Combining Two Local Searches with Crossover: An Efficient Hybrid Algorithm for the Traveling Salesman Problem

Weichen Liu
UBRI, School of Computer Science and Technology
University of Science and Technology of China
Hefei, Anhui, China 230027
lwc@mail.ustc.edu.cn

Thomas Weise*
Institute of Applied Optimization
Faculty of Computer Science and Technology, Hefei University
Hefei, Anhui, China 230601
tweise@hfuu.edu.cn

Yuezhong Wu
UBRI, School of Computer Science and Technology
University of Science and Technology of China
Hefei, Anhui, China 230027
yuezhong@mail.ustc.edu.cn

Qi Qi
UBRI, School of Computer Science and Technology
University of Science and Technology of China
Hefei, Anhui, China 230027
qqi@mail.ustc.edu.cn

ABSTRACT

The Traveling Salesman Problem (TSP) is one of the most well-known optimization problems. Ejection Chain Methods (ECM) and the Lin-Kernighan (LK) heuristic are the state-of-art local search (LS) algorithms for solving the TSP. Multi-Neighborhood Search (MNS) is known to be especially suitable for hybridization with Evolutionary Computation (EC). Hybridizing two different LS algorithms with each other (LS-LS) can combine their mutual advantages and lead to better performance. We introduce the new concept of LS-LS-X hybrids, which combines two different LS algorithms with a crossover operator. We enhance the two best LS-LS hybrids, ECM-LK and LK-MNS, with Order Based Crossover and Heuristic Crossover. We hybridize these LS-LS-X algorithms with an Evolutionary Algorithm, the most prominent EC method, and obtain highly-efficient (memetic) EC-LS-LS-X algorithms. We conduct a large-scale experimental study with many different algorithm setups on all 110 symmetric instances of the TSPLib benchmark set. We find that the LS-LS-X hybrids have significantly better performance than the original LS-LS and their component algorithms. They even outperform several memetic EC-LS-LS and EC-LS algorithm setups. The EC-LS-LS-X hybrids are the best hybrid EA-based TSP solvers by a large margin in our experiment and the wide range of algorithms available in the popular TSP Suite.

CCS CONCEPTS

• **Computing methodologies → Genetic algorithms**; Planning and scheduling; Randomized search; • **Applied computing → Transportation**; *Operations research*;

*corresponding author

KEYWORDS

Local Search, Hybrid Algorithms, Traveling Salesman Problem, Crossover, Memetic Algorithm

ACM Reference format:
Weichen Liu, Thomas Weise, Yuezhong Wu, and Qi Qi. 2017. Combining Two Local Searches with Crossover: An Efficient Hybrid Algorithm for the Traveling Salesman Problem. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071201

1 INTRODUCTION

The Traveling Salesman Problem (TSP) [1, 14, 23] is maybe the most prominent $\mathcal{NP}$-hard problem, not only in terms of practical applications, but also for being a test bed for novel optimization approaches [26]. Given a collection of n cities and the travel distance between them, solving a TSP means to find the shortest round-trip tour through all cities and back to the starting point. The TSP can be formulated based on a cost matrix $D = (D_{i,j})$, where $D_{i,j}$ is the cost of traveling from city i to j. The target then is to find a permutation t of the integers from 1 to n minimizing the sum $D_{t[1],t[2]} + D_{t[2],t[3]} + \cdots + D_{t[n],t[1]}$. In this paper, we focus on symmetric TSPs, where $D_{i,j} = D_{j,i}$ holds.

For all $\mathcal{NP}$-hard problems [14, 39], any exact algorithm which can guarantee to find the best solution has exponential worst-case runtime complexity. Many approaches have thus been devised to get good approximate solutions within acceptable runtime, including Local Search (LS) algorithms [10, 24], and Evolutionary Algorithms (EAs) [3, 7, 34]. The state-of-the-art LS approaches for the TSP are the Lin-Kernighan (LK) heuristic [24] and Ejection Chain Methods (ECM) [10]. Wu et al. [40] compared several hybrid and pure versions of the LK heuristic with EAs [13]. Liu et al. [25] proposed the Fundamental Stem and Cycle Method (FSM**), an improved ECM based on the P_SEC algorithm [30]. Memetic Algorithms (MAs) [28], combine the global search ability of an Evolutionary Computation (EC) method such as an EA with the exploitation strength of a LS, are known to be especially efficient [26]. Such "EC-LS" algorithms using Multi-Neighborhood Search (MNS) as LS performed best in [35]. Only recently, researchers started to combine different LS algorithms to LS-LS hybrids in order to exploit their mutual advantages [41].

With the present work, we make the following contributions:

(1) We introduce the new concept of LS-LS-X hybridization, which combines two different LS methods with a crossover operator. The LS algorithms produce solutions which are then recombined and serve as starting point for the next LS iteration.
(2) We combine the two best known LS-LS hybrids, LK-MNS and FSM**-LK [41], with Heuristic Crossover (HX) and Order Based Crossover (OX2). We find that the resulting LS-LS-X methods significantly outperform their LS components and the original LS-LS hybrids. It should be noted that the state-of-the-art of metaheuristic TSP solving is based on these outperformed component LS algorithms, namely LK and FSM**, i.e., the new methods are very competitive.
(3) We then hybridize our LS-LS-X hybrids with EAs. We find that the end resulting EC-LS-LS-X algorithms perform better and can solve more problem instances than any other method in our experiments, in [25, 35, 40], and in the huge algorithm portfolio of the popular *TSP Suite*.
(4) We conduct an in-depth statistically comparison of all the above algorithms based on a large-scale experimental study, where 36 algorithm setups are applied to all 110 symmetric benchmark instances from *TSPLib* [31]. We apply runtime-behavior based statistics, which provide more information than simple end-result comparisons.
(5) The LS-LS-X and EC-LS-LS-X concepts are very easy to implement and, while we use the TSP as testbed, generalize to any kind of optimization problem.

The remainder of this paper is organized as follows. In Section 2, we introduce the investigated algorithms LK, FSM**, and MNS, the LS-LS hybrids, crossover operators, as well as our new LS-LS-X and EC-LS-LS-X hybrids. We then present our experimental study and discuss its results in Section 3. Finally, the paper ends with conclusions and plans for future work in Section 4.

2 INVESTIGATED ALGORITHMS

2.1 Local Search

LS algorithms start at a random or heuristically-generated solution (tour). They remember the best solution discovered so far and try to improve it step by step by applying search operators.

A solution for a TSP is a tour and such a tour can be considered as cyclic path consisting of n edges. The most common search operators for the TSP are so-called m-opt moves [35], which replaces m of these n edges. The exchange of two cities in a tour corresponds to the replacement of (at most) four edges (4-opt) [22, 27]. Rotating a sub-sequence of cities to the left or right is a 3-opt [8, 22] move. The reversal of a sub-sequence of a tour, the maybe most common operator for the TSP, is a 2-opt move [19, 22].

If the LS cannot further improve its best tour, it may apply a larger random modification in order to escape from the local optimum, while remembering the best overall solution in an additional variable. This process is repeated until a termination criterion is reached.

2.2 The Lin-Kernighan Algorithm

The LK heuristic is a LS approach published by Lin and Kernighan [24] in 1973. Its derivatives dominate today's TSP research and many improvements have been proposed. When the Chained Lin-Kernighan (CLK) algorithm [2] arrives at a local optimum from which it cannot escape, it generates a new solution by a random 4-opt move instead of restarting at a random solution. CLK performs particularly well on TSPs with a large number n of cities. The Lin-Kernighan-Helsgaun (LKH) algorithm [17, 18] may be the most efficient LK variant.

We use the LK implementation from [40, 41] in our experiments. It applies the first improving move discovered instead of scanning a larger neighborhood for the best improving move, as the original LK heuristic does. This was found to be more efficient in [17, 40]. We furthermore use candidate sets [24], i.e., limit the choices of neighbors for any city in a tour to speed up the algorithm. When reaching a local optimum, the algorithm uses the soft restart approach described in [35, 40], where a randomly chosen sub-sequence of the current tour is randomly shuffled.

An in-depth description of the LK heuristic used in our investigation can be found in [40], while [17, 18] provides further explanations of the standard LK heuristic and its improvements. From [41], we know that LK with a candidate set of size 10, in the following abbreviated with LK10, is the most suitable version for hybridization.

2.3 Ejection Chain Method: FSM**

The Ejection Chain Method (ECM) was introduced by Glover in 1992 [10]. Unlike other LS directly searching in the space of candidate solutions, ECMs work with Stem-and-Cycle (S&C) reference structure. The structure consists of a path (called stem) attached to a cycle of nodes. The root r marks one end of the stem, and another end is called the tip t. The two neighboring nodes of r on the cycle are called sub-roots ($s1$ and $s2$).

If the stem is degenerated to become a single node (i.e., $r = t$), the S&C structure becomes a tour. Otherwise, the S&C structure can be transformed to two candidate tours by removing the edge between one of the sub-roots si and the root r, and then re-connecting si to the tip t. The better one of these two trial solutions can be chosen.

During the search, the S&C structure is iteratively refined by two search operators (called "rules") [11, 30], which leave the root r unchanged:

(1) Choose a node j on the cycle. Let the two nodes adjacent to it be $q1$ and $q2$. Select one of them and refer to it as q. Delete the edge between q and j and then add edge (t, j).
(2) Choose a node j on the stem. Let the node adjacent to j and farther away from r than the other adjacent node be called q. Connect t to j and delete edge (j, q).

P_SEC is a particularly efficient ECM proposed by Rego [30]. Here, in each step, both rules and all possible node j are tested and the combination which minimizes the overall edge length of the S&C is applied. In our experiments, we use FSM** [25], which improves P_SEC in several aspects. It does not allow an edge to be deleted a second time during the search, i.e., being deleted, added, and then deleted again from the S&C. The root of the S&C is changed after $0.45n$ steps and after $0.15n$ nodes have been used as root, the algorithm applies the same soft restart method as the tested LK implementation [35]. Each of these modifications has been confirmed to improve the overall performance [25].

2.4 Multi-Neighborhood Search

Another LS approach for the TSP is the Mult-Neighborhood Search (MNS). In each iteration, MNS performs an $O(n^2)$ scan that investigates four neighborhoods (city swap, sub-sequence rotate left, sub-sequence rotate right, and reversal) of a tour at once. It tests all pairs $\{i, j\}$ as potential indexes for cities to swap or start and end indexes of sub-sequence rotations and reversals. For each pair $\{i, j\}$, the gain is computed and all discovered improving moves enter a queue. After the scan, the best discovered move is carried out. Doing this may invalidate some other moves in the queue, e.g., if a sub-sequence reversal that overlaps with a potential sub-sequence left rotation was performed. After pruning all invalidated moves from the queue, the remaining best move is carried out, if any. If the queue becomes empty, another scan of the current solution is performed, as new moves may have become possible. If no improving moves can be found anymore, a random sub-sequence of the current tour is randomly shuffled.

This algorithm has performed the best among all of the LS methods tested in [35]. It is outperformed by both LK and FSM** in their pure form, but its hybrid versions with Ant Colony Optimization outperform theirs [25, 40].

2.5 Hybrid Local Search: LS-LS

Research on hybrid ("Memetic") algorithms is almost entirely focused on combining global search (e.g., EC) and local search algorithms (EC-LS), as done in [25, 26, 35, 40, 42], for instance. However, LS algorithms can already exhibit different behaviors which might complement each other. Some LS approaches (like MNS) are efficient to find good solutions quickly but can easily get stuck in local optima. Others (LK, FSM**) might initially be slower but find better final results [25, 40]. ECMs are considered to be able to explore parts of the search space which cannot be reached by LK [9]. The idea to hybridized different LS to combine their mutual advantages leading to better overall performance, i.e., the LS-LS concept, gained interest with the work of Wu et al. [41].

2.6 Crossover Operators

Crossover operators and populations for guarding against premature convergence are the two major innovations brought by EAs to optimization [34]. The input of a crossover operator are two different solutions which have been selected, i.e., which are good. Combining their building blocks should hopefully lead to a new, better solution uniting the positive aspects of both parents.

The heuristic crossover operator (HX) was introduced in [12, 22] and focuses on combining parental edges. It creates an offspring tour in the following way: A random city is selected to be the current city of the offspring. Then, the cheapest one of the at most four (undirected) edges connecting the current city to an unvisited city from the parent tours is chosen. If none of the parental edges leads to an unvisited city a random edge is selected. This is repeated until a complete tour has been constructed.

The order based crossover operator (OX2) selects at random several positions in a parent tour, and the order of the cities in the selected positions of this parent is imposed on the other parent [22].

2.7 Hybrid Local Search with Crossover Operator: LS-LS-X

In LS-LS hybrids, the second LS refines result solution of the first LS. If it can find an improvement, the first LS is applied to that solution. If neither can improve the solution anymore, a soft restart is applied [41]. The concept of LS-LS hybrids can be thought of a generalization of Variable Neighborhood Search [16], as it not just switches neighborhoods but complete search strategies.

In our LS-LS-X hybrids, we plug a crossover operator between the two LS algorithms. The first local search produces a tour t_1 which is refined by the second local search to t_2. Both t_1 and t_2 become the input to the crossover operator which then hopefully combines their building blocks in a meaningful way. This process is described in Algorithm 1 in detail.

Algorithm 1 LS-LS-X Hybrid Algorithm

```
1:  set best current tour $t$ to a random initial tour
2:  **while** no stopping criterion is reached **do**
3:      apply first LS to tour $t$ to get a new tour $t_1$
4:      **if** $f(t) \leq f(t_1)$ **then**
5:          go to line 15
6:      **end if**
7:      store $t_1$ in $t$
8:      apply second LS to tour $t$ to get a new tour $t_2$
9:      **if** $f(t) \leq f(t_2)$ **then**
10:         go to line 15
11:     **end if**
12:     apply crossover operator to $t_1$ and $t_2$ and get tour $t_3$
13:     store $t_3$ in $t$
14:     go to line 2
15:     conduct a soft restart method and go to line 2
16: **end while**
17: return $t$ and stop
```

Two issues should be noted: Our LS-LS-X algorithm may "lose" the best solution discovered so far due to the crossover operator (Line 12) or soft restart (Line 15 in Algorithm 1). This does not matter as the *TSP Suite*, our experimentation environment, always keeps a copy of the best solution discovered. Second, all LS algorithms are implemented to start at an input solution t and either return an improved tour of shorter length or t itself, but never a worse result.

2.8 Evolutionary and Memetic Algorithms

EAs are the most well-known EC approaches [4, 6]. $(\mu + \lambda)$-EAs first generate a set of λ random solutions. Out of these, the best $\mu \leq \lambda$ solutions will be selected as "parents" of the second generation. λ offspring solutions are created by applying either a unary (mutation) or binary (crossover) operator to the parents. From then on, the μ best individuals are selected from the joint set of λ offspring solutions and their μ parents in each generation.

We investigate MAs which combine such $(\mu + \lambda)$-EAs with LS, LS-LS, and LS-LS-X algorithms, respectively. The first population of our MAs is not generated randomly, but instead stems from the Edge-Greedy, Double Minimum Spanning Tree, Savings, Double-Ended Nearest Neighbor, and Nearest Neighbor Heuristic, in order to improve their performance [35].

It does not make sense to use the same crossover operator in the MA that we use in our LS-LS-X hybrids. We thus apply Edge Crossover [38] in the MA, which generates a new solution by picking edges belonging to either of its two parents, as recombination operator. It is considered to be one of the best crossover operators for the TSP [35]. The crossover rate is set to 1. The LS (or LS-LS or LS-LS-X) component of the MA is applied to every solution generated, both by these initialization heuristics and crossover. The LS-LS-X procedure is therefore modified to work on a solution coming from the MA instead of a random initial tour (Line 1 in Algorithm 1) and similar modifications are introduced to LS and LS-LS. We propose four families of MAs: MA$(\mu + \lambda)$-LS, MA$(\mu + \lambda)$-LS-LS, and MA$(\mu + \lambda)$-LS-LS-X.

3 EXPERIMENTS AND RESULTS

We conduct a large-scale experimental study in which we perform 30 independent runs for 36 different algorithm setups on all 110 symmetric *TSPLib* [31] benchmark cases. The maximum computational budget is set to 1 hour per run.

Most metaheuristics, including EAs, MAs, as well as all LS approaches are anytime algorithms [5]. Even several exact algorithms, such as BB [21], fall into this category. Anytime algorithms can provide a best guess of what the optimal solution of a problem could be at *any time* during their run. This means that there are no "final" solutions, as the point in time when the algorithm is stopped may be arbitrarily chosen (here: after 1h). Thus, anytime algorithms cannot just be characterized by a final solution and runtime requirement, but should be compared based on their whole runtime behavior [35].

We use the *TSP Suite* [35] to execute our experiments, gather data about algorithm runtime behavior, and to evaluate these data. The *TSP Suite* also addresses the problem of properly measuring the time consumed by an algorithm: Runtime measurements presented in terms CPU seconds can capture all the complexities and the overhead of the algorithm implementations, but are strongly machine-dependent and inherently incomparable. Even if normalized runtimes (*NT*) are calculated based on machine performance factors, they remain problem specific and may not represent the utility of black-box metaheuristics in general. Counting the number of generated solutions, i.e., objective function evaluations, or *FE*s in short, is the most-often used alternative in benchmarking. However, it neglects the fact that one *FE* in Ant Colony Optimization has quadratic complexity, in an EA many mutation steps have linear complexity, whereas in a LS algorithm using 2-opt operations, a new solution may be created in $\mathcal{O}(1)$. The *TSP Suite* thus measures runtime in four different ways, CPU time, normalized CPU time, FEs, and the number *DE* of accesses to the distance matrix *D*, in order to provide a balanced overview on algorithm performance.

The *TSP Suite* furthermore ranks algorithms according to their overall performance. This ranking includes statistical comparisons of results at different runtimes, empirical cumulative distribution functions (ECDFs) [15, 20, 33] for different goal values (see the following section), the progress of algorithms over time, estimated running time (ERT) [15] curves, final results, as well as the expected runtime to reach the optimum, amongst others. It therefore represents algorithm performance and robustness from several different angles. The ranking of all algorithm setups investigated can be found at the end of this section in Figure 4.

3.1 LS, LS-LS and LS-LS-X Performance

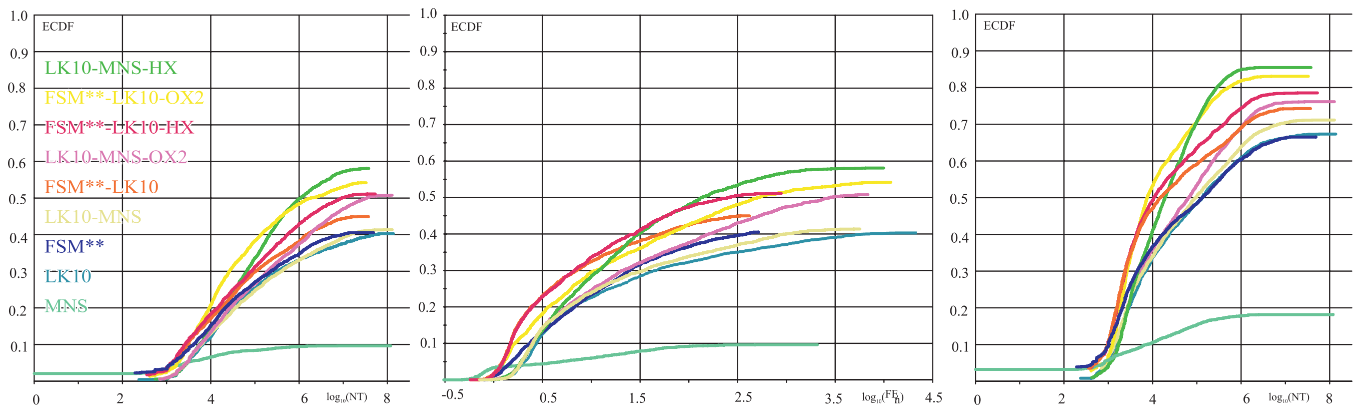
First we will investigate LS, LS-LS, and LS-LS-X hybrids. We therefore define 3 LS setups, namely LK10, FSM**, and MNS. LK10-MNS and FSM**-LK10 have been established as the best LS-LS hybrids in [41]. Therefore, we introduce the crossover operators HX and OX2 into these algorithms and obtain LK10-MNS-HX, LK10-MNS-OX2, FSM**-LK10-HX, and FSM**-LK10-OX2.

According to the automated ranking obtained from the *TSP Suite* (see Figure 4 at the end of the experiment section), LK10-MNS-HX and FSM**-LK10-OX2 have the best performance amongst the LS, LS-LS and LS-LS-X hybrids. The LS-LS hybrids are better than pure LS. In Figure 1, we plot the ECDFs of selected setups for different goal errors F_t and runtime measures. The ECDF illustrates the fraction of runs that have discovered a (best) solution with $F_b \leq F_t$, where F_b corresponds to the relative excess length of the tour length compared to the global optimum. $F_b = 0.01$ would correspond to a tour which is 1% longer than the globally optimal tour of a given benchmark instance. The illustrated ECDFs are aggregated over all 110 benchmark instances. An algorithm is the better the faster and higher its ECDF rises.

The ECDF of LK10-MNS-HX in Figure 1a, based on the normalized CPU runtime measure *NT*, reaches 0.58 for $F_t = 0$. In other words, a global optimum can be reached in about 58% of all runs on all benchmark cases under the given computational budget. Initially, the ECDF curve of FSM**-LK10-OX2 rises quicker than any others, but is overtaken by LK10-MNS-HX at end. Both LK10-MNS-OX2 and FSM**-LK10-HX are a little bit slower compared to the LS-LS hybrids at the beginning, but eventually outperform them. The LS-LS-X hybrids are very obviously better than LS-LS hybrids, which in turn are very obviously better than pure LS. This confirms that hybridizing two LS algorithms by using a crossover operators can improve the performance.

In Figure 1b, we change the runtime measure from normalized CPU time *NT* to the number of objective function evaluations, i.e., *FE*s. This measure counts the search steps but disregards their algorithmic complexity. Here, the hybrids starting with LK10 look a bit slower, but there is no big difference to Figure 1a, i.e., between the theoretical and the actual runtime behavior of the algorithms.

In Figure 1c, we increase the goal error F_t to 0.01. This goal can be reached more often. The LK10-MNS-HX hybrid, for instance, reaches 0.855, i.e., can find a solution not more than 1% longer than the global optimum in 85% of its runs. The curves of FSM**-LK10, FSM**-LK10-HX, and FSM**-LK10-OX2 initially behave the same, probably due to the fact that all of them start with executing FSM**. However, the LK10-MNS-HX algorithm again outperforms the others eventually.



(a) ECDF for *NT* and $F_t = 0.0$ (b) ECDF for *FEs* and $F_t = 0.0$ (c) ECDF for *NT* and $F_t = 0.01$

Figure 1: ECDF diagrams of LS versions for different (log-scaled) runtime measures and goal errors

Additionally, we confirm that different LS-LS hybrids have different suitable crossover operator. For LK10-MNS hybrids, using HX is better than OX2, but OX2 is a right choice for FSM**-LK10. When F_t is a little bigger such as 0.01, the curves of LS-LS-X hybrids are much similar with corresponding LS-LS, but increase faster and higher at the end. The reason may be that pure local search algorithms such as LK10 and FSM** are already efficient in finding approximate solution of that quality. Crossover operators can definitely help LS-LS hybrids to find high-quality approximate solutions.

In Figure 2, we plot the ECDF over the normalized runtime *NT* and F_t=0.0 about for problems of different scale. The ECDF of LK10-MNS-HX in Figure 2a reaches 1, meaning that LK10-MNS-HX can solve all the benchmark problems with 128 to 255 cities. FSM**-LK10-OX2 can solve 98% of them. The best LS-LS hybrid, FSM**-LK10, solves 72%. In Figure 2b, we extend the problem scale to the range 256 . . . 511. 73% of these instances can be solved by LK10-MNS-HX, while the best LS-LS hybrid, again FSM**-LK10, can only solve 28%. FSM**-LK10-OX2 can solve nearly twice as many problems. This is overwhelming evidence that crossover very significantly improves the capability of LS-LS hybrids to solve problems, which already are much better than pure LS algorithms.

3.2 EC-LS and EC-LS-LS Hybrids

After confirming that LS-LS-X hybrids outperform pure LS and LS-LS algorithms and knowing that EC-LS hybrids outperform both pure EC and LS from [25, 35, 40], we now investigate combinations of a EC and our new LS-LS-X algorithms, i.e., EC-LS-LS-X hybrids.

For each of LK10-MNS-HX, LK10-MNS-OX2, FSM**-LK10-HX, FSM**-LK10-OX2, LK10-MNS, FSM**-LK10, MNS, LK10, and FSM**, three additional setups were evaluated: namely MA(2+4), MA(2+8) and MA(16+64). This resulted in 27 setups of EC-LS, EC-LS-LS and EC-LS-LS-X algorithms.

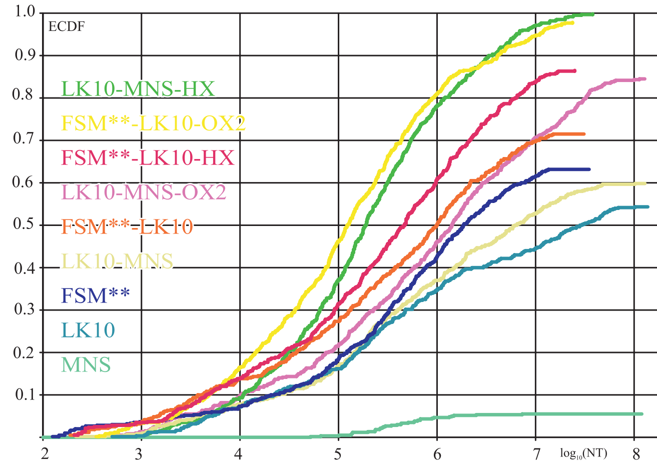
We find that different setups of the same component algorithms, e.g., MA(2+4) and MA(2+8), have relatively similar behavior for all time measures. We choose representative setups from all different EC-LS-LS-X, EC-LS-LS and EC-LS combinations for illustration to get less cluttered charts.

From the ECDF plots in Figure 3a, we observe an improvement of performance in the EC-LS-LS-X methods compared to the LS-LS-X and EC-LS-LS algorithms. MA(16+64)-LK10-MNS-HX finds the optimal solutions ($F_t = 0$) in 61% of its runs while the best MA-LS-LS hybrid, MA(16+64)-LK10-MNS, succeeds in only 58% of its runs. Initially, MA(16+64)-FSM**-LK10, MA(16+64)-FSM**-LK10-HX and MA(16+64)-FSM**-LK10-OX2 are faster than the other algorithms. However, MA(16+64)-FSM**-LK10-HX (end ECDF is 0.59) overtakes MA(16+64)-FSM**-LK10. MA(16+64)-FSM**-LK10-OX2 is slower compared with MA(16+64)-FSM**-LK10. Furthermore, the setups MA(16+64)-LK10-MNS, MA(16+64)-LK10-MNS-HX, and MA(16+64)-LK10-MNS-OX2 have lower ECDF curves than those which utilize FSM**-LK10 at beginning. Still, MA(16+64)-LK10-MNS-HX defeats all the others in the end and solves the most problems.

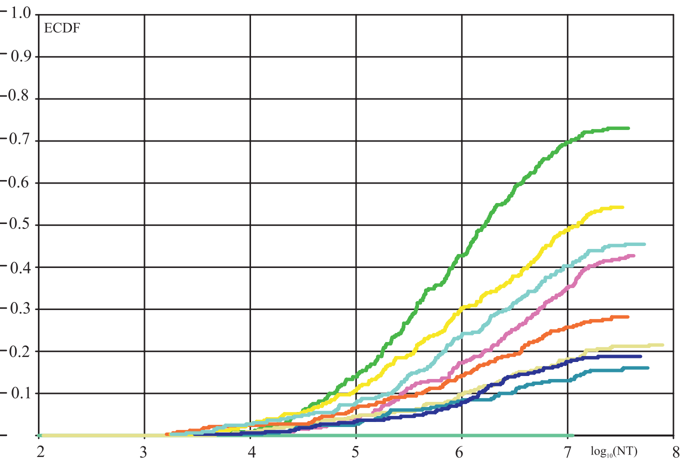
From this figure the importance of collecting and evaluating complete runtime behaviors becomes obvious, since here we clearly observe that the right choice of algorithm depends on the available runtime.

In Figure 3b, we use the number of objective function evaluations, i.e., *FE*s, instead of the normalized CPU time *NT* as time measure. The ECDF curves of MA(16+64)-FSM**-LK10 and MA(16+64)-FSM**-LK10-HX now rise much faster than other algorithms and their end results are only slightly worse than the best compared to MA(16+64)-LK10-MNS-HX.

Comparing the two figures, we find that, although the FSM**-LK10-based hybrids can make bigger advances with the same amount of *FE*s for most of their course, this does not translate 1:1 to actually faster execution. Although they do have an advantage if we measure runtime in terms of (normalized) clock time, it is much smaller than what one would expect when only considering the algorithm steps. This emphasizes the importance of using different time measures to get a clear picture when benchmarking optimization algorithms.

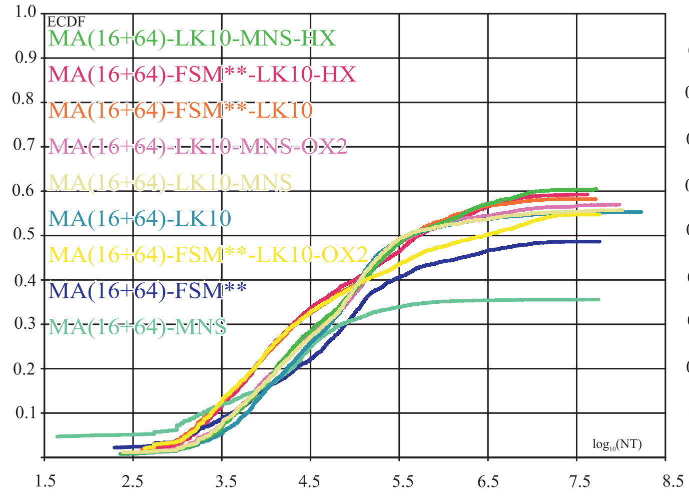


(a) ECDF for NT, $F_t = 0.0$ and $128 \le n < 255$

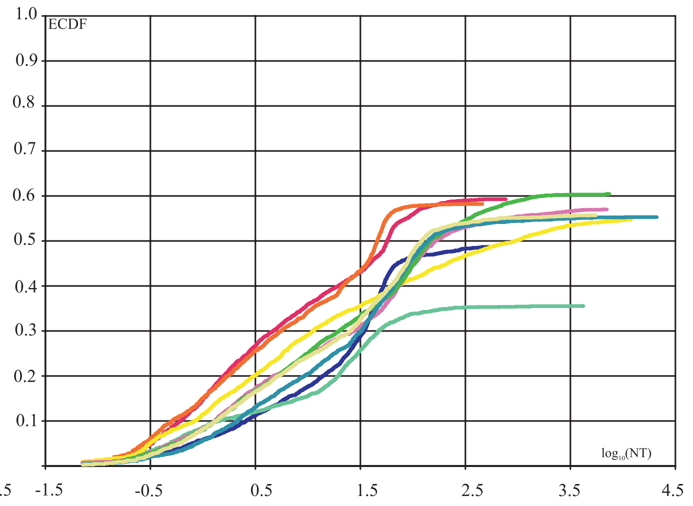


(b) ECDF for NT, $F_t = 0.0$ and $256 \le n < 511$

Figure 2: Although we conduct experiments with benchmark instances with between 14 and 85900 cities, here we present ECDF diagrams for subsets of our experiments selected by problem scale, using the normalized runtime NT and $F_t = 0.0$.



(a) ECDF for NT and $F_t = 0.0$



(b) ECDF for FEs and $F_t = 0.0$

Figure 3: ECDF diagram of hybrid EC versions for different (log-scaled) runtime measures and goal errors

3.3 Summary

In Figure 4, we present an abridged algorithm performance ranking of the representative setups obtained with the *TSP Suite*. This ranking combines a variety of different performance measures, ranging from areas under the ECDF and ERT curves to statistical tests comparing end results and expected runtimes to reach certain goal tour lengths. We observe that MA(2+4)-LK10-MNS-HX is the overall best, most robust, and versatile algorithm in the experiment, which complies with our other observations. Interestingly, the second-best method is an EC-LS-LS hybrid. The three best algorithms all employ a combination of MNS and LK10.

4 CONCLUSIONS AND FUTURE WORK

In this work, we introduced the concept of hybridizing two LS methods by combing them with a crossover operator. We explore the utility of this method on the example of the TSP. We therefore pairwise combined the three state-of-the-art TSP solvers FSM**, MNS, and the LK heuristic with Heuristic and Order Based crossover. We then hybridized the new LS-LS-X algorithms further with an EC method, namely an $(\mu + \lambda)$-EA. We conducted a large-scale experimental study applying 36 different algorithm setups to all of the 110 benchmark instances from *TSPLib*. Our experiments have led us to four major conclusions:

MA(2+4)-LK10-MNS-HX *(rank 1)*, MA(16+64)-LK10-MNS-HX *(2.5)*, MA(2+8)-LK10-MNS-HX *(2.5)*, MA(16+64)-LK10-MNS-OX2 *(4)*, MA(16+64)-LK10-MNS *(5)*, MA(16+64)-FSM**-LK10-HX *(6.5)*, MA(2+8)-LK10-MNS-OX2 *(6.5)*, MA(2+4)-LK10-MNS-OX2 *(8)*, MA(16+64)-FSM**-LK10 *(9)*, LK10-MNS-HX *(10)*, MA(16+64)-LK10 *(11)*, MA(2+8)-FSM**-LK10 *(12.5)*, MA(2+4)-FSM**-LK10-OX2 *(12.5)*, MA(2+8)-FSM**-LK10-HX *(14)*, MA(2+8)-FSM**-LK10-OX2 *(15)*, MA(16+64)-FSM**-LK10-OX2 *(16)*, MA(2+8)-LK10 *(17)*, MA(2+4)-LK10-MNS *(18)*, MA(2+8)-LK10-MNS *(19)*, MA(2+4)-FSM**-LK10 *(20.5)*, MA(2+4)-LK10 *(20.5)*, MA(2+4)-FSM**-LK10-HX *(22)*, MA(16+64)-FSM** *(23)*, MA(2+8)-FSM** *(24)*, MA(2+4)-FSM** *(25)*, FSM**-LK10-OX2 *(26)*, LK10-MNS-OX2 *(27)*, MA(16+64)-MNS *(28)*, FSM**-LK10-HX *(29)*, MA(2+8)-MNS *(30)*, LK10-MNS *(31)*, FSM**-LK10 *(32)*, FSM** *(33.5)*, LK10 *(33.5)*, MA(2+4)-MNS *(35)*, MNS *(36)*.

Figure 4: Algorithm ranking from best to worst, based on various performance measures and statistics (see [35] for details). The different algorithm types pure local search, LS-LS hybrid, LS-LS-X hybrid, EC-LS hybrid, EC-LS-LS hybrid and EC-LS-LS-X hybrid are highlighted.

(1) The new LS-LS-X hybrids are better than their pure LS algorithm and LS-LS hybrid components. This means that the new idea of combining two different LS algorithms with crossover operator is very promising.
(2) The new EC-LS-LS-X hybrids outperform the LS-LS-X algorithms as well as EC-LS and EC-LS-LS hybrids. MA(2+4)-LK10-MNS-HX becomes the new most powerful hybrid EA algorithm in the huge collection of algorithms and experimental results of the popular *TSP Suite*.
(3) Different LS-LS hybrids have different suitable crossover operators. The setup LK10-MNS-HX, for instance, outperforms LK10-MNS-OX2, while FSM**-LK10-HX yields to FSM**-LK10-OX2. Compared with OX2, the HX operator is a better choice in MAs for EC-LS-LS-X hybrids (while the MAs internally employ Edge Crossover).

In our future work, we will investigate other crossover operators for hybridizing two LS algorithms: The two most efficient operators, edge assembly crossover [29] and Generalized (Asymmetric) Partition Crossover [32, 36, 37], have not been investigated in this study. We will create implementations of them compatible to the *TSP Suite* experimental procedure in the near future.

We will also construct LS-LS-X hybrids with the efficient Tabu Search-based TSP solvers introduced in [42].

Furthermore, instead of hard-wiring the choice of the LS algorithms and crossover operators in our (EC-)LS-LS-X hybrids, we will investigate choosing them according to a random distribution. This choice will be adaptive and change according to the success rate of such refinements.

Finally, the concept of LS-LS-X and EC-LS-LS-X hybrid algorithms is also promising for other problem domains. We will therefore explore its utility on several other well-known combinatorial optimization problems.

Acknowledgments We acknowledge support from the National Natural Science Foundation of China under Grants 61673359, 61150110488, and 71520107002 and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] David Lee Applegate, Robert E. Bixby, Vašek Chvátal, and William John Cook. 2007. *The Traveling Salesman Problem: A Computational Study.* Princeton, NJ, USA: Princeton University Press.

[2] David Lee Applegate, William John Cook, and André Rohe. 2003. Chained Lin-Kernighan for Large Traveling Salesman Problems. *INFORMS Journal on Computing (JOC)* 15, 1 (Winter 2003), 82–92.

[3] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz (Eds.). 1997. *Handbook of Evolutionary Computation.* New York, NY, USA: Oxford University Press, Inc.

[4] Christian Blum, Raymond Chiong, Maurice Clerc, Kenneth Alan De Jong, Zbigniew Michalewicz, Ferrante Neri, and Thomas Weise. 2011. Evolutionary Optimization. In *Variants of Evolutionary Algorithms for Real-World Applications*, Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz (Eds.). Berlin/Heidelberg: Springer-Verlag, Chapter 1, 1–29. DOI:https://doi.org/10.1007/978-3-642-23424-8_1

[5] Mark S. Boddy and Thomas L. Dean. 1989. *Solving Time-Dependent Planning Problems.* Technical Report CS-89-03. Providence, RI, USA: Brown University, Department of Computer Science.

[6] Raymond Chiong, Ferrante Neri, and Robert Ian McKay. 2009. Nature that Breeds Solutions. In *Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science and Engineering.* Hershey, PA, USA: Information Science Reference, Chapter 1, 1–24.

[7] Kenneth Alan De Jong. 2006. *Evolutionary Computation: A Unified Approach.* Bradford Books, Vol. 4. Cambridge, MA, USA: MIT Press.

[8] David B. Fogel. 1988. An Evolutionary Approach to the Traveling Salesman Problem. *Biological Cybernetics* 60, 2 (Dec. 1988), 139–144.

[9] Fred Glover. 1992. *Ejection Chains and Combinatorial Leverage for the Traveling Salesman Problems.* Technical Report. University of Colorado-Boulder, Boulder, CO, USA.

[10] Fred W. Glover. 1992. New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems. In *Computer Science and Operations Research: New Developments in Their Interfaces*, Osman Balci and Ramesh Sharda (Eds.). Oxford, UK: Pergamon Press, 449–509.

[11] Fred W. Glover. 1996. Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. *Discrete Applied Mathematics – The Journal of Combinatorial Algorithms, Informatics and Computational Sciences* 65, 1-3 (March 1996), 223–253.

[12] John J. Grefenstette. 1987. Incorporating Problem Specific Knowledge into Genetic Algorithms. In *Genetic Algorithms and Simulated Annealing*, Lawrence Davis (Ed.). Pitman, 42–60.

[13] Michael Guntsch and Martin Middendorf. 2002. Applying Population Based ACO to Dynamic Optimization Problems. In *From Ant Colonies to Artificial Ants – Proc. of the Third Intl. Workshop on Ant Colony Optimization (ANTS'02) (LNCS)*, Vol. 2463/2002. Berlin, Germany: Springer-Verlag GmbH, Brussels, Belgium, 111–122.

[14] Gregory Z. Gutin and Abraham P. Punnen (Eds.). 2002. *The Traveling Salesman Problem and its Variations.* Combinatorial Optimization, Vol. 12. Norwell, MA, USA: Kluwer Academic Publishers.

[15] Nikolaus Hansen, Anne Auger, Steffen Finck, and Raymond Ros. 2012. *Real-Parameter Black-Box Optimization Benchmarking: Experimental Setup.* Technical Report. Orsay, France: Université Paris Sud, INRIA Futurs, Équipe TAO.

[16] Pierre Hansen and Nenad Mladenović. 2001. Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research (EJOR)* 130, 3 (May 1 2001), 449–467.

[17] Keld Helsgaun. 1998. *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic.* Datalogiske Skrifter (Writings on Computer Science) 81. Roskilde, Denmark: Roskilde University, Department of Computer Science.

[18] Keld Helsgaun. 2009. General k-opt Submoves for the Lin–Kernighan TSP Heuristic. *Mathematical Programming Computation* 1, 2-3 (Oct. 2009), 119–163.

[19] John Henry Holland. 1975. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Ann Arbor, MI, USA: University of Michigan Press.

[20] Holger H. Hoos and Thomas Stützle. 1998. Evaluating Las Vegas Algorithms – Pitfalls and Remedies. In *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence (UAI'98).* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Madison, WI, USA, 238–245.

[21] Yan Jiang, Thomas Weise, Jörg Lässig, Raymond Chiong, and Rukshan Athauda. 2014. Comparing a Hybrid Branch and Bound Algorithm with Evolutionary Computation Methods, Local Search and their Hybrids on the TSP. In *Proc. of the IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS'14), Proc. of the IEEE Symposium Series on Computational Intelligence (SSCI'14).* Los Alamitos, CA, USA: IEEE Computer Society Press, Orlando, FL, USA. DOI:https://doi.org/10.1109/CIPLS.2014.7007174

[22] Pedro Larrañaga, Cindy M. H. Kuijpers, Roberto H. Murga, Iñaki Inza, and Sejla Dizdarevic. 1999. Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators. *Journal of Artificial Intelligence Research (JAIR)* 13, 2 (April 1999), 129–170.

[23] Eugene Leighton Lawler, Jan Karel Lenstra, Alexander Hendrik George Rinnooy Kan, and David B. Shmoys. 1985. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* Chichester, West Sussex, UK: Wiley Interscience.

[24] Shen Lin and Brian Wilson Kernighan. 1973. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research* 21, 2 (March–April 1973), 498–516.

[25] Weichen Liu, Thomas Weise, Yuezhong Wu, and Raymond Chiong. 2015. Hybrid Ejection Chain Methods for the Traveling Salesman Problem. In *Proc. of the 10th Intl. Conf. Bio-Inspired Computing – Theories and Applications (BIC-TA'15)*. Communications in Computer and Information Science, Vol. 562. Berlin/Heidelberg: Springer-Verlag, Hefei, Anhui, China, 268–282. DOI: https://doi.org/10.1007/978-3-662-49014-3_25

[26] Peter Merz and Bernd Freisleben. 2001. Memetic Algorithms for the Traveling Salesman Problem. *Complex Systems* 13, 4 (2001), 297–345.

[27] Zbigniew Michalewicz. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. Berlin, Germany: Springer-Verlag GmbH.

[28] Pablo Moscato. 1989. *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. Caltech Concurrent Computation Program C3P 826. Pasadena, CA, USA: California Institute of Technology (Caltech), Caltech Concurrent Computation Program (C3P). http://www.each.usp.br/sarajane/SubPaginas/arquivos_aulas_IA/memetic.pdf

[29] Yuichi Nagata and Shigenobu Kobayashi. 2013. A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Traveling Salesman Problem. *INFORMS Journal on Computing* 25, 2 (2013), 346–363. DOI: https://doi.org/10.1287/ijoc.1120.0506

[30] César Rego. 1998. Relaxed Tours and Path Ejections for the Traveling Salesman Problem. *European Journal of Operational Research* 106, 2 (1998), 522–538.

[31] Gerhard Reinelt. 1991. TSPLIB – A Traveling Salesman Problem Library. *ORSA Journal on Computing* 3, 4 (Fall 1991), 376–384.

[32] Renato Tinós, Darrell Whitley, and Gabriela Ochoa. 2014. Generalized Asymmetric Partition Crossover (GAPX) for the Asymmetric TSP. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '14, July 12-16, Vancouver, BC, Canada*. ACM, New York, NY, USA, 501–508. DOI: https://doi.org/10.1145/2576768.2598245

[33] Dave Andrew Douglas Tompkins and Holger H. Hoos. 2004. UBCSAT: An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT. In *Revised Selected Papers from the Seventh Intl. Conf. on Theory and Applications of Satisfiability Testing (SAT'04) (LNCS)*, Vol. 3542. Berlin, Germany: Springer-Verlag GmbH, Vancouver, BC, Canada, 306–320.

[34] Thomas Weise. 2009. *Global Optimization Algorithms – Theory and Application*. Germany: it-weise.de (self-published). http://www.it-weise.de/projects/book.pdf

[35] Thomas Weise, Raymond Chiong, Ke Tang, Jörg Lässig, Shigeyoshi Tsutsui, Wenxiang Chen, Zbigniew Michalewicz, and Xin Yao. 2014. Benchmarking Optimization Algorithms: An Open Source Framework for the Traveling Salesman Problem. *IEEE Computational Intelligence Magazine (CIM)* 9, 3 (Aug. 2014), 40–52. DOI: https://doi.org/10.1109/MCI.2014.2326101

[36] Darrell Whitley, Doug Hains, and Adele E. Howe. 2009. Tunneling between Optima: Partition Crossover for the Traveling Salesman Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09, July 8-12, Montreal, Québec, Canada*, Franz Rothlauf (Ed.). ACM, New York, NY, USA, 915–922. DOI: https://doi.org/10.1145/1569901.1570026

[37] Darrell Whitley, Doug Hains, and Adele E. Howe. 2010. A Hybrid Genetic Algorithm for the Traveling Salesman Problem Using Generalized Partition Crossover. In *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature, PPSN XI, Part I, Sept. 11-15, Kraków, Poland (Lecture Notes in Computer Science)*, Vol. 6238. Springer-Verlag GmbH, Berlin, Germany, 566–575. DOI: https://doi.org/10.1007/978-3-642-15844-5_57

[38] L. Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. 1989. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. In *Proc. of the Third Intl. Conf. on Genetic Algorithms (ICGA'89)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., Fairfax, VA, USA: George Mason University (GMU), 133–140.

[39] Gerhard J. Woeginger. 2003. Exact Algorithms for NP-hard problems: A Survey. In *Combinatorial Optimization – Eureka, You Shrink!*, Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi (Eds.). Springer, 185–207. DOI: https://doi.org/10.1007/3-540-36478-1_17

[40] Yuezhong Wu, Thomas Weise, and Raymond Chiong. 2015. Local Search for the Traveling Salesman Problem: A Comparative Study. In *Proc. of 14th IEEE Conf. on Cognitive Informatics & Cognitive Computing (ICCI*CC'15)*. Beijing, China, 213–220. DOI: https://doi.org/10.1109/ICCI-CC.2015.7259388

[41] Yuezhong Wu, Thomas Weise, and Weichen Liu. 2016. Hybridizing Different Local Search Algorithms with Each Other and Evolutionary Computation: Better Performance on the Traveling Salesman Problem. In *Proceedings of the 18th Genetic and Evolutionary Computation Conference (GECCO'18), July 20-24, Denver, CO, USA*. ACM Press, New York, NY, USA, 57–58. DOI: https://doi.org/10.1145/2908961.2909001

[42] Dan Xu, Thomas Weise, Yuezhong Wu, Jörg Lässig, and Raymond Chiong. 2015. An Investigation of Hybrid Tabu Search for the Traveling Salesman Problem. In *Proc. of the 10th Intl. Conf. Bio-Inspired Computing – Theories and Applications (BIC-TA'15)*. Communications in Computer and Information Science, Vol. 562. Berlin/Heidelberg: Springer-Verlag, Hefei, Anhui, China, 523–537. DOI: https://doi.org/10.1007/978-3-662-49014-3_47