Available online at www.sciencedirect.com

SCIENCE DIRECT®

Operations Research Letters 32 (2004) 499–509

Operations Research Letters

www.elsevier.com/locate/dsw

Match twice and stitch: a new TSP tour construction heuristic

Andrew B. Kahng, Sherief Reda*

UCSD Computer Science & Engineering Department, 9500 Gilman Drive, MS 0114, La Jolla, CA 92093-0114, USA

Received 12 May 2003; accepted 14 April 2004

Abstract

We present a new symmetric traveling salesman problem tour construction heuristic. Two sequential matchings yield a set of cycles over the given point set; these are then stitched to form a tour. Our method outperforms all previous tour construction methods, but is dominated by several tour improvement heuristics.
© 2004 Elsevier B.V. All rights reserved.

Keywords: Traveling salesman problem; Heuristic; Tour construction

1. Introduction

STSP heuristics—we use TSP and STSP synonymously unless otherwise indicated—are generally classified into two categories: *tour construction* [2–4,11,22] and *tour improvement* [6,7,16,19,21]. Tour construction heuristics execute a sequence of operations until a valid tour is obtained, at which point the heuristics stop and report the constructed tour. Tour improvement heuristics start with a valid tour (e.g., the output of a tour construction heuristic [2]) and iteratively improve the tour cost, typically via local search, until some stopping criterion is reached.

While tour improvement techniques produce near-optimal tours as measured by the gap from the Held-Karp lower bound [14,15], the solution quality achieved by tour constructions is significantly worse. According to results reported by [1], the best tour constructions are approximately 8% worse than the best tour improvement methods. The proposed STSP tour construction achieves unprecedented results in narrowing this gap. We relax the tour structure constraints to allow formation of multiple cycles and then join the cycles together to form a tour. While this strategy has been pursued before for the asymmetric traveling salesman problem (ATSP) [10,18] and MAX TSP [9,12,13], it has not been applied to the traditional STSP. This is because the assignment phase from ATSP construction is ineffective in the STSP; it simply yields a minimum-cost matching, i.e., all cycles are of length two. In this paper, we propose a new way for constructing a less trivial set of cycles in the STSP, as well as new ways to compose these cycles into a tour.

- The first phase of our approach, *cycle construction*, uses two sequential matchings to construct the cycles. The first matching returns the usual minimum-cost edge set with each point incident to exactly one matching edge. We then remove all these edges and execute a second matching. The second matching effectively repeats the matching process subject to the constraint that none of the

* Corresponding author.
E-mail address: sreda@cs.ucsd.edu (S. Reda).

edges chosen in the first matching can be used again. The results of the first and second matchings together form a set of cycles.
- The second phase of our approach *stitches* the constructed cycles to form the TSP tour. We define *stitching* as the process that composes *all* cycles to form one tour, while *patching* is the process that only composes a given pair of cycles to form a larger cycle. We discuss two methods for patching a given pair of cycles. The first method is exact but slow; the second is approximate but offers a considerable speedup. Based on the costs of patching every pair of cycles, a minimum spanning tree (MST) calculation determines a way to stitch all cycles into a tour. We also report results from using the PATCH heuristic [18].

The effectiveness of our proposed tour construction methodology is confirmed by application to the TSPLIB [1] benchmarks. For example, we construct a tour for the largest instance (pla85900) that is only 1.91% over the Held–Karp (HK) bound. Our empirical results outperform all reported results of tour construction heuristics including [3] and its variants [17]. We also compare our results to a number of tour improvement heuristics: for all but a few benchmarks (e.g., u2319 and pla33810) our heuristic is dominated by some of these techniques. In the following discussion, Section 2 formulates the STSP, while Section 3 discusses Phase I (cycle construction), and Section 4 discusses Phase II (cycle stitching). Section 5 gives experimental results and closes with directions for future research.

2. TSP formulation

Let $P=\{p_1,p_2,\ldots,p_n\}$ be the given set of points, and let E be the set of edges forming a complete graph over P. For each edge $e=\{p_i,p_j\}\in E$, we are given a weight w_e that we can view as the distance between its endpoints. If $\delta(p_i)$ is the set of edges incident to point p_i, and x_e is a 0–1 variable indicating the inclusion of edge e in the tour, then the STSP problem can be formulated as

$$\min \sum_{e\in E} w_e x_e \tag{1}$$

subject to

$$\sum_{e\in\delta(p_i)} x_e = 2 \quad \forall p_i \in P \tag{2}$$

and

$$T=\{e\in E:\ x_e=1\} \text{ forms no subtours.} \tag{3}$$

The first phase of our technique constructs a set of cycles meeting objective (1) and constraint (2) but ignores constraint (3). The second phase stitches the subtours (cycles) produced by the first phase to yield a tour satisfying (3).

3. Phase I: cycle construction

The cycle construction phase produces a set of disjoint cycles $C=\{C_1,\ldots,C_m\}$ such that every point in P is incident to exactly two edges of some cycle in C. The set of edges $F=\bigcup_{i=1}^{m} C_i$ of C are typically referred to as a 2-factor. The concept of 2-factor is generalizable: a set of edges is an f-factor if every point is incident to exactly f edges.

Fact 1. The minimum-weight f-factor is computable in polynomial time ([20, Chapter 10]).

The method of [20] relies on transforming the original graph instances into a new weighted graph with O(E) new vertices, and then finding a minimum-weight matching in the new graph. A minimum-weight 2-factor can be computed in O(n^3) time [23, p. 523].

While the 2-factor is the union of a set of cycles, this does not impose any limits on the size of the cycles except that they are greater than 2. The set of cycles will better resemble a tour if we require a minimum number of points in a cycle. A 2-factor is k-restricted if none of its cycles has k or fewer points. Ultimately, a minimum-weight $(|P|-1)$-restricted 2-factor is an optimal TSP tour.

Fact 2. The problem of finding a minimum weight k-restricted 2-factor in a weighted complete graph is NP-hard for $k\geqslant 4$ [8,24].

Fact 3. The hardness status of the 3-restricted "triangle-free" minimum weight 2-factor problem remains unknown [8,12, Chapter 12].

Phase I of our approach may be viewed as constructing a heuristic 3-restricted minimum-weight 2-factor by using two sequential matchings. In the first matching, we apply minimum-weight (1-factor) matching to produce a set of edges M_1. Before applying the second matching, we remove M_1 from E, yielding a graph with edge set $E \setminus M_1$, and then reapply minimum-weight matching. The second matching effectively repeats the first matching *under the constraint that none of the edges of M_1 can be chosen.* This constraint can be practically enforced by setting the edge weights of M_1 to ∞. Let M_2 denote the set of edges in the second matching. The two sets of edges M_1 and M_2 comprise a disjoint set of cycles $C = \{C_1, \ldots, C_m\}$. That is, $M_1 \cup M_2 = \bigcup_{i=1}^{m} C_i$, such that $C_i \cap C_j = \emptyset$, $1 \leqslant i, j \leqslant m$, $i \neq j$. An important property of such cycles is stated in the following lemma.

Lemma 1. *The two sequential matchings with edges $M_1 \cup M_2$ produce a set of even-length cycles $C = \{C_1, \ldots, C_m\}$ with $|C_i| \geqslant 4$, $1 \leqslant i \leqslant m$.*

Proof. Color the edges from M_1 red and those from M_2 blue. No two edges in $M_1 \cup M_2$ that share an endpoint can have the same color by definition of matching. Thus all cycles must be even and of length at least 4.

An immediate consequence of Lemma 1 is that the two sequential matchings can be considered as a heuristic for constructing a minimum weight 3-restricted 2-factor with the additional property that all cycles are of even length. Using the two sequential matchings, we construct a set of disjoint cycles in polynomial time $O(n^3)$ for general TSP instances. For Euclidean instances, [17] reports that the observed runtime for Euclidean matching is proportional to $n^{1.25}$. For the second matching, we trap the edge weight calculation function, reporting a large number (∞) whenever this function is called with an edge from M_1. There is no noticeable increase in observed runtime due to this modification. The cycles formed from the two matchings are stitched together into one tour as described in the next section.

4. Phase II: cycle stitching

Recall that the set of edges of the two matchings form a disjoint set of cycles (subtours). In this section we investigate ways to *stitch* the cycles of C to form one tour T. First, we extend the concept of *patching* as practiced in the ATSP domain to the domain. Then, we give two algorithms that patch any given pair of cycles. These two algorithms represent a trade-off between patching runtime and solution quality. We then give a minimum spanning tree (MST) based approach for stitching all cycles into a single tour, and finally present the overall outline of the stitching process.

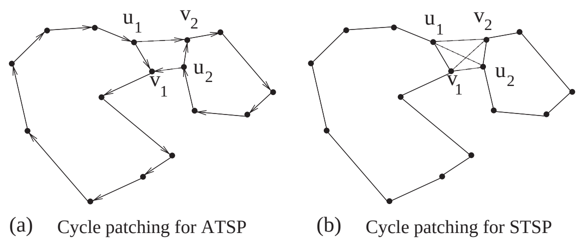

Fig. 1. Patching for ATSP versus patching for STSP.

4.1. Patching a pair of cycles in STSP

Suppose that we wish to compose cycles C_i and C_j into one larger cycle by eliminating edges $e_1 = \{u_1, v_1\} \in C_i$ and $e_2 = \{u_2, v_2\} \in C_j$. In the ATSP domain, the two cycles can be composed in only one way: this is traditionally called *patching* and is illustrated in Fig. 1a. The cost of patching the two cycles at the specified edges is

$$p(e_1, e_2) = d(u_1, v_2) + d(u_2, v_1) - d(u_1, v_1) - d(u_2, v_2). \quad (4)$$

While cycle patching in the ATSP offers only one way to compose two cycles, the STSP offers two ways as shown in Fig. 1b, with patching cost

$$s(e_1, e_2) = \min(d(u_1, v_2) + d(v_1, u_2), d(u_1, u_2) + d(v_1, v_2)) - d(u_1, v_1) - d(u_2, v_2). \quad (5)$$

We note that in Euclidean instances a non-intersecting patch always has the least cost.

4.2. Methods for patching a pair of cycles

Given two cycles C_a and C_b to be patched, we seek to determine two edges $e_{a_i} \in C_a$ and $e_{b_j} \in C_b$ that minimize the patching cost as given by Eq. (5). We give

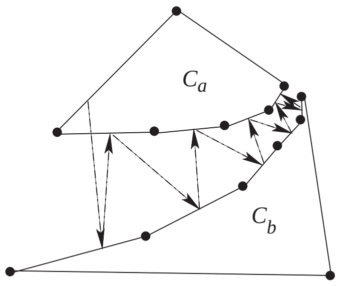

Fig. 2. A case where alternating patching almost alternates between all edges of the two cycles.

two algorithms that permit a trade-off between patching cost and runtime.

1. Exact patching: This straight-forward method evaluates the patching cost for every pair of edges $e_{a_i} \in C_a$, and $e_{b_j} \in C_b$, with $1 \leqslant i \leqslant |C_a|$ and $1 \leqslant j \leqslant |C_b|$. While this method gives the minimum-cost patching of two cycles, it has quadratic runtime of $O(|C_a||C_b|)$.

2. Alternating patching: This method starts with an arbitrary edge $e_{a_1} \in C_a$ and finds an edge $e_{b_1} \in C_b$ that minimizes the patching cost to e_{a_1} as given by Eq. (5). We refer to e_{b_1} as the *closest* edge to e_{a_1}. Ties are broken by choosing the lexicographically first edge, i.e., the edge with the smallest index in the cycle. Denote the operation of finding a closest edge by the operator $B(\cdot,\cdot)$, i.e., $B(e_{a_1}, C_b) = e_{b_1}$. We proceed to find the closest edge $e_{a_2} \in C_a$ to e_{b_1}, i.e., $B(e_{b_1}, C_a) = e_{a_2}$. As shown in Fig. 2, the method alternates between edges in cycles C_a and C_b, always decreasing the patching cost, until $B(B(e_{a_l}, C_b), C_a) = e_{a_l}$. At that point, the algorithm patches C_a and C_b using the edges e_{a_l} and $B(e_{a_l}, C_b)$ and terminates. We next prove that this procedure stops after a finite number of alternations.

Lemma 2. *The alternating patching heuristic terminates within* $O(|C_a||C_b|)$ *runtime.*

Proof. We first show that $B(\cdot,\cdot)$ can be executed at most once for any edge e_{a_i} in C_a. (The same argument also applies to any edge e_{b_i} in C_b). If e_{a_i} is the final edge examined by the algorithm before terminating then $B(e_{a_i}, C_b)$ is trivially executed once. Toward a contradiction, assume that $B(\cdot,\cdot)$ is executed twice for some non-final edge e_{a_i}, hence there exist 3 edges in C_b: $e_{b_{i-1}}$, e_{b_i} and e_{b_j} with $j > i$, such that $B(e_{b_{i-1}}, C_a) = e_{a_i}$, $B(e_{a_i}, C_b) = e_{b_i}$, and $B(e_{b_j}, C_a) = e_{a_i}$. This implies $s(e_{a_i}, e_{b_j}) < s(e_{a_i}, e_{b_i})$, a contradiction by the definition of the cost function. Hence $B(\cdot,\cdot)$ can be executed at most once for any edge. Since each call of $B(\cdot.\cdot)$ for an edge $e_{a_i} \in C_a$ (resp., $e_{b_i} \in C_b$) takes $O(|C_b|)$ (resp., $O(|C_a|)$) to execute, the total patching runtime is $O(|C_a||C_b|)$.

In the worst case, the alternating patching heuristic would need to consider all pairs of edges (a pair consists of an edge from C_a and an edge from C_b) before stopping. Our experimental results indicate that in reality the alternating patching heuristic hardly performs according to the worst case. On the other hand, Fig. 2 shows the worst-case scenario where the alternating patching heuristic keeps on alternating between edges of the two cycles until most edges are considered for patching.

4.3. MST based stitching

Given a set of cycles $C = \{C_1, C_2, \ldots, C_m\}$ produced in Phase I, we must stitch all of these cycles together to form a tour satisfying the no-subtour constraint given in Eq. (3). Stitching represents the cumulative result of patching pairs of cycles. Let c_{ij} be the patching cost of cycles C_i and C_j as calculated by either the exact or the alternating method. We construct a graph $G = (V, E)$, where vertices $v_1, v_2, \ldots, v_m \in V$, respectively represent cycles $C_1, C_2, \ldots C_m \in C$. We connect every pair of vertices $v_i, v_j \in V$ by an edge $\{v_i, v_j\}$ having weight c_{ij}. The stitching objective is achieved by constructing a MST of G. The outline of our stitching algorithm is given in Fig. 3.

In this algorithm, Lines 1–3 calculate the patching cost between every possible pair of cycles using either EXACT (Exact patching) or FAST (Alternating patching). Line 4 constructs the MST. Lines 5–6 carry out the stitching process by examining the MST. If there are two cycles to be patched and any of them has been already patched before, we re-calculate the patching cost between the two cycles, as given in Step 6b. We keep track of the patched cycles by using the function $\pi(\cdot)$. If $\pi(i) = i$, where i is the cycle index, then the cycle has never been patched before; otherwise, it has been patched and its new patched cycle index is $\pi(i)$. The intuition behind re-calculating the new cost is that perhaps the newly stitched cycle(s) offer better patching cost. Another alternative we have

```
Input: Set of cycles $C = \{C_1, C_2, \ldots, C_m\}$, patching mode $mode$
Output: Tour $T$

1. If $mode$ is EXACT then
      Calculate the patching cost $c_{ij}$ for every pair of cycles $\{C_i, C_j\} \in C \times C$ using exact patching
   else if $mode$ is FAST then
      Calculate the patching cost $c_{ij}$ for every pair of cycles $\{C_i, C_j\} \in C \times C$ using alternating patching
2. Construct the complete graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_m\}$ corresponding to
   the set of cycles $C = \{C_1, C_2, \ldots, C_m\}$ and $E = V \times V$
3. For every edge $\{v_i, v_j\} \in E$, let weight $w(\{v_i, v_j\}) = c_{ij}$
4. Find the MST of $G$, and let $M$ be the edges of the MST
5. For $i = 1 \ldots m$: Let $\pi(i) = i$
6. For each edge $\{v_i, v_j\} \in M$ in increasing-weight order order:
   a. Let $l$ be the minimum of $i$ and $j$
   b. If $\pi(i) = i$ and $\pi(j) = j$ then
         Construct a new cycle $C_{\pi(l)}$ by patching $C_{\pi(i)}$ and $C_{\pi(j)}$ using the pre-computed
   patching of cost $c_{ij}$
      else
         Construct a new cycle $C_{\pi(l)}$ by patching $C_{\pi(i)}$ and $C_{\pi(j)}$ after having recomputed the
   best patching of
         $C_{\pi(i)}$ and $C_{\pi(j)}$ and the associated cost
   c. For $k = 1 \ldots m$: If $\pi(k) = \pi(i)$ or $\pi(j)$ then $\pi(k) = \pi(l)$
7. Return $T = C_1$
```

Fig. 3. Overall stitching algorithm.

tried is to re-construct the MST every time two cycles get patched. This basically converts the MST-based stitching algorithm into a greedy one. We have found negligible improvements in quality with this alternative, at the expense of increased runtime. However, we do resort to greedy stitching in some situations, as will be explained in Section 5.

For complexity analysis, we notice that constructing G requires calculation of patching cost between all pairs of cycles, which takes $\sum_{i=1}^{m} |C_i| \sum_{j=1, i \neq j}^{m} |C_j| = \sum_{i=1}^{m} |C_i|(n - |C_i|) = O(n^2)$ time. Stitching all cycles is achieved by constructing the MST of G. Constructing the MST takes $O(m^2)$ using Prim's algorithm, where $m \leqslant n/4$. The overall runtime of the stitching phase is $O(n^2)$.

In addition to the MST-based heuristic, we can also use the PATCH heuristic [18] for stitching all cycles. In this heuristic, the two largest cycles are repeatedly patched until one tour is formed. We try this heuristic using both exact and alternating patching in the next section.

5. Practical considerations and experimental results

The overall TSP construction methodology is presented in Fig. 4. Since a perfect matching can be attained only if the number of points is even, i.e., $|P|\%2 = 0$, we must drop one point p_{rn} from P when $|P|$ is odd. Steps 1 and 2 drop one point randomly. For reproducibility of our results, we drop the *last* point in any TSPLIB benchmark suite having odd size (we later examine how to re-incorporate this point). Step 3 executes the Euclidean matching algorithm [5] of Cook and Rohe, and stores the resultant matching in

```
Input: Set of points $P$
Output: Tour $T$
1. Let $r = |P|\%2$
   // if number of points is odd
2. If $r \neq 0$ then
   a. Choose a random point $p_{rn}$ from $P$
   // remove this point temporarily from the pointset
   b. Let $P = P - \{p_{rn}\}$
3. Let $M_1$ be the result of executing minimum weight matching
4. Set all weights of edges in $M_1$ to $\infty$
5. Let $M_2$ be the result of executing minimum weight matching
6. Construct the cycles $C$ using $M_1$ and $M_2$
   // if number of points is odd, re-insert the removed point
7. If $r \neq 0$ then
    Find the best cycle $C_b \in C$ where augmenting $p_{rn}$ to $C_b$ yields the minimum increase in
   the total distance
8. Let $T$ be the result of executing the stitching algorithm of Figure 3
9. Return $T$
```

Fig. 4. Overall TSP construction methodology.

M_1. Step 4 sets all weights of M_1 to ∞. This can be enforced in Rohe's code by trapping the edge length calculation function and returning a large number whenever an edge in M_1 is passed to the function as an argument. Step 5 re-executes the Euclidean matching procedure under the previous constraint. The result of Step 5, M_2, together with M_1 comprise the edges of a disjoint set of cycles constructed in Step 6. If $|P|$ is odd, then Step 7 reinserts the randomly dropped point p_{rn} into one of the constructed cycles. The cycle and position of the augmentation are chosen such that minimum increase in total cycle length is attained. This is realized by examining every pair of consecutive points in the cycles and calculating the increase in cost due to insertion of p_{rn} between the points of this pair. Then, p_{rn} is inserted between the pair that attains the smallest increase in tour cost. Finally, Step 8 invokes the stitching algorithm of Fig. 3. If the MST is too large to be allocated in memory, the stitching process is switched into a greedy mode where each cycle is stitched with the best cycle to stitch. This stitching is repeated until there is only one cycle left. The stitching algorithm executes in either the exact stitching (ES) mode or the fast alternating stitching (AS) mode. For Euclidean instances, the observed matching runtime is proportional to $n^{1.25}$ and hence the total observed complexity is proportional to n^2. In general, the total complexity is $O(n^3)$ as it is dominated by the matching calculation.

Our implementation platform is an Intel Xeon 1.4 GHz processor with 2 GByte RAM. We use gcc version 2.96 with -O3 optimization, integrating Cook and Rohe matching code [5] as well. We execute our tour construction heuristic on the TSPLIB Euclidean benchmarks. While our technique applies to both Euclidean and non-Euclidean instances, we restrict our experiments to the Euclidean benchmarks. Since we have two different stitching algorithms: (i) the MST-based computation and (ii) the PATCH heuristic, as well as two patching methods: (i) exact and (ii) alternating, we have a total of four heuristic variants to evaluate. These are described in Table 1. We use

Table 1
Description of the different variants of the MTS heuristic

Flow	Description
MTS-1	Sequential matchings to form cycles + PATCH heuristic using alternating patching
MTS-2	Sequential matchings to form cycles + PATCH heuristic using exact patching
MTS-3	Sequential matchings to form cycles + MST stitching heuristic using alternating patching
MTS-4	Sequential matchings to form cycles + MST stitching heuristic using exact patching

Table 2
Results of all variants of the Match Twice and Stitch (MTS) heuristic on the TSPLIB instances

Instance	Cities	Cycles	Cycle cost	MTS-1		MTS-2		MTS-3		MTS-4	
				Tour	Time	Tour	Time	Tour	Time	Tour	Time
dsj1000	1000	58	1.48	8.12	0.78	6.68	0.98	6.06	0.87	5.41	1.31
pr1002	1002	86	0.10	9.09	0.26	7.65	0.51	9.57	0.32	5.46	0.87
u1060	1060	54	1.53	6.63	0.31	5.79	0.48	5.71	0.38	5.06	0.72
vm1084	1084	40	1.81	7.57	0.23	5.81	0.44	6.02	0.27	5.18	0.73
pcb1173	1173	26	2.29	4.49	0.21	3.28	0.46	4.59	0.22	3.20	0.77
d1291	1291	57	−0.41	10.14	0.74	9.06	1.18	9.68	0.82	8.15	1.71
rl1304	1304	60	1.43	8.74	0.45	5.81	0.79	6.49	0.51	5.33	1.25
rl1323	1323	51	1.03	6.34	0.37	5.01	0.80	6.59	0.46	4.70	1.29
nrw1379	1379	59	1.34	5.57	0.29	3.48	0.75	4.01	0.38	2.86	1.37
fl1400	1400	76	−4.09	10.67	1.32	9.76	1.80	9.81	1.40	9.36	2.43
u1432	1432	76	0.98	4.82	0.32	3.31	0.83	3.02	0.42	1.89	1.50
fl1577	1577	104	−2.42	12.95	1.39	11.89	1.99	16.23	1.48	15.18	2.77
d1655	1655	67	1.48	8.15	1.19	5.93	1.88	6.58	1.32	5.10	2.66
vm1748	1748	79	2.50	6.92	0.52	5.66	1.23	5.66	0.65	4.80	2.15
u1817	1817	93	2.23	10.25	0.48	7.36	1.34	8.52	0.67	5.89	2.41
rl1889	1889	98	1.34	8.55	0.70	7.06	1.36	9.14	0.84	6.69	2.55
d2103	2103	50	−1.23	7.48	1.08	6.86	2.34	4.92	1.15	3.69	3.79
u2152	2152	140	0.29	10.96	0.70	7.16	1.95	7.50	0.97	5.66	3.53
u2319	2319	79	0.16	1.09	0.40	0.19	1.84	0.97	0.74	0.16	3.59
pr2392	2392	129	2.40	11.80	0.66	10.15	2.22	7.94	0.82	6.38	4.13
pcb3038	3038	95	1.84	4.89	0.78	3.31	3.31	3.71	1.06	2.84	6.15
fl3795	3795	142	−0.40	8.65	13.44	8.08	17.15	7.76	13.79	5.23	21.55
fnl4461	4461	149	1.63	4.15	1.51	2.81	6.18	3.65	2.35	2.73	11.99
rl5915	5915	164	2.36	7.26	3.04	5.77	9.94	5.40	3.82	4.53	18.16
rl5934	5934	195	2.49	7.86	3.57	6.04	10.70	6.93	4.55	5.60	19.82
pla7397	7397	292	1.47	5.41	22.66	4.02	38.59	4.33	25.45	3.57	58.96
rl11849	11849	465	1.91	7.63	11.59	5.06	49.20	5.75	15.19	4.27	95.62
usa13509	13509	577	1.50	5.65	19.43	4.41	69.64	5.26	26.86	4.00	130.92
brd14051	14051	504	1.67	4.71	23.16	3.04	78.02	3.65	31.12	2.80	141.29
d15112	15112	517	1.50	4.30	15.59	2.77	74.77	3.53	24.68	2.67	143.60
d18512	18512	645	1.67	4.62	20.10	2.90	117.60	3.82	34.07	2.72	223.91
pla33810	33810	1116	0.96	4.67	71.51	2.62	495.14	2.88	90.28	1.65	900.07
pla85900	85900	3392	0.72	4.08	413.36	2.46	3169.94	2.66	696.85	1.91	5746.59
				7.10	632.14	5.49	4165.35	6.01	984.76	4.69	7560.16

Cities is the instance number of cities. Cycle is the number of cycles produced from the two sequential matchings. Cycle cost is the total cycle cost in terms of percentage over the HK bound. Tour is the tour cost as percentage over the HK bound. Time is the non-normalized CPU time in seconds. In the last line we give the average tour cost as percentage over the KH bound and the total CPU time in seconds.

Table 3
Comparison of the proposed TSP tour construction heuristic against all TSP tour construction techniques that are within 15% of the HK bound [1]

Instance	cost	FI	FA+	Savings	CCA	CR-S	CR-G	CR-HK	CR-JM	CR-R	MTS-1	MTS-2	MTS-3	MTS-4
u2319	tour(%)	6.77	6.85	7.24	7.18	11.45	6.82	7.02	8.98	9.24	1.09	0.19	0.97	0.16
	CPU(s)	0.55	0.32	0.04	29.63	0.12	0.12	1.69	0.07	0.29	0.60	2.76	1.11	5.38
pr2392	tour(%)	14.49	14.62	13.51	12.72	13.65	9.03	7.47	9.71	16.84	11.80	10.15	7.94	6.38
	CPU(s)	0.54	0.29	0.05	43.70	0.17	0.18	1.69	0.08	0.33	0.99	3.33	1.23	6.20
pcb3038	tour(%)	16.12	16.27	11.80	11.65	14.00	9.46	6.78	11.05	14.54	4.89	3.31	3.71	2.84
	CPU(s)	0.77	0.42	0.07	70.42	0.21	0.21	1.97	0.11	0.33	1.17	4.96	1.59	9.23
fl3795	tour(%)	—	—	—	—	—	—	8.34	—	10.59	8.64	8.08	7.76	5.23
	CPU(s)	—	—	—	—	—	—	2.09	—	0.59	20.16	25.72	20.68	32.33
fnl4461	tour(%)	2.24	12.31	11.12	10.32	14.85	9.73	7.12	10.73	15.25	4.15	2.81	3.64	2.73
	CPU(s)	1.10	0.62	0.11	151.74	0.35	0.36	2.89	0.17	0.39	2.27	9.27	3.53	17.98
rl5915	tour(%)	24.49	24.70	12.44	15.11	11.62	7.80	7.07	9.89	12.75	7.26	5.77	5.40	4.53
	CPU(s)	1.55	0.79	0.13	612.33	0.29	0.30	5.00	0.23	0.40	4.56	14.91	5.73	27.24
rl5934	tour(%)	22.04	22.11	12.86	14.44	12.13	7.80	6.75	9.52	13.82	7.86	6.04	6.93	5.60
	CPU(s)	1.57	0.81	0.14	560.57	0.34	0.35	4.87	0.23	0.41	5.35	16.05	6.82	29.73
pla7397	tour(%)	14.57	14.89	10.09	11.17	12.23	8.20	8.59	9.60	13.79	5.41	4.02	4.33	3.57
	CPU(s)	2.01	1.09	0.16	533.03	0.57	0.58	9.17	0.33	0.55	33.99	57.89	38.17	88.44
rl11849	tour(%)	—	—	—	—	—	—	6.91	—	14.32	7.63	5.06	5.75	4.27
	CPU(s)	—	—	—	—	—	—	10.84	—	0.71	17.38	73.80	22.79	143.43
usa13509	tour(%)	—	—	—	—	—	—	7.38	—	20.16	5.65	4.41	5.26	4.00
	CPU(s)	—	—	—	—	—	—	16.85	—	0.92	29.14	104.46	40.29	196.38
brd14051	tour(%)	11.82	11.91	11.61	11.12	14.59	9.52	6.42	10.85	18.68	4.71	3.04	3.65	2.80
	CPU(s)	4.25	2.12	0.38	1923.91	2.98	2.95	10.29	0.67	0.90	34.74	117.03	46.68	211.94
d15112	tour(%)	12.32	12.40	11.63	11.32	14.22	9.50	6.79	10.99	18.35	4.30	2.77	3.53	2.67
	CPU(s)	4.67	2.33	0.41	2578.46	1.65	1.65	14.14	0.73	1.09	23.38	112.16	37.02	215.40
d18512	tour(%)	12.21	12.29	11.21	11.08	14.58	9.27	6.79	11.16	18.45	4.62	2.90	3.82	2.72
	CPU(s)	5.32	2.77	0.44	3867.29	2.67	2.69	14.23	0.82	1.12	30.15	176.40	51.11	335.87
pla33810	tour(%)	17.78	17.97	10.87	10.38	16.50	9.62	6.90	11.83	19.38	4.67	2.62	2.88	1.65
	CPU(s)	9.84	4.77	0.62	21100.84	3.88	3.82	50.10	1.54	3.16	107.27	742.71	135.42	1350.11
pla85900	tour(%)	15.32	15.49	9.96	—	15.46	9.50	7.42	11.52	20.79	4.08	2.46	2.66	1.91
	CPU(s)	26.40	12.82	1.36	—	31.78	29.51	177.89	3.90	11.12	620.04	4754.91	1045.28	8619.89

All runtimes are normalized. We use the naming convention of the TSP Challenge website: Savings is the Clarke–weight savings heuristic [4]. CCA is the Golden–Stewart convex hull, cheapest insertion, angle selection algorithm [11]. FI is the farthest insertion algorithm [2]. FA+ is the farthest augment addition algorithm [2]. CR-S is the classical Christofides MST-based tour construction method [3]. The following are variants of the Christofides heuristic: CR-G is the greedy shortcut version. CR-HK is a version where the one-tree of the HK bound is combined with greedy shortcuts. CR-JM is an approximate, greedy version. CR-R is the Rohe's half-LK standard version. MTS1 through MTS4 are the proposed matching-based cycle construction technique followed by different types of stitching as described in Table 1.

the acronym MTS (match twice and stitch) to refer to the proposed tour construction methodology.

Table 2 gives the results of all four variants of our technique. In this table, we report non-normalized runtimes. From the table, we can conclude that none of the four variants dominates the others. They represent a trade-off between quality and runtime with the MTS-4 variant offering the best tour quality, and the MTS-1 variant offering the best runtime (about 10x improvement over MTS-4) but sacrificing tour quality by about 2.5% in average. The results also indicate that performance, as measured by excess over the HK bound, generally *improves* as the instance size *increases*. For the largest instance, pla85900, the tour length is only 1.91% over the HK bound.

Table 3 compares our heuristic with previous TSP construction heuristics that produce tours that are within 15% of the HK bound [1]. All runtimes are normalized according to the TSP Challenge website procedure. For space limitations, we only compare results for the largest 15 instances of the TSPLIB benchmarks. From the table, the proposed approach

Table 4
Comparison of the proposed TSP tour construction heuristic against tour improvement heuristics [1]

Instance	cost	Tabu-SC -DB	2Opt-JM -40b	2.5Opt -B	3Opt-JM -40	LK -ABCC	LK-JM -40-BD	Helsgaun	MTS-1	MTS-2	MTS-3	MTS-4
u2319	tour(%)	0.34	1.98	2.15	1.77	0.51	0.42	0.09	1.09	0.19	0.97	0.16
	CPU(s)	434.76	0.34	0.24	0.34	0.32	0.52	35.27	0.60	2.76	1.11	5.38
pr2392	tour(%)	1.83	6.65	8.47	4.81	4.56	3.09	1.45	11.80	10.15	7.94	6.38
	CPU(s)	955.97	0.32	0.35	0.35	0.18	0.77	34.87	0.99	3.33	1.23	6.20
pcb3038	tour(%)	1.71	5.58	5.73	3.70	2.99	1.85	0.97	4.89	3.31	3.71	2.84
	CPU(s)	1154.97	0.42	0.40	0.45	0.25	0.74	55.85	1.17	4.96	1.59	9.23
fl3795	tour(%)	5.25	11.03	0.00	9.37	3.32	7.36	7.83	8.64	8.08	7.76	5.23
	CPU(s)	15447.66	0.64	0.00	0.69	0.37	28.85	74.06	20.16	25.72	20.68	32.33
fn14461	tour(%)	1.34	4.39	4.71	2.32	2.31	1.68	0.62	4.15	2.81	3.64	2.73
	CPU(s)	1717.12	0.63	0.60	0.68	0.43	1.18	129.23	2.27	9.27	3.53	17.98
rl5915	tour(%)	2.61	5.63	5.84	4.04	4.93	2.80	1.95	7.26	5.77	5.40	4.53
	CPU(s)	9390.51	0.95	0.63	1.02	0.46	3.29	242.99	4.56	14.91	5.73	27.24
rl5934	tour(%)	2.61	5.69	8.04	3.84	4.03	2.67	1.60	7.86	6.04	6.93	5.60
	CPU(s)	11834.26	0.95	0.75	1.05	0.47	3.48	271.67	5.35	16.05	6.82	29.73
pla7397	tour(%)	2.57	4.76	5.67	3.16	2.53	2.05	0.94	5.41	4.02	4.33	3.57
	CPU(s)	15862.69	1.28	0.92	1.35	0.76	7.84	452.01	33.99	57.89	38.17	88.44
rl11849	tour(%)	2.44	5.21	0.00	3.48	3.51	2.17	1.37	7.63	5.06	5.75	4.27
	CPU(s)	27361.51	1.77	0.00	1.95	1.33	4.33	1311.87	17.38	73.80	22.79	143.43
usa13509	tour(%)	1.79	4.99	0.00	2.85	3.20	1.94	0.88	5.65	4.41	5.26	4.00
	CPU(s)	66554.23	1.90	0.00	2.11	1.96	6.15	1133.81	29.14	104.46	40.29	196.38
brd14051	tour(%)	1.91	4.98	4.91	2.86	2.26	1.59	0.61	4.71	3.04	3.65	2.80
	CPU(s)	27193.02	2.19	1.97	2.40	1.67	7.27	1674.24	34.74	117.03	46.68	211.94
d15112	tour(%)	1.73	4.93	4.27	2.89	2.35	1.66	0.63	4.30	2.77	3.53	2.67
	CPU(s)	35660.20	2.53	1.94	2.73	2.38	4.60	1515.99	23.38	112.16	37.02	215.40
d18512	tour(%)	1.61	4.64	4.32	2.62	2.04	1.59	0.58	4.62	2.90	3.82	2.72
	CPU(s)	45269.20	2.70	2.34	2.91	2.49	4.62	3212.32	30.15	176.40	51.11	335.87
pla33810	tour(%)	2.37	6.13	5.38	3.83	2.16	1.88	0.96	4.67	2.62	2.88	1.65
	CPU(s)	85225.86	5.43	3.14	5.97	3.48	30.60	7982.09	107.27	742.71	135.42	1350.11
pla85900	tour(%)	2.33	5.64	4.82	3.54	1.60	1.60	1.25	4.08	2.46	2.66	1.91
	CPU(s)	113625.25	14.61	7.45	15.83	8.84	46.20	48173.84	620.04	4754.91	1045.28	8619.89

All runtimes are normalized. We use the naming convention of the TSP Challenge website. Tabu-SC-DB is Zachariasen-Dam's Tabu-Search Flower/D-B implementation. 2opt-JM-40b is Johnson–McGeoch's 2-Opt implementation 40 Quadrant Neighbors, Run 2 results. 2.5opt-B is Bentley's 2.5-Opt implementation. LK-ABCC is Concorde's version of Lin-Kernighan implementation. LK-JM-40-BD is Johnson–McGeoch's Lin–Kernighan implementation. Helsgaun is Keld Helsgaun's implementation of a Lin–Kernighan variant. MTS1 through MTS4 are the proposed matching-based cycle construction technique followed by different types of stitching as described in Table 1.

clearly outperforms all reported tour construction heuristics including Christofides' heuristic [3] and its variants [17] by a significant margin. On the other hand, other heuristics offer better runtimes.

While the heuristic we propose is a tour construction heuristic, we also compare it with tour improvement techniques in Table 4. Due to space limitations, we cannot compare with all tour improvement heuristics. Hence, we select a number of the leading techniques for tour improvement. All runtimes are normalized according to the TSP challenge website. From Table 4, we see that in general the tour improvement heuristics outperform MTS variants by a small margin, but nevertheless the MTS-4 variant can outperform a number of leading tour improvement techniques on such benchmarks as u2319 and pla3381. On the other hand, most TSP tour improvement heuristics offer their results in much less time than the MTS variants. The Helsgaun heuristic that offers better tour quality results in all benchmarks does not dominate MTS-4 since MTS-4 produce its results in less runtime.

Similar results for uniform and clustered random instances are given in Table 5, where we also compare our results against two tour construction heuristics

Table 5
Summary of results for uniform, clustered and TSPLIB benchmarks for the MTS variants as described in Table 1.

		1000	3162	10 k	31 k	100 k	316 k
Heuristic	Average Percent Excess over HK Bound						
MTS1	Uniform	6.09	8.09	6.23	6.33	6.22	6.20
MTS1	Clustered	8.90	9.96	11.97	11.61	9.45	—
MTS2	Uniform	4.19	4.98	4.73	4.81	4.73	4.70
MTS2	Clustered	8.62	8.76	9.91	10.09	9.46	—
MTS3	Uniform	5.26	5.8	5.55	5.69	5.6	5.6
MTS3	Clustered	8.52	9.5	10.11	9.725	9.46	—
MTS4	Uniform	4.44	4.68	4.46	5.00	4.51	4.43
MTS4	Clustered	6.91	7.96	8.25	8.00	7.36	—
CR-S	Uniform	14.48	14.61	14.81	14.67	14.70	14.49
	Clustered	12.03	12.79	13.08	13.47	13.50	13.45
CR-HK	Uniform	7.55	7.33	7.30	6.74	6.86	6.90
	Clustered	7.27	7.78	8.37	8.42	8.46	8.56
LK-JM-40-BD	Uniform	1.99	1.88	1.94	1.95	1.89	1.89
	Clustered	1.60	2.72	3.62	3.16	3.51	3.63
Helsgaun	Uniform	0.9	0.89	0.83	0.83	—	—
	Clustered	1.25	2.00	3.32	3.58	—	—
		1000	3162	10 k	31 k	100 k	316 k
Heuristic	Average Normalized Running Time in Seconds						
MTS1	Uniform Points	0.37	2.56	17.21	213.40	1248.21	11833.82
MTS1	Clustered Points	0.78	4.19	45.09	276.43	1798.15	—
MTS2	Uniform Points	0.72	5.48	54.36	659.09	6265.64	60270.63
MTS2	Clustered Points	1.15	8.22	86.11	772.06	6941.61	—
MTS3	Uniform Points	0.46	3.55	24.65	289.06	2063.29	21716.42
MTS3	Clustered Points	0.84	4.76	49.04	337.31	2213.05	—
MTS4	Uniform Points	1.15	10.15	103.77	1103.78	10537.33	186203.79
MTS4	Clustered Points	1.65	13.50	139.05	1234.53	11124.85	—
CR-S	Uniform	0.1	0.3	1.0	4.7	21.3	99.5
	Clustered	0.2	0.8	3.2	11.0	37.8	152.8
CR-HK	Uniform	1.0	4.0	14.7	51.4	247.2	971.5
	Clustered	0.9	3.3	11.6	40.9	197.0	715.1
LK-JM-40-BD	Uniform	0.3	0.9	3.0	9.1	27.0	74.1
	Clustered	6.3	18.6	52.4	180.9	603.7	1712.7
Helsgaun	Uniform	5.6	71.5	861.7	7819.3	—	—
	Clustered	7.0	70.3	768.3	12812.5	—	—

For space limitations, we consider also results from two tour construction heuristics (CR-S and CR-HK) and two improvement heuristics (LK-JM-40-BD and Helsgaun). The vertical columns (1000, 3162, 10 k, 31 k, 100 k, and 316 k) give the number of points in each random instance.

and two tour improvement heuristics. The vertical columns of this table (1000, 3162, 10 k, 31 k, 100 k, and 316 k) give the number of points in each random instance. In this experiment, we had to resort to greedy stitching with the 316k instances due to memory limitations. We summarize our experimental results as follows.

- The proposed MTS method outperforms all reported tour construction heuristics, including Christofides' heuristic [3] and its variants [17], by a significant margin but does not dominate most of these heuristics since they offer better runtimes.
- With the exception of only the u2319 and pla33810 benchmarks, the proposed method is dominated by some of the leading TSP tour improvement heuristics.
- The proposed heuristic narrows the performance gap between the class of tour construction and tour improvement heuristics.

- The performance of the proposed method improves as the instance size increases, but runtime grows quadratically for Euclidean instances and cubically for non-Euclidean instances.

A number of interesting theoretical questions emerge from this work. Some of these questions are: How well do the two sequential matchings approximate a minimum-weight 3-restricted 2-factor? What is the approximation ratio of the two sequential matchings to the optimal tour? Or, more ambitiously, what is the approximation ratio of the MTS heuristic?

Acknowledgements

We thank I. Mandoiu and A. Zelikovsky for useful discussions, and M. Mostagir and A. Leontaris for careful reading of an earlier draft. We thank the two anonymous referees for numerous suggestions which have enhanced the paper.

References

[1] http://www.research.att.com/∼dsj/chtsp. TSP Challenge Website.
[2] J.L. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA J. Comput. 4 (1992) 387–411.
[3] N. Christofides, Worst-case analysis of a new heuristic for the traveling salesman problem, Technical Report 388, GSIA, Carnegie-Mellon University, Pittsburgh, PA, 1976.
[4] G. Clarke, J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, Oper. Res. 12 (1964) 568–581.
[5] W. Cook, A. Rohe, Computing minimum weight perfect matchings, http://www.or.uni-bonn.de/home/rohe/matching.html., INFORMS J. Comput. 11 (1999) 38–148.
[6] G.A. Croes, The traveling-salesman problem, Oper. Res. 4 (1956) 61–75.
[7] G.A. Croes, A method for solving traveling salesman problems, Oper. Res. 6 (1958) 791–812.
[8] W.H. Cunningham, Y. Wang, Restricted 2-factor polytopes, Math. Programming 87 (1) (2000) 87–111.
[9] M.L. Fisher, G.L. Nemhauser, L.A. Wolsey, An analysis of approximations for finding a maximum weight Hamiltonian circuit, Oper. Res. 27 (4) (1979) 799–809.
[10] P.C. Gilmore, R.E. Gomory, Sequencing a one state-variable machine: a solvable case of the traveling salesman problem, Oper. Res. 12 (1964) 655–679.
[11] B.L. Golden, W.R. Stewart, Empirical analysis of heuristics, in: E.L. Lawler, et al., (Eds.), The Traveling Salesman Problem, Wiley, Chichester, 1985, pp. 207–249.
[12] G. Gutin, A.P. Punnen (Eds.), The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Dordrecht, 2002.
[13] R. Hassin, S. Rubinstein, A 7/8-approximation algorithm for metric max TSP, Inform. Process. Lett. 81 (5) (2002) 247–251.
[14] M. Held, R. Karp, The traveling salesman problem and minimum spanning trees, Oper. Res. 18 (1970) 1138–1162.
[15] M. Held, R. Karp, The traveling salesman problem and minimum spanning trees: part II, Math. Programming 1 (1971) 6–52.
[16] I. Hong, A.B. Kahng, B. Moon, Improved large-step Markov chains for the symmetric TSP, J. Heuristics 3 (1997) 63–81.
[17] D. Johnson, L. McGeoch, Experimental analysis of heuristics for the STSP, in: In G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and its Variations, Kluwer Academic Publishers, Dordrecht, 2002.
[18] R.M. Karp, A patching algorithm for the non-symmetric traveling-salesman problem, SIAM J. Comput. 8 (4) (1979) 561–573.
[19] S. Lin, B. Kernighan, An effective heuristic algorithm for the traveling-salesman problem, Oper. Res. 21 (1973) 498–516.
[20] L. Lovasz, M.D. Plummer, Matching Theory, North-Holland, Amsterdam, 1986.
[21] O. Martin, S.W. Otto, E.W. Felten, Large-step Markov chains for the TSP incorporating local search heuristics, Oper. Res. Lett. 11 (1992) 219–224.
[22] L.K. Platzman, J.J. Bartholdi, Spacefilling curves and planar traveling salesman problem, J. ACM 36 (1989) 719–737.
[23] A. Schrijver, Combinatorial Optimization: Polyhedra and Efficiency, Springer, Berlin, 2003.
[24] O. Vornberger, Easy and hard cycle covers, Technical Report, Universitat Paderborn, 1980.