.2	12.56
5	3	12.78	0.3	12.78	29.8	12.78	46.2	12.78
5	4	12.43	0.3	12.40	21.1	12.36	90.0	12.40
10	0	10.54	0.7	10.54	24.2	10.53	100.2	–
10	1	10.75	0.5	10.75	24.0	10.73	49.8	–
10	2	10.57	0.5	10.57	23.8	10.56	49.8	–
10	3	10.62	0.4	10.62	23.8	10.62	36.0	–
10	4	10.48	2.3	10.48	25.2	10.47	79.8	–
Computer		Systemax Venture H524		Pentium-III PC				Unreported
RT Mult.		1		10				–

Journal ID: 10732, Article ID: 9104, Date: 2009-02-28, Proof No: 1

in the table), the genetic algorithm presented in Wang et al. (2008) (denoted *WGW-GA* in the table) and the Artificial Neural Network presented in Wang et al. (1996) (denoted *ANN* in the table)—*Sln.* and *Sec.* are, respectively, the solution and seconds of runtime. The results for the 2-P IA are from this paper's research and the other results are presented in Wang et al. (2008). For the harmony search presented in Geem et al. (2005) (denoted *HS* in the table), only a solution column is provided as no runtimes were presented for that algorithm. Additionally, the algorithm was only tested on instances with $k = 5$.

At the bottom of the table, the *Computer* row denotes the computer used to test the algorithm in the column heading. The *RT Mult.* row denotes a reasonable multiplier to account for hardware differences, based on the results presented in Dongarra (2008). For instance, the multiplier of 10 in the ANN column states that we expect the hardware used to test the ANN heuristic to execute the algorithm roughly 10 times slower than we would expect the hardware described in Sect. 3.2 to execute the algorithm.

1.2 Detailed results for small-scale OP tests

In this section, we consider the testing of instances generated from graphs published in Tsiligirides (1984) and Chao (1993). The first three graphs, presented in Tsiligirides (1984), have sizes of 32, 21, and 33 nodes, respectively, and are named 1, 2, and 3, respectively, under the *Prob.* heading in Table 7. The remaining two graphs, detailed in Chao (1993), have sizes of 66 and 64 nodes, respectively. They are named 5 and 6, respectively, under the *Prob.* heading in the table. Problem 4, as defined in Chao et al. (1996), is nearly identical to problem 1, so it was not tested. For each graph, instances were generated by using a range of d_{lim} values, which are labeled in the table. In addition to the two-parameter iterative algorithm (2-P IA), we considered two other heuristics for comparison—the record-to-record improvement approach described in Chao et al. (1996) (labeled *CR* in the table) and the stochastic algorithm described in Tsiligirides (1984) (labeled *TA* in the table). For each algorithm, the *Sln.* and *Sec.* columns respectively list the solution and runtime reported for the heuristic on the labeled instance.

At the bottom of the table, the *Computer* row denotes the computer used to test the algorithm in the column heading. The *RT Mult.* row denotes a reasonable multiplier to account for hardware differences, based on the results presented in Dongarra (2008). For instance, the multiplier of 500 in the CR column states that we expect the hardware used to test the CR heuristic to execute the algorithm roughly 500 times slower than we would expect the hardware described in Sect. 3.2 to execute the algorithm.

1.3 Detailed results for large-scale OP tests

In this next section, we consider the large-scale OP instances generated from TSPLib instances using the scoring techniques described in Fischetti et al. (1998). For each TSPLib instance, labeled *Name* in Table 8, we created three OP instances, using score generation techniques Generation 1, Generation 2, and Generation 3 detailed in Fischetti et al. (1998) and Sect. 3.4.2. For each instance, the distance limit was set as half the distance of the optimal traveling salesman tour for the TSPLib instance.

Table 7 Detailed results for the 89 small-scale OP instances tested. *Sln.* labels the solutions of the heuristics and *Sec.* labels the runtime of the heuristic in seconds

Instance		2-P IA		TA	CR	
Prob.	d_{lim}	Sln.	Sec.	Sln.	Sln.	Sec.
1	5	10	0.14	10	10	0.67
1	10	15	0.18	15	15	0.80
1	15	45	0.21	45	45	2.28
1	20	65	0.30	65	65	17.49
1	25	90	0.26	90	90	9.01
1	30	110	0.26	110	110	31.92
1	35	135	0.25	135	135	25.25
1	40	155	0.25	150	155	16.76
1	46	175	0.25	170	175	21.58
1	50	190	0.25	185	190	24.91
1	55	205	0.24	195	205	24.67
1	60	225	0.23	220	225	24.28
1	65	240	0.22	235	240	23.26
1	70	260	0.21	255	260	25.09
1	73	265	0.20	260	265	25.24
1	75	270	0.19	265	270	28.53
1	80	280	0.18	270	280	26.84
1	85	285	0.17	280	285	21.71
2	15	120	0.15	120	120	1.29
2	20	200	0.11	190	200	2.24
2	23	210	0.12	205	210	4.45
2	25	230	0.12	230	230	5.65
2	27	230	0.13	230	230	6.37
2	30	265	0.14	250	265	6.18
2	32	300	0.14	275	300	7.21
2	35	320	0.14	315	320	7.81
2	38	360	0.14	355	360	6.84
2	40	395	0.13	395	395	7.14
2	45	450	0.11	430	450	0.61
3	15	170	0.23	100	170	4.37
3	20	200	0.26	140	200	5.16
3	25	260	0.26	190	260	9.40
3	30	320	0.28	240	320	9.96
3	35	390	0.27	290	390	15.38
3	40	430	0.26	330	430	18.65
3	45	470	0.26	370	470	26.84
3	50	520	0.25	410	520	28.74
3	55	550	0.24	450	550	30.27
3	60	580	0.24	500	580	27.68

Table 7 (*Continued*)

Instance		2-P IA		TA	CR	
Prob.	d_{lim}	Sln.	Sec.	Sln.	Sln.	Sec.
3	65	610	0.22	530	610	25.02
3	70	640	0.24	560	640	29.82
3	75	670	0.23	590	670	29.25
3	80	710	0.23	640	710	30.14
3	85	740	0.32	670	740	28.30
3	90	770	0.19	690	770	24.43
3	95	790	0.20	720	790	22.33
3	100	800	0.19	760	800	0.67
3	105	800	0.18	770	800	0.60
3	110	800	0.18	790	800	0.72

Instance		2-P IA		TA		CR	
Prob.	d_{lim}	Sln.	Sec.	Sln.	Sec.	Sln.	Sec.
5	5	10	0.31	10	18.1	10	1.05
5	10	40	0.38	40	34.2	40	0.46
5	15	120	0.44	100	68.2	120	4.33
5	20	205	0.57	190	151.3	195	6.17
5	25	290	0.53	290	144.3	290	73.42
5	30	400	0.55	400	188.9	400	54.82
5	35	465	0.57	460	237.2	460	32.42
5	40	575	0.85	575	288.5	575	98.92
5	45	650	0.64	645	329.3	650	58.13
5	50	730	0.63	730	373.5	730	68.05
5	55	825	0.66	820	414.9	825	65.23
5	60	915	1.26	915	461.3	915	84.59
5	65	980	0.70	980	495.2	980	82.18
5	70	1070	0.61	1070	532.4	1070	119.00
5	75	1140	0.65	1140	566.7	1140	116.70
5	80	1215	1.06	1215	598.8	1215	108.93
5	85	1270	0.68	1265	629.1	1270	132.45
5	90	1320	0.61	1340	655.5	1340	502.41
5	95	1395	1.38	1390	682.4	1380	467.13
5	100	1465	1.59	1455	711.1	1435	128.56
5	105	1520	0.89	1515	736.4	1510	316.30
5	110	1560	1.27	1550	761.4	1550	469.94
5	115	1595	0.72	1590	783.5	1595	474.64
5	120	1635	1.10	1635	807.9	1635	357.98
5	125	1670	0.68	1655	826.2	1655	268.86
5	100	1465	1.59	1455	711.1	1435	128.56
5	105	1520	0.89	1515	736.4	1510	316.30

Journal ID: 10732, Article ID: 9104, Date: 2009-02-28, Proof No: 1

Table 7 (*Continued*)

Instance		2-P IA		TA		CR	
Prob.	d_{lim}	Sln.	Sec.	Sln.	Sec.	Sln.	Sec.
5	110	1560	1.27	1550	761.4	1550	469.94
5	115	1595	0.72	1590	783.5	1595	474.64
5	120	1635	1.10	1635	807.9	1635	357.98
5	125	1670	0.68	1655	826.2	1655	268.86
5	130	1680	0.56	1670	847.3	1680	32.05
6	15	84	0.56	90	25.1	96	13.01
6	20	294	0.53	258	107.3	294	27.86
6	25	390	0.58	354	183.9	390	238.90
6	30	474	0.64	432	180.3	474	74.48
6	35	570	0.59	516	248.9	570	139.78
6	40	714	0.66	642	316.9	714	137.90
6	45	816	0.82	732	372.9	816	204.98
6	50	900	1.43	828	423.9	900	231.57
6	55	984	0.74	906	482.9	984	246.18
6	60	1062	0.67	978	527.9	1044	264.77
6	65	1116	0.69	1020	568.5	1116	232.57
6	70	1188	0.65	1110	608.4	1176	230.95
6	75	1236	0.66	1152	645.3	1224	223.12
6	80	1284	0.94	1200	678.9	1272	212.27
Computer		Systemax Venture H524		Sun 4/370			
RT Mult.		1		500			

These distances are listed as d_{lim} in Table 8. We note that the d_{lim} for problem gr229 was incorrectly listed as 1,765 in Fischetti et al. (1998). The correct value, 67,301, is listed in Table 8. For each of the generations for each instance, *Opt.* is the optimal solution published in Fischetti et al. (1998) for the instance. For the 10 instances for which the solver in Fischetti et al. (1998) reached its time limit, 5 hours, the best solution encountered in the 5 hours of computation is listed in bold. Two algorithms, the 2-parameter iterative algorithm from this paper (*2-P IA* in the table) and the tabu search from Gendreau et al. (1998) (*TS* in the table) are compared. For each score generation technique and each heuristic, the *Sln.* column represents the solution for the specified algorithm, and the *Sec.* column represents the runtime in seconds of the specified algorithm.

All heuristics in this table were tested on the same hardware, which is described in Sect. 3.2.

1.4 Detailed results for variability to seed tests

In this last section, and corresponding Table 9, we consider the variability to seed testing on the large-scale OP instances generated from TSPLib instances using the

Table 8 Detailed results for the 126 OP instances generated from TSPLib instances. d_{lim} represents the distance limit for the instance listed under *Name*, and *Opt.* is the optimal solution for a given OP instance (or the best solution returned by the solver after 5 hours of computation if in bold). *TS* is the tabu search due to Gendreau et al. and *2-P IA* is the two-parameter iterative algorithm presented in this paper. *Sln.* and *Sec.* are respectively the solution returned and runtime in seconds for a given algorithm on the specified instance

Instance		Generation 1					Generation 2					Generation 3				
Name	d_{lim}	Opt.	TS		2-P IA		Opt.	TS		2-P IA		Opt.	TS		2-P IA	
			Sln.	Sec.	Sln.	Sec.		Sln.	Sec.	Sln.	Sec.		Sln.	Sec.	Sln.	Sec.
att48	5314	31	31	0.96	31	0.34	1717	1717	0.97	1717	0.38	1049	1044	0.76	1049	0.75
gr48	2523	31	31	0.96	31	0.33	1761	1749	0.98	1756	0.46	1480	1479	0.93	1480	0.36
hk48	5731	30	30	0.87	30	0.35	1614	1614	0.76	1614	0.54	1764	1754	0.99	1764	0.41
eil51	213	29	29	0.96	29	0.84	1674	1674	0.85	1673	0.77	1399	1399	0.96	1399	0.50
brazil58	12698	46	45	1.39	46	0.47	2220	2198	1.58	2211	0.65	1702	1702	1.50	1702	0.71
st70	338	43	43	2.06	43	0.76	2286	2285	1.57	2276	0.94	2108	2079	1.60	2097	1.06
eil76	269	47	46	1.94	46	0.83	2550	2490	1.64	2540	1.14	2467	2467	2.08	2461	1.27
pr76	54080	49	49	2.10	49	0.88	2708	2708	2.04	2705	1.35	2430	2430	2.08	2430	1.25
gr96	27605	64	60	2.28	64	2.50	3425	3156	2.32	3371	2.39	3182	2965	2.59	2947	1.32
rat99	606	52	51	2.15	51	1.01	2944	2793	2.06	2924	1.90	2908	2816	2.10	2908	1.97
kroA100	10641	56	55	3.51	55	1.10	3212	3212	2.96	3186	3.27	3211	3188	3.10	3211	1.16
kroB100	11071	58	58	3.25	55	2.27	3241	3217	2.18	3237	2.19	2804	2754	2.02	2804	2.71
kroC100	10375	56	54	2.56	55	1.73	2947	2818	2.16	2924	3.57	3155	3029	2.42	3155	2.59
kroD100	10647	59	55	2.48	59	1.34	3307	3268	2.49	3299	1.64	3167	3164	2.81	3006	3.01
kroE100	11034	57	57	2.49	55	1.16	3090	3082	2.54	3012	2.10	3049	3015	2.07	3022	4.69
rd100	3955	61	60	2.75	61	1.53	3359	3359	2.47	3351	2.15	2926	2792	2.61	2924	3.26
eil101	315	64	62	2.90	63	2.08	3655	3642	2.62	3609	3.76	3345	3286	3.21	3322	2.50
lin105	7190	66	66	3.23	66	2.55	3544	3536	3.69	3533	2.62	2986	2894	2.40	2978	1.61
pr107	22152	54	54	1.43	54	0.90	2667	2667	1.60	2667	1.96	1877	1756	1.57	1756	0.99
gr120	3471	75	73	3.57	74	3.34	4371	4355	4.02	4307	1.73	3779	3673	3.56	3723	4.89

Table 8 (*Continued*)

Instance		Generation 1					Generation 2					Generation 3				
Name	d_{lim}	Opt.	TS		2-P IA		Opt.	TS		2-P IA		Opt.	TS		2-P IA	
			Sln.	Sec.	Sln.	Sec.		Sln.	Sec.	Sln.	Sec.		Sln.	Sec.	Sln.	Sec.
pr124	29515	75	75	4.06	60	1.20	3917	3917	4.15	3917	3.46	3557	3439	3.28	3557	2.31
bier127	59141	103	103	7.94	103	2.67	5383	5309	6.37	5291	4.72	2365	2351	5.45	2344	6.81
pr136	48386	71	68	2.89	68	2.69	4309	4037	3.69	4223	6.63	4390	4132	3.60	4390	5.39
gr137	34927	81	81	6.08	80	2.14	4294	4274	4.95	4246	4.81	3979	3473	4.38	3803	3.97
pr144	29269	77	73	2.81	72	2.71	4003	3890	3.46	3786	1.90	**3809**	3404	2.59	3633	2.12
kroA150	13262	86	83	5.51	82	2.95	4918	4788	5.33	4558	6.02	5039	5039	6.61	4998	4.10
kroB150	13065	87	80	4.88	85	3.57	4869	4587	4.68	4649	3.35	5314	4749	4.31	5253	6.59
pr152	36841	77	74	3.39	75	2.22	4279	4130	3.13	4243	4.59	3905	3896	3.62	3546	4.15
u159	21040	93	83	3.88	81	4.33	4960	4804	3.82	4872	7.46	5272	5056	6.95	5266	6.66
rat195	1162	102	97	7.81	99	4.95	5791	5415	5.48	5621	8.76	6195	5730	7.68	6120	13.07
d198	7890	123	116	9.73	116	3.67	6670	6530	8.62	6588	8.45	6320	6148	9.10	5916	4.84
kroA200	14684	117	102	5.37	110	8.13	6547	6200	7.54	6480	18.55	6123	6026	7.13	5790	6.65
kroB200	14719	119	109	9.36	114	8.47	6419	6098	9.07	6272	6.70	6266	6127	9.96	6138	17.62
gr202	20080	147	136	13.46	137	9.77	7848	7627	15.73	7679	8.85	8632	8271	14.64	7915	10.99
ts225	63322	**125**	121	12.47	124	9.02	6834	6491	12.90	6634	7.92	7575	7074	16.42	6719	7.36
pr226	40185	**134**	105	8.23	96	3.18	**6615**	6085	9.35	5682	9.66	6993	4486	5.65	6600	8.53
gr229	67301	176	174	27.65	171	14.80	9187	8965	24.37	9062	8.52	6347	6186	23.88	6188	30.22
gil262	1189	158	139	13.65	144	13.53	8321	7689	9.98	7831	33.91	9246	8638	14.58	8803	30.34
pr264	24568	132	132	13.53	132	8.84	6654	6654	13.60	5946	18.76	8137	4140	9.97	6994	19.09
pr299	24096	162	152	19.36	153	20.85	**9161**	8436	19.50	8846	15.43	**10358**	10057	16.68	9977	19.58
lin318	21045	205	176	33.52	193	31.09	**10900**	9233	30.04	10783	62.63	**10382**	9007	21.79	9411	19.70
rd400	7641	239	216	46.27	215	27.85	**13648**	12114	43.05	12998	75.17	**13229**	11689	36.98	12740	48.95

Table 9 Detailed variance to seed results for this paper's 2-parameter iterative algorithm for the 126 OP instances generated from TSPLib instances. *Name* is the name of the instance being tested, and *Opt.* is the optimal solution for a given OP instance (or the best solution returned by the solver after 5 hours of computation if in bold). *Wst.* is the is the worst result returned in the 5 runs on the instance, *Best* is the best of the 5 results, *Avg.* is the average over the 5 results, and σ is the standard deviation of the 5 runs. *Sec.* is the average runtime on the instance

Instance	Generation 1						Generation 2						Generation 3					
Name	Opt.	Wst.	Best	Avg.	σ	Sec.	Opt.	Wst.	Best	Avg.	σ	Sec.	Opt.	Wst.	Best	Avg.	σ	Sec.
att48	31	31	31	31.0	0.0	0.34	1717	1717	1717	1717.0	0.0	0.38	1049	1049	1049	1049.0	0.0	0.44
gr48	31	31	31	31.0	0.0	0.34	1761	1749	1750	1749.8	0.4	0.49	1480	1480	1480	1480.0	0.0	0.36
hk48	30	30	30	30.0	0.0	0.46	1614	1604	1614	1612.0	4.5	0.57	1764	1764	1764	1764.0	0.0	0.40
eil51	29	28	29	28.8	0.4	0.54	1674	1674	1674	1674.0	0.0	0.83	1399	1399	1399	1399.0	0.0	0.50
brazil58	46	46	46	46.0	0.0	0.46	2220	2211	2220	2212.8	4.0	0.60	1702	1702	1702	1702.0	0.0	0.69
st70	43	42	43	42.8	0.4	1.04	2286	2265	2286	2281.2	9.1	1.43	2108	2095	2108	2105.4	5.8	1.09
eil76	47	46	46	46.0	0.0	0.96	2550	2514	2540	2524.8	9.7	1.47	2467	2435	2462	2454.6	11.2	1.62
pr76	49	48	49	48.8	0.4	1.00	2708	2708	2708	2708.0	0.0	1.53	2430	2430	2430	2430.0	0.0	1.46
gr96	64	64	64	64.0	0.0	2.20	3425	3362	3376	3370.2	5.1	2.17	3182	2947	3145	3064.6	107.4	1.73
rat99	52	51	51	51.0	0.0	1.05	2944	2903	2926	2917.4	9.0	2.52	2908	2877	2908	2901.6	13.8	1.81
kroA100	56	55	56	55.4	0.5	1.31	3212	3190	3212	3205.2	10.0	2.70	3211	3211	3211	3211.0	0.0	1.51
kroB100	58	53	58	57.0	2.2	1.64	3241	3223	3237	3232.4	5.4	2.72	2804	2804	2804	2804.0	0.0	2.32
kroC100	56	54	55	54.6	0.5	1.71	2947	2805	2947	2864.4	73.7	2.60	3155	3149	3149	3149.0	0.0	1.63
kroD100	59	58	59	58.8	0.4	1.88	3307	3288	3299	3293.4	4.7	2.74	3167	3106	3123	3119.6	7.6	2.40
kroE100	57	55	56	55.2	0.4	1.24	3090	2989	3009	2998.4	8.3	2.80	3049	3010	3021	3013.8	4.3	2.75
rd100	61	60	61	60.8	0.4	1.76	3359	3351	3359	3354.2	4.4	2.28	2926	2911	2924	2919.2	6.6	2.62
eil101	64	62	64	62.4	0.9	1.28	3655	3562	3634	3601.4	27.5	3.23	3345	3302	3335	3327.0	14.3	2.36
lin105	66	65	66	65.4	0.5	1.49	3544	3535	3536	3535.8	0.4	2.34	2986	2986	2986	2986.0	0.0	2.04
pr107	54	54	54	54.0	0.0	0.90	2667	2667	2667	2667.0	0.0	1.61	1877	1756	1877	1784.2	52.6	1.48
gr120	75	72	74	73.6	0.9	2.78	4371	4302	4333	4316.6	11.4	2.88	3779	3701	3737	3715.8	13.9	3.58

Table 9 (*Continued*)

Instance	Generation 1						Generation 2						Generation 3					
Name	Opt.	Wst.	Best	Avg.	σ	Sec.	Opt.	Wst.	Best	Avg.	σ	Sec.	Opt.	Wst.	Best	Avg.	σ	Sec.
pr124	75	60	75	66.0	8.2	1.54	3917	3804	3917	3851.8	40.7	3.08	3557	2482	3517	3070.0	538.0	2.29
bier127	103	103	103	103.0	0.0	2.79	5383	5308	5368	5333.6	28.5	5.80	2365	2349	2355	2351.4	2.2	4.59
pr136	71	68	69	68.4	0.5	2.16	4309	4177	4234	4206.0	22.6	5.31	4390	4390	4390	4390.0	0.0	3.97
gr137	81	80	81	80.8	0.4	2.77	4294	4086	4271	4140.0	74.4	4.84	3979	3797	3947	3902.8	60.9	4.81
pr144	77	72	73	72.2	0.4	2.20	4003	3711	4003	3863.8	113.2	3.75	**3809**	3412	3634	3567.0	91.6	3.74
kroA150	86	82	85	84.0	1.2	5.02	4918	4672	4903	4766.0	117.2	7.79	5039	4987	5037	5022.6	20.6	5.97
kroB150	87	83	86	84.8	1.3	4.50	4869	4819	4853	4837.8	12.7	7.20	5314	4811	5229	5137.6	182.9	5.47
pr152	77	74	76	75.4	0.9	4.50	4279	4198	4269	4247.6	29.2	4.89	3905	2693	3089	2998.4	171.3	3.52
u159	93	80	82	80.4	0.9	3.79	4960	4675	4863	4814.4	78.7	5.77	5272	5240	5270	5259.0	12.4	6.05
rat195	102	97	98	97.6	0.5	4.99	5791	5652	5695	5669.6	16.4	11.05	6195	6054	6140	6099.8	41.4	9.91
d198	123	112	117	115.2	1.9	7.19	6670	6446	6608	6534.2	67.7	12.76	6320	5882	5950	5907.0	26.3	9.68
kroA200	117	104	110	107.8	3.0	7.70	6547	6377	6436	6404.6	21.3	17.10	6123	5818	6047	5960.8	97.5	9.23
kroB200	119	110	116	112.4	2.5	8.24	6419	6210	6349	6270.6	52.4	14.05	6266	6080	6240	6146.8	67.6	10.22
gr202	147	135	139	136.4	1.7	10.28	7848	7652	7760	7702.4	48.7	15.11	8632	7901	8206	8015.2	126.1	15.23
ts225	**125**	124	124	124.0	0.0	9.89	6834	6565	6731	6663.8	65.8	7.90	7575	6832	6954	6891.8	46.0	11.46
pr226	**134**	96	116	108.8	8.5	9.32	**6615**	5673	6641	6168.4	442.2	12.63	6993	6600	6665	6622.2	31.1	16.30
gr229	176	167	173	170.2	2.7	13.86	9187	9051	9131	9086.4	35.1	17.42	6347	6176	6225	6194.2	18.8	18.55
gil262	158	141	147	143.8	2.2	18.32	8321	7852	8144	7964.8	116.4	37.96	9246	8804	9024	8904.4	89.7	35.60
pr264	132	132	132	132.0	0.0	7.36	6654	5619	6654	6173.6	373.4	18.74	8137	6806	7828	7498.8	406.5	23.08
pr299	162	151	154	153.0	1.2	36.76	**9161**	8880	8938	8918.8	23.9	33.03	**10358**	9674	9974	9874.4	126.0	33.27
lin318	205	184	194	189.4	4.2	34.42	**10900**	10461	10755	10628.4	152.2	85.13	**10382**	9088	9876	9489.2	311.4	66.29
rd400	239	209	218	214.0	3.3	99.27	**13648**	12624	13120	12857.4	180.0	149.47	**13229**	12236	12858	12599.2	235.8	115.97

scoring techniques described in Fischetti et al. (1998). Problem instances for this testing are as described in Sect. 1.3. For each of the generations for each instance, *Opt.* is the optimal solution published in Fischetti et al. (1998) for the instance. For the 10 instances for which the solver in Fischetti et al. (1998) reached its time limit, 5 hours, the best solution encountered in the 5 hours of computation is listed in bold. The 2-parameter iterative algorithm from this paper is the only algorithm tested for these results. For each instance, the algorithm was executed 5 times. For each instance, the *Wst.* column represents the worst of the 5 solutions, the *Best* column represents the best of the 5 solutions, the *Avg.* column represents the average of the 5 solutions, the σ column represents the standard deviation of the 5 solutions, and the *Sec.* column represents the average runtime in seconds needed to obtain the solutions.

The hardware described in Sect. 3.2 was used to collect this data.

Appendix B: Pseudocode

Pseudocode for the 2-parameter iterative algorithm follows. The algorithm is based on a Process P. The framework of the heuristic follows, and we then describe Process P.

1. Get a solution S by running Process P.
2. Repeatedly complete Step 1 until the score of the new solution returned by Process P does not exceed the score of the previous solution returned.
3. Return the best solution S encountered during iteration.

Process P pseudocode follows.

1. Input: Parameters i and t, graph $G=(V,E)$, distance matrix d for which d_{ab} is the distance between vertices a and b, start node s, destination node e, distance limit l, and $score(S)$, a function that returns the score of a solution S.
2. Initialize solution S to contain the single node s.
3. While adding node e to the end of S would not cause the length of S to exceed the distance limit l.
   (a) Randomly select i nodes (with repeats allowed), s.t. each is not in S and each is not e. Store these i nodes in set L. If all nodes except e have been added to S, then add e to the end and return the final solution.
   (b) If z is the last vertex in S, then select $b \in L$ s.t. $\forall q \in L, d_{zb}+d_{be} \le d_{zq}+d_{qe}$.
   (c) Add b to the end of S.
4. Replace the last vertex in S with e.
5. While $\exists$ edges $(a,b),(c,d) \in S$ s.t. $d_{ab}+d_{cd} > d_{ac}+d_{bd}$, remove edges (a,b) and (c,d) from S and add edges (a,c) and (b,d) to S.
6. Place the vertices not in S in a list L, such that L_m is the mth element of the list. Define function $sp(S,k)=score(T)$, where T is S with vertex k inserted at arbitrary location. Insert the elements into L such that $sp(S,L_m)<sp(S,L_o)$ implies $m>o$.
7. Define set $T=\{L_m \in L, L_m \notin S : \exists (a,b) \in S :$ the length of S is less than l if edge (a,b) is removed from S and edges (a,L_m) and (L_m,b) are added to $S\}$.

8. While $|T| > 0$
   (a) Select $L_b \in T$ s.t. $b \leq j \ \forall L_j \in T$.
   (b) Select edge $(v, w) \in S$ s.t. $d_{vL_b} + d_{L_b w} - d_{vw} \leq d_{xL_b} + d_{L_b y} - d_{xy} \ \forall (x, y) \in S$.
   (c) Remove edge (v, w) from S and add edges (v, L_b) and (L_b, w) to S.
   (d) Redefine T as in Step 7.
9. Flag current solution S as the best solution discovered and set y, the number of iterations since the last improvement in the best solution, to be 0.
10. While $y \leq t$
   (a) Randomly select i unique nodes in S, each of which is not s or e, and store them in set R.
   (b) For each $a \in R$, let $b(S, a)$ be the node in S before a and let $a(S, a)$ be the node in S after a. Remove edges $(b(S, a), a)$ and $(a, a(S, a))$ and add edge $(b(S, a), a(S, a))$.
   (c) Place the vertices not in S and not in R in a list L, such that L_m is the mth element of the list. Define function $sp(S, k)$ as in Step 6. Insert the elements into L such that $sp(S, L_m) < sp(S, L_o)$ implies $m > o$.
   (d) Add the contents of R in arbitrary order to the end of L.
   (e) Repeat Steps 7 through 8 with L to complete modified path tightening.
   (f) While $\exists$ edges $(a, b), (c, d) \in S$ s.t. $d_{ab} + d_{cd} > d_{ac} + d_{bd}$, remove edges (a, b) and (c, d) from S and add edges (a, c) and (b, d) to S.
   (g) Repeat Steps 6 through 8 to complete unmodified path tightening.
   (h) If $score(S)$ is higher than the score of the best solution yet discovered, flag current solution S as the best solution discovered and set $y = 0$. Otherwise, set $y = y + 1$.
11. Output: The solution flagged as the best solution discovered.

References

Chao, I.-M.: Algorithms and solutions to multi-level vehicle routing problems. Ph.D. thesis, University of Maryland, College Park, MD (1993)

Chao, I.-M., Golden, B.L., Wasil, E.A.: A fast and effective heuristic for the orienteering problem. Eur. J. Oper. Res. **88**(3), 475–489 (1996)

Chekuri, C., Pál, M.: A recursive greedy algorithm for walks in directed graphs. In: Proceedings of the 2005 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 245–253. IEEE Computer Society, Los Alamitos (2005)

Dongarra, J.: Performance of various computers using standard linear equations software. Technical report, University of Tennessee (2008)

Fischetti, M., Salazar-González, J.J., Toth, P.: Solving the orienteering problem through branch-and-cut. INFORMS J. Comput. **10**(2), 133–148 (1998)

Geem, Z.W., Tseng, C.-L., Park, Y.: Harmony search for generalized orienteering problem: Best touring in China. In: Advances in Natural Computation. Lecture Notes in Computer Science, vol. 3612, pp. 741–750. Springer, Berlin/Heidelberg (2005)

Gendreau, M., Laporte, G., Sémét, F.: A tabu search heuristic for the undirected selective travelling salesman problem. Eur. J. Oper. Res. **106**(2–3), 539–545 (1998)

Golden, B.L., Levy, L.J., Vohra, R.: The orienteering problem. Nav. Res. Logist. **34**(3), 307–318 (1987)

Ramesh, R., Brown, K.M.: An efficient four-phase heuristic for the generalized orienteering problem. Comput. Oper. Res. **18**(2), 151–165 (1991)

Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)

Tsiligirides, T.: Heuristic methods applied to orienteering. J. Oper. Res. Soc. **35**(9), 797–809 (1984)

Wang, Q., Sun, X., Golden, B.L.: Using artificial neural networks to solve generalized orienteering problems. In: Dagli, C., Akay, M., Chen, C., Fernández, B. (eds.) Intelligent Engineering Systems Through Artificial Neural Networks, vol. 6, pp. 1063–1068. ASME Press, New York (1996)

Wang, X., Golden, B.L., Wasil, E.A.: Using a genetic algorithm to solve the generalized orienteering problem. In: Golden, B.L., Raghavan, S., Wasil, E.A. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges, pp. 263–274. Springer, New York (2008)