Dan Xu, Thomas Weise, Yuezhong Wu, Jörg Lässig, and Raymond Chiong. An Investigation of Hybrid Tabu Search for the Traveling Salesman Problem. In Proceedings of the 10th International Conference on Bio-Inspired Computing – Theories and Applications (BIC-TA'15), September 25 – 28, 2015. Hefei, Anhui, China, volume 562 of Communications in Computer and Information Science. Berlin/Heidelberg: Springer-Verlag, pages 523 – 537, ISBN 978-3-662-49013-6. doi:10.1007/978-3-662-49014-3_47

An Investigation of Hybrid Tabu Search for the Traveling Salesman Problem

Dan Xu[1], Thomas Weise[1,2⋆], Yuezhong Wu[1], Jörg Lässig[3], and Raymond Chiong[4]

[1] Joint USTC-Birmingham Research Institute in Intelligent Computation and Its Applications (UBRI), School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China, 230027.
dandy@mail.ustc.edu.cn
tweise@ustc.edu.cn
yuezhong@mail.ustc.edu.cn

[2] Enterprise Application Development Group, Department of Computer Science, University of Applied Sciences Zittau/Görlitz, D-02826 Görlitz, Germany
jlaessig@hszg.de

[3] School of Design, Communication and IT, Faculty of Science and IT, The University of Newcastle, Callaghan, NSW 2308, Australia.
Raymond.Chiong@newcastle.edu.au

Abstract. The Traveling Salesman Problem (TSP) is one of the most well-known problems in combinatorial optimization. Due to its $\mathcal{NP}$-hardness, research has focused on approximate methods like metaheuristics. Tabu Search (TS) is a very efficient metaheuristic for combinatorial problems. We investigate four different versions of TS with different tabu objects and compare them to the Lin-Kernighan (LK) heuristic as well as the recently developed Multi-Neighborhood Search (MNS). LK is currently considered to be the best approach for solving the TSP, while MNS has shown to be highly competitive. We then propose new hybrid algorithms by hybridizing TS with Evolutionary Algorithms and Ant Colony Optimization. These hybrids are compared to similar hybrids based on LK and MNS. This paper presents the first statistically sound and comprehensive comparison taking the entire optimization processes of (hybrid) TS, LK, and MNS into consideration based on a large-scale experimental study. We show that our new hybrid TS algorithms are highly efficient and comparable to the state-of-the-art algorithms along this line of research.

Keywords: Traveling salesman Problem, Tabu search, Evolutionary algorithms, Ant colony optimization, Memetic algorithms

1 Introduction

The Traveling Salesman Problem (TSP) [1,2,3,4] is perhaps the most-studied optimization problem. Given n cities, a salesman leaves from a start city, visits

⋆ Corresponding author.

each of the other cities exactly once, and returns back to the start city. The tour of the salesman thus forms a cyclic path passing through each node in a fully-connected graph of n nodes exactly once. In the graph, each edge from city i to city j has a weight $D_{i,j}$, which represents the distance. The task is to find a tour (path) that yields the minimal total weight sum.

The TSP has been researched for decades. The optimization version of it is $\mathcal{NP}$-hard [3], so the worst-case runtime complexity of any exact TSP solver is exponential [5]. As exactly solving the TSP is not generally feasible, many approximate algorithms such as Evolutionary Algorithms (EAs) [6,7,8] and Ant Colony Optimization (ACO) [9,10,11] have been introduced to tackle it. The Lin-Kernighan (LK) heuristic [12], a Local Search (LS) method, and its derivatives are considered to be the best performers for the TSP. In [13], an alternative approach known as Multi-Neighborhood Search (MNS) was introduced and found to be a more efficient LS approach for the TSP than, e.g., Variable Neighborhood Search [14] or Hill Climbing. MNS has also been shown to perform better than pure EAs and Population-based ACO (PACO) [15].

Tabu Search (TS) [16,17] is one of the most widely known metaheuristics for combinatorial problems and is able to provide high-quality solutions for many problems. An EA hybridized with TS is shown to outperform pure TS on small-scale TSP instances in [18].

The TSP is an ideal testbed for investigating and comparing new algorithms' performances, since it is easy to understand and standard benchmarks with known solutions (like the *TSPLib*[19]) are available. Additionally, there is a large body of related work involving both the TSP and TS. However, virtually all of them either only focused on small-scale problems, only compared different parameter settings of the same tabu object [20] or did not apply sound statistics to evaluate their results.

With the present work, we make the following contributions:

1. A full in-depth analysis of the performance of a TS algorithm *over runtime* on all 110 of the symmetric *TSPLib* benchmark instances. The analysis is based on both small-scale and large-scale instances, several different performance metrics, and several different ways to measure runtime.
2. The comparison of three different tabu criteria as well as two different ways to search the neighborhood of the current solution in TS.
3. A detailed comparison of TS to state-of-the-art LS algorithms such as LK and MNS.
4. The introduction of two novel hybrid forms of the investigated TS algorithm based on both EAs and ACO.
5. A detailed comparison of these hybrid TS algorithms to hybrid variants of the above-mentioned LS approaches constructed in the same way as those of TS.

The remainder of this paper is organized as follows. We first discuss related work on automated experimentation (Section 2.1) and the TSP (Section 2.2). We then present our TS approach, the three tabu criteria and new hybrid algorithms

in Section 3. Our large-scale experimental study is discussed in Section 4 and conclusions as well as future work are given in Section 5.

2 Related Work

The focus of this paper is not only to introduce new TS algorithms for the TSP, but also on sound experimentation. Before outlining the state-of-the-art TSP algorithms, we discuss the nature of these algorithms and how they influence the experimental approach.

2.1 Related work on experimentation

All LS methods and most metaheuristics (e.g., EAs, ACO) are anytime algorithms [21]. Even some exact methods, like Branch and Bound (BB) [22], belong to 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. LS methods, for instance, begin with a random solution and iteratively refine it. Given a TSP instance, BB would maintain the best solution discovered so far and investigate a set of other solutions only if the lower bound for their tour length is better than the actual tour length of that best-so-far solution.

If an anytime algorithm $\mathcal{A}$ provides a better final solution than another anytime algorithm $\mathcal{B}$, does this make $\mathcal{A}$ better? The traditional answer would be *yes*, but what if the best guess of $\mathcal{A}$ for the solution is much worse than $\mathcal{B}$'s until after a very long (run)time? Due to their nature, anytime algorithms thus cannot be assessed just by a final solution and runtime requirement, but by their entire runtime behavior [13].

In the field of metaheuristic optimization, experimentation is the most important tool to assess and compare the performance of different algorithms. Even though this has been the case for a long time, experimentation approaches adopted in most of the existing studies rely mainly on the most basic statistics, some of which are even flawed [13]. Proper experimentation is a complex, time-demanding and cumbersome process.

The *COmparing Continuous Optimisers* (COCO) [23] system for numerical optimization, used in the Black-Box Optimization Benchmarking workshops, is one of the earliest approaches aiming to reduce the workload of an experimenter by automatizing most of the steps involved in an experimentation process. Its evaluation procedure generates statically structured papers that contain diagrams with runtime behavior information. The necessary data is automatically collected from executed experiments.

UBCSAT [24], an experimental framework for satisfiability problems, is another representative example of work in this area. UBCSAT focuses on a specific family of algorithms: the Stochastic LS [25]. In COCO, the objective function would automatically gather log data before returning its result to the algorithm. In UBCSAT, this would be done through a trigger architecture, which can also compute complex statistics online and provide them to the running algorithm.

COCO and UBCSAT both explore algorithm behavior over runtime instead of just comparing end results.

The *TSP Suite*[13] used for running the experiments in our work takes the idea one step further. First, it provides software development support such as unit testing. Second, it takes care of parallelization or distribution of workload on a multi-processor system or cluster. Third, like in COCO, an algorithm performance report can be created automatically. The difference, however, is that it includes an in-depth description of the experimental procedure and presents several different statistical analyses, such as statistical tests comparing the measured runtimes and end results, automated comparisons of the estimated running time (ERT) [23] curves over goal objective values or problem scales, and automated comparisons of empirical cumulative distribution functions (ECDFs) [23,26,24]. All of these statistics result in algorithm rankings, which are later aggregated into a global ranking list. The global ranking provides some insights on the general performance of a TSP solver.

Our *TSP Suite* is the first framework addressing the issue of runtime measures. Traditionally, runtime is either measured in CPU seconds or the number of generated candidate solutions (i.e., objective "function evaluations" or FEs in short). The problem with using CPU time as time measure is that results obtained on different machines are inherently incomparable, while the number of generated candidate solutions gives no information about the actual runtime of an algorithm, since 1 FE may have different computational complexities in different algorithms. For instance, in a LS algorithm or a mutation operator in an EA, a new solution may be obtained from an existing tour of known length by swapping two cities, which has the complexity of $\mathcal{O}(1)$. In ACO, the creation of one new solution has time complexity $\mathcal{O}(n^2)$. In the *TSP Suite*, these shortcomings have been addressed by introducing two new time measures: the normalized runtime (NT) and the number of times the distance matrix D is accessed (distance evaluations, DEs). The NT is the CPU time divided by a specific performance factor based on machine and problem instances, thus rendering (somewhat) machine independent time measure results. The DEs take into account the different complexities of 1 FE in different algorithms. Statistical analyses through the *TSP Suite* are all conducted three times, based on the FE, NT, and DE respectively. The algorithm ranking created therefore represents a more balanced and fair perspective of an algorithm's performance.

2.2 Related work on the Traveling Salesman Problem

A large body of work on the TSP exists, but the relevant literature focuses mainly on single-algorithm end result comparisons and rarely takes into account algorithms of different families. Notable exceptions for the latter can be found in [27,28], while the runtime behavior of different pure and hybrid metaheuristics as well as LS methods were studied in [13]. BB, LS and Evolutionary Computation (EC) methods were considered in [22], and the state-of-the-art LS methods LK and MNS were compared in [29].

Generally speaking, LS algorithms dominate the research on TSP. They start at a random or heuristically-generated solution. They remember the best solution discovered so far and try to improve it. These improvements usually take place in the form of modifications to the tour. The most prominent examples of such modifications within the TSP domain are m-opt moves [13]: the exchange of two cities in a tour, for instance, corresponds to the deletion and addition of four edges (m = 4-opt) [30], the rotation of a sub-sequence of cities to the left or right results in the deletion and addition of three edges (3-opt) [**?**,31], while the reversal of a sub-sequence of a tour requires deleting and adding two edges (2-opt) [**?**,32]. However, LS is likely to be trapped by local optima.

TS [33,16] is a LS method attempting to avoid this premature convergence problem. In the last 30 years, TS was used to solve the TSP and its variants several times as discussed in the comprehensive survey by Basu [20]. According to Basu's survey, most of the past TS-related studies have focused on just small-scale TSP instances with no comparison of different tabu criteria, and typically used only brittle end-of-run statistics. To fill this gap, we will conduct a large-scale experimental study considering problem instances with more than 1000 nodes. We will also compare different tabu objects and final results with robust statistical tests before applying statistics over the entire optimization processes. In Section 3.1, where our algorithm and its components are introduced, we will provide additional references to corresponding related studies on TS for the TSP.

3 Investigated Algorithms

3.1 Tabu search

In each iteration, a LS algorithm will look for a better solution s' in the neighborhood $N(s)$ of the current (best) solution s. $N(s)$ is spanned by the available search operators. For a very simple LS algorithm that only accepts s' when it is better than s, the algorithm is likely to get trapped in a local optimum, i.e., a solution whose neighborhood does not contain any better solution while itself is not the globally optimal one.

To prevent this, the LS algorithm should be able to accept a move from s to s' even if s' is worse than s – Simulated Annealing [34] is an example of such an algorithm. However, this may lead to a "cycling" effect in the search space, if the move leading out of the local optimum is undone in the next algorithm iteration. TS therefore incorporates a memory structure, called tabu list T, to forbid certain moves that would return to a recently visited solution. Instead of searching the neighborhood $N(s)$ of s, TS investigates $N(s) \setminus T$. For this general procedure, we analyze the performance impact of the following design criteria.

Tabu object One design criterion for TS is the data structure that is used to represent prohibited solutions and the tabu list. We investigate three such tabu criteria [35]:

Solutions The simplest way is to "tabu" solutions that have been accepted in the process of the algorithm. Our implementation does this by forbidding not only a specific solution, such as $(1, 2, 3, 4, 5)$, but also all *equivalent* solutions in the symmetric TSP, such as $(5, 4, 3, 2, 1)$, $(3, 4, 5, 1, 2)$ or $(4, 3, 2, 1, 5)$. T here is implemented as a hash table with $\mathcal{O}(n)$ cost to compute the hash code of a tour. We refer to TS using this method as TSS.

Moves We also prevent certain *moves*, i.e., applications of search operators, that could lead back to a recently visited solution. Our TS uses three different search operators, namely 2-, 3-, and 4-opt, each of which has a tuple of two node indexes as parameters. We implement this tabu object by storing the nodes corresponding to the indexes in T and preventing using them again for the tabu tenure. If we apply a 2-opt move (sub-tour reversal) at index 2 and 4 to tour $(5, 3, 2, 1, 4)$, we would get $(5, 1, 2, 3, 4)$ and prevent the two nodes 3 and 1 from being the start or end node of subsequent moves. We refer to TS using this method as TSM.

Objective Values We forbid repetitive objective values, i.e., tour lengths. If a tour with length 783 is discovered, then for the tabu tenure $|T|$, the search must not move to any tour with the same length (regardless of whether this tour is the one previously discovered, equivalent to it, or entirely different). We refer to TS using this method as TSO.

Tabu tenure The number of times (algorithm iterations) a certain move or solution is prohibited by the tabu list is called the tabu tenure. It corresponds to the maximum amount of elements in T. This tenure can be either fixed or based on the problem scale. We compare both methods in our study. In the notation of setup names used in our experiments, we append a tuple $(\alpha+\gamma)$ to the algorithm names, e.g., TSO$(10 + 10)$. The tabu tenure then equals $\alpha + \gamma\sqrt{n}$, i.e., α is an absolute value and γ will be multiplied by the square root of the problem scale.

Soft restarts If the algorithm cannot improve its best solution any further, it can be restarted. Instead of restarting at a completely random solution, we randomly shuffle a uniformly chosen part of the current best solution. This policy, introduced by us in [13], leads to sufficient randomness while having a chance of preserving some good building blocks. At a restart, the tabu list T will *not* be emptied. This means two restarts that happen to start at the same solution (by chance) will still take different paths in the search space.

Investigating the neighborhood In TS, the non-tabu neighborhood $N(s)-T$ of the current solution s is scanned for a better solution s'. If no such solution can be found, our algorithms will immediately perform a soft restart (see above). For all three tabu objects, we search the entire neighborhood for the best possible s'. An alternative way is to search the neighborhood and if an s' better than

s is discovered then accept that one immediately. Since this approach is often considered as inferior [29,36], we only test it for the solution tabu object setups. We refer to it as TSF.

3.2 The Lin-Kernighan algorithm

The LK heuristic [12] and its derivatives dominate today's TSP research [37,36,38]. A tour can be considered as β-optimal when it is impossible to improve the tour quality by replacing β edges. LK can be considered as a variable β-opt LS. At each step, the algorithm tests whether replacing β edges may achieve a shorter tour (for increasing values of β). Let s be the current tour. The algorithm will then, in each iteration, construct the sets $X = \{X_1, \ldots, X_\beta\}$ of edges to be deleted from s and $Y = \{Y_1, \ldots, Y_\beta\}$ of edges to be added to s, such that the resulting tour would be valid and shorter. The interchange of these edges is then a β-opt move. In the beginning, X and Y are empty. Pairs of edges are added to X and Y such that the end node of the edge added to X is the start node of the edge added to Y, whose end node will then become the start node of the edge added to X in the next iteration, if any. The LK heuristic we compare our TS to is based on that by Wu et al. [29], which uses the restart policy defined in [13], the same as our TS.

3.3 Multi-neighborhood search

MNS is another efficient LS method for the TSP. In each iteration, MNS performs an $\mathcal{O}(n^2)$ scan that investigates the same neighborhoods as our TS at once. It therefore tests all indexes i and j as potential indexes of start or end of a search operation. For each pair $\{i, j\}$, the gain of each operation is computed and all discovered improving moves enter a queue. The access to distance matrix D is minimized by remembering (and updating) the lengths of all n edges in the current tour and avoiding the check of redundant moves (swapping the cities at indexes i and $i+1$ is equivalent to a reversal of the sub-sequence from i to $i+1$, for instance).

After the scan, the best discovered move is carried out. Doing this may invalidate some other moves in the queue, e.g., a sub-sequence reversal performed overlaps with a potential sub-sequence left rotation. 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 will be performed, as new moves may have become possible. During this scan, only moves that at least intersect with the previously modified sub-sequence(s) of the current best solution need to be considered in order to speed up the search. If no improving moves can be found anymore, a random sub-sequence of the current tour will be randomly shuffled – the same soft restart method as used in our TS and LK.

MNS scans for all 2-opt and some 3- and 4-opt moves, thus investigates the same neighborhoods as our TS. LK, on the other hand, investigates a larger sub-set of possible m-opt moves.

3.4 Evolutionary algorithms

EAs are population-based EC methods that start by generating a set of λ random solutions. Out of these, the best $\mu \leq \lambda$ solutions are selected as "parents" of the second generation: λ offspring 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 λ offspring and their μ parents in each generation in the case of a $(\mu + \lambda)$-EA. A (μ, λ)-EA selects only from the μ offspring. In this paper, we investigate such EAs using the neighborhoods also used by our TS as mutation operators. Edge Crossover [39], which generates a new solution by picking edges belonging to either of its two parents, is applied as the recombination operator at a crossover rate of 1/3.

Hybridization of LS and EAs has a long tradition. Such hybrid algorithms, where the LS algorithm either takes the place of the mutation operator or is applied to each new solution (stemming from mutation or crossover), are called Memetic Algorithms (MAs). Especially in the TSP domain, MAs are considered to perform well. We propose MAs based on our TS, so called hMA$(\mu \stackrel{+}{,} \lambda)$TS, and compare them with likewise-structured hybrids using LK and MNS as LS. The little h in the name indicates that the first population of the MAs is not generated randomly, but instead stemmed from the Edge-Greedy, Double Minimum Spanning Tree, Savings, Double-Ended Nearest Neighbor, and Nearest Neighbor Heuristics, as in [13].

3.5 Ant colony optimization

ACO is another EC approach first introduced by Dorigo in 1992 [9]. It took inspiration from the way ants find and reinforce short paths during foraging using pheromones for communication. Although such algorithms are able to perform well in many small-scale combinatorial problems, they suffer from quadratic memory requirements as well as quadratic complexity of the process of creating solutions.

In this paper, we investigate a state-of-the-art ACO variant, Population-based ACO (PACO) [15], which has linear memory requirements. The PACO algorithm maintains a population of k solutions. Pheromones are defined by the edges occurring in these solutions. In each algorithm iteration, m solutions are created as in standard ACO and the "oldest" solution in the whole population is replaced by the best of the new generated solutions. Limited hybrid ACO approaches have been applied to the TSP, although it was shown in [13] that they perform particularly well. We therefore propose hybrid hPACO(k, m)TS and similar variants with LK and MNS (which are heuristically initialized in the same way as the hMAs).

4 Experiments and Results

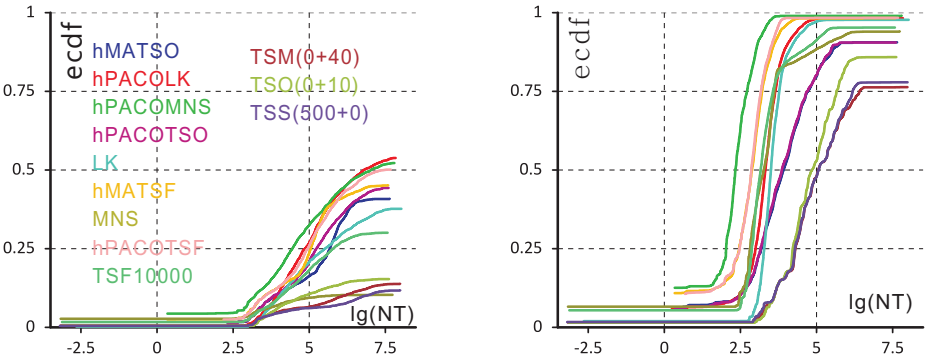
4.1 Experimental setup

We conducted experiments using the symmetric *TSPLib* benchmark cases, for which all optima are known. We thus can measure the quality of a solution as relative error f, i.e., the factor by which a solution (tour) is longer than the optimum. Here, $f = 0$ stands for the optimal solution, and $f = 1$ indicates one that is twice as long. We obtained results for all 110 problem instances up to 85,900 cities, and we considered all the results in the evaluation.

A total of 45 TS setups were built: 25 pure algorithms, 10 hybrids with PACO, and 10 hybrid MAs. For each type of the tabu objects, we tested seven tabu tenure settings: (10+0), (100+0), (500+0), (10000+0), (0+10), (0+20), and (0+40). For TSF, we only tested four tabu tenure settings: (10+0), (100+0), (500+0), and (10000+0). For the hybrid algorithms, we tested the best known settings from [13], i.e., hPACO(3,10) and hMA(16+64). We hybridized them with all four TSF settings and tabu tenures (10+10) and (500+0) for each of the three tabu criteria.

Additionally, we investigated the pure LK and MNS as well as hybrids of PACO with LK and MNS. Obviously, there are too many setups to present all of them in a single plot. Therefore, we divided our analysis into comparisons of pure and hybrid algorithms. We present only the setups that have performed the best in our experiments each time.

4.2 Pure algorithm performance



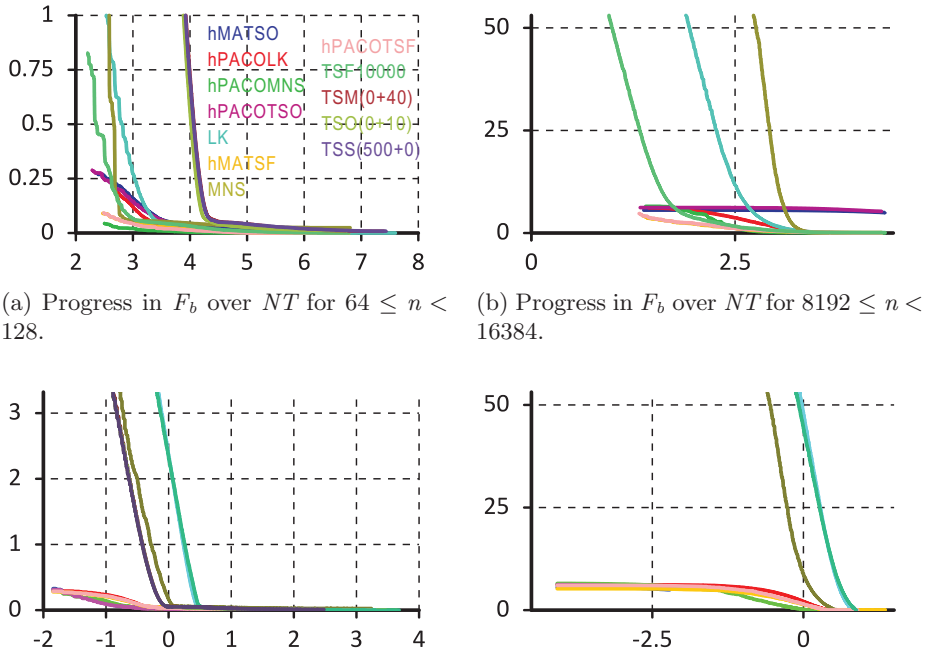
(a) ECDF for NT and $F_t = 0.0$.

(b) ECDF for NT and $F_t = 0.1$.

Fig. 1: ECDF diagrams for different (log-scaled) runtime measures and goal errors.

Let us first explore the performance of the pure TS algorithms. In Figure 1, we plot the ECDF for different goal errors F_t and runtime measures. The ECDF illustrates the fraction of runs that have discovered a solution with $F_b \leq F_t$ at a given point in time. If the ECDF reaches 1, all runs have found such a solution. In Figure 2, we plot the best normalized objective value F_b discovered by an algorithm over runtime, where $F_b = 0$ means that the global optimal has been reached and $F_b = 1$ indicates the discovery of a tour twice as long.

Tabu object We first analyze the three types of tabu criteria introduced in Section 3.1. In Figure 1, whether based on runtime measure *NT* or *FE*, the ECDF of TSO increases faster in the beginning and reaches higher end values. TSM and TSS are not distinguishable.



(a) Progress in F_b over NT for $64 \leq n < 128$.

(b) Progress in F_b over NT for $8192 \leq n < 16384$.

(c) Progress in F_b over FE for $64 \leq n < 128$.

(d) Progress in F_b over FE for $8192 \leq n < 16384$.

Fig. 2: Progress diagrams for different (log-scaled) time measures and problem scales.

Figure 2 shows the improvement of solution quality over time, As can be seen, all three criteria have very similar performances. A tabu object actually begins

to control the search at a relatively late stage, at the point when the algorithm reaches a local optimum (before that, it would not try to visit the same solution twice). It seems that at this point, which is not visible in Figure 2 (due to the large range of y-axis), TSO performs better.

Tabu tenure Not only the choice of the tabu object contributes to the performance of TS algorithms, but also the choice of the tabu tenure. In Figure 3, the seven ECDF curves of TS using the objective value as the tabu object separate into two groups depending on whether the tabu tenure is fixed or based on the problem scale. In the beginning, the two groups share similar curves, but in the end, the algorithms using the problem-scale based tabu tenure reach higher ECDF values.

However, the impact of tabu tenure strongly depends on the tabu object. In the experiments, we see that the best settings for the three different types of tabu objects are TSM$(0 + 20)$, TSO$(0 + 10)$, and TSS$(500 + 0)$.

Investigating the neighborhood The most interesting finding in this set of experiments is that the way our TS algorithms search the neighborhood has a huge impact on the performance. In Figure 1, TSF has much better performance than the TS algorithms that accept only the best solution in the neighborhood. Its curve rises the fastest in the beginning and reaches the highest ECDF in the end among all the TS algorithms. This proves that it can solve the most benchmark instances.

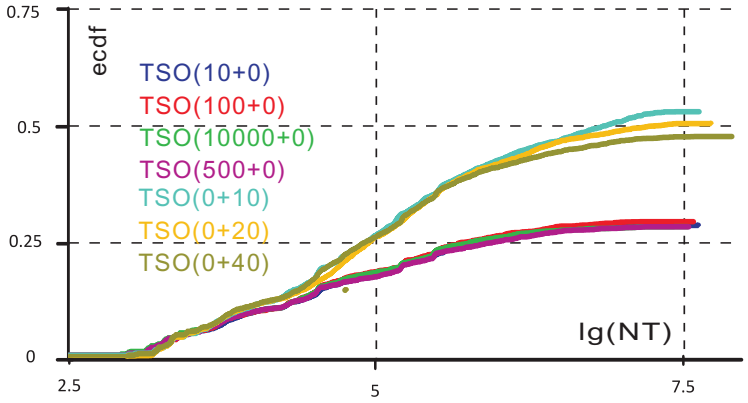
Comparison with LK and MNS In Figure 1, we see that LK always as higher ECDF values than TS and MNS in the end. MNS starts a little faster in the beginning but its ECDF in the end is lower than TS and LK. All in all, we find that the best TS setup is better than MNS and worse than LK. Consequently, the *TSP Suite* ranks the algorithms as follows: LK, TSF, TSO$(0 + 10)$, MNS, TSS$(500 + 0)$, and TSM$(0 + 20)$.

4.3 Hybrid algorithm performance

We also investigated our newly proposed hybridized versions of TS with EAs and PACO. Both of the hybrids were tested using two different settings for each of TSS, TSO and TSM, as well as four different settings for TSF. We compared them with the best known settings of the hybrid versions of LK and MNS, namely hPACO(3,10)LK and hPACO(3,10)MNS.

Two of the best hybrid setups for TS accepting only the best solution in the neighborhood in each iteration are hPACO$(3, 10)$TSO$(10 + 10)$ and hMA$(16 + 64)$TSO$(10 + 10)$. For TSF, the best settings are hPACO$(3, 10)$TSF(10+0) and hMA$(16 + 64)$TSF(10+0).

Interestingly, in our experiments, the best tabu tenure for pure TSF is 10000, but for hybrid TSF it is 10. A potential reason for this could be that operators



(a) Progress in F_b over NT for $64 \leq n < 128$.

Fig. 3: ECDF diagrams for different (log-scaled) runtime measures and goal errors.

of the EA, such as crossover and selection, already somewhat avoid convergence and a long tabu list would require more runtime for less additional benefits in a hybrid algorithm.

From Figure 1, we can see significant improvement of TS after being hybridized with EAs and PACO. The hybrids of TS outperform pure TS both in terms of speed and end results: the ECDF curves of the hybrids start slightly earlier, increase more rapidly, and finally reach at higher end points than those of the pure algorithms. The pure TSO is able to solve less than 20% of the benchmark instances, while hPACO(3, 10)TSO can solve almost 45% and hMA(2, 4)TSO 41%, making PACO the better global optimization method to hybridize TS with than an EA.

The same observations can be made with the hybrids of LK and MNS. We see that hPACO(3, 10)LK and hPACO(3, 10)MNS outperform the pure LK and MNS significantly. Besides that, hPACO(3, 10)LK and hPACO(3, 10)MNS perform better than hPACO(3, 10)TS.

When we set $F_t = 0.1$, the hybrid algorithms again outperform the pure ones and can reach ECDF values higher than 0.9. The PACO hybrids of TSO and TSF behave very similarly to the MAs. The hybrids of TSF even start earlier than hPACO(3, 10)LK. If we measure time with *FE*, the hybrids of TSO start the earliest among the investigated algorithms.

From Figure 2, we see that the hybrids algorithms always start at a lower F_b, mainly due to their heuristic initialization. For small-scale problems, both the hybrids and pure algorithms can decrease F_b towards 0 quickly, i.e., find the optima quickly. However, in large-scale problems (Figure 2b), the hybrids of

TSO start at a lower F_b and barely decrease it, while pure TSO gets a smaller F_b in the end.

The aggregated algorithm ranking provided by the *TSP Suite* when comparing all setups regarding ECDF, final results, expected runtime to the optimum, and progress according to different runtime measures is:

hPACO(3, 10)MNS, hPACO(3, 10)LK, algohPACO(3, 10)TSF(10+0),
hEA(16 + 64)TSF(10+0),hPACO(3, 10)TSO(10 + 10), LK,
hEA(16 + 64)TSO(10 + 10), TSF10000, MNS, TSO(0 + 10), TSS(500 + 0), and TSM(0+20).

5 Conclusions and Future Work

In this work, we have presented a large-scale experimental study investigating the performance of several TS approaches. We analyzed the impact of three different tabu criteria and introduced new and highly-efficient hybrid algorithms. Our experiments have led us to the following major conclusions:

1. The pure TS algorithm works well on small- and medium-scale TSP instances, but it cannot get very close to the global optimal for large-scale problems.
2. As a simple algorithm, TS performs worse than LK but better than MNS, while LK and MNS are both significantly more complicated.
3. The tabu object used in the tabu list can influence the performance.
4. Using a tabu object that forbids repeating objective values works better than forbidding candidate solutions, even though the relevant literature has focused largely on the latter.
5. Hybridization provides some considerable performance improvement.
6. Hybrid PACO works better than MAs (hybrid EAs), although the relevant literature has focused largely on the latter.

We will continue to investigate TS algorithms on the TSP. We next plan to implement additional tabu criteria mentioned in the literature such as [16]. In addition, we will try more strategies to improve the performance of TS.

Acknowledgments

We acknowledge support from the Fundamental Research Funds for the Central Universities, the National Natural Science Foundation of China under Grant 6115 0110488, Special Financial Grant 201104329 from the China Postdoctoral Science Foundation, the Chinese Academy of Sciences (CAS) Fellowship for Young International Scientists 2011Y1GB01, and the European Union 7th Framework Program under Grant 247619. The experiments reported in this paper were executed on the supercomputing system in the Supercomputing Center of University of Science and Technology of China.

References

1. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press (2007)
2. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization. Chichester, UK: Wiley Interscience (September 1985)
3. Gutin, G.Z., Punnen, A.P., eds.: The Traveling Salesman Problem and its Variations. Norwell, MA, USA: Kluwer Academic Publishers (2002)
4. Jiang, H., Sun, W., Ren, Z., Lai, X., Piao, Y.: Evolving hard and easy traveling salesman problem instances: A multi-objective approach. In: Simulated Evolution and Learning. Springer (2014) 216–227
5. Woeginger, G.J.: Exact algorithms for np-hard problems: A survey. In: Revised Papers of the 5th Intl. Works. on Combinatorial Optimization, Aussois, France, Berlin, Germany: Springer (March 5–9, 2001) 185–207
6. Weise, T.: Global Optimization Algorithms – Theory and Application. Germany: it-weise.de (self-published) (2009)
7. Bäck, T., Fogel, D.B., Michalewicz, Z., eds.: Handbook of Evolutionary Computation. New York, NY, USA: Oxford University Press (1997)
8. De Jong, K.A.: Evolutionary Computation: A Unified Approach. Cambridge, MA, USA: MIT Press (2006)
9. Dorigo, M.: Optimization, Learning and Natural Algorithms. PhD thesis, Milano, Italy: Dipartimento di Elettronica, Politecnico di Milano (1992)
10. Dorigo, M., Birattari, M., Stützle, T.: Ant colony optimization – artificial ants as a computational intelligence technique. IEEE Computational Intelligence Magazine **1**(4) (2006) 28–39
11. Gambardella, L.M., Dorigo, M.: Solving symmetric and asymmetric tsps by ant colonies. In: Proc. of IEEE Intl. Conf. on Evolutionary Computation, Nagoya, Japan (May 20–22, 1996) 622–627
12. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Operations Research **21**(2) (1973) 498–516
13. Weise, T., Chiong, R., Tang, K., Lässig, J., Tsutsui, S., Chen, W., Michalewicz, Z., Yao, X.: Benchmarking optimization algorithms: An open source framework for the traveling salesman problem. IEEE Computational Intelligence Magazine **9**(3) (August 2014) 40–52
14. Hansen, P., Mladenović, N., Moreno Pérez, J.A.: Variable neighbourhood search: Methods and applications. Annals of Operations Research **175**(1) (2010) 367–407
15. Guntsch, M., Middendorf, M.: Applying population based aco to dynamic optimization problems. In: Proc. of the 3rd Intl. Works. on Ant Colony Optimization, Brussels,(2002) 111–122
16. Glover, F.W., Taillard, É.D., de Werra, D.: A user's guide to tabu search. Annals of Operations Research **41**(1) (1993) 3–28
17. Misevičius, A.: Using iterated tabu search for the traveling salesman problem. Information technology and control **32**(3) (2015)
18. Osaba, E., Diaz, F.: Comparison of a memetic algorithm and a tabu search algorithm for the traveling salesman problem. In: 2012 Federated Conf. on Computer Science and Information Systems. (2012) 131–136
19. Reinelt, G.: Tsplib 95. Technical report, Heidelberg, Germany: Universität Heidelberg, Institut für Mathematik (1995)

20. Basu, S.: Tabu search implementation on traveling salesman problem and its variations: A literature survey. American Jour. of Operations Res. **2**(2) (2012) 163–173
21. Boddy, M.S., Dean, T.L.: Solving time-dependent planning problems. Technical Report CS-89-03, Providence, RI, USA: Brown University (1989)
22. Jiang, Y., Weise, T., Lässig, J., Chiong, R., Athauda, R.: 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 Series on Computational Intelligence, Orlando, FL, USA (December 9–12, 2014)
23. Hansen, N., Auger, A., Finck, S., Ros, R.: Real-parameter black-box optimization benchmarking: Experimental setup. Technical report, Orsay, France: Université Paris Sud, INRIA, Équipe TAO (2012)
24. Tompkins, D.A.D., Hoos, H.H.: Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat. In: 7th Intl. Conf. on Theory and Applications of Satisfiability Testing, Berlin, Springer (2004) 306–320
25. Hoos, H.H., Stützle, T.: Stochastic Local Search: Foundations and Applications. San Francisco, CA, USA: Morgan Kaufmann (2005)
26. Hoos, H.H., Stützle, T.: Evaluating las vegas algorithms – pitfalls and remedies. In: Proc. of the 14th Conf. on Uncertainty in AI, Madison, WI, USA, San Francisco, CA, USA: Morgan Kaufmann (July 24–26, 1998) 238–245
27. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the stsp. In: The Traveling Salesman Problem and its Variations. (2002) 369–443
28. Johnson, D.S., Gutin, G.Z., McGeoch, L.A., Yeo, A., Zhang, W., Zverovitch, A.: Experimental analysis of heuristics for the atsp. In: The Traveling Salesman Problem and its Variations. (2002) 445–487
29. Wu, Y., Weise, T., Chiong, R.: Local search for the traveling salesman problem: A comparative study. In: Proc. of 14th IEEE Conf. on Cognitive Informatics & Cognitive Computing. (July 68, 2015) 213–220
30. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Berlin, Germany: Springer (1996)
31. Fogel, D.B.: An evolutionary approach to the traveling salesman problem. Biological Cybernetics **60**(2) (1988) 139–144
32. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. University of Michigan Press (1975)
33. Glover, F.W.: Future paths for integer programming and links to artificial intelligence. Computers & Operations Research **13**(5) (1986) 533–549
34. Kirkpatrick, S., Gelatt, Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science Magazine **220**(4598) (1983) 671–680
35. Wang, D., Wang, J., Wang, H., Zhang, R., Guo, Z.: Intelligent optimization methods (2007)
36. Helsgaun, K.: An effective implementation of the lin-kernighan traveling salesman heuristic. Technical report, Denmark: Roskilde University (1998)
37. Applegate, D.L., Cook, W.J., Rohe, A.: Chained lin-kernighan for large traveling salesman problems. INFORMS Journal on Computing 15(1) (2003) 82–92
38. Helsgaun, K.: General k-opt submoves for the lin-kernighan tsp heuristic. Mathematical Programming Computation 1(2-3) (2009) 119–163
39. Whitley, L.D., Starkweather, T., Fuquay, D.: Scheduling problems and traveling salesman: The genetic edge recombination operator. In: Proc. of the 3rd Intl. Conf. on Genetic Algorithms, (1989) 133–140

Dan Xu, Thomas Weise, Yuezhong Wu, Jörg Lässig, and Raymond Chiong.
An Investigation of Hybrid Tabu Search for the Traveling Salesman Problem.
In Proceedings of the 10th International Conference on Bio-Inspired Computing – Theories and Applications (BIC-TA'15), September 25 – 28, 2015. Hefei, Anhui, China, volume 562 of Communications in Computer and Information Science. Berlin/Heidelberg: Springer-Verlag, pages 523 – 537, ISBN 978-3-662-49013-6. doi:10.1007/978-3-662-49014-3_47