Evaluation of the Contract-or-Patch Heuristic for the Asymmetric TSP

Gregory Gutin and Alexey Zverovich

Abstract

In this paper we further investigate tour construction algorithms for the Asymmetric Traveling Salesman Problem (ATSP). In [1] we introduced a new algorithm, called Contract-or-Patch (COP). We have tested the algorithm together with other well-known and new heuristics on a variety of families of ATSP instances. In our study, COP has demonstrated good performance, clearly outperforming all other algorithms on robustness. It has either produced the shortest tours or came close to the leader on each of the seven families tested, while each of the remaining algorithms failed on at least two families of instances.

In this paper we introduce three new variants of the COP algorithm, and perform an extensive computational study of the original as well as new versions of the algorithm on a variety of ATSP instances. We also study the influence of the threshold parameter on the quality of tours produced by COP. We conclude the study by recommending one of the new versions of COP as a replacement for the original algorithm. The modified algorithm generally produces higher-quality tours than the original version, and has a nice property of being a much simpler algorithm. We also recommend a good universal choice of the parameter value.

Keywords: Traveling Salesman, Heuristics, Construction Heuristics

1 Introduction

In our earlier work [1] we examined some well-known tour construction algorithms for the Asymmetric Traveling Salesman Problem (ATSP)[1]. Tour construction heuristics build a tour without attempting to improve it once it is constructed. They can be used to provide starting tours for local search

[1]By the ATSP we understand the whole variety of TSP instances including symmetric instances as a special case.

algorithms, find approximate solutions of the ATSP when little time is available, etc.

In [1] we have also proposed three new algorithms, one of which, Contract-or-Patch (denoted by COP), has been shown to be a robust heuristic for a variety of classes of ATSP instances. COP combines modifications of two other new algorithms, Recursive Path Contraction (RPC), originating from [2], and Greedy Karp-Steele Patching (GKS) [1]. The latter is based on the Karp-Steele Patching heuristic (KSP) [3, 4], but considers a larger set of patching operations. The combined algorithm, which in this paper is denoted by COP/GKS, was shown to be more robust than any of well-known tour construction heuristics evaluated in [1]. This algorithm is shown (in computational experiments) to outperform the well-known tour construction heuristics for most families of instances considered in [1], and it did not fall far behind other heuristics for some special families of instances.

An unfortunate feature of COP/GKS is its considerable programming complexity, which results from the complexity of an efficient implementation of GKS as a computer code. A desire to alleviate this complexity has motivated us to analyze comparative performance of COP/GKS and a combination of COP with more simple KSP algorithm. We denote this combination by COP/KSP. Somewhat surprisingly, it turned out that both algorithms perform very similarly in terms of the quality of tours they produce and their running times. The simplicity of COP/KSP makes it a good candidate for replacing COP/GKS in most, if not all, potential applications.

We introduce an improved version of the COP algorithm, which applies a cycle patching procedure (such as KSP or GKS) at a different stage of processing. We perform a computational evaluation of this new algorithm, combined with KSP and GKS, and show that it produces higher quality tours compared to the original COP procedure.

We also perform an extensive computational study of the impact of the value of a COP parameter, the threshold, on the tour quality and running time of both versions of COP combined with both KSP and GKS, and recommend a robust choice for the value of the threshold.

The evaluation is performed on a diverse set of ATSP instances, comprising seven different families of instances.

2 Terminology and notation

The vertex set of a weighted complete digraph $\overset{\leftrightarrow}{K}_n$ (on n vertices) is denoted by $V(\overset{\leftrightarrow}{K}_n)$. The weight of an arc xy of $\overset{\leftrightarrow}{K}_n$ is denoted by $w_{\overset{\leftrightarrow}{K}_n}(x, y)$. We say that $\overleftrightarrow{K}_n$ is *complete* if for every pair x, y of distinct vertices of $\overleftrightarrow{K}_n$, both xy and yx are arcs of $\overleftrightarrow{K}_n$. The *length* of a cycle C (path P) is the number of arcs in C (P). The *Asymmetric Traveling Salesman Problem* is defined as follows: given a weighted complete digraph $\overleftrightarrow{K}_n$ on n vertices, find a Hamiltonian cycle (*tour*) H of $\overleftrightarrow{K}_n$ of minimum weight. A *cycle factor* of $\overleftrightarrow{K}_n$ is a collection of vertex-disjoint cycles in $\overleftrightarrow{K}_n$ covering all vertices of $\overleftrightarrow{K}_n$. A cycle factor of $\overleftrightarrow{K}_n$ of minimum weight can be found in time $O(n^3)$ using assignment problem (AP) algorithms for the corresponding weighted complete bipartite graph [3–5]. Clearly, the weight of the lightest cycle factor of $\overleftrightarrow{K}_n$ provides a lower bound to the solution of the ATSP *(the AP lower bound)*.

We will use the operation of *contraction* of a (directed) path $P = v_1 v_2 ... v_s$ of $\overleftrightarrow{K}_n$. The result of this operation is the weighted complete digraph $\overleftrightarrow{K}_n/P$ with vertex set $V(\overleftrightarrow{K}_n/P) = V(\overleftrightarrow{K}_n) \cup \{p\} - \{v_1, v_2, ..., v_s\}$, where p is a new vertex. The weight of an arc xy of $\overleftrightarrow{K}_n/P$ is

$$w_{\overleftrightarrow{K}_n/P}(x,y) = \begin{cases} w_{\overleftrightarrow{K}_n}(x,y) & \text{if } x \neq p \text{ and } y \neq p \\ w_{\overleftrightarrow{K}_n}(v_s,y) & \text{if } x = p \text{ and } y \neq p \\ w_{\overleftrightarrow{K}_n}(x,v_1) & \text{if } x \neq p \text{ and } y = p \end{cases}$$

Sometimes, we contract an arc a considering a as a path of length one.

We also use the operation of *patching* of two cycles C_1 and C_2, which is defined as follows: two fixed arcs x_1y_1 from C_1 and x_2y_2 from C_2 are deleted and two arcs joining the cycles together (x_1y_2 and x_2y_1) are added. The *weight* of patching of C_1 and C_2 using x_1y_1 and x_2y_2 is defined as $w(x_1, y_2) + w(x_2, y_1) - w(x_1, y_1) - w(x_2, y_2)$. This weight is the difference between the sum of the weights of the inserted arcs and the sum of the weights of the deleted arcs.

3 Algorithms Under Consideration

This section briefly outlines the original and the greedy versions of the Karp-Steele Patching algorithm, and the family of Contract-or-Patch heuristics.

3.1 Karp-Steele Patching

The Karp-Steele Patching heuristic [3,4] (KSP) uses the following procedure to build a tour:

1. Construct a cycle factor F of minimum weight.

2. Select the two longest cycles C_1 and C_2 in F, and perform a patching operation of C_1 and C_2 that has minimum weight.

3. Repeat Step 2 until F is reduced to a single cycle. Use this cycle as an approximate solution for the ATSP.

3.2 Greedy Karp-Steele Patching

The Greedy Karp-Steele Patching algorithm is based on the KSP heuristic, but considers a larger set of patching operations. It works as follows:

1. Construct a cycle factor F of minimum weight.

2. Choose the lightest patching operation possible in F, and perform it, thus reducing the number of cycles in F by one.

3. Repeat Step 2 until F is reduced to a single cycle. Use this cycle as an approximate solution for the ATSP.

The difference from the Karp-Steele Patching algorithm is that GKS considers all possible pairs of arcs as candidates for patching, while KSP only looks at arcs that belong to two longest cycles.

In [1] we have shown that, although the patching steps (Steps 2-3) have $O(n^3)$ worst-case complexity, in practice the running time of the algorithm can be significantly reduced by pre-computing and incrementally updating information on the cheapest available patching that involves each individual arc in F. The reader is referred to [1] for details.

Note that both KSP and GKS can be used as cycle patching procedures if Step 1 is excluded from both procedures, and the algorithms are supplied with a starting cycle factor F. We utilise this property in the Contract-or-Patch algorithm presented in the next section.

3.3 Contract-or-Patch

While evaluating tour construction heuristics, we observed that KSP and, to a lesser extent, GKS fail to construct good tours in situations when the solution of the assignment problem produces a large number of short cycles (especially those of length two). To address this problem we proposed an algorithm, called Contract-or-Patch (COP), which is aimed at reducing the number of such cycles. The algorithm is presented below.

1. Fix a threshold t.

2. Find a minimum weight cycle factor F.

3. If there is a cycle in F of length less than t (a *short* cycle), delete a heaviest arc in every short cycle, contract the obtained paths (the vertices belonging to long cycles are unaffected by the contraction), and go to Step 2.

4. Apply a cycle patching algorithm (for example, GKS) to transform F into a cycle C.

5. Expand the arcs of C contracted on Step 3 to obtain an approximate solution for the original problem.

In essence, the above algorithm calls a cycle patching algorithm on a smaller problem that is obtained from the original problem by recursively contracting short paths. In [1] we evaluated a combination of COP with the GKS heuristic (this combination is denoted here by COP/GKS), and have shown that it produces good results on a wide variety of ATSP instances. In this paper we also present computational results for the combination of COP with KSP (COP/KSP).

If Step 4 is omitted from the above procedure, then the algorithm produces a cycle factor that is obtained by eliminating any short cycles from the cycle factor produced on Step 1. A cycle patching algorithm (such as KSP or GKS) can then be applied to the resulting cycle factor to transform it to a tour. The procedure becomes as follows.

1. Fix a threshold t.

2. Find a minimum weight cycle factor F.

3. If there is a cycle in F of length less than t (a *short* cycle), delete a heaviest arc in every short cycle, contract the obtained paths (the vertices of the long cycles are left intact), and go to Step 2.

4. Expand the arcs of F contracted on Step 3 to obtain a cycle factor for the original problem.

5. Apply a cycle patching algorithm (for example, GKS) to transform F into a cycle C. Use C as an approximate solution for the original problem.

We denote the combinations of the above procedure with KSP (GKS, respectively) by *KSP/COP* (*GKS/COP*, respectively). Our computational results show that this modification results in an improvement in tour quality over the original version of COP.

4 Computational results

In [1] we have evaluated six tour construction algorithms for the asymmetric ATSP on a diverse set of ATSP instances. These algorithms included Karp-Steele Patching, Greedy Karp-Steele Patching, Recursive Path Contraction [2], COP/GKS, and well-known Random Insertion and Greedy heuristics [6]. In our study, COP/GKS has outperformed all other algorithms on four out of seven families of instances, producing significantly better tours than any other algorithm; it was also quite competitive on the remaining three families. Every other tested algorithm has failed on at least two out of seven families, producing tours of unacceptable quality, that ranged from 160% to 2200% above the optimum or lower bound. For comparison, even on the most difficult classes of instances among those tested, COP/GKS has consistently stayed within 50% above the lower bound, while for easier families it often produced tours within 1% of the lower bound. The reader is referred to [1] for further details on the study.

In this work we concentrate on further improving the Contract-or-Patch algorithm by studying a modified version of COP, and combinations of both the original and the modified versions with two cycle patching algorithms, KSP and GKS. We also study the impact of the COP parameter value, the threshold, on the quality of tours produced.

We have implemented four variants of the COP algorithm: COP/KSP, COP/GKS, KSP/COP and GKS/COP (see Section 3.3). Every algorithm has been tested with the value of threshold varied from 3 to 15 (note that when the threshold value is less than 3, the algorithms effectively degenerate into the corresponding cycle patching algorithms). All four algorithms have been tested on the following seven families of ATSP instances:

1. all asymmetric TSP instances from TSPLIB [7] (26 instances);
2. all Euclidean TSP instances from TSPLIB with the number of vertices not exceeding 3000 (57 instances);
3. asymmetric TSP instances with weights matrix $W = [w(i,j)]$, with $w(i,j)$ independently and uniformly chosen random numbers from $\{0, 1, 2, ..., 10^5\}$;

4. asymmetric TSP instances with weights matrix $W = [w(i,j)]$, with $w(i,j)$ independently and uniformly chosen random numbers from $\{0, 1, 2, ..., i \times j\}$;

5. symmetric TSP instances with weights matrix $W = [w(i,j)]$, with $w(i,j)$ independently and uniformly chosen random numbers from $\{0, 1, 2, ..., 10^5\}$ $(i < j)$;

6. symmetric TSP instances with weights matrix $W = [w(i,j)]$, with $w(i,j)$ independently and uniformly chosen random numbers from $\{0, 1, 2, ..., i \times j\}$ $(i < j)$;

7. sloped plane instances [8], which are defined as follows: for a given pair of vertices p_i and p_j, defined by their planar coordinates $p_i = (x_i, y_i)$ and $p_j = (x_j, y_j)$, the weight of the corresponding arc is

   $$w(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} - \max(0, y_i - y_j) + 2 \times \max(0, y_j - y_i)$$

   We have tested the algorithms on sloped plane instances with independently and uniformly chosen random coordinates from $\{0, 1, 2, ..., 10^5\}$.

For the families 3-7, we have performed a preliminary study of all algorithms on instances of sizes 500, 1000 and 2000, and have found the results to be quite similar for each of these sizes. Therefore we have decided to perform our detailed evaluation on instances of a fixed size. We have chosen 1000 as instance size, since it represents instances of moderate size and at the same time allows us to run a large number of trials in reasonable time. We varied the threshold value between 3 and 15, and for the families 3-7 the results presented below represent an average of 50 trials for every algorithm with each of the threshold values.

All tests were executed on a Pentium III 500 MHz computer with 256 MB of RAM. For the TSPLIB instances, the families 1 and 2, the results are compared to the optima. For the asymmetric instance families 3, 4 and 7 we used the AP lower bound, the weight of the lightest cycle factor. The AP lower bound is known to be of high quality for the pure asymmetric TSP [9]. For the symmetric families 5 and 6, we exploited the Held-Karp (HK) lower bound [10,11], which is known to be very effective for this type of TSP instances [12].

For the TSPLIB families 1 and 2, the tests were performed on all asymmetric instances in TSPLIB, and all Euclidean instance of not more than 3000 cities. Each algorithm was executed once with every parameter value on each of the tested instances. The average excess over optimum was then calculated in each of the families, and the results were plotted on the top two graphs of Figure 1. The overall time taken by every algorithm to solve all problems in each of the families 1 and 2 is presented on top two graphs of Figure 1.

For each of the families 3-7, fifty problem instances of size 1000 were randomly generated. All four algorithms were executed on each of these instances with every parameter value. The excess over the corresponding lower bound and the running time were then averaged over fifty trials in each family, and plotted in Figures 1 and 2 respectively.

The rest of this section is organised as follows. First, we compare the results produced by the heuristics based on the GKS algorithm to algorithms based on KSP. Then we compare the original version of COP with the modified version (see Section 3.3). Finally, we examine the impact of the threshold value on the quality of tours and execution times of all four algorithms, and make some recommendations for choosing the value of the threshold.

As it can be seen in Figure 1, the quality of tours produced by the GKS-based variants of COP is very close to the tour quality of their KSP-based counterparts. Figure 2 shows that GKS-based algorithms are usually somewhat slower that those based on KSP. Although in some cases algorithms based on GKS do offer a minor improvement in tour quality, the overall complexity of GKS may not always justify its use. We suggest that in cases when small degradation of in the quality of the tours produced by a construction heuristic can be tolerated, algorithms based on KSP can be used instead of their GKS-based counterparts.

Comparison of the original COP with its modified version reveals that the modified version generally performs slightly better, at the expense of increased execution time. The increase in the execution time of the KSP-based version of the modified COP is smaller than that of the algorithms based on GKS. Based on this, and bearing in mind simplicity of the KSP-based algorithms, we recommend KSP/COP as a replacement for the original version of COP (COP/GKS).

It can be seen in the Figures 1 and 2 that in most cases the quality of tours deteriorates, sometimes quickly, when the threshold is increased. The only cases when increasing the threshold clearly results in shorter tours are the random asymmetric families 3 and 4. For these families the tours produced even with small values of the threshold are already very close to the lower bound (within 2.2% and 1.3% of the LB). This suggests that a small value of the threshold will result in the best all-round performance.

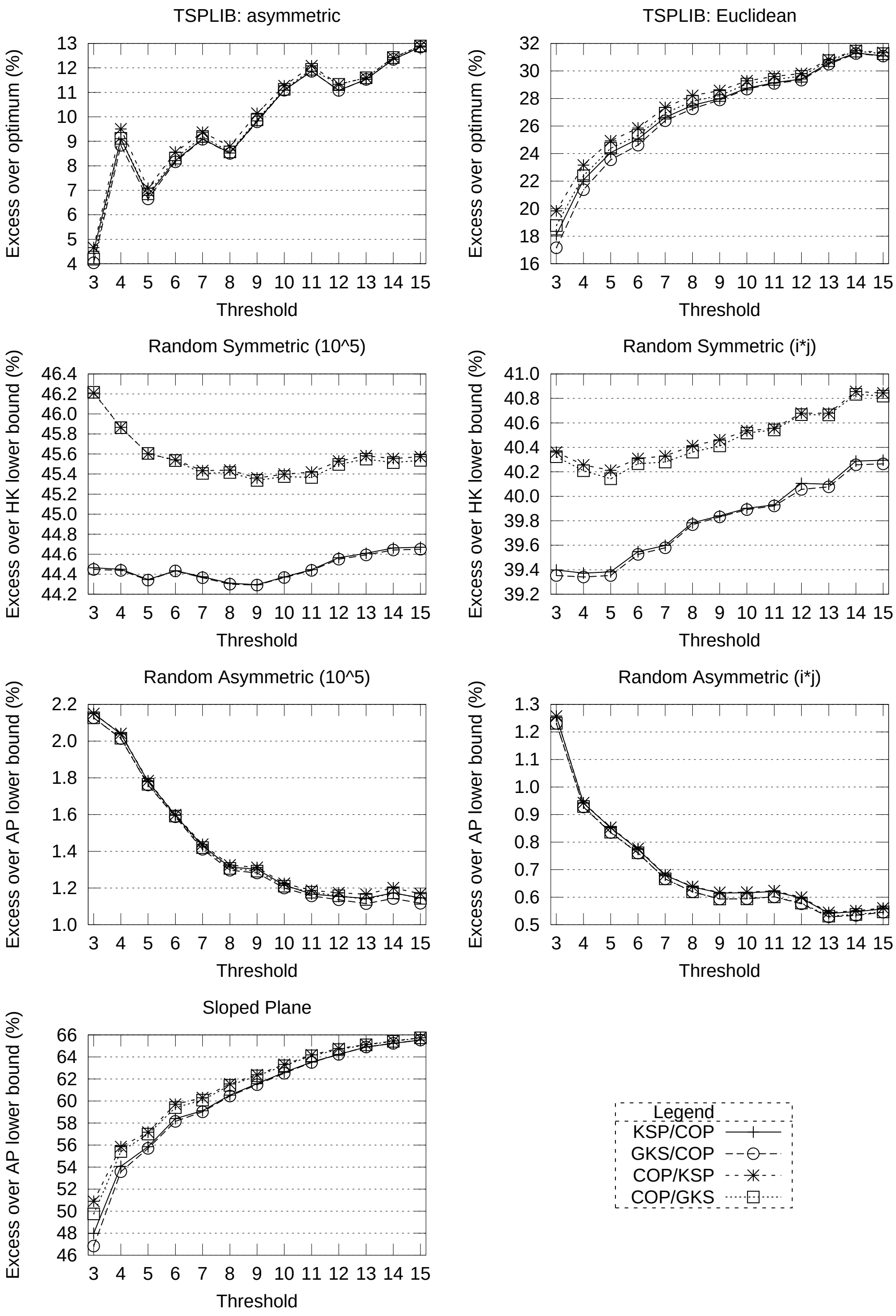

Figure 1: Tour quality

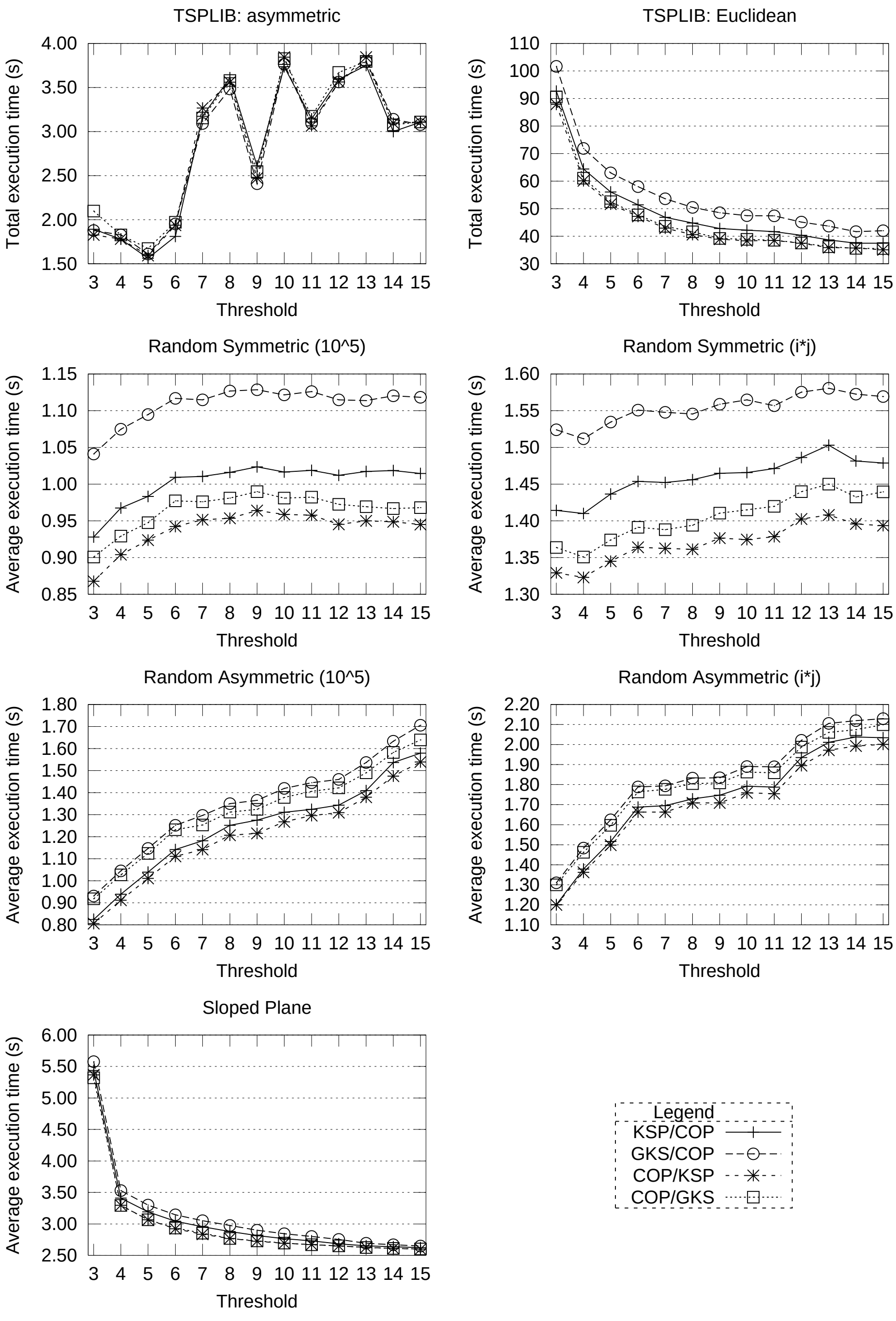

Figure 2: Execution time

We therefore recommend that the threshold value of three is used as a good universal choice.

To summarise, the KSP/COP heuristic with the threshold parameter fixed to 3 appears to be a reliable tour construction heuristic for a wide variety of families of TSP instances.

5 Conclusions

In this paper we proposed a new version of the Contract-or-Patch algorithm and have shown that it offers an improvement over the original COP heuristic in the quality of the tours it produces. We have performed an extensive computational evaluation of combinations of the original and new versions of COP with both Karp-Steele and Greedy Karp-Steele patching procedures. The results of our evaluation show that the heuristics based on KSP perform very similarly to their GKS counterparts, while being much simpler algorithms. We proposed the new version of the COP algorithm combined with Karp-Steele Patching (KSP/COP) as a replacement for the GKS-based original COP algorithm introduced in [1] (COP/GKS). We have also studied the influence of the COP parameter, threshold, on the quality of the tours produced by the algorithms and their execution times. Based on the results of our computational experiments, we proposed that the KSP/COP algorithm with the threshold value of three can serve as a good universal tour construction heuristic for the asymmetric Traveling Salesman Problem.

References

[1] F. Glover, G. Gutin, A. Yeo, and A. Zverovich. Construction heuristics for the asymmetric TSP. To appear in European Journal of Operational Research, 2000.

[2] A. Yeo. Large exponential neighbourhoods for the traveling salesman problem. Technical Report 47, Dept of Maths and CS, Odense University, Odense, Denmark, 1997.

[3] R. M. Karp. A patching algorithm for the nonsymmetric traveling-salesman problem. *SIAM Journal on Computing*, 8(4):561–573, November 1979.

[4] R.M. Karp and J.M. Steele. Probabilistic analysis of heuristics. In Lawler et al. [13], pages 181–205.

[5] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. John Wiley & Sons, 1997.

[6] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer Verlag, 1994.

[7] G. Reinelt. TSPLIB – a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[8] D.S. Johnson. Private communication, 1998.

[9] E. Balas and P. Toth. Branch and bound methods. In Lawler et al. [13], chapter 10, pages 361–401.

[10] M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

[11] M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[12] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: A case study in local optimization. In E. L. Aarts and J. K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, 1997.

[13] E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, and D.B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, 1985.