European Journal of Operational Research 211 (2011) 427–441

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Invited Review

Traveling salesman problem heuristics: Leading methods, implementations and latest advances

César Rego [a,*], Dorabela Gamboa [b], Fred Glover [c], Colin Osterman [a]

[a] School of Business Administration, University of Mississippi, MS 38677, USA
[b] Escola Superior de Tecnologia e Gestão de Felgueiras, CIICESI, GECAD, Instituto Politécnico do Porto, Apt. 205, 4610-156 Felgueiras, Portugal
[c] University of Colorado, Boulder, CO 80309-0419, USA

ARTICLE INFO

Article history:
Received 19 December 2009
Accepted 6 September 2010
Available online 21 September 2010

Keywords:
Traveling salesman problem
Heuristics
Ejection chains
Local search

ABSTRACT

Heuristics for the traveling salesman problem (TSP) have made remarkable advances in recent years. We survey the leading methods and the special components responsible for their successful implementations, together with an experimental analysis of computational tests on a challenging and diverse set of symmetric and asymmetric TSP benchmark problems. The foremost algorithms are represented by two families, deriving from the Lin–Kernighan (LK) method and the stem-and-cycle (S&C) method. We show how these families can be conveniently viewed within a common *ejection chain* framework which sheds light on their similarities and differences, and gives clues about the nature of potential enhancements to today's best methods that may provide additional gains in solving large and difficult TSPs.

1. Introduction

The traveling salesman problem (TSP) is undoubtedly the most extensively studied problem in combinatorial optimization. In popular language, the TSP can be described as the problem of finding a minimum distance tour of n cities, starting and ending at the same city and visiting each other city exactly once. In spite of the simplicity of its problem statement, the TSP is exceedingly challenging and has inspired well over a thousand publications devoted to analyses and algorithms attempting to solve it more effectively.

In graph theory, the problem can be defined on a graph $G = (V,A)$, where $V = \{v_1,\ldots,v_n\}$ is a set of n vertices (nodes) and $A = \{(v_i,v_j) | v_i,v_j \in V,\ i \neq j\}$ is a set of arcs, together with a non-negative cost (or distance) matrix $C = (c_{ij})$ associated with A. The problem is considered to be symmetric (STSP) if $c_{ij} = c_{ji}$ for all $(v_i,v_j) \in A$, and asymmetric (ATSP) otherwise. Elements of A are often called edges (rather than arcs) in the symmetric case. The version of STSP in which distances satisfy the triangle inequality $(c_{ij} + c_{jk} \geqslant c_{ik})$ is the most studied special case of the problem. The STSP (ATSP) consists in determining the Hamiltonian cycle (circuit), often simply called a *tour, of minimum cost*.

The TSP is a classic NP-complete combinatorial problem, and therefore there is no known polynomial-time algorithm (and unless $P = NP$, none exists) that is able to solve all instances of the problem. Consequently, heuristic algorithms are used to provide solutions that are of high quality but not necessarily optimal. The importance of identifying effective heuristics to solve large-scale TSP problems prompted the "8th DIMACS Implementation Challenge", organized by Johnson et al. [21] and solely dedicated to TSP algorithms.

In this paper we survey leading heuristics for the TSP, which come from the families called Lin–Kernighan (LK) and stem-and-cycle (S&C) methods, and report their computational performances. These foremost heuristics have proven to dominate other known approaches, solving TSP problems of vastly greater size and difficulty than would have been imagined possible before the advent of recent algorithmic developments. We also describe the state-of-the-art data structures used in the implementation of TSP algorithms, which play a key role in their efficiency.

As we demonstrate, the LK and S&C families of methods are members of a broader class known as *ejection chain* (EC) methods. Although there are several individual publications on ejection chain approaches to the TSP, with this paper we adopt the unifying EC perspective to give a convenient foundation for highlighting and differentiating the key features of the best current algorithms. Other general survey publications concerning heuristics for TSP, such as the excellent book chapters of Johnson and McGeoch [18,20], are no longer up to date and we include algorithms in our analysis that are not considered in these previous treatments. We also introduce and report computational outcomes for additional algorithms that represent new advances for solving problems in the ATSP class. Finally, we summarize latest

* Corresponding author. Tel.: +1 662 915 5519.
E-mail addresses: crego@bus.olemiss.edu (C. Rego), dgamboa@estgf.ipp.pt (D. Gamboa), fred.glover@colorado.edu (F. Glover), costerman@bus.olemiss.edu, colin.j.osterman@navy.mil (C. Osterman).

0377-2217/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2010.09.010

developments in data structures that are providing greater efficiencies in solving TSP problems.

Taking advantage of the ejection chain perspective, the following sections provide a brief survey of the most prominent algorithms for the TSP and discuss their salient performance characteristics, together with a summary of computational results that demonstrate the remarkable efficacy of these methods.

2. Ejection chain fundamentals

In the general context of combinatorial optimization, ejection chains are constructions to create variable-depth neighborhoods efficiently for local search procedures. The underlying technique consists of decomposing a very large neighborhood into a sequence of component neighborhood structures that can be evaluated in polynomial-time. Each component neighborhood structure in the sequence does not usually correspond to a feasible solution but constitutes a reference structure that permits a feasible solution to be obtained efficiently.

More advanced EC variants are typically based on specialized decision rules that adaptively take the search to promising regions of the solution space. In this paper, we specifically refer to ejection chains defined on graph structures relevant to the TSP setting.

2.1. Graph theory representation

We briefly overview the nature of an ejection chain process on a graph, as a foundation for a more specific description relative to the TSP problem. An ejection chain of L levels on a graph G consists of a sequence of simple operations, called *ejection moves*, $\langle e_1,\ldots,e_m,\ldots,e_L\rangle$, that iteratively transform a subgraph G_m of G into another subgraph G_{m+1} by disconnecting a subpath and reconnecting it with different components. At each level of the chain the subgraph composing the reference structure leads to a feasible solution via an extra operation called a *trial move*. Therefore, a neighborhood search ejection chain procedure consists in generating a sequence $\langle e_1,t_1,\ldots,e_m,t_m,\ldots,e_L,t_L\rangle$, where $\langle e_m,t_m\rangle$ represents the paired ejection and trial moves at level m of the chain. The new solution that constitutes the starting point of the next iteration is obtained from the compound move $\langle e_1,e_2,\ldots,e_m,t_m\rangle$, where the subscript m identifies the chain level that produced the best trial solution. Such a sequence can be efficiently evaluated by a simple recursive process. An extensive description of ejection chain methods for the TSP can be found in [33], and a recent survey of ejection chain methods and applications to other combinatorial optimization problems can be found in [34]. See also [30] for additional applications.

2.2. TSP ejection chains

In the TSP setting, the EC framework provides a hierarchy of four types of reference structures: (1) a detached stem (or chain) structure; (2) a stem-and-cycle structure; (3) a doubly-rooted stem-and-cycle structure; (4) a multi-cycle structure. Only the first two of these (the detached stem and the stem-and-cycle) have been extensively studied in the symmetric TSP setting, and only an intermediate variant along with modified instances of the second and third structures have been studied (much less extensively) in the asymmetric TSP setting. Consequently, there is a possibility that an exploration of elements of the ejection chain framework so-far unexamined may yield additional advances. We will not examine the issues surrounding these unexamined variants, but instead focus on the details of the widely-studied structures and the accompanying strategic devices that have produced their most effective implementations.

The detached stem procedure, which is better known as the Lin–Kernighan (LK) method [23], is far and away the most thoroughly investigated of the EC approaches for the TSP, and has been fortified and supplemented over the years with special auxiliary schemes to enhance its performance. Until the recent emergence of the stem-and-cycle (S&C) approach, the "name of the game" in TSP algorithmic development has largely consisted of efforts to find ever more subtle and ingenious ways to coax out another increment of speed or solution quality from implementations of the LK approach. Now, as will be seen, the game has acquired a new dimension, as the stem-and-cycle method has been found to match or surpass the best results of the LK approach without including several of the supporting techniques that have contributed to the LK successes. This raises the question for future study concerning whether the auxiliary strategies that will provide the greatest enhancements for the S&C approach will turn out to have a different character than those used to enhance the LK method. Currently, this is a realm for speculation, and no real answers are known.

An interesting discovery that surfaced from the EC framework is a relationship between the LK and S&C methods that binds them to specialized graph theory algorithms for certain problems of polynomial complexity. In particular, the operations of the LK approach result in generating classical alternating paths, having the same structure (though derived from different mechanisms) as the alternating paths implicitly and explicitly exploited by well known network assignment algorithms and matroid intersection algorithms. The S&C approach also generates alternating paths, but of a more general class that is not considered by the classical graph theory constructions. Over the subclass that consists of ordinary alternating paths, the S&C method additionally generates instances that are not accessible to the LK method.

These and other similarities and differences between these two fundamental EC methods will be elaborated in the next sections.

3. Symmetric TSP

We first summarize the main components of the most effective instances of the LK and S&C algorithms for the symmetric TSP and analyze their performance. The basic difference between these two methods derives from the difference between their reference structures. The detached stem reference structure of the LK method is a Hamiltonian path (arising by dropping an edge of the TSP tour), while the S&C reference structure consists of a node simple cycle attached to a path. In each case, the reference structures include all nodes of the graph. An interesting theoretical analysis of the differences between the types of paths generated by S&C and LK procedures, disclosing properties of the increased range of solutions accessible to the S&C approach, is provided in Funke et al. [10].

In the following, we first describe leading variants of the LK method, and then follow with a description of the S&C method and its performance characteristics.

3.1. The LK method and its variants

The Lin–Kernighan neighborhood search is designed as a method to generate k-opt moves (which consist in deleting k edges and inserting k new edges) in a structured manner that provides access to a relevant subset of these moves by an efficient expenditure of computational effort. The approach is based on the fact that any k-opt move can be constructed as a sequence of 2-opt moves [6], and a restricted subset of those move sequences can be produced in a systematic and economic fashion.

The method starts by generating a low order k-opt move (with $k \leqslant 4$) and then creates a Hamiltonian path by deleting an edge adjacent to the last one added. This path provides the first $k-1$ levels of the ejection chain used by the LK process. Fig. 1 illustrates the three types of starting moves for initiating the LK ejection chain procedure (i.e. 2-, 3-, and 4-opt moves). The node sequence $\langle t_1, t_2, \ldots, t_L \rangle$ identifies the succession of add-drop operations used in the ejection move and correspondingly indicates the order that nodes have been selected. Dotted lines show edges that are dropped from the original tour, and solid lines cutting across the "interior" of the tour (in the layout of the diagram) show new edges that are added to the tour. In each case, the resulting structure of solid lines is a Hamiltonian path from which the associated k-opt move can be obtained by linking the highest indexed t node to the initial node t_1.

At succeeding levels each new move consists of adding an edge to the node that was met by the last edge deleted (the "opposite end" of the path of solid lines starting from t_1, represented by the highest indexed t node) and dropping the unique resulting edge that again will recover a Hamiltonian path. This type of ejection move is also available at levels 2 and 3 of the ejection chain illustrated above as inherent options to generate 3-opt and 4-opt moves, respectively.

Additional sophistication of the basic method is provided by a backtracking process that allows testing different alternatives for 2-, 3-, and 4-opt moves that derive from the same base node t_1 (as illustrated in Fig. 1) and then proceeding iteratively until reaching level L.

3.1.1. Johnson and McGeoch Lin–Kernighan (LK–JM)

The LK implementation of Johnson and McGeoch [19] is featured among the lead papers of the "8th DIMACS Implementation Challenge" [21]. The results reported for this implementation use four supplementary components to enhance the method's efficacy: greedy initial solutions, the k-quadrant neighbor list [26] with $k = 20$, the don't-look-bits strategy [4], and the 2-level tree data structure [7]. We provide an abbreviated description of each of these algorithmic components to give an indication of their nature. To provide a complete tabulation, we also list the two auxiliary strategies embodied in the original LK proposal, and one additional strategy that is also incorporated into all of the best LK implementations.

(1) The Greedy heuristic (aka multi-fragment heuristic [4]) is a constructive algorithm that creates a tour from scratch by adding at each step the shortest valid edge among those not already included in the tour being constructed-where an edge is considered valid if it does not create a degree-3 node and hence produces a subtour (i.e. a cycle of length less than n).
(2) The k-quadrant neighbor list is a device for restricting attention to a subset of the edges of the graph as candidates for edges that may be incorporated into the alternating path and hence become edges of a new tour produced by an ejection chain move. The list provides an alternative to the k-nearest neighbor list classically used for this purpose, and operates by choosing the k neighbors of a node evenly from each quadrant of a 2-dimensional Euclidian plane that takes the indicated node as its origin. If not enough neighbors are found in some quadrant, then the list is completed by adding the nearest neighbors not already included.
(3) The don't-look-bits strategy is a special type of adaptive memory technique used to restrict the neighborhood space (by limiting the choices for a LK base node t_1) and to help guide the search to regions of the problem graph where further improvements are more likely to be found. This is done by associating a binary variable (or flag) with each node, where the bit for a node is turned off or on, respectively, to indicate that the node is permitted, or not permitted, to serve as a base node to start an ejection chain. A bit for a node is turned on (to prevent its selection as a base node) the first time the selection of this node does not produce an improving move. Conversely, it is turned off (to permit its selection as a base node) when one of its adjacent nodes is used as a base node that produces an improving move. (According to the personal communication revealed by Neto [27] the strategy is implemented by keeping an active list of turned-off nodes organized in a first-in-first-out queue, which supports constant time membership test and update, somewhat similar to the sparse set representation described in [5].)
(4) The 2-level tree is a specialized data structure to represent tours in TSP local search algorithms in order to update the tour structure efficiently. In particular, the implementation of the neighborhood structure often requires reversing the recorded sequence of a tour subpath in order to allow the nodes of the tour to be listed sequentially. For example, closing the 2-opt move of Fig. 1 (by linking t_4 to the base node t_1) requires reversing the subpath from t_2 to t_4. Because these subpaths can be very long in large TSP graphs, path reversal operations are computationally expensive. Due to its practical efficiency, the 2-level tree became a standard data structure in the implementation of LK algorithms, and likewise has been used with other types of ejection chain methods as will be discussed later. Finally, we conclude the tabulation by listing the auxiliary strategies previously mentioned that come from the original LK proposal and the special strategy more recently used by the best methods to overcome a limitation subsequently identified in the LK method.
(5) The backtracking method.
(6) The implicit k-opt moves for $k = 2$, 3 and 4.
(7) The double-bridge kick moves (discussed later).

In addition to these components, the Johnson and McGeogh implementation incorporates the *legitimacy restrictions* introduced

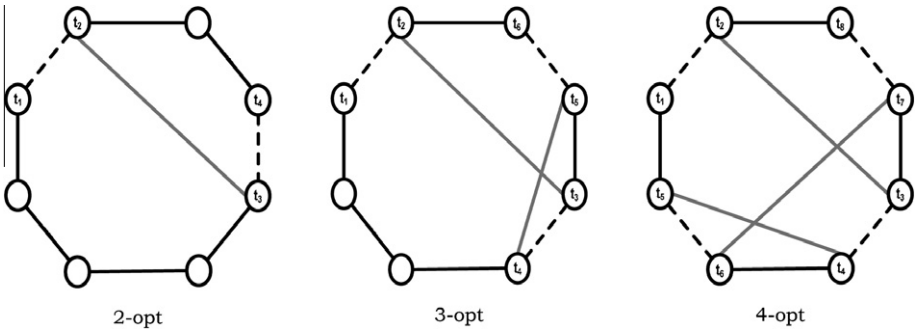


Fig. 1. Possible moves at the first level of the Lin–Kernighan ejection chain method.

by Lin and Kernighan which constitute a guiding mechanism that limits the moves permitted to be generated throughout the ejection chain process. These restrictions have the result of maintaining a classical (static) alternating path neighborhood structure by preventing the method from deleting an edge that has been newly added during the construction of the current ejection chain. This restriction implicitly bounds the ejection chain length L to n levels, but Johnson and McGeogh additionally found it useful to stop the search at level 50. Following the original proposal, backtracking (item (5) above) is also used any time an ejection chain fails to produce an improved solution, in which case all alternatives for t_2 to t_6 are tested, but limiting consideration for each of them to examining only one possible alternative for the pair (t_7, t_8).

The champion version of the Johnson and McGeogh implementation (with respect to solution quality) is the chained (or iterated) Lin–Kernighan algorithm (ILK), which extends the basic version with a perturbation mechanism and the use of multiple starts. Instead of terminating the basic algorithm upon reaching a LK local optimum, the iterated method looks for an improving double-bridge move (item (7) above), and if one is found a new LK search is launched from this improved solution. In this implementation, double-bridge moves to be examined are generated at random over the entire set of nodes in the graph. If the move generated does not improve the current best tour, the move is not applied and a new double-bridge move is attempted. This process is repeated for as many iterations as desired.

The structure of the double-bridge move derives from a special 4-opt neighborhood where edges added and dropped need not be successively adjacent, thus overcoming the potential limitation of the LK method which imposes such a successive adjacency requirement. As shown in Fig. 2, this move consists of a sequence of two disconnected 2-exchange moves and does not entail subpath reversals. The first exchange deletes edges (v_1, v_2), (w_1, w_2) and links their endpoints by adding edges (v_1, w_2), (w_1, v_2), which results in two subtours. The second exchange deletes edges (w_3, w_4), (w_5, w_6) and adds edges (w_3, w_6), (w_5, w_4) to create a feasible tour. Each ILK iteration initializes the don't-look-bits active list with the eight nodes involved in the improving double-bridge move, as suggested by Martin et al. [24,25].

In what follows, we will indicate the primary algorithms that incorporate one or more of these strategies in their design, including the best algorithms as determined by the 8th DIMACS Implementation Challenge. We also describe new algorithms that incorporate more innovative structures and that achieve the highest levels of performance among the methods now available.

3.1.2. Neto's Lin–Kernighan (LK–N)

This implementation, described in [27], differs from the LK–JM implementation primarily in its incorporation of special cluster compensation routines. In general, problems in which nodes are not uniformly distributed but may clump and cluster pose a challenge to the classical LK heuristic. The reason conjectured for this LK weakness is that once an edge that links two distinct clusters is deleted, the search for new Hamiltonian paths tends to focus on nodes in the particular cluster that contains the new "end node" of the path under construction. It has been found that adding an edge that links distant clusters, coupled with dropping an edge in one of these clusters, is likely to produce a significant gain. The opposite result is found when the two ends of the Hamiltonian path (lying in different clusters) have to be connected to produce a new tour. Therefore, Neto's strategy consists of altering the neighborhood evaluation function so that the potential harm of the cross-cluster linking move in the LK procedure is not completely ignored, keeping the search from getting too greedy in choosing the move that links to the best neighboring solution at each level of the ejection chain. Instead of choosing the reference structure that gives the lowest cost ejection move or the one that gives the best tour from the associated trial solution, the move evaluation is adjusted to incorporate an auxiliary term that reflects the magnitude of the costs of the linking move expected to be encountered in further levels of the chain. This term is defined by the cost of the shortest edge in the set that contains the longest edge of any path linking the two nodes under consideration. This cost is efficiently obtained by observing that the required edge corresponds to the longest edge of the unique path linking the two nodes in the minimum spanning tree of the underlying graph G. Hence, the task is reduced to computing a minimum spanning tree and then efficiently computing the longest edge between pairs of nodes in the tree.

Once a minimum spanning tree is constructed the query for the longest edge between two nodes can be answered in constant time [16] by finding the least common ancestor of the two nodes in the binary tree representation of the spanning tree. On the flip side this operation adds to the algorithm a preprocessing time bounded by $O(n \log n)$ as well as $O(n)$ extra space for data structures. However, results reported in [27] show improvements produced by the cluster compensation technique that significantly outweigh its overhead. In these experiments, Neto reports savings of 50–100% in running times without compromising solution quality. Although this advantage is more prevalent in clustered instances the technique also proved effective with uniformly structured instances when embedded in an Iterative LK implementation, finding solutions of similar quality and sometimes better in shorter running times. In sum, Neto's LK implementation with cluster compensation is superior to its classical counterpart that does not incorporate the strategy. The other aspects of the code are similar to that of LK–JM implementation [19] discussed above, except that it aggregates 20 quadrant neighbors and 20 nearest neighbors into a single candidate list.

3.1.3. Applegate, Bixby, Chvatal, and Cook Lin–Kernighan (LK–ABCC)

This implementation is part of the Concorde library [1] and is based on [2]. It uses Q-Boruvka starting tours, 12-quadrant neighbor candidate lists, the don't-look-bits technique, and the 2-level tree data structure. LK–ABCC bounds the LK searches by 50 moves, and the backtracking technique is slightly deeper than that of the LK–JM implementation. The don't-look-bits strategy is implemented using first-in-first-out priority queue similar to that used in the codes described above.

A number of subtleties of this code deserve consideration. Instead of selecting double-bridge moves randomly as in the previously discussed approaches, the authors developed two different types of cost-restricted double-bridge moves called *close kick* and *geometric kick*. For what follows we refer to the double-bridge diagrams of Fig. 2. Both variants of the kicking strategies start with the selection of a node v_1 that is locally optimum among a small

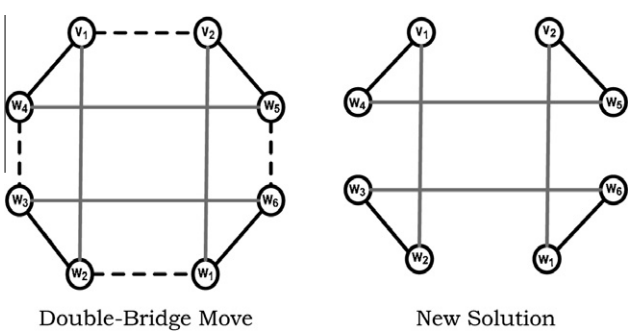


Fig. 2. The double-bridge neighborhood structure.

number r of randomly generated nodes (e.g. $0.003n \leqslant r \leqslant 0.03n$). The local optimality criterion is the one that minimizes the cost of a hypothetic partial move that adds an arc (v_1, w_2) and deletes the arc (v_1, v_2), where v_2 is the successor of v_1 in the current orientation of the tour and the w_2 is the nearest neighbor of v_1. Once v_1 has been selected, the close kicking strategy finds the 4-opt double-bridge move by selecting edges (w_1, w_2), (w_3, w_4), and (w_5, w_6) in a restricted neighborhood where nodes w_{2i-1} $(i = 1,2,3)$ must be among the six nearest neighbors of v_1. (Note that the initial node w_2 used in the selection of v_1 is not necessarily part of the best double-bridge move found in the restricted neighborhood.) The geometric kick changes the restricted neighborhood by allowing the w_{2i-1} nodes to be selected at random among the k-nearest neighbors of v_1. In the computational analysis reported in [33] the authors conclude that while geometric kicks outperform the general random kicks for instances having more than 10,000 nodes, the random kicks are usually preferable for the smaller instances. Therefore, the algorithm switches between random and geometric kicks depending on the problem size.

3.1.4. Applegate, Cook and Rohe Lin–Kernighan (LK–ACR)

The implementation of this method is very similar to that of the preceding LK–ABCC approach, but the backtracking strategy is even deeper and broader. The depth of the LK searches, by contrast, is half that of the LK–ABCC approach (25 moves). This implementation is based on the design reported in [2,3]. The main new ingredient seems to be the introduction of a new method to generate restricted double-bridge moves called *random-walk kick*. The method consists of three random walks from v_1 in the 12-quadrant neighbor graph where nodes w_{2i-1} are those reached in a walk i for $i = 1,2,3$. This perturbation procedure can be executed much faster than the geometric kicks used in LK-ABCC, especially for general (non-geometric) instances; hence it is adopted for the solution of the largest instances involving several millions of nodes.

3.1.5. Helsgaun's Lin–Kernighan variant (LK–H)

This implementation, described in [17], modifies several aspects of the original Lin–Kernighan heuristic. The most notable difference is found in the search strategy. The algorithm uses larger (and more complex) search steps than the original procedure. Also, sensitivity analysis is used to direct and restrict the search. The algorithm does not employ backtracking, but uses the don't-look-bits technique and the 2-level tree data structure.

LK–H is based on 5-opt moves restricted by carefully chosen candidate sets. Helsgaun's method for creating candidate sets may be the most valuable contribution of the algorithm. The rule in the original algorithm restricts the inclusion of links in the tour to the five nearest neighbors of a given city. LK–JM includes at least 20 nearest quadrant neighbors. Helsgaun points out that edges selected simply on the basis of length may not have the highest probability of appearing in an optimal solution. Another problem with the original type of candidate set is that the candidate subgraph need not be connected even when a large fraction of all edges is included. This is the case for geometrical problems in which the point sets exhibit clusters.

Helsgaun therefore develops the concept of an α-nearness measure that is based on sensitivity analysis of minimum spanning 1-trees. This measure tries to better reflect the probability that an edge will appear in an optimal solution. It also handles the connectivity problem, since a minimum spanning tree is (by definition) always connected. The key idea, in brief, is to assign a value to each edge based on the length of a minimum 1-tree containing it. A candidate set of edges can then be chosen for each city by selecting edges with the lowest values. The effectiveness of α-nearness in selecting promising edges can be further improved by transforming the graph. For this, a subgradient optimization method is utilized that strives toward obtaining graphs in which minimum 1-trees are close to being tours.

By using the α-measure, the cardinality of the candidate set may generally be small without reducing the algorithm's ability to find short tours. In fact, Helsgaun claims that for his initial set of test problems, the algorithm was able to find optimal tours using as candidate edges the 5 α-nearest edges incident to each node.

3.1.6. Nguyen, Yoshihara, Yamamori and Yasunaga Lin–Kernighan variant (LK–NYYY)

A short description of this implementation can be found in [21]. This variant starts with a 5-opt move but uses 3-opt moves in the LK searches as opposed to the LK–H approach that uses 5-opt as a basic move. The LK–NYYY variant also uses the don't-look-bits technique, greedy starting solutions, and 12-quadrant neighbor lists, but uses a data structure with properties similar to segment trees [9]. The results reported from this algorithm were submitted to the DIMACS Challenge after the summary chapter [21] was finished. A highly significant difference from the Helsgaun variant is that LK–NYYY is able to run instances up to 1,000,000 nodes whereas LK–H only manages instances up to 85,900 nodes and consumes a significant amount of computational time as is evident in Table 4.

3.2. Stem-and-cycle implementations

Because of its relatively recent emergence on the scene, by comparison to the LK procedure, the stem-and-cycle method has to date been embodied in only two different implementations. A first attempt was independently conducted by Pesch and Glover [31] and Rego [32], the latter incorporating aspects to improve efficiency. An updated version of Rego's implementation was later undertaken by Gamboa et al. [11,12]. We describe the fundamental ideas underlying the two more effective versions and then describe their details and the way that they differ.

3.2.1. Rego, Glover and Gamboa stem-and-cycle (S&C-RGG)

The S&C-RGG algorithm implements an ejection chain method that differs from the LK procedure in several key ways, which chiefly derive from the differences in the reference structure employed by the two methods. The Hamiltonian path reference structure used by the LK approach is very close to being a valid TSP solution (only requiring the single edge to be added that "closes" the path to produce a tour). As a result, the structure implicitly limits the types of moves it can generate. At the next rung above the LK reference structure in the ejection chain hierarchy, the S&C reference structure is able to generate moves that the LK structure cannot.

Drawing on the characterization of the stem-and-cycle structure proposed by Glover in [13], the implementation reported here was designed by Rego [32] and subsequently enhanced by Gamboa et al. [11,12]. The S&C reference structure is a spanning subgraph of G consisting of a path called a stem $ST = (v_t, \ldots, v_r)$ connected to a cycle $CY = (v_r, v_{s_1}, \ldots, v_{s_2}, v_r)$. A diagram of this reference structure is given in Fig. 3. The vertex v_r shared by the stem and the cycle is called the root, and the two vertices of the cycle adjacent to v_r are called subroots. Vertex v_t, which terminates the stem (at the opposite extremity from the root v_r), is called the tip of the stem.

The S&C method creates its initial reference structure from a TSP tour, by adding a non-tour edge to link two non-adjacent nodes of the tour and then removing one of the two tour edges adjacent to the added edge. As shown in Fig. 3, ejection moves are carried out within this reference structure by adding an edge to join the tip node v_t to any other node v_p of the graph except for the unique node adjacent to the tip. Two different ejection moves are possible depending whether v_p lies on the stem or on the part of the cycle that excludes the root v_r. The former situation

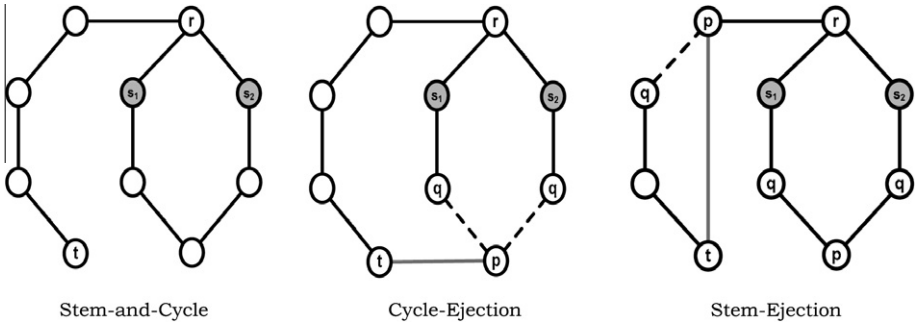


Fig. 3. The stem-and-cycle structure and ejection moves.

gives rise to a *stem-ejection* move and the latter gives rise to a *cycle-ejection* move. Trial solutions yielding valid TSP tours are obtained by adding an edge (v_t, v_{s_i}) linking the current tip v_t to one of the subroots v_{s_i} $(i \in \{1,2\})$ and deleting the adjacent edge (v_{s_i}, v_r).

The results reported here come from the updated version of the S&C implementation, and improve upon those for the S&C method reported in the DIMACS challenge by incorporating changes outlined in [12] and by adding a variant of the don't-look-bits strategy. We present results using greedy initial solutions, 12-quadrant neighbor candidate lists concatenated with a list generated by the construction of Delaunay triangulations and the 2-level tree data structure. In its present form the algorithm executes a local search procedure without resorting either to the backtracking strategy or the double-bridge kick strategy, as incorporated into the best LK implementations. Similarly, no recourse is made to the implicit k-opt moves ($k = 2,3,4$) that are used in these best implementations, although such moves arise as natural options within the first levels of the ejection chain generated by the S&C algorithm.

Because of the differences between the LK and S&C reference structures, the node structure of the 2-level tree (used to update the tour representation) must be slightly modified to efficiently accommodate the peculiarities of the S&C structure. In addition, specialized requirements are incorporated in the organization and content of the data segments of the S&C structure in order to facilitate the execution of each type of ejection move and its associated trial moves. A comprehensive description of the necessary operations to manipulate the 2-level tree data structure for the S&C ejection chain are given in [9].

In summary, apart from the fact that our implementation of the S&C method has excluded three of the strategies used to support the leading LK implementations, the essential differences between the two methods reside in the types of trial moves and trial solutions available at each step.

3.3. Comparative analysis of performance

We now evaluate the performance of the heuristic algorithms referenced above using the results obtained for the "8th DIMACS Implementation Challenge" [21] (submitted during and after the conference was held), together with the updated results for S&C-RGG. We restrict attention to the evaluation of the results reported for the algorithms relevant to this paper's main focus, hence do not include results for methods that did not perform at a competitive level. A complete list of algorithm results and other information related to the generation of the testbed instances, the scale factors to compare running times for different computer systems, and other characteristics of the challenge can be found on the Challenge web site [21].

The Challenge testbed consists of three sets of instances: uniformly distributed problems (sizes between 1000 and 10,000,000 nodes), clustered problems (sizes between 1000 and 316,228 nodes), and those from the TSP Library [35] with at least 1000 nodes. In the current study we limited the number of problems to instances up to 3,000,000 nodes.

A benchmark code was provided for Challenge participants that was run on the same machines used to run the competing algorithms of the participants, in order to obtain a more accurate comparison of running times. The tests for the updated version of S&C have been run on the same machine used to run the first S&C version for the DIMACS Challenge, and the same scale factor has been used to normalize the new implementation running times. An exception was made for the 3 million-node problem whose results were obtained on a Dual Intel Xeon, 3.06 GHz with 2 GB of memory. A scale factor of 2.89 was used to compute our normalized time for this problem.

Tables 1–4 summarize the results of the aforementioned algorithms. The values presented are averages of solution quality and computational times (in seconds), where instances are grouped by size. This grouping is similar to the one used by Johnson and McGeoch [20] to design the tables of results in their book chapter summarizing the Challenge's submissions. Again it is important to stress that a number of algorithms and results described here were submitted or updated after the chapter was published. In the solution quality tables, in addition to reporting average percentage excess over the optimal solution or over the Held-and-Karp lower bound, we present the number of best solutions (NBS) found by each algorithm, i.e., the number of problem instances for which the associated algorithm obtained a solution of higher quality than other algorithms. We utilize the notation $x|y$ to signify that the associated algorithm found x better solutions than all other algorithms (excluding S&C-RGG$^+$ from the comparison) and y better solutions than all other algorithms (excluding S&C-RGG from the comparison) in the corresponding group of problems. To facilitate the analysis of the tables, we replace zeros with dashes and likewise for cases where $x = y$ we use only a single number to avoid repetition. For example, Figs. 1,2 for LK–JM in the group 1000/10 indicate that this algorithm found one better solution than all the others (excluding S&C-RGG$^+$) and two better solutions than all the others (excluding S&C-RGG). Accordingly, when either S&C-RGG or S&C-RGG$^+$ are considered in the analysis, we see that each of them finds seven and five better solutions than all other algorithms, respectively. The values in bold in the tables indicate the best averages.

We separate the basic LK algorithmic variants and the S&C approach from the two "iterated" LK variants since the latter employ an extended set of auxiliary strategies and consume considerably greater amounts of computation time as a result of switching back and forth between these strategies and the basic LK routine, consequently placing them in a different category of method. The basic LK variants and the S&C method alike determine moves by deleting one edge and inserting another one, completing the 2-exchange with a trial move. The NYYY and Helsgaun variants additionally search for valid 3-exchange and 5-exchange moves. To make this

Table 1
Basic LK and S&C – solution quality.

Problem size/number of instances – 25 uniformly distributed problems																		
	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1		1000000/1		3000000/1		Total	
Algorithm	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK–JM	1.18	1|2	1.27	1|2	2.02	–	2.02	–	1.97	–	1.96	–	1.96	–	1.92	–	1.79	2|4
LK–N	1.17	–|2	1.26	–	1.99	–	1.88	–	1.95	–	1.97	–	1.92	–	1.878	–	1.75	–|2
LK–ABCC	1.47	2|1	1.71	–	2.60	–	2.48	–	2.54	–	2.67	–	2.68	–	2.55	–	2.34	2|1
LK–ACR	1.61	–	2.18	–	2.72	–	2.72	–	2.74	–	2.75	–	2.77	–	2.67	–	2.52	–
S&C-RGG	**0.79**	7	**0.95**	4	**1.68**	3	**1.61**	2	**1.65**	2	**1.86**	1	**1.91**	1	**–**	–	**1.49**	**20**
S&C-RGG$^+$	**0.93**	5	**0.98**	3	**1.55**	3	**1.66**	2	**1.72**	2	**1.84**	1	**1.90**	1	**1.875**	1	**1.56**	**17 + 1**

Problem size/number of instances – 23 clustered problems														
	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1		Total	
Algorithm	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK–JM	**1.21**	6	2.32	4|2	3.41	2	3.72	–	**3.63**	1|2	**3.67**	1	**2.99**	14|13
LK–N	1.97	1	3.55	–	4.76	–	4.42	–	4.78	–	–	–	3.90	1
LK-ABCC	3.22	–	5.58	–	5.70	–	6.38	–	5.31	–	5.45	–	5.27	0
LK-ACR	3.34	–	5.48	–	5.92	–	6.28	–	5.55	–	5.54	–	5.35	0
S&C-RGG	1.35	3	2.57	1	**3.24**	1	**3.16**	2	3.69	1	3.99	–	3.00	**8**
S&C-RGG$^+$	1.79	3	**2.24**	3	**3.27**	1	**3.29**	2	3.72	–	3.81	–	3.02	**9**

Problem size/number of instances – 11 TSPLIB problems												
	1000/4		3162/3		10000/2		31623/1		100000/1		Total	
Algorithm	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK–JM	1.40	–	1.28	–	1.38	–	1.23	–	1.213	–	1.30	–
LK–N	1.43	–	1.44	–	1.34	–	1.49	–	–	–	1.43	–
LK–ABCC	2.56	–	2.41	–	1.86	–	1.65	–	1.208	–|1	1.94	–|1
LK–ACR	3.49	–	2.59	–	3.17	–	2.40	–	2.00	–	2.73	–
S&C-RGG	**0.52**	4	**0.60**	3	**0.91**	2	**1.02**	1	**1.17**	1	**0.84**	**11**
S&C-RGG$^+$	**0.89**	4	**0.79**	3	**0.94**	2	**1.21**	1	1.57	–	**1.08**	10

Table 2
Basic LK and S&C – computational time.

Problem size/number of instances – 25 uniformly distributed problems								
	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1	1000000/1	3000000/1
Algorithm	CPU	CPU	CPU	CPU	CPU	CPU	CPU	CPU
LK–JM	0.16	0.53	1.77	6.81	27.74	108.87	493.42	2049.28
LK–N	0.19	0.87	3.35	14.40	89.58	574.42	3577.74	17660.51
LK–ABCC	0.09	0.34	1.49	5.95	21.43	60.79	307.17	1332.79
LK–ACR	0.07	0.29	0.93	2.95	16.40	76.32	318.10	1289.25
S&C-RGG	4.04	19.82	100.92	733.93	5804.09	33239.39	255971.44	–
S&C-RGG$^+$	2.95	13.49	80.88	313.11	2872.61	13584.87	69542.57	336304.79

Problem size/number of instances – 23 clustered problems						
	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1
Algorithm	CPU	CPU	CPU	CPU	CPU	CPU
LK–JM	1.30	3.62	11.99	57.65	211.30	916.91
LK–N	4.35	15.04	51.17	138.59	558.07	–
LK–ABCC	0.20	0.72	2.55	11.04	37.91	107.67
LK–ACR	0.11	0.45	1.40	4.49	24.97	114.19
S&C-RGG	4.17	18.41	135.12	956.39	5416.85	60199.97
S&C-RGG$^+$	2.58	16.25	89.02	445.76	3818.27	39353.15

Problem size/number of instances – 11 TSPLIB problems					
	1000/4	3162/3	10000/2	31623/1	100000/1
Algorithm	CPU	CPU	CPU	CPU	CPU
LK–JM	0.29	0.54	3.61	14.57	35.90
LK–N	0.41	1.08	10.26	47.09	–
LK–ABCC	0.10	0.29	1.22	3.48	8.84
LK–ACR	0.08	0.23	0.74	1.74	5.42
S&C-RGG	7.59	26.47	134.99	665.85	3253.16
S&C-RGG$^+$	4.14	21.67	78.11	505.99	1461.85

search possible without consuming excessively large amounts of computation time, these two latter procedures use special and highly sophisticated candidate lists as previously noted.

In order to assess the potential effect of using restricted neighborhood search of the type employed by the don't-look-bits strategy considered in the LK implementations, we report results for a first attempt to incorporate this technique in the S&C algorithm. In the tables, S&C-RGG$^+$ refers to the version of the S&C algorithm that adds restricted neighborhood search to S&C-RGG.

Table 3
Helsgaun & NYYY – solution quality.

Problem size/number of instances – 24 uniformly distributed problems																
	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1		1000000/1		Total	
Algorithm	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK–H	0.16	10	0.19	5	0.83	3	0.83	2	–	–	–	–	–	–	0.50	20
LK–NYYY	0.73	–	0.74	–	1.57	–	1.48	–	1.48	2	1.53	1	1.49	1	1.29	4

Problem size/number of instances – 23 clustered problems														
	1000/10		3162/5		10000/3		31623/2		100000/2		316228/1		Total	
Algorithm	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK–H	0.71	8	1.38	4	3.32	1	3.58	1	–	–	–	–	2.25	14
LK–NYYY	1.22	2	2.18	1	3.08	2	3.45	1	3.51	2	3.49	1	2.82	9

Problem size/number of instances – 11 TSPLIB problems												
	1000/4		3162/3		10000/2		31623/1		100000/1		Total	
Algorithm	%	NBS	%	NBS	%	NBS	%	NBS	%	NBS	Average	NBS
LK–H	0.24	4	0.15	3	0.24	2	0.46	1	0.85	1	0.39	11
LK–NYYY	1.15	–	0.86	–	0.72	–	0.99	–	1.03	–	0.95	0

Table 4
Helsgaun & NYYY – computational time.

Problem size/number of instances – 24 uniformly distributed problems							
	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1	1000000/1
Algorithm	CPU	CPU	CPU	CPU	CPU	CPU	CPU
LK–H	5.64	71.49	861.71	7819.27			
LK–NYYY	0.16	0.57	1.76	4.97	20.86	84.73	507.62

Problem size/number of instances – 23 clustered problems						
	1000/10	3162/5	10000/3	31623/2	100000/2	316228/1
Algorithm	CPU	CPU	CPU	CPU	CPU	CPU
LK–H	6.93	70.28	768.31	12812.46		
LK–NYYY	0.50	1.36	3.96	9.68	38.81	147.20

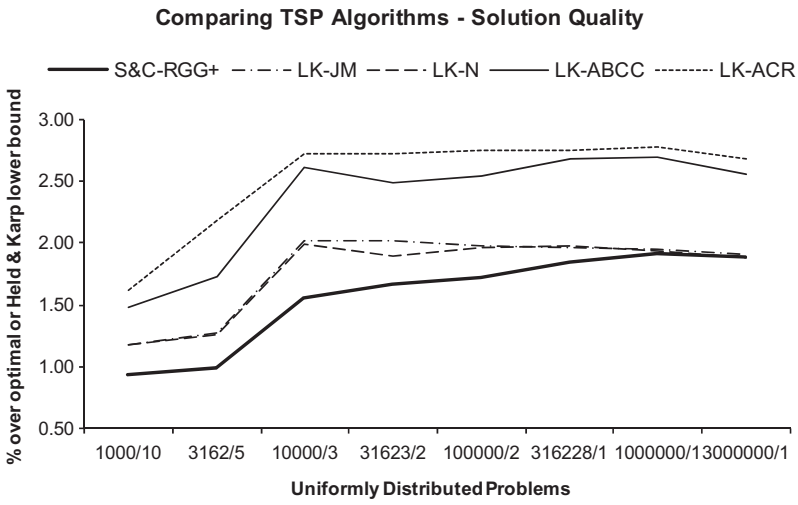
Problem size/number of instances – 11 TSPLIB problems					
	1000/4	3162/3	10000/2	31623/1	100000/1
Algorithm	CPU	CPU	CPU	CPU	CPU
LK–H	7.82	73.32	1063.13	7982.09	48173.84
LK–NYYY	0.26	0.66	1.96	5.09	13.06

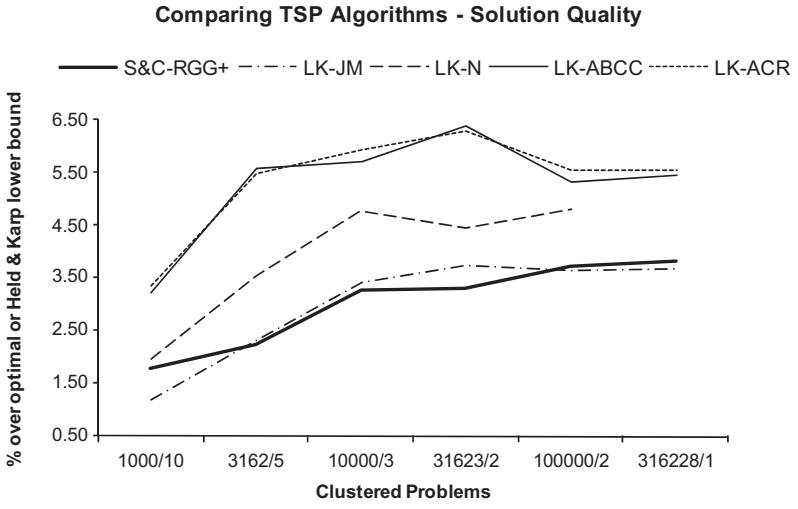
Tables 1 and 2 disclose that the S&C approach is better than all other implementations for generating high quality solutions. Figs. 4 and 5 provide a graphical visualization of the results summarized in Tables 1 and 2, respectively. Note that besides achieving better solution quality on average, both S&C variants find a significantly larger number of best solutions across all problems and tables, at the expense of longer running times.[1]

The graphics in Fig. 6 show the effect of the don't-look-bits strategy on the S&C algorithm, showing solution quality in Table 1 and computation times in Table 2. We see that even a straightforward implementation of the don't-look-bits candidate list strategy yield appreciable reductions in the running times for the S&C algorithm without sacrificing the solution quality. In some cases the quality of the solutions is even better when restricting the neighborhood, suggesting that more elaborate implementations of the don't-look-bits strategy can have a dual effect on the performance of the S&C by simultaneously improving the efficiency and effectiveness of the algorithm. For the uniformly distributed problems, the variant of the S&C algorithm that makes use of don't-look-bits (S&C-RGG$^+$) performs better than its counterpart (S&C-RGG), in three out of the seven problems in this group, while performing comparably on the remaining problems. Also as illustrated in the graphics of Fig. 6, the running times with the don't-look-bits strategy grow sublinearly with the problem size while these times grow much more rapidly in the absence of this strategy. A similar advantage should be expected for clustered problems, as occurred in the case of the LK implementations; hence this topic invites special attention in future developments.

We conjecture that additional improvements can be made by using more effective neighbor lists that restrict the neighborhood size without omitting arcs that may be critical to perform potentially good moves. A neighbor list that is not designed with utmost care can prevent the best solutions from being found by failing to include some of the arcs in this solution, and can also cause the search to take much longer to find these solutions when arcs are not made accessible at the appropriate time. These observations raise the possibility that some of the more advanced forms of candidate list constructions and strategies that abound in tabu search proposals may prove useful in the TSP setting.

[1] There is a lesson here for those who create TSP implementations. As disclosed by later timing tests, the increased time required by the S&C implementation was largely due to an unsophisticated implementation of the don't-look-bits strategy, coupled with a failure to factor the sparsity of the list into the design of the data structure. In short, the difference between a "straightforward" and a carefully designed implementation – even of a supporting routine – can spell big differences for run times in the TSP setting.





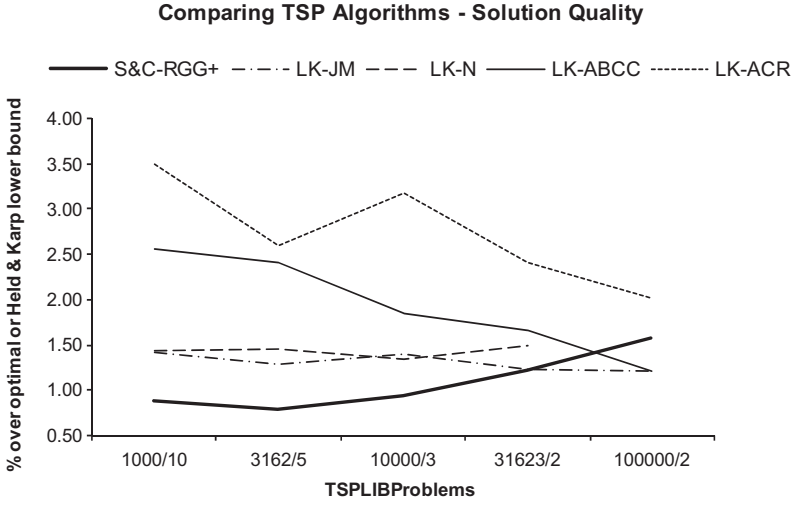


Fig. 4. Basic LK and S&C (values from Table 1).

Finally, the consideration of sophisticated techniques like *caching of distances* and other implementation tricks that proved efficient in LK implementations can likewise be incorporated in the S&C algorithm. (For details on these techniques, we refer to Johnson and McGeoch [19].)

The tables also suggest that LK–JM has some advantages in application to the clustered instances. Tables 3 and 4 show that LK–H achieves higher solution quality but with very heavy computational times. This high computational burden is a serious drawback by limiting the size of the problems the method can

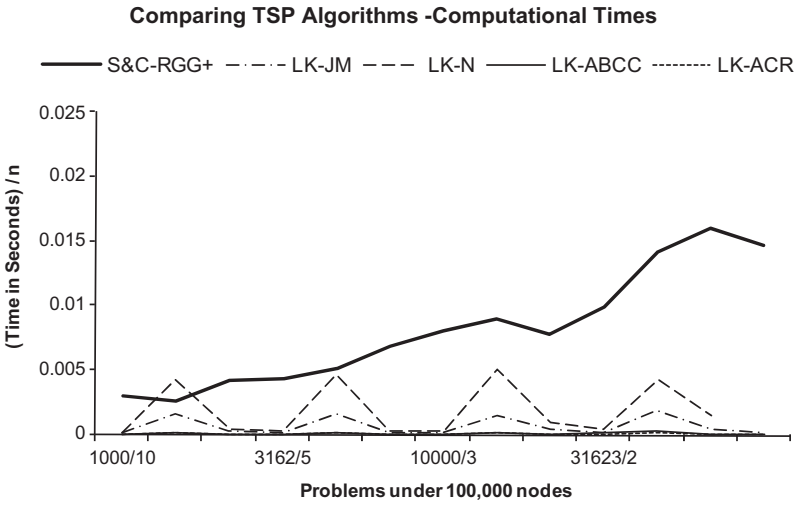


Fig. 5. Basic LK and S&C (values from Table 2).

address. LK–NYYY obtains reasonably good results in this group of algorithms and is able to report solutions to all instances.

We now examine a new development that may hold promise for better implementations.

3.4. Advances in data structures for large STSPs

The problem of data representation is fundamental to the efficiency of search algorithms for the TSP and particularly important for large STSP instances. The nature of these algorithms necessitates the execution of certain basic tour operations involving subpath reversal and traversal. The computational effort that must be devoted to these operations becomes increasingly pronounced with larger problem instances. For example, if the tour is represented as an array (or doubly linked list) of nodes, a subpath reversal takes time $O(n)$, where n is the problem size.

A new data structure called the *k-level satellite tree* [28] has recently been developed for the purpose of minimizing the contribution of tour management toward the overall runtime cost of a given search.

The 2-level tree [9] has for many years been considered the most practical choice for representing the tour, yielding a worst-case cost of $O(\sqrt{n})$ for tour operations. The idea is to divide the tour into roughly $\sqrt{n}$ segments, where each segment is maintained as a doubly linked list and the segments are connected in a doubly linked list.

The k-level satellite tree takes the segmentation idea a step further: the tour is divided into segments containing roughly $n^{1/k}$ nodes each, and the resulting segments are grouped into *parent segments* containing about $n^{1/k}$ segments each. Ultimately, $k-1$ groupings are performed, giving the tree k levels with at least $n^{1/k}$ parents on the top level. The leveraging effect achieved by this grouping of nodes into segments is the same as that achieved by the 2-level tree, except that we no longer assume that "2" is always the appropriate number of levels.

The 2-level tree representation reduces the time complexity of move operations at the price of incurring slightly larger constant costs, also called *overhead*. As one might guess choosing higher values for k (making the tree "taller") will further reduce complexity while driving up overhead. It turns out that when these costs are balanced, the best value for k increases logarithmically with n, but only approximately, since k must be integer. A related property is that, in most cases, the ideal size of a segment will remain the same as problem size increases. This relationship can be shown algebraically under the assumption that a given algorithm will splice the tree during moves about as often as it will traverse parents. Therefore, the key to choosing k is discovering the ideal segment size. This value, however, varies depending on the design and tuning of a given algorithm, and therefore must be determined experimentally.

The *satellite* design [28,29] makes it possible to defray some of the overhead associated with introducing additional levels. A satellite list is similar to a doubly linked list but has the advantage of being symmetric in that an orientation is not inherent. If this structure replaces the doubly linked lists of the traditional 2-level tree, many of the query operations required in the course of a given algorithm may be performed more quickly. This benefit becomes more pronounced when the tree is expanded to include more than two levels.

The best value for k for a k-level satellite tree can be calculated according to the size of the instance and the ideal segment size, which is unique to each algorithm implementation. Hence experimentation to establish best k values as a function of instance size and segment size would be a useful contribution.

Recent experiments show the k-level satellite tree representation to be far more efficient than its predecessors. Particularly outstanding reductions in algorithm running times occur with large problem instances. When the tree is created with k chosen optimally in comparison to $k=2$, the average running time reduction balloons from a modest 7% for 1000 node problems to 27% for 10,000 node problems and to 71% for 100,000 node problems. For these tests, a S&C algorithm implemented with the k-level satellite tree was run on Euclidean instances from the DIMACS Challenge [21].

Fortunately, leading ejection chain algorithms for the TSP are similar enough that they may all make use of the same data structures. Consequently, the improvement offered by the k-level satellite tree may be shared as a common advantage in the same way that the 2-level tree has been incorporated in multiple implementations.

4. Asymmetric TSP

4.1. Ejection chain based algorithms

The Kanellakis–Papadimitriou (KP) heuristic [22] (based on the LK procedure) was the only local search ejection chain algorithm for the ATSP submitted to the "8th DIMACS Implementation Challenge" as reported by Johnson and McGeoch in the Challenge summary chapter [18]. The other two algorithms presented here

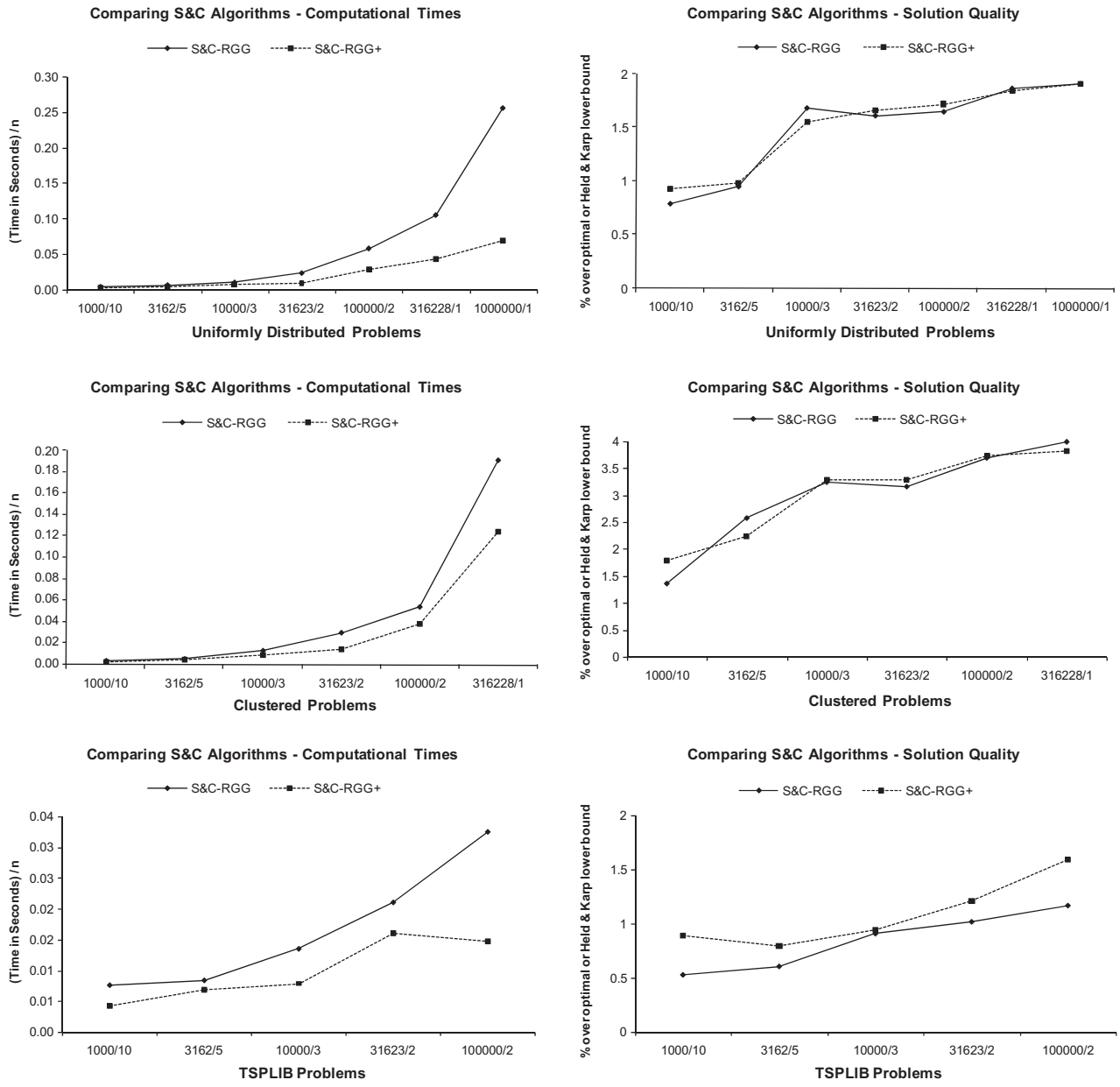


Fig. 6. The effect of don't-look-bits strategy on the S&C algorithm.

were not submitted to the Challenge. These are the ATSP version of the S&C algorithm and a new approach for the ATSP using the doubly-rooted stem-and-cycle reference structure [14].

4.1.1. Kanellakis–Papadimitriou heuristic (KP-JM)

Lin and Kernighan were not concerned with the ATSP when they developed their TSP heuristic in 1973 [23]. LK is based on 2-opt moves which always imply segment reversals that entail exceedingly high computational effort, and hence this method can not be directly applied to the ATSP. A variant of the LK approach presented by Kanellakis and Papadimitriou in 1980 [22] solved this problem by using segment reordering instead of segment reversals (creating and breaking cycles so that the resulting sequence corresponds to a sequence of 3-opt moves). The KP method starts with a variable-depth search based on LK but where the moves performed correspond to k-opt moves for odd values of $k \geqslant 3$. When the variable-depth search fails to improve the solution, the method searches for an improving double-bridge move (with no reversals). Then KP returns to variable-depth search and iterates in this manner until neither of the searches improves the tour. Fig. 7 depicts a 3-opt and a 5-opt move that could be created from the same initial tour. Diagrams A and B of Fig. 7 show the ejection moves involved in a 3-opt move while diagrams C to F are for the 5-opt move. For convenience of illustration we select different nodes to create the initial structures in diagrams A and C for the 3-opt and 5-opt moves, respectively. In both cases the ejection chain starts by selecting a base node t_1 and a node t_3 that identify the arcs (t_1, t_2) and (t_3, t_4) to be dropped and the arc (t_3, t_2) to be added, where t_4 is the endpoint of the path from t_1 to t_4. The result is a disconnected subpath and cycle structure. A 3-opt move is obtained by linking t_4 to a node t_5 in the cycle and deleting one of its adjacent arcs. This rule is the same as that used to create the 3-opt move for the STSP, as illustrated in Fig. 1; however in the case of the ATSP only arcs that preserve the current tour orientation are chosen, thus leaving only one option for the selection of adjacent arc that breaks the cycle in the ATSP as opposed to two

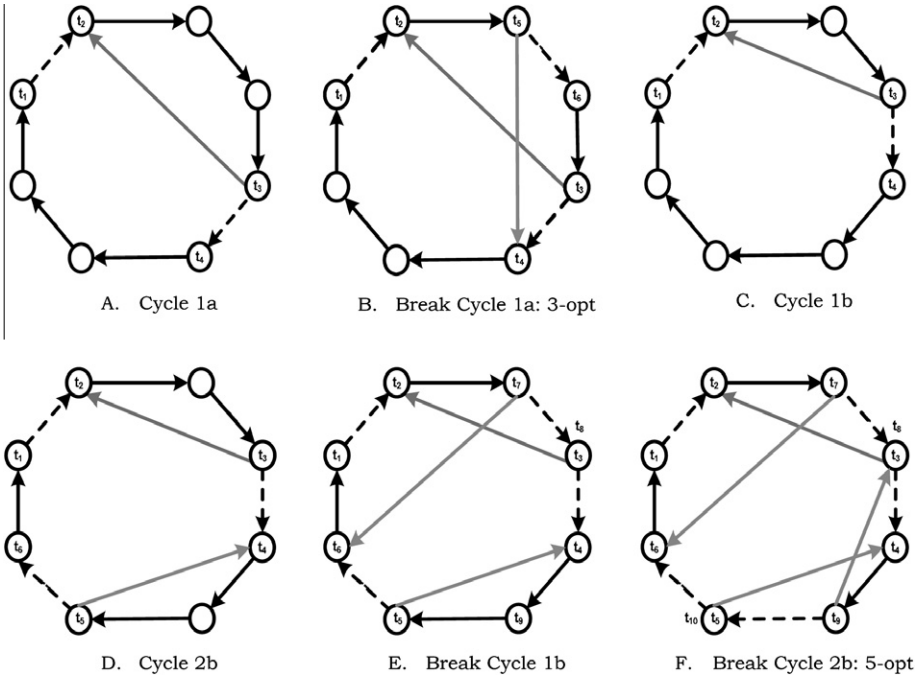


Fig. 7. Possible Lin–Kernighan (KP) ejection chain moves for ATSP.

possible edges in the STSP setting. In the example of Fig. 7, once arc (t_5,t_4) is added and (t_5,t_6) is deleted, a 3-opt move is obtained using a standard LK close up move, which adds arc (t_1,t_6). The 5-opt move operates similarly, except that instead of immediately breaking the current cycle, it first creates another cycle around the current path and disconnects the cycle from path, and then breaks the two cycles in order. In the example, the first and second structures are as shown in diagrams C and D of Fig. 7. Then, the first and second cycles are broken as shown in diagrams E and F. Finally, node t_1 is linked to the other end of the current Hamiltonian path to close up the tour – in the example, t_1 is linked to t_{10} (formerly t_5), thus adding arc (t_1,t_{10}) to form a 5-opt exchange move.

The KP algorithm implementation analyzed in this paper is due to Johnson and McGeoch and described in Cirasella et al. [8]. It takes advantage of the same speedup techniques used in the authors' LK implementation [19], including neighbor lists and the don't-look-bits candidate list strategy. It also uses the dynamic programming approach introduced by Glover [15] to find the best double-bridge 4-opt move in $O(n^2)$ time. It also allows for temporary decreases in the net gains of ejection (add-drop) moves along the chain.

4.1.2. Rego, Glover, and Gamboa stem-and-cycle (S&C-RGG)

This algorithm is based on the S&C ejection chain algorithm for the STSP (S&C-RGG) presented above but ignores moves that generate path reversals. This implementation does not use candidate lists to reduce the neighborhood size, thereby penalizing the computation times as discussed in the following subsection.

4.1.3. Rego, Glover, and Gamboa doubly-rooted S&C (DRS&C-RGG)

The main distinguishing feature of this approach is its use of the doubly-rooted stem-and-cycle reference structure characterized in Glover [14], which generalizes the S&C structure by allowing for additional moves on each level of the ejection chain. The doubly-rooted structure has two forms: a *bicycle* in which the roots are connected by a single path, joining two cycles, and a *tricycle* in which the two roots are connected by three paths, thereby generating three cycles (see Fig. 8 where r_1 and r_2 indicate the roots).

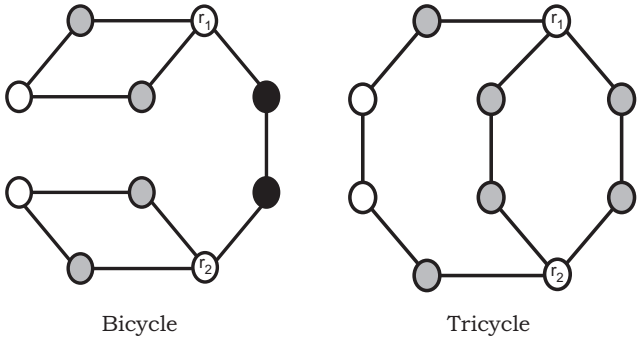


Fig. 8. The doubly-rooted S&C reference structure.

Ejection moves consist of adding a new edge (v_s,v_j) where v_s is a subroot and deleting the edge (v_s,v_r) resulting in node v_j as

Table 5
Solution quality.

Problem size/number of instances – 28 TSPLIB problems								
	100/16		300/5		500/7		Total	
Algorithm	%	NBS	%	NBS	%	NBS	Average	NBS
KP–JM	2.08	8|–	0.52	5|–	1.23	7|4	1.28	20|4
S&C-RGG	1.81	7	1.21	–	2.44	–	1.82	7
DRS&C-RGG	**0.29**	14	**0.02**	4	**1.11**	3	**0.47**	**21**

Table 6
Computational time.

Problem size/number of instances – 28 TSPLIB problems			
	100/16	300/5	500/7
Algorithm	CPU	CPU	CPU
KP-JM	5.73	6.33	72.44
S&C-RGG	2.47	60.64	275.79
DRS&C-RGG	50.50	54.46	1785.10

the new root. The trial moves correspond to those made available from each stem-and-cycle structure obtained by removing one edge linking one root to one of the subroots. Due to the symmetric relationship between the resulting structures, duplicated moves can be eliminated by testing around one of the roots. This leaves us with four trial solutions in the bicycle structure and six trial solutions in the tricycle structure for the STSP. For the ATSP wherein arcs orientation must be preserved the number of possible trial solutions is reduced to one for the bicycle and two for the tricycle structure.

In order to assess the effectiveness of the doubly-rooted S&C neighborhood structure compared to the KP variant, we have adopted for our implementation a similar strategy that alternates between the ejection chain search and the 4-opt double-bridge neighborhood. All the implementations use *Nearest Neighbor* starting tours.

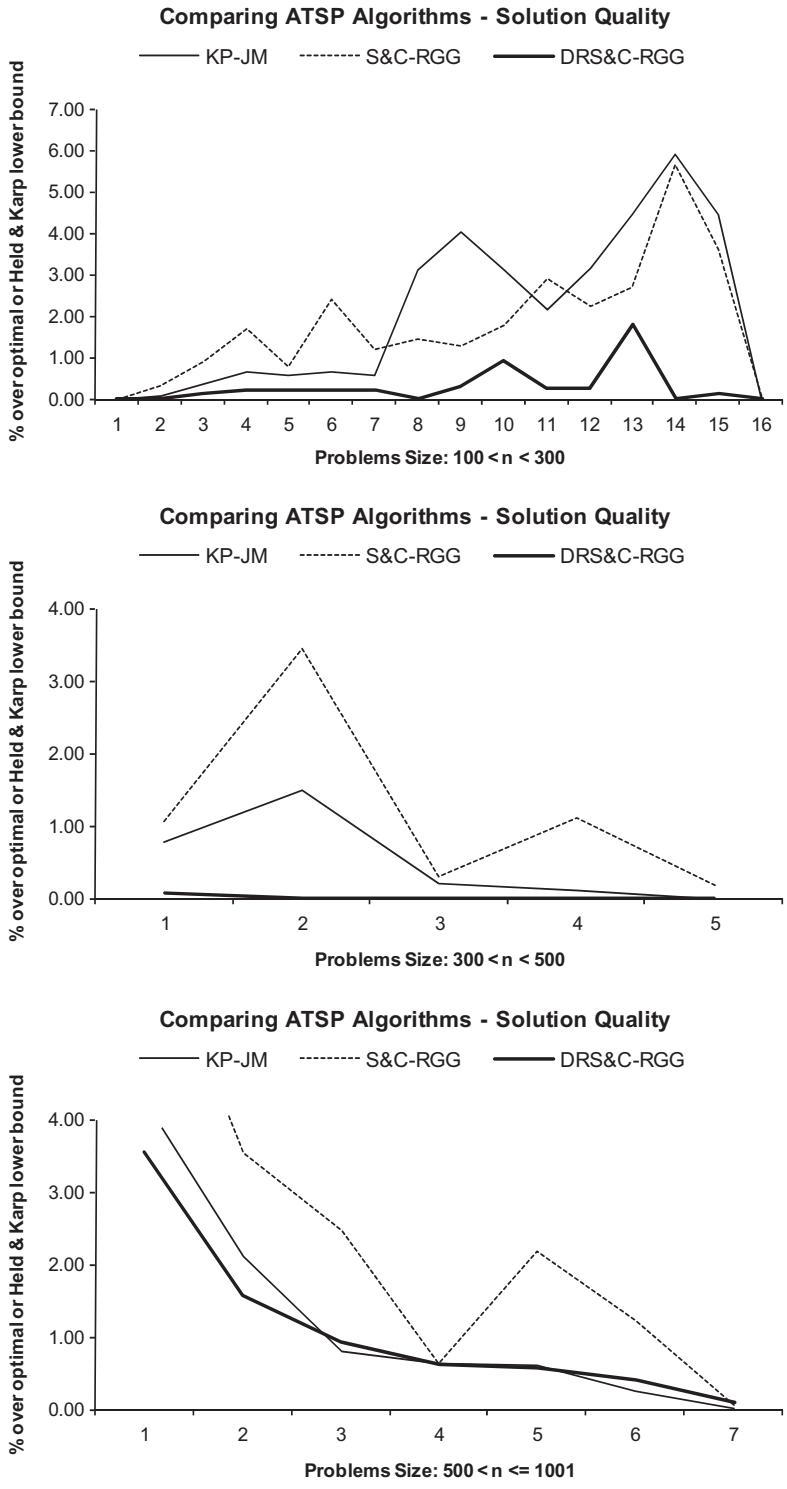


Fig. 9. Comparison of ATSP algorithms: Solution quality.

4.2. Comparative analysis of performance

Tables 1 and 2 summarize the results of the aforementioned algorithms on all the TSP Library [35] asymmetric instances of size ranging from at least 100 to slightly over 1000 nodes. The tables are organized similarly to those presented in Section 3.3 for the symmetric instances. Problems are grouped by size and the $x|y$ notation is used to indicate that the associated algorithm found x better solutions than S&C-RGG and y better solutions than DRS&C-RGG in the corresponding group of problems. The remaining values are averages of percentage gap above the best known solutions (%), the number of instances one algorithm outperforms the others (NBS) when compared individually with either S&C-RGG or DRS&C-RGG, and averages of computational times (in seconds). The values in bold indicate the best averages.

Since our ATSP experiments were not conducted on the same computer considered for the above STSP results (and reported in the Challenge), it is important to explicitly identify the machines used to carry out the tests. The S&C-RGG and DRS&C-RGG algorithms were run on an Intel Centrino 1.5 GHz processor with 128 MB of memory. The results for the KP-JM algorithm were obtained on the TSP Challenge reference machine, a Silicon Graphics Power Challenge with 31 196 MHz MIPS R10000 processors, 1 MB 2nd level caches and 7.6 GB of main memory shared by all processors. To be consistent with the analysis reported for the STSP, we provide normalized running times derived from runs of the standard benchmark code available in the Challenge website [21]. We note that the benchmark code used here corresponds to an implementation of the "Hungarian method", and is different from the Greedy (or Multi-Fragment) geometric benchmark code used above in the normalizations of STSP algorithms. As suggested in [18] such a specialized method for the solution of linear assignment problems is more likely to reflect the pattern of ATSP computations. We encountered relative factors of 1.000, 1.476 and 2.3429 for $n = 100$, 316 and 1000, respectively; hence we found 1.6 a reasonable compromise for the actual factors of the two machines.

It is important also to mention that S&C-RGG and DRS&C-RGG results were obtained in a single run of the algorithms with fixed parameters. By contrast, results for the KP-JM algorithm are averages over at least five runs for each instance as reported in [8]. Also, the S&C-RGG procedure corresponds to a version of the S&C method created by removing features from its STSP version that do not apply to the ATSP, and no effort has been undertaken to create a specialized S&C variant for asymmetric instances to take advantage of the principles that gave rise to the KP variant of the LK method. Similarly, our 4-opt search used in the current DRS&C implementation corresponds to the procedure having a potential $O(n^4)$ complexity that was considered in the original KP algorithm [22], as opposed to Glover's efficient $O(n^2)$ procedure used in its recent KP implementations [8] analyzed here.

From Table 5 we can infer that the S&C-RGG algorithm obtains competitive results but the DRS&C-RGG approach is clearly more effective in producing high quality solutions. Note that besides achieving better solution quality on average, DRS&C-RGG finds a significantly larger number of best solutions across all groups of problems. For the 100-node instances, KP–JM is unable to find even one solution better than DRS&C-RGG, and can only match the best solution found by DRS&C-RGG in two instances out of the 16 in the group. Similar results are observed for the 300-node instances, but for the largest instances of at least 500 nodes KP-JM seems more competitive. Over all the 28 instances tested KP-JM can only find four better solutions than DRS&C-RGG and match on three. The overall percentage deviation average is also considerably better for the doubly-rooted S&C approach, although the computational times are higher as shown in Table 6. The results on the 28 instances of the complete testbed are displayed in the graphics of Figs. 9 and 10.

5. Concluding remarks

The most effective and efficient local search ejection chain algorithms for the TSP, which are examined in this paper, concern six variants of the Lin–Kernighan (LK) approach and two variants of the stem-and-cycle (S&C) ejection chain method for symmetric TSPs, in addition to three generalizations of these methods for the asymmetric version of the problem. We find that the S&C approaches clearly outperform the basic LK implementations in terms of solution quality, although using longer running times to achieve the best solutions.

For symmetric instances, the S&C approach finds better solutions than all (four) of the leading LK variants for about 70% of the problems tested. Conspicuously, the 70% advantage of the S&C approach refers to a comparison with the most effective variant of the LK procedure. The second best variant of this approach is dominated by the S&C approach in approximately 97% of the problems. Some other variants failed to find even a single solution better than the S&C approach over all 59 problems tested.

Similar success was achieved by our doubly-rooted S&C variant applied to the asymmetric setting of the problem. Tests on 28 standard instances revealed 21 best solutions for our doubly-rooted S&C algorithm as opposed to 4 best solutions obtained by a specialized LK variant for these asymmetric instances.

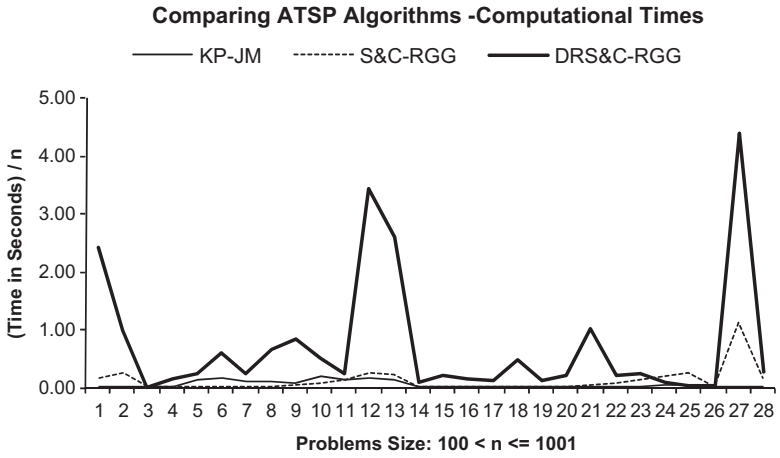


Fig. 10. Comparison of ATSP algorithms: Computation times.

We conjecture that gains in performance from the more advanced ejection chain methods result from the fact that they embrace k-opt moves for $k \geqslant 4$ that are not accessible to the LK approaches. The S&C-based ejection chain methods not only perform better than the local search TSP algorithms based on the LK framework, but also give the overall best solutions when the local search algorithms described here are used as engines for iterated local search heuristics. We anticipate that future gains will result by introducing more effective candidate lists that narrow the neighborhood size without causing solution quality to deteriorate. The advantages of the doubly-rooted S&C method over the single rooted method within the asymmetric TSP setting also raise the possibility that a variant of the doubly-rooted S&C approach will prove highly effective in the symmetric TSP setting as well.

References

[1] D. Applegate, R. Bixby, V. Chvátal, W. Cook, Concorde: a code for solving Traveling Salesman Problems, 1999. <http://www.math.princeton.edu/tsp/concorde.html>.

[2] D. Applegate, R. Bixby, V. Chvátal, W. Cook, Finding tours in TSP, Research Institute for Discrete Mathematics, Universitat Bonn, Bonn, Germany, 99885, 1999.

[3] D. Applegate, W. Cook, A. Rohe, Chained Lin–Kernighan for large traveling salesman problems, INFORMS Journal on Computing 15 (2003) 82–92.

[4] J.L. Bentley, Fast algorithms for geometric traveling salesman problems, ORSA Journal on Computing 4 (1992) 347–411.

[5] P. Briggs, L. Torczon, An efficient implementation for sparse sets, ACM Letters on Programming Languages and Systems 2 (1993) 59–69.

[6] N. Christofides, S. Eilon, Algorithms for large-scale traveling salesman problems, Operations Research Quarterly 23 (1972) 511–518.

[7] M. Chrobak, T. Szymacha, A. Krawczyk, A data structure useful for finding Hamiltonian cycles, Theoretical Computer Science 71 (1990) 419–424.

[8] J. Cirasella, D.S. Johnson, L.A. McGeoch, W. Zhang, The asymmetric traveling salesman problem: algorithms, instance generators and tests, in: Proceedings of the Algorithm Engineering and Experimentation, Third International Workshop, ALENEX 2001, 2001, pp. 32–59.

[9] M.L. Fredman, D.S. Johnson, L.A. McGeoch, G. Ostheimer, Data structures for traveling salesman, Journal of Algorithms 18 (1995) 432–479.

[10] B. Funke, T. Grünert, S. Irnich, A note on single alternating cycle neighborhoods for the TSP, Journal of Heuristics 11 (2005) 135–146.

[11] D. Gamboa, C. Rego, F. Glover, Data structures and ejection chains for solving large-scale traveling salesman problems, European Journal of Operational Research 160 (2005) 154–171.

[12] D. Gamboa, C. Rego, F. Glover, Implementation analysis of efficient heuristic algorithms for the traveling salesman problem, Computers and Operations Research 33 (2006) 1154–1172.

[13] F. Glover, New ejection chain and alternating path methods for traveling salesman problems, Computer Science and Operations Research (1992) 449–509.

[14] F. Glover, Ejection chains, reference structures and alternating path methods for traveling salesman problems, Discrete Applied Mathematics 65 (1996) 223–253.

[15] F. Glover, Finding a best traveling salesman 4-opt move in the same time as a best 2-opt move, Journal of Heuristics 2 (1996) 169–179.

[16] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM Journal on Computing 13 (1984) 338–355.

[17] K. Helsgaun, An effective implementation of the Lin–Kernighan traveling salesman heuristic, European Journal of Operational Research 126 (2000) 106–130.

[18] D.S. Johnson, G. Gutin, L.A. McGeoch, A. Yeo, W. Zhang, A. Zverovitch, Experimental analysis of heuristics for the ATSP, in: G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Boston, 2002, pp. 445–487.

[19] D.S. Johnson, L.A. McGeoch, The traveling salesman problem: a case study in local optimization, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, John Wiley and Sons, Ltd., 1997, pp. 215–310.

[20] D.S. Johnson, L.A. McGeoch, Experimental analysis of heuristics for the STSP, in: G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Boston, 2002, pp. 369–443.

[21] D.S. Johnson, L.A. McGeoch, F. Glover, C. Rego, 8th DIMACS Implementation Challenge: The Traveling Salesman Problem, 2000. <http://www.research.att.com/~dsj/chtsp/>.

[22] P.C. Kanellakis, C.H. Papadimitriou, Local search for the asymmetric traveling salesman problem, Operations Research 28 (1980) 1086–1099.

[23] S. Lin, B. Kernighan, An effective heuristic algorithm for the traveling salesman problem, Operations Research 21 (1973) 498–516.

[24] O.C. Martin, S.W. Otto, E.W. Felten, Large-step Markov chains for the traveling salesman problem, Complex Systems 5 (1991) 299–326.

[25] O.C. Martin, S.W. Otto, E.W. Felten, Large-step Markov chains for the TSP incorporating local search heuristics, Operations Research Letters 11 (1992) 219–224.

[26] D.L. Miller, J.F. Pekny, A staged primal–dual algorithm for perfect B-matching with edge capacities, ORSA Journal on Computing 7 (1995) 298–320.

[27] D. Neto, Efficient Cluster Compensation for Lin–Kernighan Heuristics, Department of Computer Science, University of Toronto, 1999.

[28] C. Osterman, C. Rego, The Satellite List and New Data Structures for Symmetric Traveling Salesman Problems, University of Mississippi, HCES-03-06, 2004.

[29] C. Osterman, C. Rego, D. Gamboa, The satellite list: a reversible doubly-linked list, in: Proceedings of the 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2005), Coimbra, Portugal, 2005, pp. 542–546.

[30] E. Pesch, Learning in Automated Manufacturing: A Local Search Approach (Production and Logistics), Physica-Verlag HD, 1994.

[31] E. Pesch, F. Glover, TSP ejection chains, Discrete Applied Mathematics 76 (1997) 165–181.

[32] C. Rego, Relaxed tours and path ejections for the traveling salesman problem, European Journal of Operational Research 106 (1998) 522–538.

[33] C. Rego, F. Glover, Local search and metaheuristics, in: G. Gutin, A. Punnen (Eds.), The Traveling Salesman Problem and Its Variations, Kluwer Academic Publishers, Dordrecht, 2002, pp. 309–368.

[34] C. Rego, F. Glover, Ejection chain and filter-and-fan methods in combinatorial optimization, Annals of Operations Research 175 (2010) 77–105.

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