Comparing a Hybrid Branch and Bound Algorithm with Evolutionary Computation Methods, Local Search and their Hybrids on the TSP

Yan Jiang*, Thomas Weise*†, Jörg Lässig‡, Raymond Chiong§, and Rukshan Athauda§
*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. Emails: ljjy23@mail.ustc.edu.cn, tweise@ustc.edu.cn
†Corresponding Author.
‡Department of Computer Science, University of Applied Sciences Zittau/Görlitz;
D-02826 Görlitz, Germany. Email: jlaessig@hszg.de
§Faculty of Science and Information Technology, The University of Newcastle;
Callaghan, NSW 2308, Australia. Emails: Raymond.Chiong@newcastle.edu.au, Rukshan.Athauda@newcastle.edu.au

***Abstract*—Benchmarking is one of the most important ways to investigate the performance of metaheuristic optimization algorithms. Yet, most experimental algorithm evaluations in the literature limit themselves to simple statistics for comparing end results. Furthermore, comparisons between algorithms from different "families" are rare. In this study, we use the *TSP Suite* – an open source software framework – to investigate the performance of the Branch and Bound (BB) algorithm for the Traveling Salesman Problem (TSP). We compare this BB algorithm to an Evolutionary Algorithm (EA), an Ant Colony Optimization (ACO) approach, as well as three different Local Search (LS) algorithms. Our comparisons are based on a variety of different performance measures and statistics computed over the entire optimization process. The experimental results show that the BB algorithm performs well on very small TSP instances, but is not a good choice for any medium to large-scale problem instances. Subsequently, we investigate whether hybridizing BB with LS would give rise to similar positive results like the hybrid versions of EA and ACO have. This turns out to be true – the "Memetic" BB algorithms are able to improve the performance of pure BB algorithms significantly. It is worth pointing out that, while the results presented in this paper are consistent with previous findings in the literature, our results have been obtained through a much more comprehensive and solid experimental procedure.**

This is a preview version of paper [1] (see page 9 for the reference). It is posted here for your personal use and not for redistribution. The final publication and definite version is available from IEEE (who hold the copyright) at http://www.ieee.org/. See also http://dx.doi.org/10.1109/CIPLS.2014.7007174

I. Introduction

The Traveling Salesman Problem (TSP) [2–4] is the most well-known combinatorial optimization problem. It can be described as follows: Given are n cities, indexed from 1 to n, and the distances $D_{i,j} \in \mathbb{N}$ (with $i, j \in 1, 2, \ldots n$) between them. A salesperson aims to visit each city exactly once and then return back to his original location. In which order should this salesperson visit the cities to minimize the total travel distance? A candidate solution to a TSP is a tour $t = (t_1, t_2, \ldots, t_n)$, and a permutation of the cities to visit. The objective function f, subject to minimization, computes the total round trip distance $f(t) = D_{t_n,t_1} + \sum_{i=1}^{n-1} D_{t_i,t_{i+1}}$ of such a tour.

This optimization version of the TSP is NP-hard [4], and the worst-case runtime complexity of any existing exact TSP solver is exponential [5]. In order to obtain close to optimal solutions within feasible time, various approaches have been proposed, ranging from metaheuristics such as Evolutionary Algorithms (EAs) [6], Ant Colony Optimization (ACO) [7], and Estimation of Distribution Algorithms (EDAs) [8], to Local Search (LS) [9], Branch and Bound (BB) [10] and cutting plane algorithms [11]. The problem is well-known, easy-to-understand, and standard benchmarks with known solutions (like the *TSPLib* [12]) are available. This makes the TSP an ideal option for investigating and comparing the performance of new algorithms. To date, however, not many comparisons of members of different algorithm families have been conducted (notable exceptions are [13, 14]). Moreover, the experimental approaches and result comparisons in the literature are often limited to simple key statistics about their final results, ignoring the progress of algorithms over time.

In our recent work [15], a more rigorous experimental procedure has been introduced. Through an open source software framework, the *TSP Suite*, we have been able to conduct comprehensive experimental analysis and show that Evolutionary Computation (EC) methods such as EAs and EDAs do not perform well on the TSP when compared to LS algorithms. Yet, we found that hybridization of EC methods with LS algorithms can lead to greatly enhanced performances. In this paper, we use the same software framework to answer two research questions: (1) *How does the BB algorithm from [10], which was created five decades ago, from a time where 40-city problems were considered "large-scale", compare to contemporary metaheuristics?* and (2) *Would hybridizing BB with LS*

have similar positive effects like those observed through the EC methods?".

The contributions of this paper can be summarized as follows:

1) An in-depth performance analysis of the BB algorithm *over time* on 83 (smaller instances) of the 110 symmetric *TSPLib* benchmark instances, according to several different time measures such as function evaluations (FEs), normalized CPU times and different performance statistics.
2) A detailed comparison of the BB algorithm to modern metaheuristics such as state-of-the-art EAs, ACO, and LS algorithms.
3) The introduction of new hybrid forms of the investigated BB algorithm based on the same hybridization scheme used in [15].
4) A detailed comparison of the new hybrid BB algorithms to hybrid variants of the above mentioned EC approaches.
5) The implementation of all the tested algorithms will be provided online at `http://www.logisticPlanning.org/tsp/` as part of the open source framework *TSP Suite*.
6) The measured performance data will be provided online too, contributing to probably the largest collection of benchmark data on TSP solvers (with log files of already more than 20GB in size, obtained from about 200 algorithm setups).

The remainder of this paper is organized as follows. In the next section, we discuss related work on automated experimentation (Section II-A) and on solving TSPs (Section II-B) respectively. We then describe the BB algorithm as well as its new hybrid variants in Section III. The conducted experiments are discussed in Section IV. Section V ends the paper with conclusions and plans for future work.

II. Related Work

A. Related Work on Experimentation

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, the experimentation approaches adopted in most of the previous studies have relied mainly on the most basic statistics, some of which are even flawed. The reason is that proper experimentation itself is actually a cumbersome, time-demanding and complex process.

The *COmparing Continuous Optimisers* (COCO) [16] system for numerical optimization, used in the Black-Box Optimization Benchmarking (BBOB) workshops, is one of the first 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 [17] is an experimental framework for satisfiability problems. It focuses on a specific family of algorithms: the stochastic local search [18]. In COCO, the objective function will automatically gather log data before returning its result to the algorithm. In UBCSAT, this is 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* [15] takes the idea one step further. First, it provides software development support such as unit testing. Second, the *TSP Suite* will take care of parallelization or distribution of workload on a multi-processor system or cluster. It does not require any additional support or third-party software and the experimenter can implement their algorithm in a normal, non-parallelized way. Third, like in COCO, an algorithm performance report can be created automatically. The difference 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) [16] curves over goal objective values or problem scales and automated comparisons of empirical cumulative distribution functions (ECDFs) [16, 17, 19]. Each of these statistics results in algorithm rankings, which are later aggregated into a global ranking list. The global ranking will provide some insights on the general performance of a TSP solver.

To the best of our knowledge, the *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 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 machine and problem instance specific performance factor, thus rendering time results (somewhat) machine independent. 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 rankings created therefore represent a more balanced and fair perspective on an algorithm's performance.

B. Related Work on the TSP

The first BB approach for solving the TSP was published by Little et al. [10] in 1963. This algorithm is the basis of

our study. It is already 50 years old, and many improvements have been made since then. For example, the efficient BB algorithms designed by Zhang [20, 21] have been able to provide good solutions. A myriad of other ideas have also been tested. More details of the BB algorithm will be provided in Section III. In this section, we focus on related work using algorithms to which we would like to compare the BB algorithm to, which include LS, EC and hybrid algorithms.

1) Solving TSPs with LS: LS algorithms maintain a single solution and try to improve it iteratively by investigating its "neighborhood", i.e., the set of solutions that can be reached by applying a single modification to it. Examples of operators that can do the job include those that reverse a sub-sequence of a tour, rotate a sub-sequence one step to the left or right, or a swap move simply exchanging two nodes [15].

One of the most successful general LS approaches is Variable Neighborhood Search (VNS) [22]. VNS investigates a set of neighborhoods by searching the first neighborhood until no further improvement is possible, then trying the second neighborhood, the third, and so on. As soon as an improvement is found, it reverts back to the first neighborhood. In [15], a Random Neighborhood Search (RNS) algorithm that randomly picks a different neighborhood in each step as well as a Multi-Neighborhood Search (MNS) algorithm that scans all neighborhoods of a given solution and collects multiple improving moves at once were tested and found to have produced good performances. In this paper, we apply these three algorithms (i.e., VNS, RNS and MNS) with restarts, exactly as defined in [15].

2) Solving TSPs with EC: EAs are the most well-known EC approaches [23]. They manage a set (population) of solutions by iteratively selecting its best members and creating new solutions by mutation and crossover operations. Mutation means to randomly generate a solution out of the neighborhood of a parent solution. Crossover means to combine two solutions. Several different mutation and crossover operators of EAs for the TSP have been proposed [24]. In this paper, we investigate an EA that uses the same four neighborhoods introduced in [15] for mutation. We apply Edge Crossover [6], which tries to create a new solution by using edges occurring in either of its two parents and is considered to have performed well [24].

The Population-based ACO (PACO) [25], another member of the EC family, is a variant of the ACO algorithm that maintains a set (population) of k solutions. The edges present in those solutions define the pheromones. In each iteration, m solutions are generated as in standard ACO and the best of them replaces the "oldest" solution in the population. PACO is known to be amongst the best ACO approaches [25, 26] and was the best tested pure EC method in [15] for the TSP.

3) Solving TSPs with Hybrid Algorithms: EC methods can be hybridized with LS algorithms for improving their performances: Memetic Algorithms (MAs) [27] are EAs where a LS algorithm is applied to every new solution created. MAs are known to have performed well on the TSP [15, 28]. Other metaheuristics (like PACO) can be hybridized as well. In [15], it was shown that LS outperforms pure EC methods, but hybrid EC-LS algorithms are the best. While this is common knowledge in the field of EC, only limited attempts, such as [29], have been made to hybridize BB with LS. To the best of our knowledge, the new hybrid BB introduced in this paper is the first such approach for the TSP.

III. Methods

A. BB in General

A BB approach for the TSP initially considers all possible tours as potential solutions, i.e., a set $\mathbf{T}_A$ of size $(n-1)!$, in the asymmetric case. A tour can be created either randomly or by using a heuristic. The best tour t^* known to the algorithm is used as the starting solution. In the *branch* step, according to some criterion φ, a set $\mathbf{T}$ (initially $\mathbf{T}_A$) of solutions is divided into two subsets $\mathbf{T}_1$ and $\mathbf{T}_2$. A lower *bound* ℓ of the objective function for each of these sets is calculated. Clearly, a set $\mathbf{T}_i$ can only contain a better solution than t^* if $\ell(\mathbf{T}_i) < f(t^*)$. Only those sets that may potentially contain better solutions are considered in the further course of the algorithm.

B. BB by Little et al.

The BB algorithm by Little et al. [10] was designed for solving asymmetric TSPs. In their algorithm, each set $\mathbf{T}$ of solutions is defined by a corresponding set E of *directed* edges that are allowed as part of the tours $t \in \mathbf{T}$, i.e., $E = \subseteq \{(i,j) : i,j \in 1 \ldots n \wedge i \neq j\}$.

Branching is done by choosing an edge e^*, which must be included in the solutions of one subset and excluded from those in the other subset. The branching criterion φ maps each edge to a natural number. For a given edge (i,j), $\varphi(i,j)$ equals to the sum of the distances of the shortest allowed edge from node i and the shortest allowed edge to node j. In the branch step, φ is evaluated for each edge $e \in E$ and the edge $e^* = (k,l)$ with the maximal φ-value is selected. The current set of allowed candidate solutions $\mathbf{T}$ is then divided into two subsets, $\mathbf{T}_1$ and $\mathbf{T}_2$. All tours in $\mathbf{T}_1$ must contain e^* while those in $\mathbf{T}_2$ must not. When branching to $\mathbf{T}_1$, a new edge set E_1 is created as $E_1 = E \setminus \{(i,l) : i \neq k\} \setminus \{(k,j) : j \neq l\}$, i.e., by removing all edges either starting in city k or ending in city l. If branching to $\mathbf{T}_2$, the corresponding set of allowed edges E_2 is created as $E_2 = E \setminus \{e^*\}$. The subset with the best lower bound ℓ is investigated first, while the other subset enters a queue. How ℓ is designed can be found in [10].

This process is recursively applied, until it arrives at sets containing only a single solution. If such a solution is better than t^*, t^* is updated. Then, the queue of tour sets awaiting investigation is pruned by removing all sets with a lower bound ℓ greater than or equal to $f(t^*)$. From the remaining candidates, the one with the smallest corresponding lower bound is extracted and used for the next branching step. We will refer to this basic algorithm as `BB` from here on.

Given enough runtime, `BB` will always return the globally optimal (shortest possible) tour. However, in a worst case scenario, the branches may form a full binary tree with depth of at least n. This leads to a worst-case time and memory requirement in $\Omega(2^n)$.

Like all common metaheuristics, BB algorithms are anytime algorithms [30], i.e., algorithms that can provide an approximate solution at any point during their course [21]. The quality of the approximation should improve over runtime. The difference between BB and other EC and LS methods is that BB can guarantee in finding the optimum solution eventually (unless terminated earlier). In our study, we thus do not consider only the end result (in this case, success or premature termination), but the progress an algorithm makes over runtime.

C. New Hybrid BB

In Section II-B1, we introduced three LS algorithms: VNS, RNS, and MNS. The hybrid PACO and the hybrid EA (i.e., MA) setups in [15] refine every constructed candidate solution by applying one of these three. This straightforward scheme can also be used to hybridize BB: whenever the algorithm has branched to a set $\mathbf{T}$ containing only $|\mathbf{T}| = 1$ solution, this solution is passed to the selected LS algorithm, which proceeds until it arrives at a (different) local optimum. The selected LS algorithm is also applied to the initial solution in order to provide a tight upper bound. We create three hybrid variants of BB by combining it with VNS, RNS, and MNS, which we abbreviate as BBVNS, BBRNS, and BBMNS respectively.

These hybrids retain the exact property of BB, but they have a higher worst-case time complexity, since each "leaf" of the search tree is additionally processed by LS where *each search step* has (at least) quadratic complexity, leading to a worst-case time complexity of $\Omega(n^2 \times 2^n)$. The actual worst-case complexity could be higher than this lower bound, since the LS algorithm would usually perform several more steps. A detailed theoretical analysis of the algorithm's complexity is out of the scope of this work. A potential benefit of the LS is that it may be able to provide tighter upper bounds $f(t^*)$, which may allow the algorithm to skip more branches earlier, and thus reduce the average runtime. In this paper, we aim to investigate whether this is true. It is worth noting that the aforementioned complexities would come into play only if the algorithm is granted enough runtime to complete its search steps. In our experiments, we apply it as an anytime algorithm with a limited computational budget.

An alternative way to decrease the (initial) upper bound without tangibly affecting the worst-case time complexity is to not obtain it from a random solution but from a solution created by a simple constructive heuristic. We tested this approach with the Double Minimum Spanning Tree (name prefix M) and Savings heuristics (name prefix S) [2]. These heuristics have the time complexity of $\mathcal{O}(n^2)$, which is negligible compared to $\mathcal{O}(2^n)$. We combine either of these heuristics with any of the previous setups and signify this with the corresponding name prefix, e.g., SBB is BB initialized with Savings, MBBMNS is a MNS hybrid of the BB algorithm obtaining its initial upper bound from a solution created by the Double Minimum Spanning Tree heuristic, and so on. All in all, this leads to $4 \cdot 3 = 12$ BB setups.

IV. Experiments and Results

We conducted experiments on the symmetric *TSPLib* benchmark cases for the 12 BB setups discussed in the previous section using the *TSP Suite* introduced before. In our experiments, 30 runs were performed for each benchmark case. Initial tests showed that the memory requirement of the algorithm quickly increases with the problem scale, but seemingly not exponentially as it would be the worst case scenario. We could only obtain results for the 83 smaller problem instances up to 3795 cities and therefore only consider these in the evaluation (which are still about 100 times more than the "largest-scale" original experiment [10]).

The comparison data for the pure LS and EC methods as well as their hybrids are taken from [15], in which detailed descriptions of the corresponding experiments can be found. In particular, we compared the algorithm performance of BB with the two best setups of pure EA and ACO found in that paper. These are EA128+256e, an $(128 + 256)$ EA with truncation selection and Edge Crossover, and PACO3,25, a population-based ACO with population size 3 and sample size 25. From these two configurations, the best variants having initial populations seeded with solutions obtained from constructive heuristics, referred to as hEA128+256e and hPACO3,25 respectively, are derived.

Finally, we also compared BB with the two best seeded EA and ACO setups hybridized with MNS and RNS according to [15]. For the EA, these are hMA16+64mnse, a $(16 + 64)$ EA with Edge Crossover hybridized with MNS, and hMA2+8rnss, a $(2+8)$ EA with Savings Crossover [15] hybridized with RNS. For PACO, the best hybrid setups with different LS algorithms are hPACO3,10mns, with population size 3, sample size 10, and MNS, as well as hPACO3,10rns, which uses RNS instead. Furthermore, we compared the performance of the BB algorithms with the best setups of the pure LS algorithms (MNS, RNS1, VNS1) and their best seeded setups (MNSm, RNSbm, and VNSbm, all of which were seeded with the Double Minimum Spanning Tree heuristic).

Different benchmark cases have different globally optimal tour lengths $f^\star$. To ease the comparison and understanding of results, we considered the algorithm's progress in terms of the relative error $F = {}^{(f(t^*)-f^\star)}/_{f^\star}$ instead of the best discovered objective value $f(t^*)$. If a run reaches $F = 0$, it has discovered the global optimum. $F = 1$ means the best currently known tour t^* is twice as long as the optimal one.

Similarly, we introduced relative "goal errors", denoted as F_t, to compute several statistics: the ERT [16], for instance, estimates the runtime needed to reach a solution with $F \leq F_t$, while the ECDF [16, 17, 19] estimates the probability to which a run obtains a solution with $F \leq F_t$ within a given time budget.

A. Pure Algorithm Performance

Let us first explore the performance of the pure BB algorithm and compare it with the pure EC algorithms initialized with different heuristics. We found that BB and its heuristically-initialized derivatives receive the worst ranking,

aggregated from a variety of statistics. The heuristically initialized SBB and MBB outperform BB.

We illustrate two ECDFs of these setups in Figure 1. The ECDF of BB in Figure 1a, based on the normalized CPU runtime measure NT, increases slowly and never reaches 0.5 for $F_t = 0.1$. In other words, even a 10% margin cannot be reached in more than half of the runs under the given computational budget. The ECDF curves of BB, MBB and SBB all increase slowly and approach each other as time increases. If we decrease F_t (not illustrated), the ECDF value decreases rapidly as well. If we measure time in terms of FEs (Figure 1c) instead of (normalized) CPU seconds, we see that the tested BB algorithms can only create relatively few solutions within the granted computational budget and thus, the ECDF stops increasing early.

These observations are based on all the instances with the size n up to 3795. The findings are not surprising, as problem scales above 40 were more or less out of reach by BB when it was designed. In terms of normalized CPU time (NT), BB can find solutions of instances with $n = 32$ and below quite well, and the ECDF increases quickly. The ERT in terms of DEs and NT is low in this problem class. When the problem scale n becomes bigger, BB finds fewer solutions.

The *TSP Suite* provides separate diagrams for the TSP instances grouped by the number of cities n. In other words, we can analyze the relationship between the performance of BB and the problem scale. In the graphs presented in this paper, lines with the same color represent different algorithms belonging to the same category (e.g., MNS and VNS belong to the LS group and thus they are illustrated with the same color).

From Figures 2a and 2b, we see that the BB algorithm can find the optimal solutions for instances with n between 16 and 31 quite rapidly, and for $32 \leq n < 63$, although not as good as LS algorithms, it still can find solutions with $F_t \geq 0.05$. As the problem scale increases, however, its performance decreases.

While the NT is related to the actual consumed runtime, we can also measure the progress of an algorithm relative to the number of candidate solutions it has constructed, i.e., in FEs. In Figures 2c and 2d, we present the ECDF in terms of FEs, based on the same groups of small-scale instances as Figures 2a and 2b illustrated. The BB algorithm again shows good performance for $n \leq 31$. We can see from Figure 2c that it can reach the optima with few FEs and outperform the other algorithms except MBB, SBB, SBBMNS, SBBVNS and SBBRNS. As the instance scale gets larger, the BB algorithm can only perform a few FEs until the maximum runtime is exhausted, and it manages to find only solutions with a longer tour length.

In Figure 3, we plot the ERT in terms of the normalized runtime NT (y-axis) to achieve given goal errors F_t (x-axis). The trends for the ERT are similar for DEs, FEs, and NT. BB and its heuristically-initialized derivatives take longer to find good solutions compared to the other algorithms. This becomes, again, even more obvious when we look at larger instances. For problems with $n < 256$, BB can reach $F \leq 0.15$. The smaller n is, the smaller the F_t value BB can reach within the given computational budget.

We also tested two initialized variants of BB: MBB and SBB. Initialization does improve the performance of BB a little, at least during the early stage of the search, by giving the algorithm a better initial upper bound and initial t^*. It does not, however, increase the speed of improving the existing solutions. The changes of gradients in the performance curves of BB, MBB, and SBB are almost identical.

From Figure 1, we can clearly see that the BB algorithms are initially better than EA128+256e, but EA128+256e later outperforms them. The heuristically-initialized EA, hEA128+256e, is better than the BB group from the start. Moving on to Figure 2, we can see that pure EC methods behave differently when comparing the improvement of F over NT. For instances with $8 \leq n < 32$, the BB algorithms outperform the PACO algorithms all the time and also outperform EA128+256e most of the time, but are outcompeted by hEA128+256e completely. For instances with $32 \leq n < 64$, the BB algorithms perform the worst. Again, the larger the scale n is, the worse the set of tested BB methods performs in comparison with the tested EC methods.

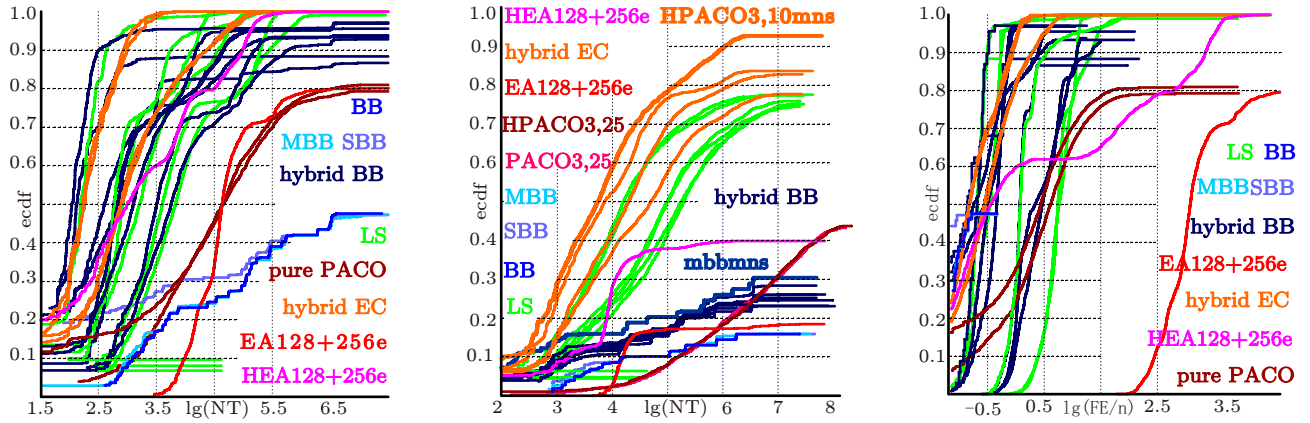
The tested LS setups MNS, RNS1 and VNS1 are much better than the BB algorithms and the pure EC methods.

B. Hybrid Algorithm Performance

Next, we investigated the BB variants hybridized with MNS (called BBMNS), VNS (BBVNS), and RNS (BBRNS). Each of these three hybrids was tested with and without heuristic initialization.

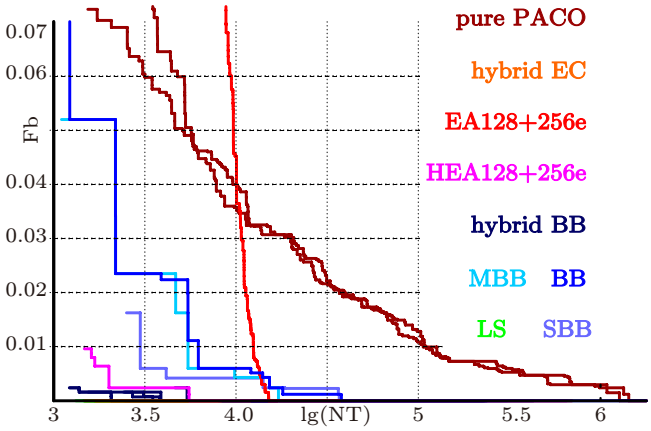
The ECDF over all tested benchmark instances has a similar shape for both DEs and NT, and we plot it for $F_t = 0.01$ in Figure 1b. From the figure, we can visually classify all the curves into two main categories. The first category contains the BB variants, of which MBBMNS (highlighted in the figure) performs the best. The other category contains the hybrid EC and LS algorithms, of which hPACO3,10mns (also highlighted) is the best algorithm. The hybrid EC methods perform significantly better than the pure EC and hybrid BB algorithms. They have achieved better ECDF values eventually and their ECDF also increases more rapidly. The ECDF value of the tested hybrid BB algorithms fails to reach 0.35 for $F_t = 0.01$ within the granted computational budget. The ECDF of the hybrid EC and LS algorithms, however, eventually exceeds 0.7. This suggests that there is a huge performance gap between the hybrid BB variants and hybrid EC algorithms or LS algorithms. The hybrid BB variants are better than the pure EA (EA128+256e) but worse than the heuristically-initialized EA (hEA128+256e). They were better than the pure and heuristically initialized variants of PACO for most of the given time, but are eventually outperformed by them.

The ERT figures in terms of DEs and NT for a given F_t also share similar shapes with those in Figure 3. Here, we observe three visually distinguishable algorithm classes: the pure and initialized BB algorithms belong to the worst class, their hybrid versions and pure EC algorithms are in a better class,

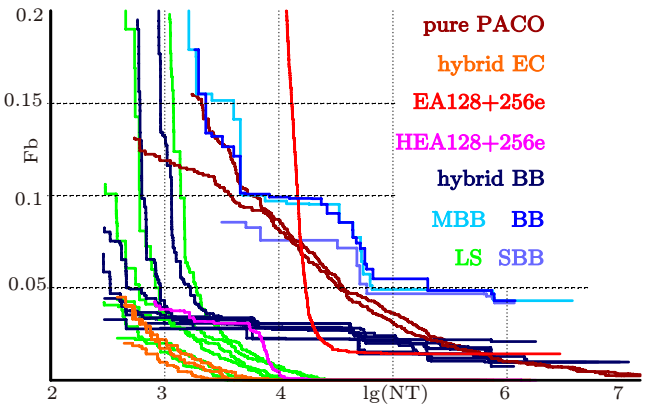


(a) ECDF for $F_t = 0.1$ and NT (b) ECDF for $F_t = 0.01$ and NT (c) ECDF for $F_t = 0.1$ and FEs

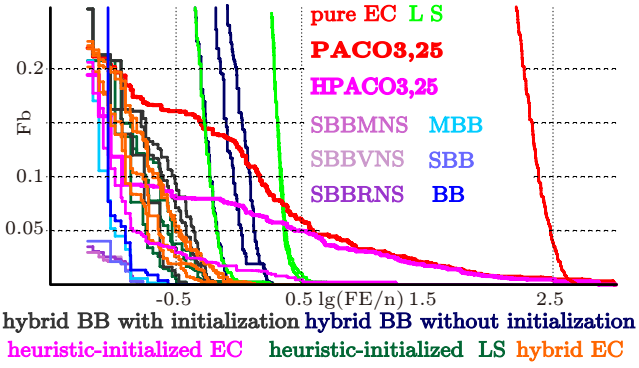
Fig. 1: We plot the ECDFs for goal error $F_t = 0.1$, i.e., the fraction of runs that have discovered a tour that is not more than 10% longer than the global optimum and $F_t = 0.01$ over the two time measures NT and FE (scaled by n). The x-axes are logarithmically scaled. In these graphs (in fact all the graphs presented in this paper), lines with the same color denote algorithm variants within the same category.



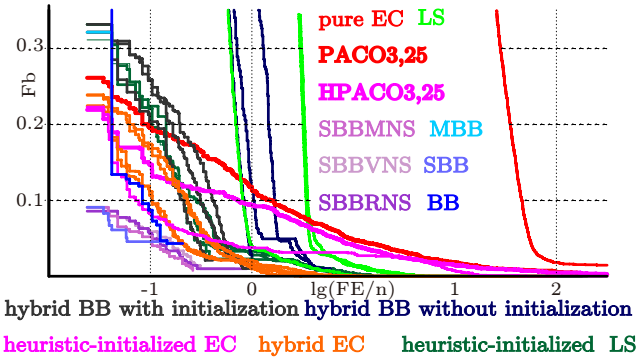
(a) Progress over NT for all instances with $16 \leq n < 32$



(b) Progress over NT for all instances with $32 \leq n < 64$



(c) Progress over FEs/n for all instances with $16 \leq n < 32$



(d) Progress over FEs/n for all instances with $32 \leq n < 64$

Fig. 2: Progress diagrams showing how the relative error F, i.e., the fraction the best known tour t^* is longer than the optimal one, improves over runtime. The diagrams are aggregated over smaller-scale subsets of the investigated TSP instances. Their x-axes hold the logarithmically scaled runtimes and the y-axes are F.

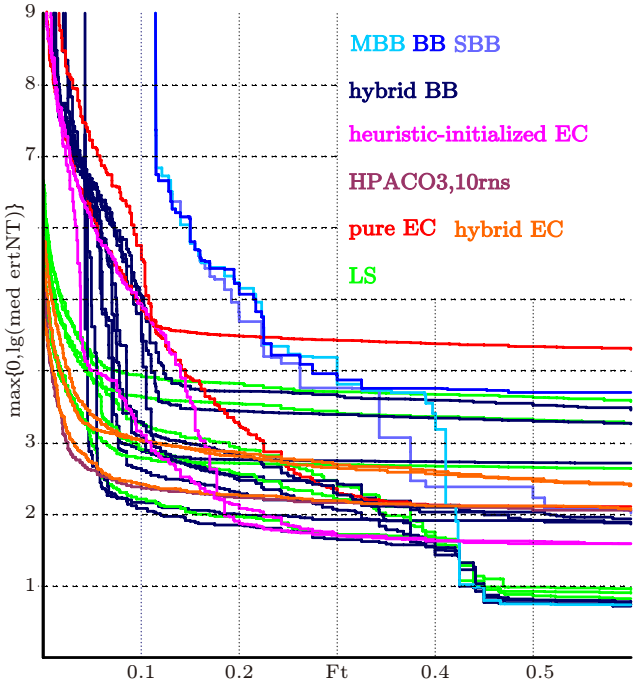


Fig. 3: (log-scaled) ERT in terms of NT over F_t for all tested instances.

and the hybrid EC algorithms as well as the LS algorithms end up in the best category. The hybrid BB algorithm group takes longer to reach a given goal error F_t. An interesting observation is that the last two classes behave similarly when F_t is larger than 0.1, but as F_t enters the interval between 0.1 and 0.05 (representing better solutions), their performances start to diverge. The speed at which the hybrid BB variants find better solutions then declines dramatically. The hybrid EC algorithms and the LS algorithms also suffer a great decline in the speed of finding better solutions at this stage, but not nearly as much as the hybrid BB group.

From Figures 2a and 2b, we see that all the tested algorithms can acquire good results on the smallest instances with $8 \leq n < 63$, although both the EC and LS methods have obtained slightly better results within the given computational budget.

A detailed analysis of the figures shows that all the tested algorithms are quick to find long tours, i.e., to reach higher goal errors F_t. However, the hybrid BB algorithms take much longer than the hybrid EC algorithms (or LS algorithms) to locate good tours.

One reason for this observation could be that, although we have introduced hybridization into BB the same way in which it is introduced in EAs and PACO, its "utilization" is very different. An MA (hybrid EA) uses the results of the LS as input for the search operations in the next generation and ACO uses it to update its pheromone matrix, i.e., the way new solutions are generated. The hybrid BB only uses it to update the upper bound $f(t^*)$ for the optimum and to update its variable t^* holding the best known solution. It does not use hybridization to generate new solutions. In other words, it cannot reap as much benefit from the LS as the EC methods do.

If we look at Figure 2a as well as figures on other scales, we notice that the performance of the compared algorithms differs most significantly when the instance size n is between 32 and 512. Within this range, hybrid BB algorithms with heuristic initialization can find good solutions with an F_b around 0.05 (0.1 if n is bigger than 256) in an early stage, although the results are still not better than the heuristically-initialized EC methods. The entire BB algorithm group fails to improve its solutions continuously compared to hybrid EC algorithms and LS algorithms. All in all, for small instance sizes, all the algorithms have good performance. For medium size instances, the hybrid EC algorithms and LS offer better performance. For relatively large problems, however, none of the tested algorithms performs well.

We found that initialization does not alter the final results much, but will it influence the process of finding the solutions? Figures 2c and 2d along with the diagrams on different instance scales provide a positive answer to this question. To reach some fixed goal error F_t, for algorithms from the hybrid BB family and the LS algorithms, the initialized variants tend to have good starting solutions and take less FEs. The exceptions are PACO3,25 and hPACO3,25, for which initialization neither provides any obvious enhancement (compared to other algorithms) on the starting solutions nor does it reduce the consumed FEs significantly. The reason may be that the way PACO creates a solution when its pheromone matrix is empty (the initial state) is similar to how a constructive heuristic works. In general, when the instance size n increases, the initial solutions generated by heuristic initialization become worse but are still better than their random counterparts.

C. Performance Classes

Based on all the above analyses, we can classify the tested algorithms according to their behaviors into four groups, which are, in descending order according to their performance: hybrid EC algorithms, LS algorithms, the hybrid BB, and pure BB. The globally aggregated ranking, over all the performance measures, computed by the *TSP Suite* is: hPACO3,10mns, hMA16,64mnse, hPACO3,10rns, MNSm, MNS, hMA2+8rnss, RNSbm, RNS1, VNSbm, VNS1, hEA128+256e, MBBMNS, SBBRNS, hPACO3,25, MBBRNS, PACO3,25, MBBVNS, SBBMNS, RBBMNS, SBBVNS, RBBRNS, RBBVNS, EA128+256e, SBB, MBB and BB.

V. Conclusions and Future Work

Our experiments have led us to four major conclusions:

1. The traditional BB algorithm still works well on small-scale TSP instances (for which it was designed) compared to current methods, but it does not perform well on medium or larger scale instances. This is true for its hybrid variants too.

2. Hybridization is able to improve the performance of BB considerably.

3. We are able to classify TSP solvers into four groups according to their runtime behavior and confirm that hybrid algorithms are better than their corresponding pure algorithms.

4. Heuristically-initialized algorithms normally have good starting solutions and take less FEs to reach the same F_t and the hybrid EC algorithms are better than our new hybrid BB algorithms.

Future work will involve implementing the modern variants of BB by Zhang [20, 21] and hybridizing them with LS. We will also investigate the use of Lin-Kernighan heuristic [31], one of the best known metaheuristic strategies [31] for the TSP, for benchmarking purposes.

Acknowledgments: We acknowledge support from the National Natural Science Foundation of China under Grant 61150110488, Special Financial Grant 201104329 from the China Postdoctoral Science Foundation, the Chinese Academy of Sciences (CAS) Fellowship for Young International Scientists 2011Y1GB01, the European Union 7th Framework Program under Grant 247619, and the University of Newcastle's Faculty of Science and IT Strategic Initiatives Research Fund under Grant 1031415. 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] Y. Jiang, T. Weise, J. Lässig, R. Chiong, and R. Athauda, "Comparing a hybrid branch and bound algorithm with evolutionary computation methods, local search and their hybrids on the tsp," in *Proceedings of the IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS'14), Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI'14)*. Orlando, FL, USA: Caribe Royale All-Suite Hotel and Convention Center: Los Alamitos, CA, USA: IEEE Computer Society Press, December 9–12, 2014.

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook, *The Traveling Salesman Problem: A Computational Study*, ser. Princeton Series in Applied Mathematics. Princeton, NJ, USA: Princeton University Press, February 2007. [Online]. Available: http://books.google.de/books?id=nmF4rVNJMVsC

[3] E. L. G. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, ser. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Chichester, West Sussex, UK: Wiley Interscience, September 1985. [Online]. Available: http://books.google.de/books?id=BXBGAAAAYAAJ

[4] G. Z. Gutin and A. P. Punnen, Eds., *The Traveling Salesman Problem and its Variations*, ser. Combinatorial Optimization. Norwell, MA, USA: Kluwer Academic Publishers, 2002, vol. 12. [Online]. Available: http://books.google.de/books?id=TRYkPg_Xf20C

[5] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in *Revised Papers of the 5th International Workshop on Combinatorial Optimization – Eureka! You shrink! Papers dedicated to Jack Edmonds*, ser. Lecture Notes in Computer Science (LNCS), M. Jünger, G. Reinelt, and G. Rinaldi, Eds., vol. 2570. Aussois, France: Berlin, Germany: Springer-Verlag GmbH, March 5–9, 2001, pp. 185–207. [Online]. Available: http://faculty.cs.tamu.edu/chen/courses/689/2006/reading/w1.pdf

[6] L. D. Whitley, T. Starkweather, and D. Fuquay, "Scheduling problems and traveling salesman: The genetic edge recombination operator," in *Proceedings of the Third International Conference on Genetic Algorithms (ICGA'89)*, J. D. Schaffer, Ed. Fairfax, VA, USA: George Mason University (GMU): San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., June 4–7, 1989, pp. 133–140.

[7] M. Guntsch and M. Middendorf, "Applying population based aco to dynamic optimization problems," in *From Ant Colonies to Artificial Ants – Proceedings of the Third International Workshop on Ant Colony Optimization (ANTS'02)*, ser. Lecture Notes in Computer Science (LNCS), M. Dorigo, G. A. Di Caro, and M. Samples, Eds., vol. 2463/2002. Brussels, Belgium: Berlin, Germany: Springer-Verlag GmbH, September 12–14, 2002, pp. 111–122. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.

[8] S. Tsutsui, "Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram," in *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature (PPSN VII)*, ser. Lecture Notes in Computer Science (LNCS), J. J. Merelo-Guervós, P. Adamidis, H. Beyer, J. L. Fernández-Villacañas Martín, and H. Schwefel, Eds., vol. 2439/2002. Granada, Spain: Berlin, Germany: Springer-Verlag GmbH, September 7–11, 2002, pp. 224–233. [Online]. Available: http://www2.hannan-u.ac.jp/ tsutsui/ps/ppsn2002.pdf

[9] K. Helsgaun, "General k-opt submoves for the lin–kernighan tsp heuristic," *Mathematical Programming Computation*, vol. 1, no. 2-3, pp. 119–163, October 2009.

[10] J. D. C. Little, K. G. Murty, D. W. Sweeny, and C. Karel, "An algorithm for the traveling salesman problem," Cambridge, MA, USA: Massachusetts Institute of Technology (MIT), Sloan School of Management, Sloan Working Papers 07-63, March 1, 1963. [Online]. Available: http://dspace.mit.edu/bitstream/handle/1721.1/46828/algorit

[11] W. J. Cook, D. G. Espinoza, and M. Goycoolea, "Computing with domino-parity inequalities for the tsp," Atlanta, GA, USA: Georgia Institute of Technology, Industrial and Systems Engineering, Tech. Rep., 2005. [Online]. Available: http://www2.isye.gatech.edu/ wcook/papers/DP_paper.pdf

[12] G. Reinelt, "Tsplib 95," Heidelberg, Germany: Universität Heidelberg, Institut für Mathematik, Tech. Rep., 1995. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/DOC.P

[13] D. S. Johnson and L. A. McGeoch, "Experimental analysis of heuristics for the stsp," in *The Traveling Salesman Problem and its Variations*, ser. Combinatorial Optimization, G. Z. Gutin and A. P. Punnen, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002, vol. 12, ch. 9, pp. 369–443. [Online]. Available: http://www2.research.att.com/ dsj/papers/stspchap.pdf

[14] D. S. Johnson, G. Z. Gutin, L. A. McGeoch, A. Yeo, W. Zhang, and A. Zverovitch, "Experimental analysis of heuristics for the atsp," in *The Traveling Salesman Problem and its Variations*, ser. Combinatorial Optimization, G. Z. Gutin and A. P. Punnen, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002, vol. 12, ch. 10, pp. 445–487. [Online]. Available: http://www2.research.att.com/ dsj/papers/atspchap.pdf

[15] T. Weise, R. Chiong, K. Tang, J. Lässig, S. Tsutsui, W. Chen, Z. Michalewicz, and X. Yao, "Benchmarking optimization algorithms: An open source framework for the traveling salesman problem," *IEEE Computational Intelligence Magazine (CIM)*, vol. 9, no. 3, pp. 40–52, August 2014, featured article and selected paper at the website of the IEEE Computational Intelligence Society (http://cis.ieee.org/).

[16] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking: Experimental setup," Orsay, France: Université Paris Sud, Institut National de Recherche en Informatique et en Automatique (INRIA) Futurs, Équipe TAO, Tech. Rep., March 24, 2012. [Online]. Available: http://coco.lri.fr/BBOB-downloads/download11.05/bbobdocexpe

[17] D. A. D. Tompkins and H. H. Hoos, "Ubcsat: An implementation and experimentation environment for sls algorithms for sat and max-sat," in *Revised Selected Papers from the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, ser. Lecture Notes in Computer Science (LNCS), H. H. Hoos and D. G. Mitchell, Eds., vol. 3542. Vancouver, BC, Canada: Berlin, Germany: Springer-Verlag GmbH, May 10–13, 2004, pp. 306–320. [Online]. Available: http://ubcsat.dtompkins.com/downloads/sat04proc-ubcsat.pdf

[18] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*, ser. The Morgan Kaufmann Series in Artificial Intelligence. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005. [Online]. Available: http://books.google.de/books?id=3HAedXnC49IC

[19] H. H. Hoos and T. Stützle, "Evaluating las vegas algorithms – pitfalls and remedies," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI'98)*, G. F. Cooper and S. Moral, Eds. Madison, WI, USA: San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., July 24–26, 1998, pp.

238–245, also published as Technical Report "Forschungsbericht AIDA-98-02" of the Fachgebiet Intellektik, Fachbereich Informatik, Technische Hochschule Darmstadt, Germany. [Online]. Available: http://www.intellektik.informatik.tu-darmstadt.de/TR/1998/98-02.ps.Z

[20] W. Zhang, "Truncated branch-and-bound: A case study on the asymmetric traveling salesman problem," in *Proceedings of the AAAI-93 Spring Symposium on AI and NP-Hard Problems*. Stanford, CA, USA: Menlo Park, CA, USA: AAAI Press, March 23–25, 1993, pp. 160–166. [Online]. Available: www.cs.wustl.edu/ zhang/publications/atsp-aaai93-symp.ps

[21] W. Zhang, "Truncated and anytime depth-first branch and bound: A case study on the asymmetric traveling salesman problem," in *AAAI Spring Symposium Series: Search Techniques for Problem Solving Under Uncertainty and Incomplete Information*, ser. AAAI Technical Report, W. Zhang and S. König, Eds., vol. SS-99-07. Menlo Park, CA, USA: AAAI Press, March 1999, pp. 148–155. [Online]. Available: https://www.aaai.org/Papers/Symposia/Spring/1999/SS-99-07/SS99-07-026.pdf

[22] P. Hansen, N. Mladenović, and J. A. Moreno Pérez, "Variable neighbourhood search: Methods and applications," *Annals of Operations Research*, vol. 175, no. 1, pp. 367–407, March 1, 2010.

[23] R. Chiong, T. Weise, and Z. Michalewicz, Eds., *Variants of Evolutionary Algorithms for Real-World Applications*. Berlin/Heidelberg: Springer-Verlag, 2011. [Online]. Available: http://books.google.de/books?id=B2ONePP40MEC

[24] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Journal of Artificial Intelligence Research (JAIR)*, vol. 13, no. 2, pp. 129–170, April 1999. [Online]. Available: http://www.dca.fee.unicamp.br/ gomide/courses/EA072/artigos/Genetic_Algorithm_TSP_review_Larranaga_1999.pdf

[25] M. Guntsch and M. Middendorf, "A population based approach for aco," in *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN (EvoWorkshops'02)*, ser. Lecture Notes in Computer Science (LNCS), S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. R. Raidl, Eds., vol. 2279. Kinsale, Ireland: Berlin, Germany: Springer-Verlag GmbH, April 2–4, 2002, pp. 72–81. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.2514

[26] S. M. Oliveira, M. S. Hussin, T. Stützle, A. Roli, and M. Dorigo, "A detailed analysis of the population-based ant colony optimization algorithm for the tsp and the qap," Brussels, Belgium: Université Libre de Bruxelles, Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), IRIDIA – Technical Report Series TR/IRIDIA/2011-006, February 2011. [Online]. Available: http://iridia.ulb.ac.be/IridiaTrSeries/rev/IridiaTr2011-006r001.pdf

[27] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Pasadena, CA, USA: California Institute of Technology (Caltech), Caltech Concurrent Computation Program (C3P), Caltech Concurrent Computation Program C3P 826, 1989. [Online]. Available: http://www.each.usp.br/sarajane/SubPaginas/arquivos_aulas_IA/memetic.pdf

[28] P. Merz and B. Freisleben, "Memetic algorithms for the traveling salesman problem," *Complex Systems*, vol. 13, no. 4, pp. 297–345, 2001. [Online]. Available: http://www.complex-systems.com/pdf/13-4-1.pdf

[29] M. J. Streeter and S. F. Smith, "Exploiting the power of local search in a branch and bound algorithm for job shop scheduling," in *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling (ICAPS'06)*, D. Long, S. F. Smith, D. Borrajo, and L. McCluskey, Eds. Cumbria, UK: Menlo Park, CA, USA: AAAI Press, June 6–10, 2006, pp. 324–333. [Online]. Available: http://www.cs.cmu.edu/ matts/Research/mstreeter_icaps_2006.pdf

[30] M. S. Boddy and T. L. Dean, "Solving time-dependent planning problems," Providence, RI, USA: Brown University, Department of Computer Science, Tech. Rep. CS-89-03, February 1989. [Online]. Available: ftp://ftp.cs.brown.edu/pub/techreports/89/cs89-03.pdf

[31] S. Lin and B. W. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations Research (Oper. Res.)*, vol. 21, no. 2, pp. 498–516, March–April 1973. [Online]. Available: https://en.wikipedia.org/wiki/Lin%E2%80%93Kernighan_heuristic

This is a preview version of paper [1] (see page 9 for the reference). It is posted here for your personal use and not for redistribution. The final publication and definite version is available from IEEE (who hold the copyright) at http://www.ieee.org/. See also http://dx.doi.org/10.1109/CIPLS.2014.7007174.

@inproceedings{JWLCA2014CAHBABAWECMLSATHOTT,
  author   = {Yan Jiang and Thomas Weise and J{\"{o}}rg L{\"{a}}ssig and Raymond Chiong and Rukshan Athauda},
  title   = {Comparing a Hybrid Branch and Bound Algorithm with Evolutionary Computation Methods, Local Search and their Hybrids on the TSP},
  booktitle = {Proceedings of the IEEE Symposium on Computational Intelligence in Production and Logistics Systems (CIPLS'14), Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI'14)},
  publisher = {Los Alamitos, CA, USA: IEEE Computer Society Press},
  pages   = {148--155},
  address   = {Orlando, FL, USA: Caribe Royale All-Suite Hotel and Convention Center},
  year   = {2014},
  month   = dec # {~9--12, },
  ISBN   = {978-1-4799-5375-2},
  doi   = {10.1109/CIPLS.2014.7007174},
},