A dual local search framework for combinatorial optimization problems with TSP application

Citation for published version:
Ouenniche, J, Ramaswamy, PK & Gendreau, M 2017, 'A dual local search framework for combinatorial optimization problems with TSP application' Journal of the Operational Research Society, vol. 68, no. 11, pp. 1377-1398. DOI: 10.1057/s41274-016-0173-4

Digital Object Identifier (DOI):
10.1057/s41274-016-0173-4

Published In:
Journal of the Operational Research Society

A Dual Local Search Framework for Combinatorial Optimization Problems with TSP Application

Jamal Ouenniche[1*], Prasanna K. Ramaswamy[1] and Michel Gendreau[2]

[1]Business School, University of Edinburgh, 29 Buccleuch Place, Edinburgh EH8 9JS, UK;

[2]CIRRELT and Ecole Polytechnique de Montreal, Montreal, Canada

Manuscript submitted to Journal of the Operational Research Society (JORS) and accepted for publication. It is currently online (DOI: 10.1057/s41274-016-0173-4)

Abstract: In practice, solving realistically sized combinatorial optimization problems (COPs) to optimality is often too time-consuming to be affordable; therefore, heuristics are typically implemented within most applications software. A specific category of heuristics has attracted considerable attention, namely, local search methods. Most local search methods are primal in nature; that is, they start the search with a feasible solution and explore the feasible space for better feasible solutions. In this research, we propose a dual local search method and customise it to solve the traveling salesman problem (TSP); that is, a search method that starts with an infeasible solution, explores the dual space – each time reducing infeasibility, and lands in the primal space to deliver a feasible solution. The proposed design aims to replicate the designs of optimal solution methodologies in a heuristic way. To be more specific, we solve a combinatorial relaxation of a TSP formulation, design a neighborhood structure to repair such an infeasible starting solution, and improve components of intermediate dual solutions locally. Sample-based evidence along with statistically significant t-tests support the superiority of this dual design compared to its primal design counterpart.

Keywords: Dual Local Search, Relaxation, Optimization, Travelling Salesman, Routing and Scheduling

1. Introduction

Nowadays, operational research is a well-established discipline with applications in very many different areas both in the public and the private sectors. One application area that has attracted the attention of both academics and practitioners for several decades is the transportation of people and merchandise. Regardless of whether transportation services are designed for people or merchandise and whether they are provided by public or private entities, transport is both a major economic driver and a major cost factor. In fact, in the United Kingdom, about 10.25% to 12.63% of national expenditure is accounted for by transportation between 2008 and 2014 (HM Treasury, 2014) – these figures highlight the importance of transportation in the economy. From an optimisation perspective, transportation accounts for some of the most challenging combinatorial optimisation problems (COPs) such as the traveling salesman problem (TSP), the Chinese postman problem, and their generalizations to incorporate more realistic settings as encountered in real-life applications. Both optimal and heuristic approaches have been proposed to address such COPs. In practice, however, either heuristics or hybrids that combine optimal and heuristic methodologies are typically implemented within most applications software to realistically manage the computational requirements of the sizes of the instances practitioners have to solve. A specific category of heuristics has attracted considerable attention, namely, local search methods. Most local search methods – whether classical local search or metaheuristics – are primal in nature; that is, they start the search with a feasible solution and explore the feasible space for better feasible solutions. In this research, we propose a dual local search (DLS) method and customise it to solve the TSP. Recall that the TSP is concerned with determining a minimal cost Hamiltonian cycle; that is, a minimum cost route for a single uncapacitated vehicle that starts at the depot, visits each customer once and only once, and returns to the depot. In sum, the proposed DLS starts with an infeasible solution, explores the dual space – each time reducing infeasibility, and lands in the primal space to deliver a feasible solution, which could be improved further. As the proposed design aims to replicate the designs of optimal solution methodologies in a heuristic way, the components of intermediate dual solutions are locally improved using an equivalent of primal local search that we refer to as Type II moves. Conceptually, Type II moves are the means by which more children of a node in a branch-and-bound tree are explored – see section 3.2 for details. To be more specific, we solve a combinatorial relaxation of a TSP formulation and design a neighborhood structure defined by what we refer to as Type I moves to repair such an infeasible starting solution and locally improve its components using classical local improvement moves that we refer to as Type II moves.

Sample-based performance along with statistically significant t-tests support the superiority of DLS compared to its primal design counterpart. Thus, the proposed dual design offers a viable alternative to primal search designs.

The remainder of this paper is organised as follows. In section 2, we provide the landscape of research on solution approaches and methods for the TSP and position our contribution with respect to the literature. In section 3, we present our dual local search framework along with its implementation decisions, discuss the rationale behind the proposed design, provide a comparative analysis with branch-and-bound (B&B) algorithms, and summarise some theoretical insights. In section 4, we provide statistical evidence that dual local search outperforms primal local search and discuss the performance of the different implementation schema proposed. Finally, section 5 concludes the paper.

2. Solution Approaches and Methods for TSP

In this section, we present the outcome of our literature survey on optimal and heuristic solution approaches and methods designed to address the TSP in the form of a classification (see sections 2.1 and 2.2). Then, we position our contribution with respect to the literature after introducing new classification criteria.

2.1 Optimal Approaches and Methods

The design of optimal solution procedures, also referred to as exact methods or algorithms, for the TSP dates back to the 1950s. Optimal solution procedures can be divided into three main categories; namely, branch-and-bound (B&B) algorithms, cutting plane algorithms, and their hybrids such as branch-and-cut (B&C) algorithms.

B&B is a generic optimal design and as such its implementation for solving a particular COP such as the TSP requires customisation; in sum, a number of decisions have to be made such as the choice of the bounding scheme to use and the choice of the branching rule to use. The design of bounding schema for B&B algorithms is of prime importance as the computational time requirements strongly depend on the quality of the bounding scheme. Recall that a bounding scheme consists of a couple of bounds; namely, a primal bound and a dual bound, where the primal bound corresponds to the objective function value of the best feasible solution found so far during the course of the algorithm, and the dual bound corresponds to the objective function value of the best solution found so far during the course of the algorithm to a chosen relaxation of the problem. For the TSP, the primal bound is typically computed using one of the construction heuristics proposed in the literature, which could be tightened using an improvement heuristic such as classical local search or metaheuristics. For example, Padberg and Rinaldi (1987, 1991) use Lin Kernighan type of heuristic, whereas Miller and Pekny (1992), Carpaneto et al. (1995), and Turkensteen et al. (2006) use Patching heuristics. As to the dual bounds for the TSP, several types of relaxations have been used in the literature such as Assignment Problem-based relaxations (e.g., Eastman, 1958; Shapiro, 1966; Bellmore and Malone, 1971; Carpaneto and Toth, 1980; Balas and Christofides, 1981; Germs et al., 2012), 2-Matching Problem-based Relaxations (e.g., Bellmore and Malone, 1971), 1-Tree Problem-based Relaxations (e.g., Held and Karp, 1971; Helbig et al, 1974; Volgenant and Jonker, 1982; Gavish and Srikanth, 1983), and Shortest n-Arc Path Problem-based Relaxations (e.g., Houck et al., 1980). On the other hand, with respect to the choice of the branching rule, which often depends on the type of relaxation and the sub-tour breaking constraints used, several branching rules have been proposed; for example, within a B&B framework that makes use of an Assignment Problem-based relaxation, Eastman (1958), Shapiro (1966), and Bellmore and Malone (1971) used branching rules that are based on the sub-tour elimination constraints proposed by Dantzig et al. (1954) while Murty (1968), Bellmore and Malone (1971), and Carpaneto and Toth (1980) used branching rules that are based on the sub-tour elimination constraints commonly referred to as the connectivity constraints. Note that these two categories of branching rules both exploit the TSP structure and are based on sub-tour elimination constraints; however, the second category generates more tightly constrained sub-problems, as proved by Bellmore and Malone (1971), which is a desirable feature of branching rules. For a comprehensive coverage of the main branching rules for the TSP, the reader is referred to Lawler et al. (1985).

Cutting plane algorithms have also been proposed for the TSP. Cutting plane algorithms are also generic designs and their customisation for a specific problem requires an understanding of the polyhedral structure of the problem to design effective cuts. Examples of cuts for the TSP include Comb inequalities (Chvatal, 1973), Brush inequalities (Naddef and Rinaldi, 1991), Star inequalities (Fleischmann, 1988), Path inequalities (Cornuejols et al., 1985), Binested inequalities (Naddef, 1992), Clique Tree inequalities (Grötschel and Pulleyblank, 1986), Bipartition inequalities (Boyd and Cunningham, 1991), Ladder inequalities (Boyd and Cunningham, 1991), and Chain inequalities (Padberg and Hong, 1980).

In order to further strengthen B&B and cutting plane algorithms, hybrids have been proposed whereby one would typically use cuts within a B&B framework resulting in B&C algorithms (e.g., Crowder and Padberg, 1980; Padberg and Rinaldi, 1991; Fischetti and Toth, 1997; Fischetti et al., 2003; Applegate et al., 2007). To the best of our knowledge, the B&C algorithm proposed by Applegate et al. (2007), which is commonly referred to as "concorde" code, remains the state-of-the-art code for the TSP.

Although optimal methodologies guarantee the delivery of an optimal solution, for large scale TSPs either heuristics or hybrids that combine optimal and heuristic methodologies are typically used in practice. In the next sub-section, we shall provide an outlook of heuristics for TSP.

2.2. Heuristic Approaches and Methods

The design of heuristics – sometimes referred to as approximate solution procedures – for the TSP dates back to the 1960s. Recall that heuristics are solution procedures that deliver a feasible solution to a problem, but without any guarantee of optimality. Heuristics for the TSP could be divided into two main categories depending on whether they are construction procedures or improvement procedures.

For the TSP, several construction procedures have been proposed including the Nearest Neighbor procedure (Rosenkrantz et al., 1977), the Clark and Wright Savings procedures (Clark and Wright, 1964; for Complexity see Ong, 1981), Insertion procedures such as Arbitrary, Farthest, Nearest, and Cheapest insertions (Rosenkrantz et al., 1977), the Minimal Spanning Tree procedure (Kim, 1975), Christofides' heuristic (Christofides, 1976), the Partitioning procedure (Karp, 1977), the Nearest Merger procedure (Rosenkrantz et al., 1977), the Patching algorithm (Karp, 1979), the Modified Patching algorithm (Glover et al., 2001), the Contract-or-Patch heuristics (Glover et al., 2001; Goldengorin et al. 2006; Gutin and Zverovich, 2005), and GENI (Gendreau et al., 1992).

On the other hand, improvement procedures could be further divided into two sub-categories; namely, classical local search methods and metaheuristics. Note that, as compared to construction methods which are problem-specific, improvement methods are rather generic frameworks that need customisation. Well-known classical local search methods for the TSP include 2-Opt and 3-Opt heuristics (Lin, 1965), r-Opt heuristic (Lin and Kernighan, 1973), and Or-Opt heuristic (Or, 1976). By design, classical local search methods typically get stuck in a local optimum. In order to address this design issue, metaheuristics have been proposed. Recall that metaheuristics are generic solution procedures equipped with strategies or mechanisms for avoiding getting and remaining stuck in local optima. Note that most metaheuristics were inspired by natural phenomena and designed as imitations of such phenomena. Metaheuristics for the TSP could be divided into several sub-categories depending on the chosen classification criterion or criteria. For example, one might divide metaheuristics into two categories depending on whether they are pure or hybrid. Examples of pure metaheuristics for the TSP include Simulated Annealing (Kirkpatrick et al., 1983; Malek et al., 1989), Tabu Search (Malek, 1988; Malek et al., 1989; Tsubakitani and Evans, 1998a), Guided Local Search (Voudouris and Tsang, 1999), Jump Search (Tsubakitani and Evans, 1998b), Randomized Priority Search (DePuy, Moraga and Whitehouse, 2005), Greedy Heuristic with Regret (Hassin and Keinan, 2008), Genetic Algorithms (Jayalakshmi et al., 2001; Tsai et al., 2003; Albayrak and Allahverdi, 2011; Nagata and Soler, 2012), Evolutionary Algorithms (Liao et al., 2012), Ant Colony Optimization (Dorigo and Gambardella, 1997), Artificial Neural Networks (Leung et al., 2004; Li et al., 2009), Water Drops Algorithm (Alijla et al., 2014), Discrete Firefly Algorithm (Jati et al., 2013), Invasive Weed Optimization (Zhou et al., 2015), Gravitational Search (Dowlatshahi et al., 2014), and Membrane Algorithms (He et al., 2014). Examples of hybrid metaheuristics include Simulated Annealing with Learning (Lo and Hsu, 1998), Genetic Algorithm with Learning (Liu and Zeng, 2009), Self-Organizing Neural Networks and Immune System (Masutti and de Castro, 2009), Genetic Algorithm and Local Search (Albayrak and Allahverdi, 2011), Genetic Algorithm and Ant

Colony Optimization (Dong at al., 2012), Honey Bees Mating and GRASP (Marinakis et al., 2011), and Particle Swarm Optimization and Ant Colony Optimization (Elloumi et al., 2014).

One might also divide metaheuristics into two categories depending on whether they are individual-based or population-based. In this paper, an individual-based metaheuristic refers to a search method that starts with a single or individual solution, often referred to as the seed, and explores its neighborhood in search for a better solution to become the seed – this process is repeated until a stopping condition is met. On the other hand, a population-based metaheuristic refers to a search method that starts with a set of solutions, often referred to as a population, a colony or a swarm, that communicate through a variety of mechanisms to exchange information about solution features to generate a new set of solutions of a better quality. Examples of individual-based metaheuristics include Simulated Annealing (e.g., Kirkpatrick et al., 1983), Tabu Search (Malek, 1988; Malek et al., 1989; Tsubakitani and Evans, 1998a) and Guided Local Search (Voudouris and Tsang, 1999). On the other hand, examples of population-based metaheuristics include Genetic Algorithms (e.g., Jayalakshmi et al., 2001; Tsai et al., 2003; Albayrak and Allahverdi, 2011; Nagata and Soler, 2012), Evolutionary Algorithm (Liao et al., 2012), and Artificial Neural Networks (Leung et al., 2004).

In this paper, we propose a classification criterion of the literature on metaheuristics that is more relevant to our research; that is, primal metaheuristics, dual metaheuristics, and primal-dual metaheuristics. In this paper, a primal metaheuristic refers to a search method that starts the search from within the feasible space and explores it until a stopping condition is met without allowing the method to leave the feasible space. A dual metaheuristic refers to a search method that starts the search from within the infeasible or dual space, explores the dual space – each time reducing infeasibility, and lands in the primal space to deliver a feasible solution or a set of feasible solutions depending on whether the search method is individual-based or population-based. Finally, a primal-dual metaheuristic refers to a search method that could start the search either from within the feasible space or the dual space and during the search for an optimal or near optimal solution it is allowed to explore both spaces. Examples of primal metaheuristics include the papers mentioned within the above classifications. As to dual and primal-dual metaheuristics, to the best of our knowledge, there are no published journal articles. Thus, as far as the TSP is concerned, this paper is a first contribution to the class of dual search methods. In the next section, we shall present the main elements of such contribution.

3. A Dual Local Search Framework

In this section, we present our dual local search (DLS) framework and discuss the rationale behind the proposed design along with its implementation decisions (see section 3.1), and provide a comparative analysis with branch-and-bound (B&B) algorithms and summarise some theoretical insights (see section 3.2).

3.1 The General Framework, Its Underlying Rationale and Its Implementation Decisions

As suggested by our classification of the literature on search methods into primal, dual, and primal-dual heuristics, and the scarcity of contributions within the sub-category of dual heuristics, we fill such a gap by proposing a dual search heuristic framework and discuss the underlying rationale along with its implementation decisions – see Figure 2 for pseud-code. The proposed dual search algorithm customised for the TSP is summarised hereafter.

As compared to primal local search (PLS)-based heuristics' designs – whether classical local search or metaheuristics – we integrate design features of optimal algorithms. To be more specific, we start the search with an infeasible solution; namely, the solution of a relaxation of the problem under consideration. In this paper, we use an assignment problem-based relaxation to generate an initial dual solution, say $\{S_k, k = 1, \dots, N_{sbt}\}$, and progress towards the feasible space – each time reducing infeasibility, where S_k denote the k^{th} sub-tour and N_{sbt} denote the total number of sub-tours in the seed.

The search progress towards the feasible space requires a repairing mechanism or set of moves, referred to in this paper as Type I moves, which define a relevant neighborhood structure for our application; namely, the TSP. We refer to this neighborhood structure as the Dual s-subtour-r-edge-exchange Neighborhood. Moves in this neighborhood consist of $(s + r_1 + \dots + r_s)$-tuples, where the first s entries correspond to the s sub-tours to be merged ($s \leq N_{sbt}$), the next r_1 entries correspond to the edges of the first sub-tour to be broken, say S_1, the following r_2 entries correspond to the edges of the second sub-tour to be broken, say S_2, and so on until the last r_s entries that correspond to the edges of the last sub-tour to be broken, say S_s. In sum, a move could be formally represented as follows:

$$(S_1, \dots, S_s, e_{S_1}^1, \dots, e_{S_1}^{r_1}, \dots, e_{S_s}^1, \dots, e_{S_s}^{r_s}),$$

where $S_1, \dots, S_s$ are the sub-tours to be merged, $e_{S_1}^{r_1}, \dots, e_{S_s}^1$ are the edges of sub-tour S_1 to be broken, and $e_{S_s}^1, \dots, e_{S_s}^{r_s}$ are the edges of sub-tour S_s to be broken. A couple of decisions need to be made to fully operationalize this neighborhood. These decisions are concerned with addressing the following questions: How to choose the s sub-tours to be merged? and How to merge them? To address the first question, in our empirical experiments, we tested three criteria for selecting sub-tours to merge; namely, the farthest distance between sub-tours; the nearest distance between sub-tours; and the cheapest cost of merger of sub-tours. As to the second question, sub-tours are merged or connected in the best possible way to form a larger and cheaper sub-tour. A graphical example is provided in Figure 1 to illustrate Type I moves, where the dual solution consists of two sub-tours which are merged by breaking one edge in each sub-tour and connecting the resulting paths to form a single tour.

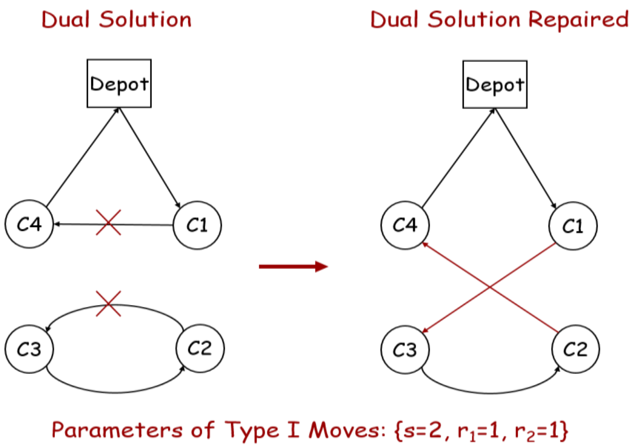

Figure 1: An Illustrative Example of Type I Moves

The above described Dual s-subtour-r-edge-exchange Neighborhood is a parameterized neighborhood structure, which allows one to control the rate at which the process converges to a feasible solution, on one hand, and to intensify or diversify the search depending on whether its parameters are set to relatively low or relatively high values, on the other hand. In fact, the number of iterations required for this dual search framework to converge to a feasible solution depends on the number of sub-tours in the solution to the relevant relaxation of the problem formulation (e.g., assignment-based relaxation of a TSP formulation), say N_{sbt}^{0} , and one of the parameters of the proposed dual neighborhood; i.e., the number of sub-tours to merge at a time, s. Let N_{sbt}^{k} denotes the number of sub-tours at iteration k. Then, the number of possible ways to choose s sub-tours to be merged amongst N_{sbt}^{k} is:

$$\binom{N_{sbt}^{k}}{s} = \frac{N_{sbt}^{k}!}{s!\left(N_{sbt}^{k} - s\right)!}$$

Obviously, the number of iterations, say K, required for this dual search framework to converge to a feasible solution is upper bounded by the number of sub-tours in the solution to the relevant relaxation of the problem formulation (e.g., assignment-based relaxation of a TSP formulation), N_{sbt}^{0}:

$$K \leq N_{sbt}^{0}.$$

However, a tighter upper bound, $K(N_{sbt}^{0}, s^{0})$, that takes account of the initial choice of s^{k}, say s^{0}, could be obtained as follows, where s^{k} denotes the number of sub-tours to merge at iteration k:

$$K(N_{sbt}^{0}, s^{0}) = \left\lceil \frac{N_{sbt}^{0} - 1}{s^{0} - 1} \right\rceil.$$

Note that, in a static implementation where the parameters of the algorithm do not change (e.g., s^{k}), depending on the values of N_{sbt}^{0} and s^{0}, the value of s^{k} might have to be changed just before the start of the last iteration. To be more specific, one would use s^{0} up to iteration $\lfloor (N_{sbt}^{0} - 1)/(s^{0} - 1) \rfloor$ and then change s^{K} to N_{sbt}^{K-1}. Notice that, for a given value of N_{sbt}^{0}, $K(N_{sbt}^{0}, s^{0})$ decreases as s^{0} increases. Therefore, from a computational perspective, a trade-off should be made between choosing relatively high values for parameter s^{0}, which would require a relatively small number of iterations K to converge but would require exploring a relatively large number of possibilities for breaking s^{0} sub-tours and merging them, or choosing relatively low values for parameter s^{0}, which would require a relatively large number of iterations K to converge but would require exploring a relatively small number of possibilities for breaking s^{0} sub-tours and merging them.

Recall that the basic move of the proposed dual neighborhood structure (i.e., Type I moves) consists of breaking some edges of some sub-tours and connecting such sub-tours in the best possible way to form a larger one. Type I moves lead to successive partial solutions that have many similarities. In order to diversify the structure of our solutions, we use a second type of moves to perturb or locally improve their components using classical local improvement moves that we refer to in this paper as Type II moves. In our experiments, we used 2-opt and 3-opt moves and the improvement moves used in GENI (Gendreau et al., 1992) – referred to in this paper as US moves – as Type II moves.

Initialization Step

Choose and solve an appropriate relaxation and use its (typically) infeasible solution to initialize the seed, say x_0, and record the corresponding objective function value, say $z(x_0)$;

Choose the neighborhood structure to use for repairing the dual solution; that is, Type I moves;

Choose the criterion for selecting sub-tours to merge;

Choose the neighborhood structure to use for improving locally the components of intermediate dual solutions; that is, Type II moves;

Iterative Step

REPEAT until stopping condition = true // (e.g., x_0 is feasible)

  Search the neighborhood of x_0, denoted $N(x_0)$, for the "best" neighbour x with respect to the sub-tours selection criterion, perform the merge operation, improve the resulting larger sub-tour using Type II moves, and update the seed; that is, set $x = x_0$;

END REPEAT

Figure 2: Pseudo-Code of Dual Local Search

In order to reduce the computational requirements of the proposed framework, one might call upon a local search mechanism based on Type II moves according to a proportion of use, say p $(0 \leq p \leq 1)$, which could be either static or dynamic and could be implemented in a deterministic fashion (e.g., using deterministic decision rule) or a stochastic fashion (e.g., using probabilistic decision rule). In the deterministic and static scheme, one would fix to a pre-specified value the proportion of use of local improvement for the entire search. The feasible values for p should satisfy the following conditions: $K(N_{sbt}^0, s^0)\ mod\ p \equiv 0$ and $p < K(N_{sbt}^0, s^0)$. On the other hand, in the deterministic and dynamic scheme, the proportion of use of local improvement varies during the course of the search according to a deterministic decision rule. The feasible values for p should satisfy the following conditions: $K(N_{sbt}^0, s^0)\ mod\ p \equiv 0$ and $p < K(N_{sbt}^0, s^0)$; let m denote the number of values of p that satisfy these conditions, $p_1,\ldots,p_m$ denote such feasible values and π_k denote the value of p at iteration k. In our numerical experiment, we used the following deterministic decision rule: set π_0 to p_1 and update π_k to p_2 after p_1 iterations, then to p_3 after p_2 iterations and so on until its value is updated to p_m. Note that, when $\sum_{j=1}^{m} p_j < K(N_{sbt}^0, s^0)$, this decision rule is implemented in a cyclical manner. Note also that, in case the last value of π_k used does not allow for improving the last tour, such final tour is exceptionally improved. Finally, in the stochastic scheme, at each iteration one would generate a random number between 0 and 1 and if such number is greater than a pre-specified threshold (e.g., 0.5, 0.7, 0.9), then local improvement is called upon.

3.2 Comparative Analysis with B&B and Some Theoretical Insights

In this section, we perform a conceptual comparative analysis with B&B, and summarise some theoretical insights – see Table 1 for a snapshot summary. For ease of exposition, we shall present the comparative analysis for a specific B&B design; namely, B&B with Bellmore and Malone (1971) branching rule. In sum, we shall address the following question: How the use of DLS with Type I and Type II moves compares to B&B with Bellmore and Malone (1971) branching rule? Before proceeding with the comparative analysis, we would like to remind the reader that, at each node of the B&B tree, the Bellmore and Malone (1971) branching rule consists of generating several successors where the first successor excludes a first arc from the arc set of a minimum cardinality sub-tour of the solution of the parent node, the second successor includes the previously excluded arc and excludes a new arc, the third successor includes the previously excluded arcs and excludes a new arc, and so on until all arcs are included but one. Note that including (respectively, excluding) an arc consists of setting the decision variable associated with that arc to 1 (respectively, 0) before solving the assignment problem at the successor nodes.

Hereafter, we shall break the comparative analysis of DLS with Type I and Type II moves and B&B with Bellmore and Malone (1971) branching rule into several points according to their main design features to highlight their similarities and differences. First, B&B breaks one sub-tour at a time, whereas DLS breaks two or more sub-tours at a time. Second, B&B

examines all possible ways of breaking one sub-tour – one edge at a time, as compared to DLS where Type I moves allow one to examine all possible ways of breaking two or more sub-tours (e.g., breaking one or more edges in each sub-tour) and connecting them. Third, within a B&B framework, at each level of the B&B tree – except level 0, the first branch excludes one edge of one of the sub-tours and the following branches each includes the edges previously excluded, excludes a new edge, and keeps all the remaining edges free except those fixed at higher levels of the tree, if any. On the other hand, within the proposed DLS framework, at each iteration Type I moves exclude two or more edges from two or more sub-tours, respectively, and connects such sub-tours. Note that this type of moves partially preserves the structure of the current infeasible solution in that the sequence(s) that have not been affected remain unchanged; in other words, Type I moves include in the next solution all the edges in the sub-tours that have not been excluded. Note also that Type I moves do not allow one to explore the "equivalent" of as many branches as those explored within a B&B framework. In order to explore the "equivalent" of those branches, we use Type II moves which consist of edge exchanges of the newly formed sub-tour. Fourth, at each node of the B&B tree, a re-optimization process is invoked, which could lead to a new infeasible or feasible solution, whereas at each iteration of the DLS, a "restricted" optimization process is invoked, which could lead to a new infeasible or feasible solution, but with potentially more similarity to the solution of the previous iteration as compared to B&B where the optimization process refers to the way the broken sub-tours could possibly be connected or equivalently the way to choose the sets of edges to include and exclude in the B&B terminology. Note that the restrictive nature of the optimization process depends on the values of the parameters chosen; in sum, the higher the values of the parameters s and r's, the less restrictive is the optimization process. Last, but not least, within a B&B framework, a new branch would not necessarily lead to a reduction in the number of sub-tours, whereas in DLS each dual s-subtour-r-edge-exchange neighborhood move of type I systematically reduces the number of sub-tours by one or more. Therefore, convergence to a feasible solution is guaranteed in a finite number of iterations.

B&B	DLS
Break one sub-tour at a time	Break two or more sub-tours at a time
Examine all possible ways of excluding or including one edge at a time of one sub-tour	Examine all possible ways of breaking two or more sub-tours (e.g., breaking one or more edges in each sub-tour) and connecting them
At each level of the B&B tree – except level 0, exclude (resp., include) one edge of one of the sub-tours & keep all the remaining edges free except those fixed at higher levels of the tree, if any	Exclude two edges, one from each sub-tour, and include in the next solution all the remaining edges in the sub-tours
At each node of the tree, a re-optimization process is invoked, which could lead to a new infeasible or feasible solution	At each node of the tree, a "restricted" optimization process is invoked, which could lead to a new infeasible or feasible solution, but with potentially more similarity to the solution of the parent node as compared to B&B. In order to diversity in terms of structure of partial solutions and explore more nodes as done in B&B, we use a second type of moves similar in spirit to branch exchange improvement
A new branch would not necessarily lead to a reduction in the number of sub-tours	Each dual s-subtour-r-edge-exchange neighborhood move systematically reduces the number of sub-tours by one or more

Table 1: Comparative Analysis between B&B and DLS

To conclude this section, we hereafter summarise a couple of important theoretical insights. The first theoretical insight is summarized in the following axiom:

Axiom: The size of the primal search space is initial solution-independent as compared to the size of the dual search space. To be more specific, the size of the primal search space is the same regardless of the construction method used for initializing PLS. However, in the dual case, the size of the search space depends on the type of relaxation used.

This axiom suggests that within a DLS framework the quality of the dual solution the search starts with would have an impact on the search process and where it would potentially land in the primal space. Furthermore, the size of the dual search space depends on the starting solution or equivalently the type of relaxation used. For example, if one uses a linear programming relaxation as compared to an assignment problem-based relaxation, one would have to explore a much larger dual search space. As to comparing the sizes of the primal search space and the dual search space, the following proposition summarizes the second theoretical insight.

Proposition: The dual search space is much larger than the primal search space.

Proof: The number of feasible solutions to a TSP of size n is $(n-1)!$ Notice that, within a PLS framework, only a subset of these solutions are visited and the cardinality of such subset depends on the type of neighborhood used and its parameters, if any. Furthermore, the cardinality of such subset is independent of the solution that PLS starts with. Note that the number of dual solutions could potentially be infinite. However, within a dual local search (DLS) framework initialized with an assignment problem-based relaxation, and assuming that during the search solution components (i.e., x_{ij}s) remain binary, the number of dual solutions that could potentially be reached from the initial dual solution becomes finite but remains larger than the $(n-1)!$ number of primal solutions. For illustration purposes, consider for example a situation where one starts with a dual solution consisting of two sub-tours of sizes n_1 and n_2, respectively, and the parameters of DLS are set to $s = 2$ and $r = (1, 1)$. Then, the size of the dual search space is $\frac{1}{2}\sum_{n_1=2}^{n-2}\binom{n}{n_1}(n_1-1)!\,(n-n_1-1)!$, which is much higher than the size of the entire primal search space $(n-1)!$, which is easily proven by induction. □

3.3 Comparative Analysis with The Literature

As part of positioning our contribution, we shall hereafter compare our dual local search (DLS) to some of the contributions that have some comparable features. Within the category of construction methods, heuristics such as the nearest merger procedure (Rosenkrantz et al., 1977), the patching heuristic (Karp, 1979; Karp and Steele, 1985) and the modified patching heuristic (Glover et al., 2001) have some similarities with our DLS in that they all are cycle merging methods. However, they differ with respect to the basic ideas behind their designs. To be more specific, the nearest merger procedure, the patching heuristic and the modified patching heuristic are all "pure" construction methods as opposed to our DLS which is a parameterized search method designed to replicate an optimal search design; namely, the B&B design. In addition, the sets of moves used within our design are inclusive of the rather restricted "set of moves" used by these pure construction methods. Finally, our DLS could also be viewed or categorized as a construction method because of its dual nature.

4. How Dual Search Compares to Primal Search

In this section, we shall provide statistical evidence of the superiority of the proposed dual local search to its primal counterpart. Such evidence is based on a sample of 43 TSP instances from the TSPLIB along with a set of statistically significant t-tests. The choice of this number of instances is the result of limiting the size of problems to solve to less than or equal to 200 nodes, which was motivated by the application context of urban logistics. We also discuss the performance of the different implementation schema proposed. Both the primal local search and the dual local search frameworks were implemented in C++ on a Dell Inspiron machine with 2.26GHz Core i3 processor and 4GB of RAM, and the assignment problem-based relaxations were solved using CPLEX 12.4.

Recall that the implementation of the proposed DLS framework for the TSP requires one to address several questions. First, how to choose the s sub-tours to be merged? In our empirical experiments, we tested three criteria for selecting sub-tours to merge; namely, the farthest distance between sub-tours; the nearest distance between sub-tours; and the cheapest cost of merger of sub-tours. Our empirical results revealed that the farthest distance criterion produces the best results, then the cheapest cost criterion produced the second best results and finally the nearest distance criterion does not perform as well as the other two criteria. For space constraints, in the remainder of this section we shall only present the numerical results for the first criterion, but conclusions are inclusive of the other results. The second question to be addressed is related to the choice of the parameters of the proposed parameterized neighborhood structure; namely, s and r. In our empirical experiments, the following parameter choices were made: $\{s = 2, r_1 = 1, r_2 = 1\}$, $\{s = 2, r_1 = 2, r_2 = 1\}$ and $\{s = 3, r_1 = 1, r_2 = 1, r_3 = 1\}$. The choice of these values has been motivated by seeking an acceptable balance between intensification, diversification, convergence rate, and computational time. The third question to be addressed is concerned with how often to call upon a local search mechanism based on Type II moves to explore those branches of the B&B tree not explored by Type I moves. Let p $(0 \leq p \leq 1)$ denote the proportion of use of a local search mechanism based on Type II moves. As previously mentioned, in order to reduce the computational requirements of the proposed framework, the value of p could be either static or dynamic and could be implemented in a deterministic fashion (e.g., using a deterministic decision rule) or a stochastic fashion (e.g., using a probabilistic decision rule). The reader is referred to section 3.1 for a detailed description of these implementation schema. As to the Type II moves used in our experiments, they are divided into two categories; namely, 2-opt and 3-opt moves and the improvement moves used in GENI, see Gendreau et al. (1992) for details, that we refer to in this paper as US moves.

The performance of the proposed dual local search framework is benchmarked against the performance of the classical primal local search framework where the initial primal solution is computed using the nearest merger construction method and improved using the same Type II moves; namely, 2-opt moves, 3-opt moves or US moves. The choice of the initial solution for the primal local search is motivated by "fair" benchmarking; that is, benchmarking against a method that is conceptually similar in spirit. Note however that several other initial solutions were also tested for; namely, farthest insertion, nearest insertion, and random insertion. Again, for space constraints, in the remainder of this section we shall only present the numerical results for the nearest merger solution as the initial solution for primal local search, but conclusions are inclusive of the other results.

First, we shall provide statistical evidence that DLS outperforms PLS. To be more specific, we tested the hull hypothesis (H_0) that the average percentage increase in distance of DLS solutions over PLS solutions is greater than or equal to zero. Therefore, the alternative hypothesis (H_1) states that the average percentage increase in distance of DLS solutions over PLS solutions is less than zero; that is, DLS outperforms PLS. The null hypothesis H_0 is tested for all combinations of search parameters; i.e., s and r, and local improvement schema resulting in a total of 36 statistical tests. The chosen hypothesis test is a one-tailed t-test performed under the p-value approach. The p-values are summarised in Table 2. Under the first set of parameters; that is, $\{s = 2, r_1 = 1, r_2 = 1\}$, all results are statistically significant at 0.1% regardless of the local improvement scheme and type of move used. Under the second set of parameters; that is, $\{s = 2, r_1 = 2, r_2 = 1\}$, all results are statistically significant at 0.1%, except for the combination { Deterministic & Static Local Improvement Scheme, 2-opt} whose result is statistically significant at 5% and the combination {Deterministic & Dynamic Local Improvement Scheme, US} whose result is statistically significant at 1%. Finally, under the third set of parameters; that is, $\{s = 3, r_1 = 1, r_2 = 1, r_3 = 1\}$, all results are statistically significant at 0.1% or 1%, except for the combinations {Deterministic & Dynamic Local Improvement Scheme, 3-opt} and {Deterministic & Dynamic Local Improvement Scheme, US} whose results are not statistically significant; however, when TSP instances hk48, kroB150, pr144 and si175 are dropped from the sample, the result of the first combination becomes statistically significant at 0.1% (p-value = 0.0040) and the result of the second combination becomes statistically significant at 0.5% (p-value = 0.0279). In sum, hypothesis testing proves that DLS outperforms PLS under most of the settings considered in our computational experiments.

Move	$s = 2, r_1 = 1, r_2 = 1$	$s = 2, r_1 = 2, r_2 = 1$	$s = 3, r_1 = 1, r_2 = 1, r_3 = 1$
Local Improvement at Each Iteration			
2-opt	0.0002***	0.0001***	0.0041**
3-opt	0.0000***	0.0000***	0.0000***
US	0.0000***	0.0001***	0.0001***
Deterministic & Static Local Improvement Scheme			
2-opt	0.0000***	0.0157*	0.0020**
3-opt	0.0001***	0.0001***	0.0021**
US	0.0000***	0.0001***	0.0002***
Deterministic & Dynamic Local Improvement Scheme			
2-opt	0.0006***	0.0003***	0.0021**
3-opt	0.0000***	0.0002***	0.1351
US	0.0002***	0.0052**	0.2270
Stochastic Local Improvement Scheme			
2-opt	0.0002***	0.0001***	0.0021**
3-opt	0.0002***	0.0001***	0.0017**
US	0.0000***	0.0000***	0.0001***

*5% significant at $p < 0.05$; **1% significant at $p < 0.01$; ***0.1% significant at $p < 0.001$

Table 2: p-values of one-tailed t-tests of hypothesis

Hereafter, we shall discuss the performance of the different implementation schema and values of search parameters based on sample evidence. First, both when a local search mechanism based on Type II moves is called upon at each iteration and when the dynamic frequency-based improvement scheme is used, on most choices of parameters s and r, DLS outperforms Random, Farthest, and Nearest Insertions as well as Nearest Merger solutions improved with PLS using 2-Opt, 3-Opt, and US moves – see, for example, Figures 4b-6b and 13b-15b. As some of the commonly reported statistics in these Figures are affected by outliers, we provide a more reliable “picture” in Figures 4a-6a and 13a-15a, where the x-axis represents TSP instances and the y-axis represents the percentage increase in the objective function value of a DLS solution over the PLS solution. These figures clearly show that DLS outperforms PLS on most problem instances (i.e., more negative spikes than positive ones) and the difference in performance could be substantial. Notice that, with the exception of very few outlier instances where US moves require a prohibitive amount of time, computational requirements are comparable. Second, under both the deterministic static and the stochastic frequency-based improvement schema, on most choices of parameters, DLS outperforms Nearest, Farthest and Random Insertions as well as Nearest Merger solutions improved with PLS using 2-Opt, 3-Opt, and US moves – see, for example, Figures 7a-9a and 10a-12a and Figures 7b-9b and 10b-12b. **Third**, the performance of DLS as compared to PLS depends on the structure of the solution with which PLS starts the search, on one hand, and the frequency with which DLS intermediate solutions are perturbed, on the other hand. In fact, numerical results reveal that DLS is often outperformed whenever the structure of the solution with which PLS starts leaves room for substantial improvement by PLS moves; for example, initializing PLS with a random insertion solution tends to leave substantial room for improving such initial primal solution by some Type II moves. Furthermore, numerical results reveal that improving the dual solution too frequently (i.e., at each iteration) tends to perturb the solution structure in a relatively unattractive way in that the advantage of starting with a "good" relaxation solution is mildly lost in comparison to the deterministic static and stochastic improvement schema – see, for example, Figures 4a-6a and 7a-12a and Figures 4b-6b and 7b-12b. Also, when the dual solution is perturbed at increasingly large and irregular intervals (i.e., dynamic frequency based improvement), the resulting change in structure turns out to be relatively unattractive as one would "miss out" on improvement opportunities as the search process progresses – see, for example, Figures 13a-15a and Figures 13b-15b. Finally, between these two "extreme" cases lies deterministic static and stochastic frequency based improvements, which tend to perform the "right" amount of perturbation needed to converge towards a good primal solution – see, for example, Figures 7a-9a and 10a-12a and Figures 7b-9b and 10b-12b.

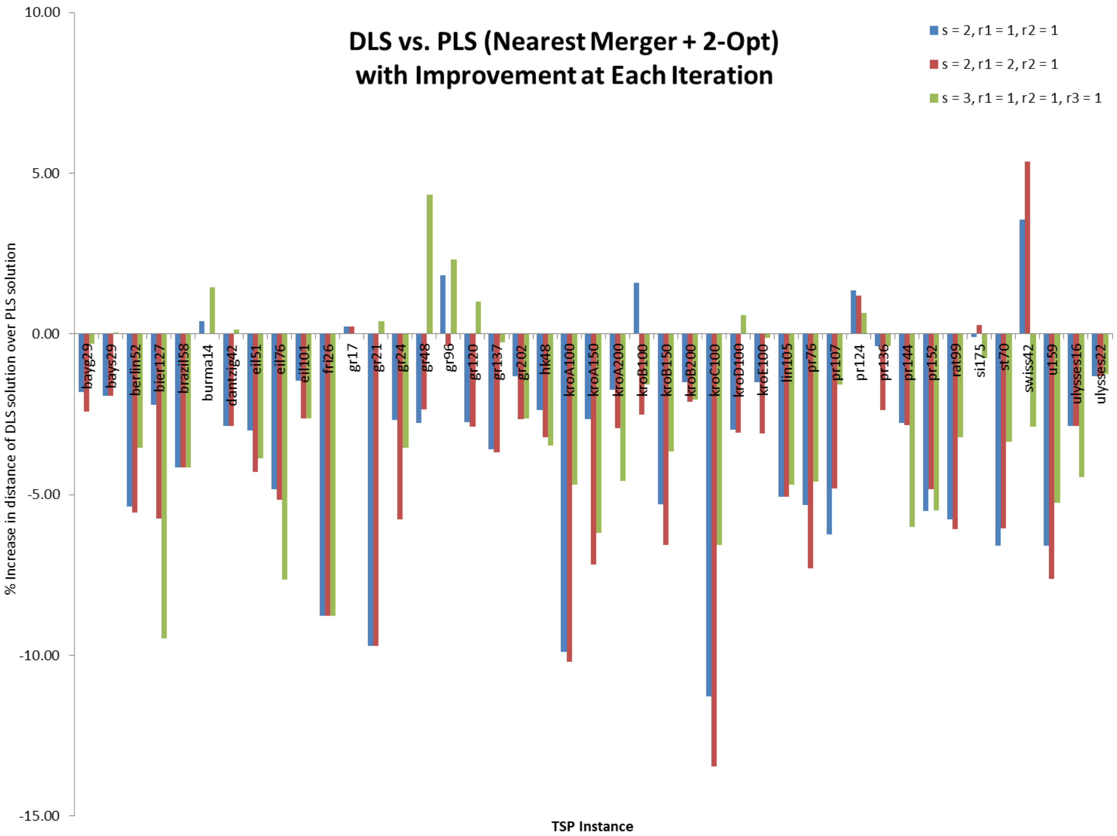

Figure 4a: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed at Each Iteration

Parameters of DLS	Statistics*		CPU Difference**	
$s = 2, r_1 = 1, r_2 = 1$	Min Max Mean Std. Dev.	-11.27 +3.55 -3.21 +3.20	Min Max Mean Std. Dev.	-1.027 +0.572 -0.169 +0.357
$s = 2, r_1 = 2, r_2 = 1$	Min Max Mean Std. Dev.	-13.46 +5.36 -3.94 +3.32	Min Max Mean Std. Dev.	-0.665 +1.345 +0.037 +0.480
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min Max Mean Std. Dev.	-9.48 +4.34 -2.62 +3.00	Min Max Mean Std. Dev.	-1.38 +0.523 -0.359 +0.436

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

**Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 4b: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed at Each Iteration

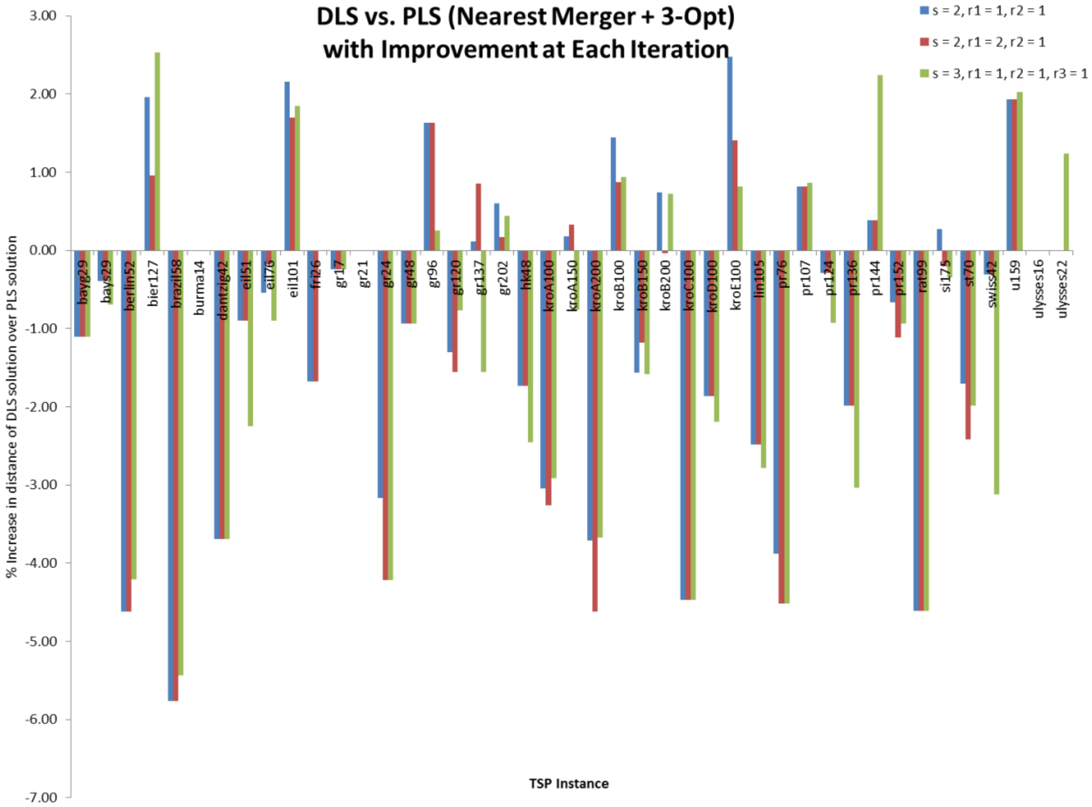

Figure 5a: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed at Each Iteration

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-5.76	Min	-1.957
	Max	+2.48	Max	+20.654
	Mean	-0.98	Mean	+1.584
	Std. Dev.	+2.02	Std. Dev.	+4.468
$s = 2, r_1 = 2, r_2 = 1$	Min	-5.76	Min	-0.981
	Max	+1.93	Max	+17.839
	Mean	-1.15	Mean	+2.344
	Std. Dev.	+2.02	Std. Dev.	+4.459
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-5.44	Min	-1.164
	Max	+2.54	Max	+34.934
	Mean	-1.22	Mean	+5.371
	Std. Dev.	+2.07	Std. Dev.	+8.908

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 5b: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed at Each Iteration

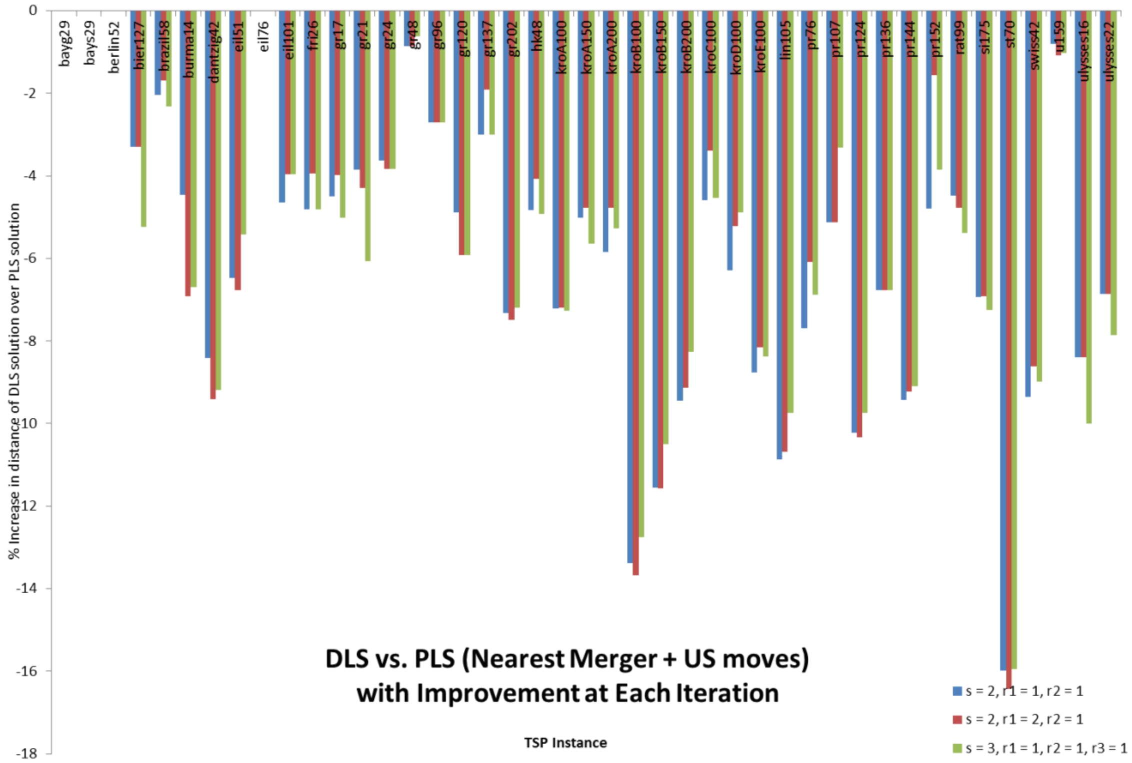

Figure 6a: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed at Each Iteration

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-15.98	Min	-0.002
	Max	0.00	Max	+794.983
	Mean	-6.00	Mean	+128.355
	Std. Dev.	+3.50	Std. Dev.	+204.205
$s = 2, r_1 = 2, r_2 = 1$	Min	-16.44	Min	0.131
	Max	0.00	Max	+1029.559
	Mean	-5.81	Mean	+158.915
	Std. Dev.	+3.62	Std. Dev.	+254.520
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-15.95	Min	+0.05
	Max	+15.95	Max	+1262.519
	Mean	-6.01	Mean	+151.604
	Std. Dev.	+3.39	Std. Dev.	+249.728

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 6b: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed at Each Iteration

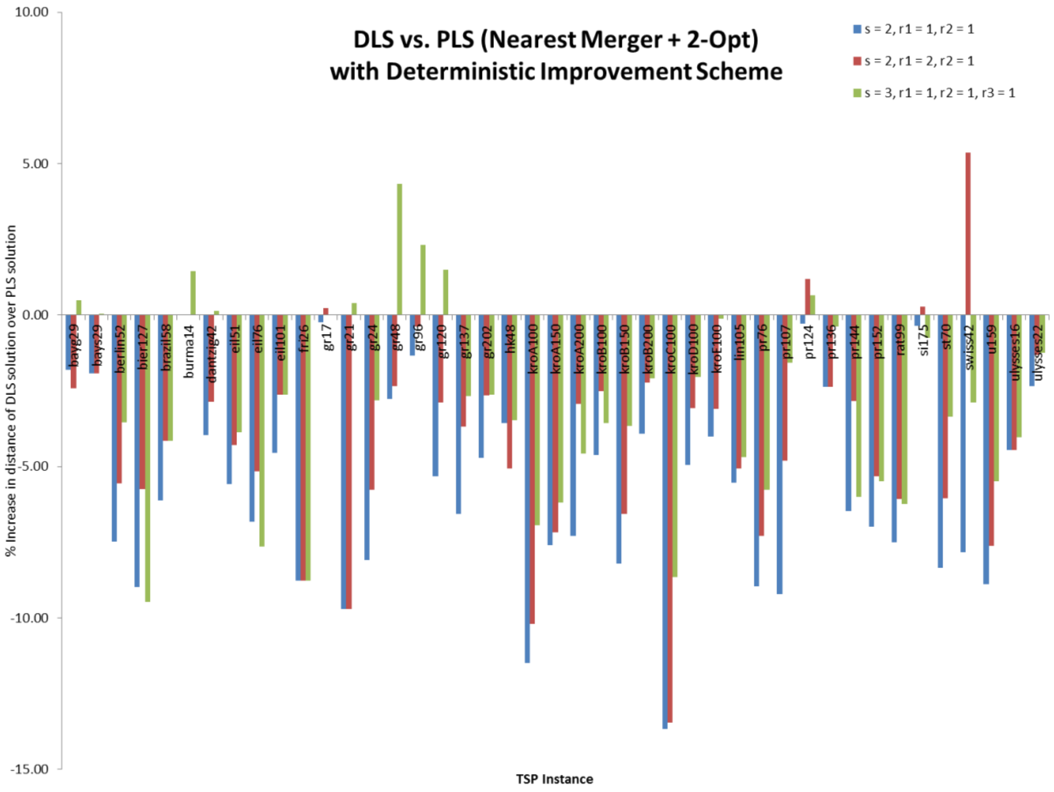

Figure 7a: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed according to The Deterministic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-13.67	Min	-0.525
	Max	0.00	Max	+27.19
	Mean	-5.67	Mean	+3.850
	Std. Dev.	+3.18	Std. Dev.	+6.119
$s = 2, r_1 = 2, r_2 = 1$	Min	-13.46	Min	-17.337
	Max	+5.36	Max	+47.315
	Mean	-4.03	Mean	+6.938
	Std. Dev.	+3.32	Std. Dev.	+11.270
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-9.48	Min	-3.66
	Max	+4.34	Max	+44.061
	Mean	-2.93	Mean	+5.803
	Std. Dev.	+3.16	Std. Dev.	+10.964

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 7b: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed according to The Deterministic Scheme

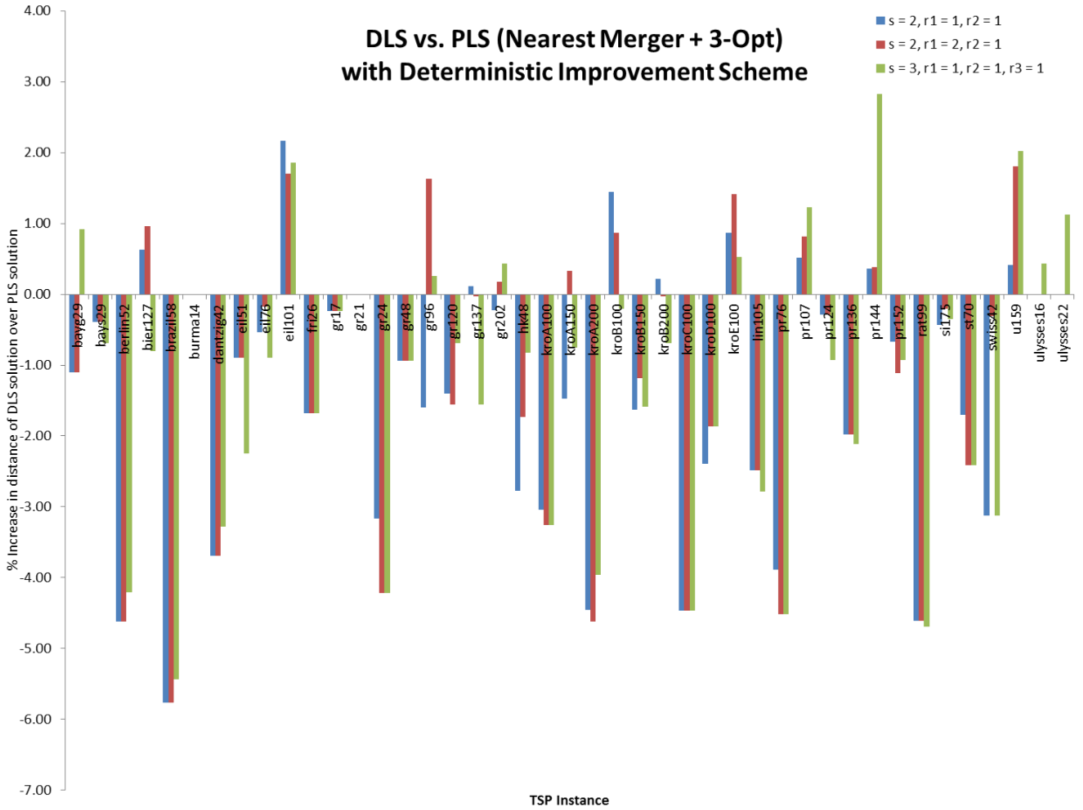

Figure 8a: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed according to The Deterministic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-5.76	Min	-20.435
	Max	+2.16	Max	+3.158
	Mean	-1.37	Mean	-1.840
	Std. Dev.	+1.85	Std. Dev.	+4.522
$s = 2, r_1 = 2, r_2 = 1$	Min	-5.76	Min	-19.054
	Max	+1.80	Max	+4.108
	Mean	-1.17	Mean	-0.266
	Std. Dev.	+2.00	Std. Dev.	+3.350
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-5.44	Min	-2.021
	Max	+2.82	Max	+105.507
	Mean	-1.27	Mean	+12.110
	Std. Dev.	+1.99	Std. Dev.	+20.301

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 8b: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed according to The Deterministic Scheme

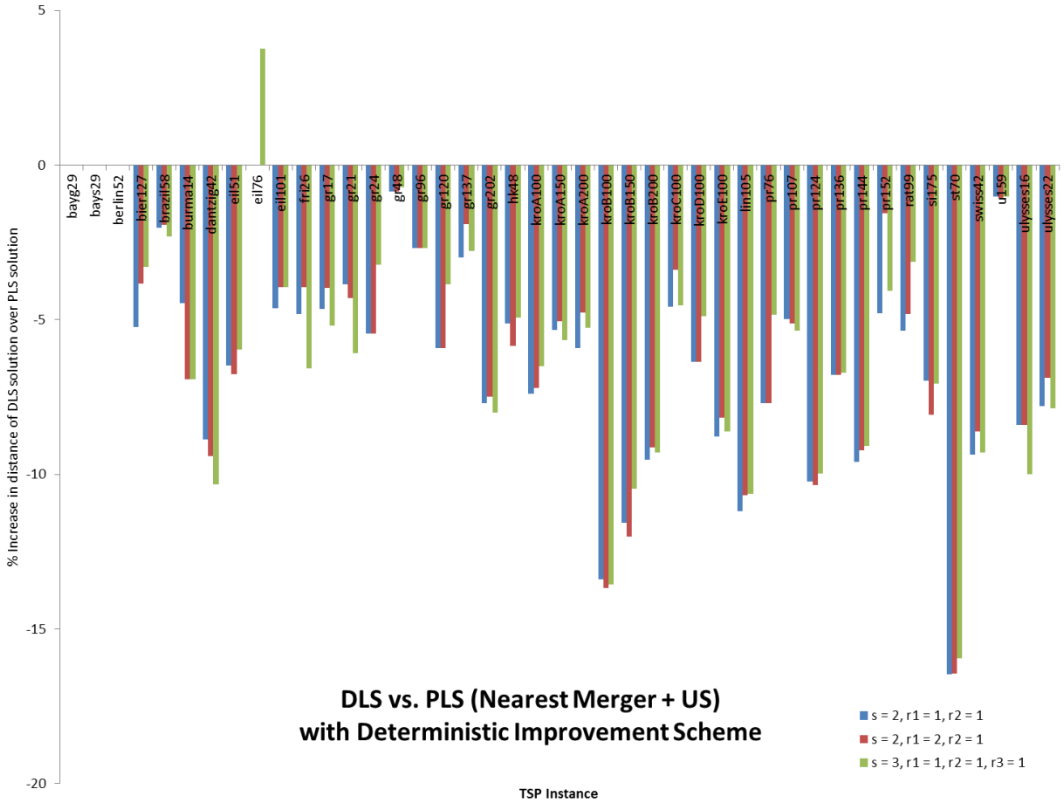

Figure 9a: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed according to The Deterministic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-16.46	Min	-2.041
	Max	0.00	Max	+1019.37
	Mean	-6.24	Mean	+124.414
	Std. Dev.	+3.49	Std. Dev.	+217.785
$s = 2, r_1 = 2, r_2 = 1$	Min	-16.44	Min	+0.261
	Max	0.00	Max	+931.186
	Mean	-6.01	Mean	+127.148
	Std. Dev.	+3.62	Std. Dev.	+210.612
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-15.95	Min	-0.11
	Max	+3.76	Max	+548.603
	Mean	-5.91	Mean	+88.311
	Std. Dev.	+3.78	Std. Dev.	+143.835

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 9b: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed according to The Deterministic Scheme

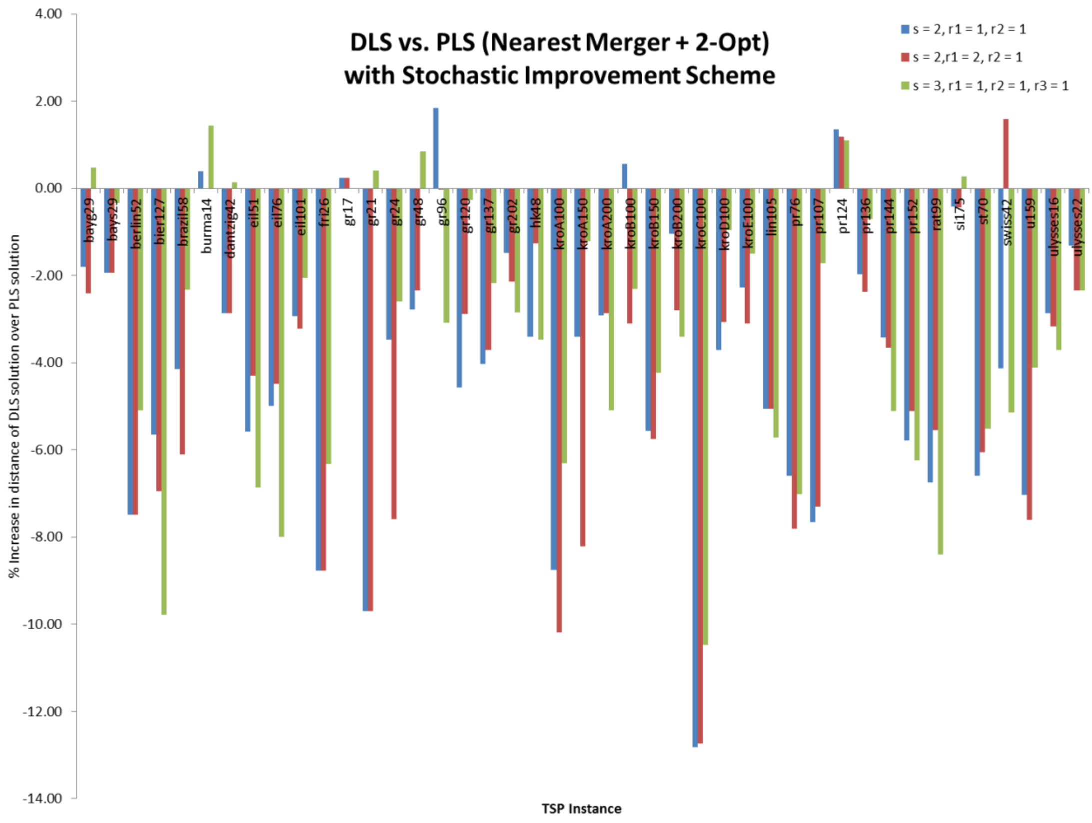

Figure 10a: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed according to The Stochastic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-12.82	Min	-1.859
	Max	+1.83	Max	+0.052
	Mean	-3.98	Mean	-0.498
	Std. Dev.	+3.10	Std. Dev.	+0.550
$s = 2, r_1 = 2, r_2 = 1$	Min	-12.73	Min	-1.006
	Max	+1.59	Max	+12.406
	Mean	-4.27	Mean	+0.993
	Std. Dev.	+3.20	Std. Dev.	+2.942
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-10.48	Min	-1.925
	Max	+1.44	Max	+12.365
	Mean	-3.30	Mean	+0.616
	Std. Dev.	+3.05	Std. Dev.	+3.240

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 10b: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed according to The Stochastic Scheme

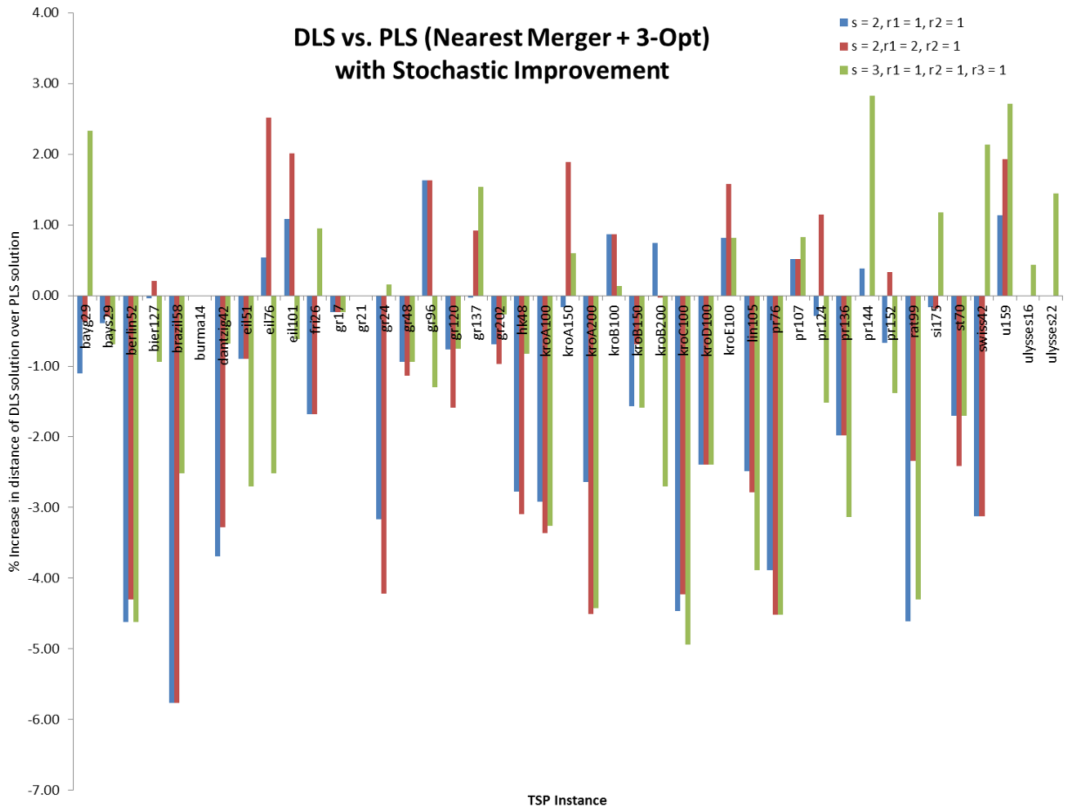

Figure 11a: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed according to The Stochastic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-5.76	Min	-16.445
	Max	+1.63	Max	+0.838
	Mean	-1.21	Mean	-1.811
	Std. Dev.	+1.83	Std. Dev.	+3.506
$s = 2, r_1 = 2, r_2 = 1$	Min	-5.76	Min	-12.326
	Max	+2.52	Max	+4.216
	Mean	-1.04	Mean	-1.175
	Std. Dev.	+2.11	Std. Dev.	+2.814
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-4.94	Min	-7.603
	Max	+2.82	Max	+10.178
	Mean	-0.96	Mean	-0.391
	Std. Dev.	+2.09	Std. Dev.	+3.017

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 11b: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed according to The Stochastic Scheme

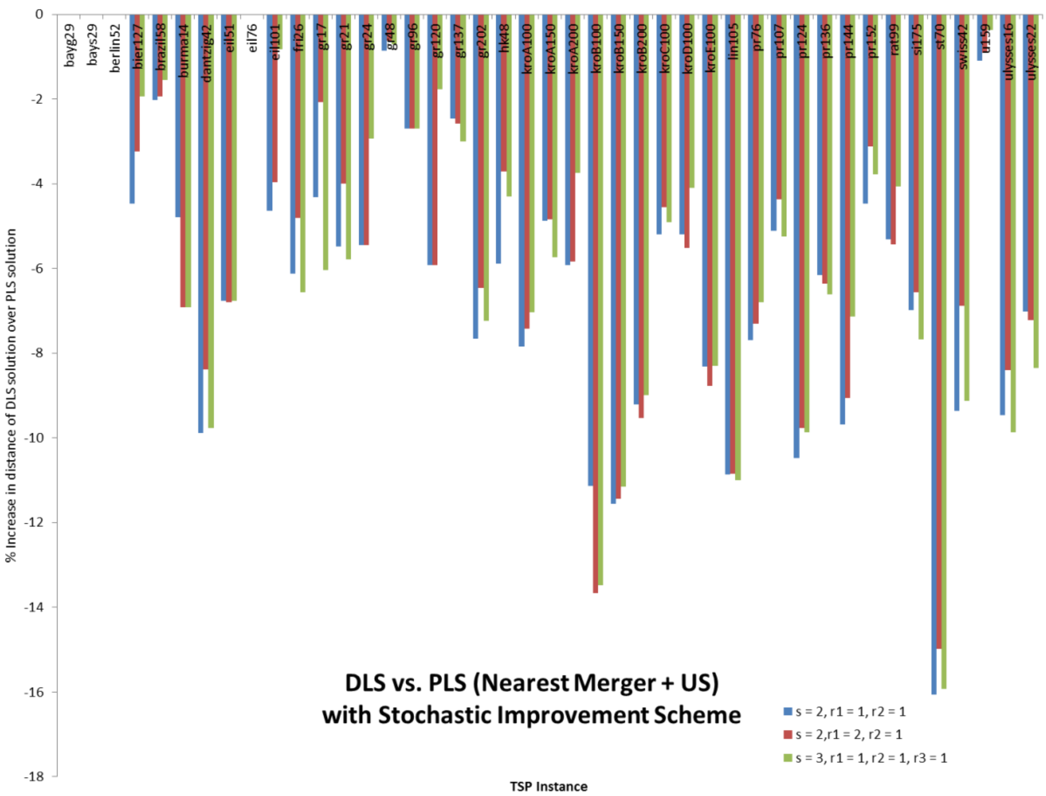

Figure 12a: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed according to The Stochastic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-16.07	Min	-1.33
	Max	0.00	Max	+1014.537
	Mean	-6.20	Mean	+121.055
	Std. Dev.	+3.38	Std. Dev.	+220.051
$s = 2, r_1 = 2, r_2 = 1$	Min	-14.99	Min	-0.418
	Max	0.00	Max	+451.444
	Mean	-5.85	Mean	+67.394
	Std. Dev.	+3.45	Std. Dev.	+108.029
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-15.92	Min	-3.853
	Max	+0.20	Max	+238.543
	Mean	-5.77	Mean	-29.351
	Std. Dev.	+3.71	Std. Dev.	+49.289

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 12b: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed according to The Stochastic Scheme

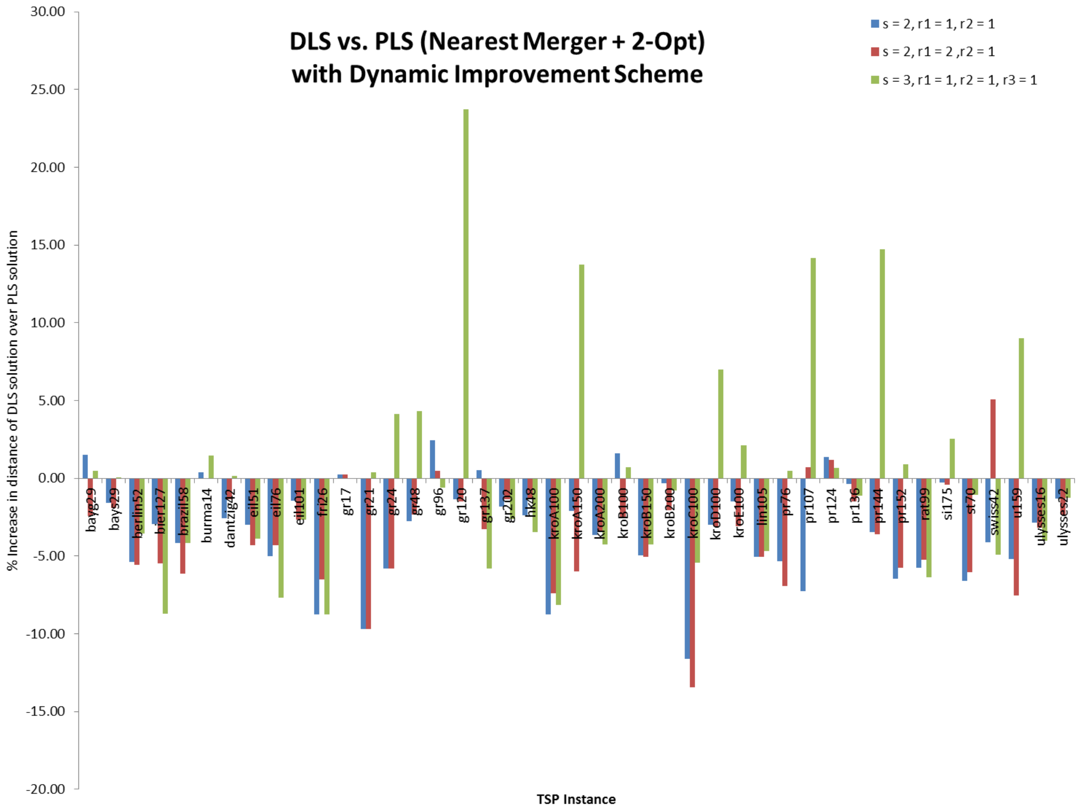

Figure 13a: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed according to The Dynamic Scheme

Parameters of DLS	Statistics*		CPU Difference**	
$s = 2, r_1 = 1, r_2 = 1$	Min	-11.61	Min	+0.033
	Max	+2.43	Max	+44.821
	Mean	-3.27	Mean	+7.118
	Std. Dev.	+3.21	Std. Dev.	+10.211
$s = 2, r_1 = 2, r_2 = 1$	Min	-13.46	Min	+0.009
	Max	+5.07	Max	+49.512
	Mean	-3.53	Mean	+8.657
	Std. Dev.	+3.22	Std. Dev.	+12.305
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-8.76	Min	+0.003
	Max	+23.71	Max	+44.754
	Mean	+0.06	Mean	+8.667
	Std. Dev.	+6.73	Std. Dev.	+11.743

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 13b: DLS vs. PLS Improving Nearest Merger Solution with 2-Opt Moves, and Improvement is Performed according to The Dynamic Scheme

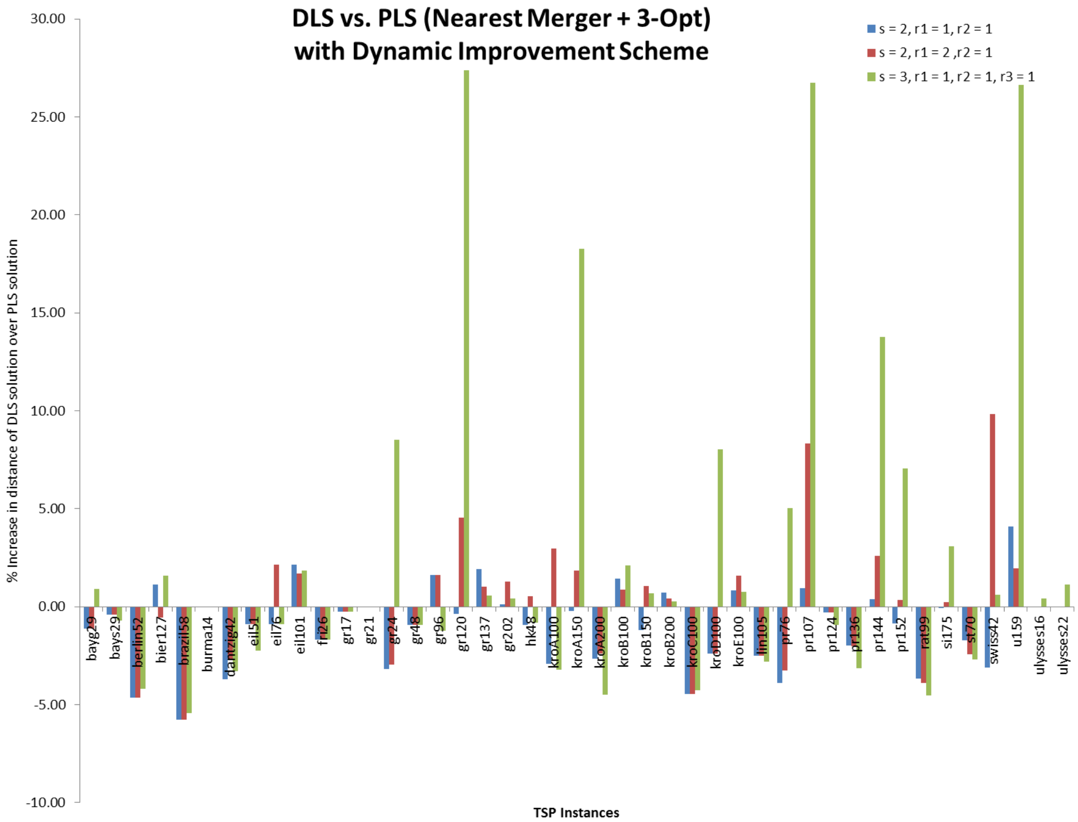

Figure 14a: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed according to The Dynamic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min	-5.76	Min	-18.114
	Max	+4.10	Max	+14.185
	Mean	-0.96	Mean	-1.222
	Std. Dev.	+2.04	Std. Dev.	+4.493
$s = 2, r_1 = 2, r_2 = 1$	Min	-5.76	Min	-16.264
	Max	+9.82	Max	+29.878
	Mean	-0.01	Mean	+3.476
	Std. Dev.	+3.00	Std. Dev.	+8.682
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min	-5.44	Min	-7.603
	Max	+27.38	Max	+10.178
	Mean	+2.51	Mean	+0.391
	Std. Dev.	+8.20	Std. Dev.	+3.017

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 14b: DLS vs. PLS Improving Nearest Merger Solution with 3-Opt Moves, and Improvement is Performed according to The Dynamic Scheme

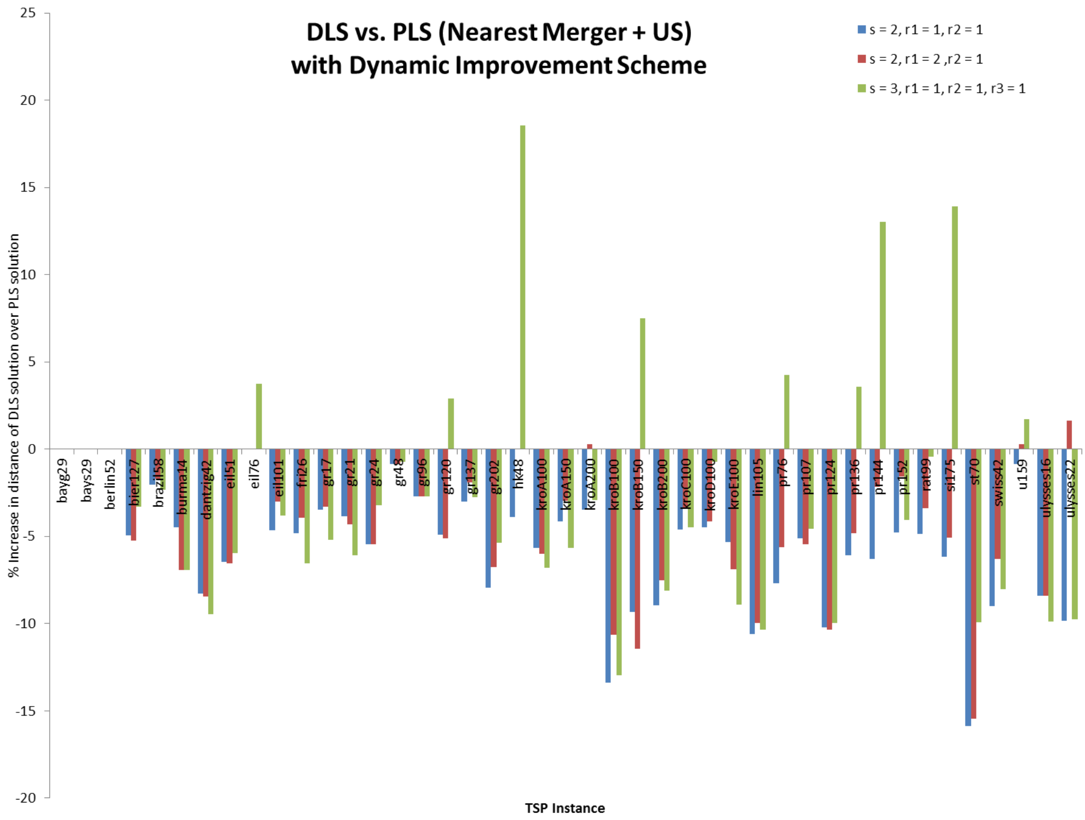

Figure 15a: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed according to The Dynamic Scheme

Parameters of DLS	**Statistics***		**CPU Difference****	
$s = 2, r_1 = 1, r_2 = 1$	Min Max Mean Std. Dev.	-15.88 0.00 -5.66 +3.35	Min Max Mean Std. Dev.	-0.26 +326.478 +61.436 +84.905
$s = 2, r_1 = 2, r_2 = 1$	Min Max Mean Std. Dev.	-15.44 +1.63 -4.71 +3.61	Min Max Mean Std. Dev.	-5.992 +335.461 +54.358 +71.608
$s = 3, r_1 = 1, r_2 = 1$ & $r_3 = 1$	Min Max Mean Std. Dev.	-12.94 +18.53 -2.31 +7.14	Min Max Mean Std. Dev.	+0.177 +1191.159 +73.805 +185.187

*Statistics on Percentage Increase in Distance of DLS Solution over PLS Solution, where a negative value of a measure reflects that DLS outperforms PLS

** Statistics on the Increase in CPU time required by DLS over PLS, where a negative value of a measure reflects that DLS outperforms PLS

Figure 15b: DLS vs. PLS Improving Nearest Merger Solution with US Moves, and Improvement is Performed according to The Dynamic Scheme

With respect to delivering the optimal solution or a near optimal solution (i.e., within 1%), DLS seems to achieve a relatively high performance given the design limitations of local search. In fact, the percentage of optimal solutions delivered reaches up to 35% depending on the choice of the parameters of DLS and the nature of type II moves. Furthermore, the percentage of near optimal solutions delivered ranges from 23% to 79% depending on the choice of the parameters of DLS and the nature of type II moves. Last, but not least, the dual search framework performs competitively with respect to CPU as compared to the primal search framework with the exception of very few outlier instances where US moves require a prohibitive amount of time. Once again, CPU requirements depend on the choice of the parameters of DLS and the nature of type II moves.

5. Conclusion

In practice, heuristics are typically used to solve realistically sized combinatorial optimization problems such as the traveling salesman problem (TSP). A specific category of heuristics has attracted considerable attention; namely, local search methods. Most local search methods are primal in nature in that they start the search with a feasible solution and explore the feasible space for better feasible solutions. In this research, we designed a dual local search method to solve the TSP; that is, a search method that starts with an infeasible solution, explores the dual space – each time reducing infeasibility, and lands in the primal space to deliver a feasible solution. The basic idea behind the proposed design is to replicate the designs of optimal solution methodologies in a heuristic way. To be more specific, our dual local search framework first solves an assignment problem relaxation of a TSP formulation and then repairs its typically infeasible solution using a new parameterized neighborhood and intermediate dual solutions are improved locally. Statistically significant t-tests support the superiority of this dual design compared to its primal design counterpart. Thus, the proposed dual local search method is a promising search framework.

References

Albayrak M and Allahverdi N (2011). Development a new mutation operator to solve the Traveling Salesman Problem by aid of Genetic Algorithms**.** *Expert Systems with Applications* **38**(3): 1313-1320.

Alijla B. O., Wong L-P, Lim C. P., Khader A. T. and Al-Betar M. Z. (2014). A modified Intelligent Water Drops algorithm and its application to optimization problems. *Expert Systems with Applications* **41**(15): 6555–6569.

Applegate D, Bixby R, Chvatal V and Cook W (2007). *The traveling salesman problem: a computational study*. Princeton University Press: Princeton.

Balas E and Christofides N (1981). A restricted Lagrangean approach to the travelling salesman problem. *Mathematical Programming* **21**(1): 19-46.

Bellmore M and Malone J C (1971). Pathology of travelling-salesman problem subtour-elimination algorithms. *Operations Research* **19**(2): 278-307.

Boyd S and Cunningham W (1991). Small Travelling salesman polytopes. *Mathematics of operations Research* **16**(2): 259-271.

Carpaneto G, Dell'Amico M and Toth P (1995). Exact Solution of Large Scale, Asymmetric Travelling Salesman Problems. *ACM Transactions on Mathematical Software* **21**(4): 394-409.

Carpaneto G and Toth P (1980). Some new Branching and Bounding Criteria for the Asymmetric Travelling Salesman Problem. *Management Science* **26**(7): 736-743.

Christofides N (1976). *Worst-Case Analysis of a new heuristic for the travelling salesman problem.* Report 388. Graduate School of Industrial Administration: Carnegie Mellon University.

Chvatal V (1973). Edmond Polytopes and Weakly Hamiltonian Graphs. *Mathematical Programming* **5**(1): 29-40.

Cornuejols G, Fonlupt J and Naddef D (1985). The travelling salesman on a graph and some related polyhedra. *Mathematical Programming* **33**(1): 1-27.

Crowder H and Padberg M W (1980). Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality. *Management Science* **26**(5): 495-509.

Dantzig G B, Fulkerson D R and Johnson S M (1954). Solution of a large-scale travelling-salesman problem. *Operations Research* **2**(4): 393-410.

DePuy G W, Moraga R J and Whitehouse G E (2005). Meta-RaPS: A Simple and Effective Approach For Solving The Traveling Salesman Problem. *Transportation Research Part E: Logistics and Transportation Review* **41**(2): 115-130.

Dong G, Guo W W and Tickle K (2012). Solving the traveling salesman problem using cooperative genetic ant systems. *Expert Systems with Applications* **39**(5): 5006-5011.

Dorigo M and Gambardella L M (1997). Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem. *IEEE Transactions on Evolutionary Computation* **1**(1): 53-66

Dowlatshahi M. B., Nezamabadi-pour H. and Mashinchi M. (2014). A discrete gravitational search algorithm for solving combinatorial optimization problems. *Information Sciences* **258**, February: 94–107.

Eastman W (1958). Linear Programming with Pattern *Constraints*. PhD thesis, Harvard University.

Elloumi W., El Abed H., Abraham A. and Alimi A. M. (2014). A comparative study of the improvement of performance using a PSO modified by ACO applied to TSP. *Applied Soft Computing* **25**, December: 234–241.

Fischetti M and Toth P (1997). A Polyhedral Approach to the Asymmetric Traveling Salesman Problem. *Management Science* **43**(11): 1520-1536.

Fischetti M, Lodi A and Toth P (2003). *Solving real-world ATSP instances by branch-and-cut*. Lecture Notes in Computer Science 2570. Springer: Berlin.

Fleischmann B (1988). A new class of cutting planes of the symmetric travelling salesman problem. *Mathematical Programming* **40**(1-3): 225-246.

Gavish B and Srikanth K (1983). *Efficient Branch and Bound Code for Solving Large Scale Travelling Salesman Problems to Optimality*. Working Paper QM 8329, University of Rochester.

Germs R., Goldengorin B. and Turkensteen M. (2012). Lower tolerance-based Branch and Bound algorithms for the ATSP. *Computers & Operations Research* **39**(2): 291–298.

Gendreau M, Hertz A and Laporte G (1992). New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research* **40**(6):1086-1094.

Glover F, Gutin G, Yeo A and Zverovich A (2001). Construction heuristics for the asymmetric TSP. *European Journal of Operational Research* **129**(3): 555-568

Goldengorin B, Jager G and Molitor P (2006). *Tolerance based Contract-or-Patch heuristic for the asymmetric TSP*. Combinatorial and Algorithmic Aspects of Networking, Lecture Notes in Computer Science, Volume 4235, pp 86-97.

Grotschel M and Pulleyblank W (1986). Clique tree inequalities and the symmetric traveling salesman problem. *Mathematics of Operations Research* **11**(4): 537-569.

Gutin G and Zverovich A (2005). Evaluation of the Contract-Or-Patch Heuristics for the Asymmetric TSP. *INFOR* **43**(1): 23–31.

Hassin R and Keinan A (2008). Greedy heuristics with regret, with application to the cheapest insertion algorithm for the TSP. *Operations Research Letters* **36**(2): 243-246.

He J., Xiao J. and Shao Z. (2014). An Adaptive Membrane Algorithm for Solving Combinatorial Optimization Problems. *Acta Mathematica Scientia* **34B(5)**:1377–1394.

Helbig H K and Krarup J (1974). Improvements of the Held-Karp Algorithm for the Symmetric Travelling-Salesman Problem. *Mathematical Programming* **7**(1): 87-96.

Held M and Karp R (1971). The traveling salesman problem and minimum spanning trees Part II. *Mathematical Programming* **1**(1): 6-25.

Houck D, Picard J, Queyranne M and Vemuganti R (1980). The travelling salesman problem as a constrained shortest path problem: theory and computational experience. *Operations Research* **17**(2&3): 93-109.

HM Treasury (2014). Public Expenditure Statistical Analyses, UK Government Publications, London.

Jati G. K., Manurung R. and Suyanto (2013). Discrete Firefly Algorithm for Traveling Salesman Problem: A New Movement Scheme. In *Swarm Intelligence and Bio-Inspired Computation - Theory and Applications*: 295–312, edited by Xin-She Yang, Zhihua Cui, Renbin Xiao, Amir Hossein Gandomi and Mehmet Karamanoglu. Elsevier.

Jayalakshmi G A, Sathiamoorthy and Rajaram R (2001). A Hybrid Genetic Algorithm – A new approach to solve travelling salesman problem. *International Journal of Computational Engineering Science* **2**(2): 339-355

Karp R M (1977). Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane. *Mathematics of Operations Research* **2**(3): 209-224.

Karp R M (1979). A patching algorithm for the non-symmetric travelling salesman problem. *SIAM Journal on Computing* **8**(4): 561–573.

Karp M R and Steele J M (1985). Probabilistic Analysis of Heuristics. In *The traveling salesman problem*, edited by Lawler et al (1985). John Wiley and Sons: Chichester.

Kim C (1975). *A Minimal Spanning Tree and Approximate tours for a travelling salesman.* Computer Science Technical Report: University of Maryland.

Kirkpatrick S, Gelatt C D and Veechi M P (1983). Optimization by Simulated Annealing. *Science* 220(4598): 671-680.

Lawler E L, Lenstra J K, Rinnooy Kan A H G and Shmoys D B (1985). *The traveling salesman problem.* John Wiley and Sons: Chichester.

Liao Y-F, Yau D-H, and Chen C-L (2012). Evolutionary algorithm to traveling salesman problems. *Computers and Mathematics with Applications* **64**(5): 788-797.

Leung K-S, Jin H-D and Xu Z-B (2004). An expanding self-organizing neural network for the traveling salesman problem. *Neurocomputing* **62**, December: 267-292.

Li M., Yi Z and Zhu M. (2009). Solving TSP by using Lotka–Volterra neural networks. *Neurocomputing* **63**(16): 3873–3880.

Lin S (1965). Computer Solutions of the travelling salesman problem. *Bell System Technical Journal* **44**(10): 2245-2269.

Lin S and Kernighan B (1973). An effective heuristic algorithm for the travelling salesman problem. *Operations Research* **21**(2): 498-516.

Liu F. and Zeng G. (2009). Study of genetic algorithm with reinforcement learning to solve the TSP. *Expert Systems with Applications* **36**(3): 6995–7001.

Lo C-C and Hsu C-C (1998). An Annealing Framework with Learning Memory. *IEEE Transactions on Systems, Man and Cybernetics – Part A: Systems and Humans* **28**(5): 648-661

Malek M (1988). *Search Methods for the Travelling Salesman Problems*. University of Texas: Austin.

Malek M, Guruswamy M, Pandya M and Owens H (1989). Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research* **21**(1): 59-84.

Marinakis Y, Marinaki M and Dounias G (2011): Honey bees mating optimization algorithm for the Euclidean traveling salesman problem. *Information Science* **181**(20): 4684-4698.

Masutti T A S and de Castro L N (2009). A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. *Information Science* **179**(10): 1454-1468

Miller D and Pekny J (1992). A Parallel Branch-and-bound for solving large scale asymmetric travelling salesman problem. Mathematical Programming 55 (1-3): 17-33.

Murty K (1968). An Algorithm for ranking all the assignments in the order of increasing cost. *Operations Research* **16**(3): 682-687.

Naddef D (1992). The binested inequalities of the symmetric traveling salesman polytope. *Mathematics of Operations Research* **17**(4): 882-900.

Naddef D and Rinaldi G (1991). The Symmetric Travelling Salesman Polytope and its Graphical relaxation: Composition of Valid Inequalities. *Mathematical Programming* **51**(1-3): 359-400.

Nagata Y and Soler D (2012). A new genetic algorithm for the asymmetric traveling salesman problem. *Expert Systems with Applications* **39**(10): 8947-8953.

Ong H (1981). *Design and analysis of heuristics for some routing and packing problems*. Ph.D. Thesis: University of Waterloo.

Or I (1976). *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking*. Ph.D. Thesis: Northwestern University.

Padberg M and Hong S (1980). On the Symmetric Travelling Salesman Problem: A Computational Study. In Combinatorial Optimization: 78-107. Springer Berlin Heidelberg.

Padberg M and Rinaldi G (1987). Optimization of a 532 city Symmetric Travelling Salesman Problem using Branch-and-cut. *Operations Research Letters* **6** (1): 1-7.

Padberg M and Rinaldi G (1991). A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Travelling Salesman Problems. *SIAM Review* **33**(1): 60-100.

Rosenkratz D, Sterns R and Lewis P (1977). An analysis of several Heuristics for the Travelling Salesman Problem. *SIAM Journal of computer* **6**(3): 563-581.

Shapiro D (1966). *Algorithms for the Solution of the Optimal Cost and Bottleneck Travelling Salesman Problems*. Sc.D Thesis, Washington University.

Tsai H K, Yang J M, Tsai Y F and Kao C Y (2003). Heterogeneous selection genetic algorithms for traveling salesman problems. *Engineering Optimization* **35**(3): 297-311.

Tsubakitani S and Evans J R (1998a). Optimizing tabu list size for the Travelling Salesman Problem. *Computers and Operations Research* **25**(2): 91-97.

Tsubakitani S and James R E (1998b). An empirical study of a new metaheuristic for the traveling salesman problem. *European Journal of Operational Research* **104**(1): 113-128.

Turkensteen M, Ghosh D, Goldengorin B and Sierksma G (2006). Iterative patching and the asymmetric traveling salesman problem. *Discrete Optimization* **3**(1): 63-77.

Volgenant T and Jonker R (1982). A branch and bound algorithm for the symmetric travelling salesman problem based on the 1-tree relaxation. *European Journal of Operational Research* **9**(1): 83-89.

Voudouris C and Tsang E (1999). Guided local search and its application to the traveling salesman problem. European Journal of Operational Research **113**(2): 469-499

Wright J and Clarke G (1964). Scheduling of Crews from a central depot to a number of delivery points. *Operations Research* **12**(4): 568-581.

Zhou Y., Luo Q., Chen H., He A. and Wu J. (2015). A discrete invasive weed optimization algorithm for solving traveling salesman problem. *Neurocomputing* **151**, March: 1227–1236.