Evolutionary Algorithm for the k-Interconnected Multi-Depot Multi-Traveling Salesmen Problem

Carlos E. Andrade
University of Campinas
Av. Albert Einstein, 1251
Campinas, SP, Brazil
andrade@ic.unicamp.br

Flávio K. Miyazawa
University of Campinas
Av. Albert Einstein, 1251
Campinas, SP, Brazil
fkm@ic.unicamp.br

Mauricio G. C. Resende
AT&T Labs Research
180 Park Avenue
Florham Park, NJ, USA
mgcr@research.att.com

ABSTRACT

We introduce the k-Interconnected Multi-Depot Multi-Traveling Salesmen Problem, a new problem that resembles some network design and location routing problems but carries the inherent difficulty of not having a fixed set of depots or terminals. We propose a heuristic based on a biased random-key genetic algorithm to solve it. This heuristic uses local search procedures to best choose the terminal vertices and improve the tours of a given solution. We compare our heuristic with a multi-start procedure using the same local improvements and we show that the proposed algorithm is competitive.

Categories and Subject Descriptors

I.2.8 [Artificial Intelligence] [**Problem Solving, Control Methods, and Search**]: [Heuristic methods]

General Terms

Algorithms, Experimentation, Performance.

Keywords

Network Design; Routing Problems; Biased Random-Key Genetic Algorithms.

1. INTRODUCTION

Many practical routing and scheduling problems seek to determine sequences of actions subject to one or more constraints. Most of these problems can be modeled as variant of the Traveling Salesman Problem (TSP). Given an undirected weighted graph, the TSP consists in finding a Hamiltonian cycle of minimum weight. Other practical applications extend this concept, requiring the salesman to visit a given subset of the vertices. Problems such as the *Vehicle Routing Problem* and the *Location Routing Problem* are examples of these extensions.

We propose a variant of this family of problems called the *k-Interconnected Multi-Depot Multi-Traveling Salesmen Problem* (k-IMDMTSP) where we choose a subset of *terminal vertices* of size k from a base graph and connect these vertices with a cycle (called the *inner cycle*). Furthermore, for each terminal vertex, we build a cycle that begins and ends at the terminal and covers a subset of vertices from the remaining graph (called the *outer cycles*). All outer cycles must be pairwise disjoint, span all vertices, and have at most C vertices (C is the *capacity* of the outer cycle). The objective is to minimize the sum of edge weights of the inner and outer cycles. This problem occurs in the design of ring networks used to prevent communication failures in presence of single link-fault events. Another application is in transportation networks that use several echelons with different external constraints, such as vehicle types, transportation rules, and service-level agreements.

The paper is organized as follows. In Section 2 we present related work. In Section 3 we formalize the k-IMDMTSP. We then address biased random-key genetic algorithms in Section 4 and follow up with a description of our heuristic in Section 5. Experimental results are provided and discussed in Section 6. Finally, concluding remarks are made in Section 7.

2. RELATED WORK

To the best of our knowledge, this is the first paper to deal with the k-IMDMTSP. A closely related problem is the Multi-Depot Multi-Traveling Salesmen Problem (MDMTSP) introduced in [21]. In the MDMTSP, an unlimited number of salesmen have to visit a set of customers using routes that can be based on a subset of available depots. In [16], a 2-approximation algorithm is presented for the Generalized MDMTSP, a variant where each depot has m salesmen available but at most p can be used. In [31], a Lagrangean-based algorithm is presented to deal with a variation of GMDMTSP where each tour must have at least three vertices.

Another family of problems related to the k-IMDMTSP are the Location Routing Problems (LRP). In LRPs, we seek to find a set of routes based at a given set of depots to serve customer demands with vehicles of limited capacity. Depots have opening costs and limited capacity to serve its routes. There is an abundant literature with respect to these problems and their variations and a recent survey can be found in [20]. In [23], a hybrid heuristic based on Lagrangean relaxation and tabu search is developed for the LRP. This heuristic is considered to be the most effective to date for the LRP. The best results of an exact algorithm are from [5]

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

where a branch-and-cut procedure is developed with a series of new inequalities tailored for the LRP.

Another interesting problem related to the k-IMDMTSP is the 2-Echelon Vehicle Routing Problem (2E-VRP), introduced in [22]. In this problem, freight is dispatched from a distribution center on long-haul vehicles to small satellite depots. From the satellite depots, the freight is transported in small vehicles to their final destinations. In the first echelon, routes are set between the distribution center and the satellites. In the second, routing occurs between the satellites and customers. Each route must start at the distribution center (resp. satellite) and visit a subset of satellites (resp. customers), to supply the required demand.

The main difference between the k-IMDMTSP and these related problems is that the latter have a specific set of vertices where the depots are located, while in the former, each vertex is a potential depot/terminal. Furthermore, with respect to the MDMTSP and LRP, the k-IMDMTSP requires an extra "inner cycle" connecting the terminals. With respect to LRP and 2E-VRP, the k-IMDMTSP permits only a single cycle connecting the terminals and a single outer cycle (possibly empty) for each terminal. The k-IMDMTSP plays an important role in networking planning specially because it enables flexibility in the optimization of the internal gateway locations and leads to cheaper networks with link-failure tolerance.

To obtain good solutions for the k-IMDMTSP, we implemented a biased random-key genetic algorithm (BRKGA) [10]. Our choice was mainly grounded on recent successes with classical hard combinatorial optimization problems such as covering [25], packing [11], scheduling [9], among others [10]. The major features that distinguish BRKGAs from traditional genetic algorithms are: 1) the standardized chromosome encoding that adopts a vector with m uniformly drawn random keys (*alleles*) over the interval $[0, 1]$; and 2) a well-defined evolutionary process which uses parameterized uniform crossover [29] for intensification and substitutes the application of the mutation operator on existing chromosomes with newly introduced *mutants* – defined as m-vectors of (uniformly drawn) random keys – for diversification. The only connection between this general-purpose metaheuristic and the underlying problem occurs when a chromosome is *decoded*, i.e., when a solution is constructed from a chromosome. Analogously, decoders indirectly map the chromosome space $[0, 1]^m$ into the set of feasible solutions to the optimization problem.

3. DEFINITIONS

Let $G = (V, E)$ be an undirected complete simple graph with vertex set V and edge set E, where each $e \in E$ has a weight or cost $c_e \in \mathbb{R}^+$. W.l.g, consider $V = \{1, 2, \ldots, n\}$ where $n = |V|$. Consider also $V[A]$ and $E[A]$, respectively, as subsets of vertices and edges induced by structure A. A cycle $\mathcal{C}$ of size t is subgraph of G such that vertices $V[\mathcal{C}]$ form a sequence of distinct vertices $v_1, \ldots, v_t$ and edge $(v_i, v_{(i \bmod t)+1}) \in E[\mathcal{C}]$ for all $1 \leq i \leq t$. Note that, by this definition, a cycle of size two is a simple edge. We also consider a cycle of size 1 (without edges) and call it *empty cycle*.

In the k-IMDMTSP, we seek a subset of vertices $T \subseteq V$, called *terminals*, such that the size of T is exactly a constant k. A cycle I formed by the terminals T is called the *inner cycle*. For each terminal $t \in T$, an *outer cycle* O_t is built such that for all $t' \in T$ and $t \neq t'$, $V[O_t] \cap V[O_{t'}] = \emptyset$, i.e., the outer cycles are pairwise disjoint. We assume that the terminal $t \in V[O_t]$ is accounted for in the size of O_t. Therefore, $|V[O_t]| \geq 1$, for all $t \in T$. Each external cycle must have no more than C vertices. A valid solution $\mathcal{S}$ for the *k-Interconnected Multi-Depot Multi-Traveling Salesmen Problem* is a set of terminals T, of size k, connected by an inner cycle I and a collection of spanning outer cycles O_t such that $|V[O_t]| \leq C$, for all $t \in T$. The cost of a solution $\mathcal{S}$ is defined as

$$\sum_{e \in E[\mathcal{S}]} c_e + \sum_{e \in E[\mathcal{C}]} c_e \tag{1}$$

where $\mathcal{C} = \{O \in \{O_{\forall t \in T}, I\} : |E[O]| = 1\}$. The second term of the sum corresponds to the edges that belong to a cycle of size 2. In this case, we consider a round trip and the cost of these edges are doubled. The objective of the k-IMDMTSP is to minimize this cost.

We can address degenerate cases for the k-IMDMTSP. With respect to the external cycles, we have two cases. In the first, the outer cycle has size one indicating that only the terminal is in the cycle. We call this case an *empty cycle*. In the second case, the outer cycle has size two, indicating that it has a terminal and a non-terminal vertex connected by an edge. Such case is called *fur*. Figure 1 shows examples of a simple case and the two degenerate cases using a TSP instance and six terminals. The edges of inner cycle are dashed while the edges of outer cycles are solid. In Figure 1(a), all terminals are attached by an outer cycle. In Figure 1(b), vertices 1, 9, and 19 are not attached to any cycle and we therefore consider they lead empty cycles. In Figure 1(c), each terminal is connected to a single vertex outside the inner cycle (having the appearance of fur of the inner cycle).

With respect to the inner cycle, one can address three special cases all related to parameter k. For $k = 2$, the inner cycle is formed by one edge as a "bridge" between two outer cycles. For $k = 1$ or $k = n$, we have a single cycle (either an outer cycle or an inner cycle, respectively). If the capacity $C \geq n$, the problem reduces to the Traveling Salesman Problem. In fact, using this reduction, one can easily argue that k-IMDMTSP is $\mathcal{NP}$-hard.

Note that the maximum capacity of the outer cycles and the number of terminals are closely related. A low capacity requires a greater number of short cycles to cover all the vertices in a valid solution. Therefore, feasible instances have the number of terminals and the capacity of the outer cycles related according to $C \geq \left\lceil \frac{n}{k} \right\rceil$.

4. BIASED RANDOM-KEY GENETIC ALGORITHMS

Algorithm 1 summarizes a typical BRKGA framework. Basically, we generate p chromosomes as initial individuals[1] using vectors with m uniformly drawn random keys over the interval $[0, 1]$. In each iteration, a problem-specific *decoder* extracts the fitness of the chromosomes. To build a new generation, we copy the p_e best individuals (the *elite* set), add p_μ random chromosomes (the *mutants*) and generate, by applying the crossover operator, $p - p_e - p_\mu$ offspring. Crossover is done between a random individual from the

[1]The pair formed by a chromosome and its fitness is called *individual*.

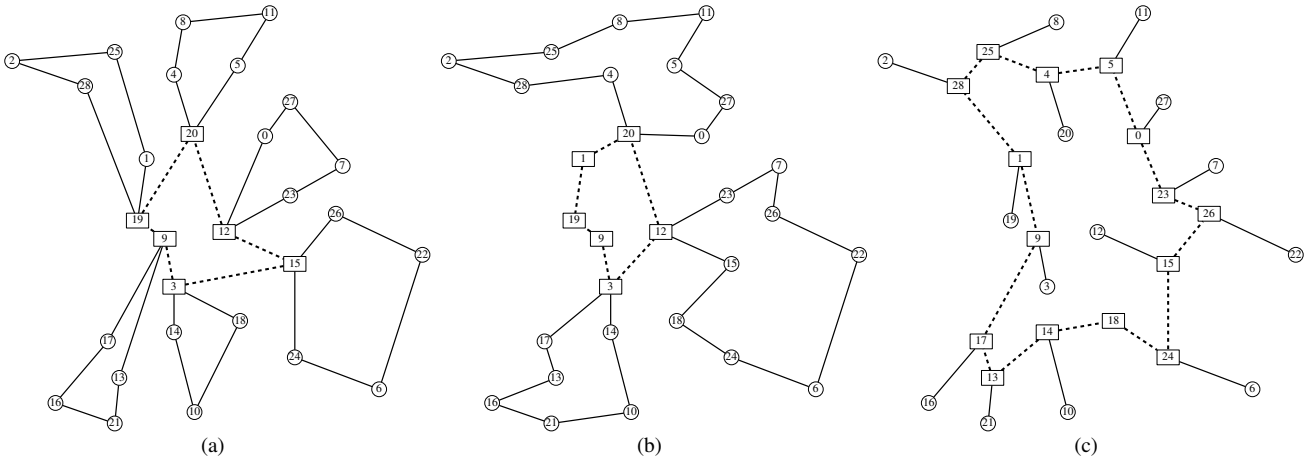

Figure 1: Examples of a simple and degenerate cases of k-IMDMTSP on `bayg29` TSP instance.

elite set and an individual from the remainder of the population: an offspring is generated by *mating*, where we take each allele from the elite parent with probability ρ_e or from the other parent with probability $1 - \rho_e$. With $\rho_e = 0.5$, the standard uniform crossover occurs. With $\rho_e > 0.5$ by definition, intensification happens at two levels: when parents are selected, because one is drawn from the elite set, and when offspring are conceived, because their alleles are inherited from the elite parent with greater probability. Diversification happens with the introduction of mutants at each generation, since they are vectors with t uniformly drawn random keys. The usual mutation operators on individual genes are not employed by the BRKGA. Observe that the above scheme prevents infeasibility since, by definition, the resulting chromosomes – both offspring and mutants – are always vectors of random keys over $[0, 1]$. One characteristic of BRKGAs is the biased selection of keys in the crossover process which enables the offspring to inherit characteristics of the elite parent with controlled probability. In [10], several experiments corroborate with this observation.

Algorithm 1: BRKGA scheme.

```
1 Generate the initial population P;
2 while a stopping criteria is not reached do
3     Decode each chromosome of P and extract their
      solutions and fitness;
4     Sort the population P in non-increasing order of
      fitness. Consider the top p_e individuals as the elite
      group E;
5     Copy E to the next generation Q, unaltered;
6     Add p_μ randomly-generated new chromosomes
      (mutants) to Q;
7     Generate p − p_e − p_μ chromosomes (offspring) by
      parameterized crossover, selecting a random parent
      from E and another from P \ E. Add them to Q;
8     P ← Q;
9 return best individual found.
```

A common approach to genetic algorithms is to evolve several populations in parallel and exchange their best individuals given a certain number of generations. This improves the variability of individuals and usually, speeds up the convergence and avoids getting stuck in local optima. It is straightforward to adapt the BRKGA framework to evolve parallel populations and exchange their best individuals.

In conclusion, the parameters that must be specified beforehand are the problem-dependent size of the chromosomes m, the size of the population p, the size of elite set p_e, the number of mutants p_μ introduced in each generation, and the inheritance probability ρ_e. If using parallel populations, we must set the number of populations Π and the generation threshold to exchange the best individuals. Advice for the parameter setup can be found in [10].

5. BRKGA FOR THE K-IMDMTSP

To use a BRKGA, we need to represent a solution with a chromosome and implement a decoder using this representation. The decoding phase is the only connection between the BRKGA and the problem itself. In this phase, it is also possible to intensify the search applying a local search procedure on the decoded solution. Note that to find a better solution in the neighborhood implies in the change the allele values of the chromosome to reflect this new solution. The fitness of a decoded chromosome is the cost of the cycles found. In following sections, we describe these particularities.

5.1 Representation

In most BRKGA implementations, a chromosome is a real vector $\mathbf{v}' \in [0, 1]^m$ following the procedure given in [4]. Here, we use an integer vector $\mathbf{v} \in \mathbb{N}^m$. The projection of $\mathbf{v}' \to \mathbf{v}$ is done with the function $f(\mathbf{v}') = \lceil \alpha \mathbf{v}' \rceil$ with $\alpha > 1$ sufficiently large. Recall that k is the number of terminals, n is the number of vertices, and C is the maximum number of vertices in an outer cycle.

A chromosome is a $(k + n)$-vector $\mathbf{v}$ of integers. The first alleles $\mathbf{v}_1, \ldots, \mathbf{v}_k \in \{1, \ldots, C\}^k$ represent the size of the outer cycles. The remaining alleles $\mathbf{v}_{k+1}, \ldots, \mathbf{v}_{k+n} \in \mathbb{N}^n$ represent the position of a vertex in an outer cycle.

To extract a solution from this chromosome, we sort the alleles $\mathbf{v}_{k+1}, \ldots, \mathbf{v}_{k+n}$ in a non-increasing order, generating a permutation π of the vertices. Vertices $\pi_1, \ldots, \pi_k$ represent the terminals T. Each terminal π_i, for $i = 1, \ldots, k$, has an outer cycle formed by vertices $\pi_{k+o(i)+j}$, for $j = 1, \ldots, \mathbf{v}_i$

and $\mathbf{v}_i \geq 2$, such that

$$o(i) = \begin{cases} o(i-1) + \mathbf{v}_{i-1} - 1 & \text{if } i > 1, \\ 0 & \text{otherwise.} \end{cases}$$

The function $o(\cdot)$ indicates the offset in the alignment of cycles in the permutation π. Note that $\mathbf{v}_i = 1$ indicates terminal i is alone in an empty cycle.

For instance, consider the chromosome of Figure 2, where $n = C = 6$ and $k = 3$. To extract a solution, we sort the indices representing the vertices, using their corresponding keys and generate the permutation π. The first terminal is the vertex 3 and it leads the cycle O_3: $3 \rightarrow 6 \rightarrow 4$. The second cycle O_5 is an empty cycle formed only by terminal 5. Cycle O_1 is a degenerate cycle formed by terminal 1 and vertex 2.

The advantage of this representation is that any modification in the values of keys $\mathbf{v}_{k+1}, \ldots, \mathbf{v}_{k+n}$ will not destroy the feasibility of the solution extracted from the chromosome. This approach was used in several papers, such as [27, 28]. On other hand, arbitrary modifications of keys $\mathbf{v}_1, \ldots, \mathbf{v}_k$ can result in invalid cycle sizes. In this case, it is necessary to use the repair procedure described in Section 5.2.

Note that distinct chromosomes may decode to the same solution either by rotation of the position of the alleles or by different allele values in the same order. However, this in unlikely to occur because of the range of the integers used in practice.

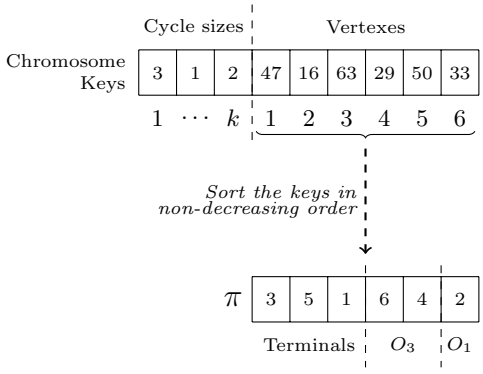

Figure 2: Example of solution extraction.

5.2 Decoding a solution

In BRKGAs, the decoder extracts a solution from the chromosomes as well as tries to improve the solution with some local search procedure. Using the representation of the previous section requires that we guarantee that the sizes of the outer cycles are correct, i.e., that the total sum of the keys equals the number of vertices. Algorithm 2 increments or decrements the sizes of each cycle based on the difference between the sum of the key values and the number of vertices. Note that the modifications are done among randomly selected cycles with the objective of preserving the "structures" inherited from the parents. Therefore, we avoid deep changes in a particular cycle. The complexity of this repair procedure is $O(|kC - n|)$.

After the repair phase, the decoder proceeds with solution extraction as described in Section 5.1. At this point, it is possible (although rare) that two or more keys have identical values. This could generate different solutions with different values from the same chromosome. To avoid this situation, the decoder modifies the keys to have different values and, therefore, guarantee a unique solution. However, note that this modification must be done only on the keys corresponding to the node ordering. The keys corresponding to the cycle size are kept untouched. Algorithm 3 summarizes the entire process, including the local search procedures described in next section.

Algorithm 2: adjustCycleSizes($\mathbf{v}, k, n, C$)

Input: a vector $\mathbf{v} \in \mathbb{N}^{k+n}$, integers $k, n, C \in \mathbb{N}$
Output: modified $\mathbf{v}$

```
1  Let A = {1, ..., k};
2  diff ← Σ_{i∈A} v_i − n;
3  if diff < 0 then sign ← −1;
4  else sign ← 1;
5  while diff ≠ 0 do
6      Choose i from A uniformly;
7      if (v_i > 2 and sign < 0) or (v_i < C and sign > 0)
       then
8          v_i ← v_i + sign;
9          diff ← diff + sign;
10     else
11         A ← A \ {i};
```

5.2.1 Local Improvements

Local improvement procedures have been shown to help convergence of genetic algorithms. Potentially, each chromosome can be in the neighborhood of a different local minimum. In such a case, local search would be able to reach these minima [13].

To perform local improvements, we explore two different neighborhoods. The first is the well-known "edge exchange" neighborhood obtained from 2-opt and 3-opt movements (it is also known as the "λ-opt" neighborhood). Let (u, v) and (u', v') be two edges of the cycle (appearing in the orientation). If $c_{(u,v')} + c_{(u',v)} < c_{(u,v)} + c_{(u',v')}$, we remove (u, v) and (u', v'), and insert (u, v') and (u', v), performing a 2-opt move. Note that the path $v, \ldots, v'$ is reversed in this process becoming $v', \ldots, v$. A λ-opt move is done in a similar way but involves the exchange of λn edges. Lin and Kernighan [14] generalize these methods resulting in one of best heuristics for the traveling salesman problem. Their procedure is adaptive and at each step decides dynamically what type of λ-opt should be applied. In [17], the Lin-Kernighan heuristic was improved using, at the start of each iteration, a "guess" or "kick" tour instead of only doing restarts. The *double-bridge move*, cheaper than a 4-exchange move, was also proposed. This heuristic is known as the *Chained Lin-Kernighan procedure*. In [12] several refinements of Lin-Kernighan are described. A sequential 5-opt step, restricted to a neighborhood of five α-nearness members, is used. This is computed using 1-trees on a modified weight distance matrix.

To explore the "edge exchange" neighborhood, we apply the Chained Lin-Kernighan heuristic to each outer cycle as well as the inner cycle but with a limited number of iterations. One can note that an exchange in positions of vertices in a cycle implies that the allele values of the vertices must

Algorithm 3: Decoder

Input: a vector $\mathbf{v} \in \mathbb{N}^{k+n}$, integers $k, n, C \in \mathbb{N}$
Output: modified $\mathbf{v}$, the *fitness* of $\mathbf{v}$

1 `adjustCycleSizes(`$\mathbf{v}, k, n, C$`);`
2 Sort the alleles $\mathbf{v}_{k+1}, \ldots, \mathbf{v}_{k+n}$ in non-decreasing order, generating a vector $\mathbf{s}$ of sorted keys and a permutation π of the vertices;
3 Keeping the order, modify the values of $\mathbf{s}$ to ensure all different;
4 Extract the inner cycle I and the outer cycles $\mathcal{O}$ using π and $\mathbf{s}$;
5 **foreach** $O \in (\mathcal{O} \cup \{I\})$ **do**
6 `  ChainedLinKernighan(`O`);`
7 **foreach** $O_t \in \mathcal{O}$ **do**
8 $\quad t' \leftarrow$ `findBestTerminal(`O_t`);`
9 $\quad O_{t'} \leftarrow O_t$;
10 $\quad I \leftarrow (I \setminus \{t\}) \cup \{t'\}$; `// Keeping the order`
11 $\quad \mathcal{O} \leftarrow (\mathcal{O} \setminus \{O_t\}) \cup \{O_{t'}\}$;
12 `ChainedLinKernighan(`I`);`
13 $i \leftarrow 1$;
14 **foreach** $O_t \in \mathcal{O}$ **do**
15 $\quad \mathbf{v}_{k+t} \leftarrow \mathbf{s}_i$;
16 $\quad i$++;
17 $\quad$ **foreach** $v \in O_t$ **do**
18 $\quad\quad \mathbf{v}_{k+v} \leftarrow \mathbf{s}_i$;
19 $\quad\quad i$++;
20 **return** *the solution value according to Equation* (1)

be also swapped. In particular, the modification in the positions of terminals in the inner cycle must be done carefully so that the remaining structure of the chromosome is not destroyed. Lines 5 and 6 of Algorithm 3 summarize this procedure.

The second neighborhood "rotates" the outer cycles looking for new terminal vertices capable of reducing the cost of the solution. Let t be the terminal of outer cycle O, t_s, and t_p be, respectively, the successor and predecessor terminals of t in the inner cycle I. Rotating cycle O means finding a vertex $t' = \text{argmin}_{t'' \in O} c_{(t_p,t'')} + c_{(t'',t_s)}$, i.e., finding a vertex to be the new terminal that minimizes the cost of the inner cycle. Note that the outer cycles are kept intact but the terminal and, consequently, the cost of inner cycle is changed. In fact, the vertex exchange can result in a new inner cycle to which is applied the Chained Lin-Kernighan procedure. Lines 7–11 of Algorithm 3 summarize this procedure. One can note that these operations can be done efficiently, but we prefer to describe it this way for notational convenience.

It is important to note that all these modifications on the solution implies that the keys of the corresponding chromosome must be modified to reflect the new solution. Lines 14–19 copy back the key values from ordered vector $\mathbf{s}$ to reflect these modifications.

5.3 Initial Population

The common approach to initialize the population of a BRKGA is to generate the chromosomes with uniformly drawn random keys over the interval $[0, 1]$. This results in highly heterogeneous individuals that may slow down the convergence of the algorithm. On the other hand, these individuals may lie in different local minima neighborhoods and help avoid premature convergence of the algorithm. In an attempt to speed up the search, we use a fast heuristic to generate some good initial solutions and then, complete the initial population with random chromosomes. An advantage of using such individuals in the starting population is that they are probably closer to a good solution than most random individuals.

The heuristic is based on k-means clustering algorithm that partitions the vertices in k clusters such that each vertex belongs to the cluster with the nearest mean [15]. This is a well-know algorithm, used in several fields, such as machine learning and computational geometry. Each cluster generated by k-means is considered as an outer cycle and the terminal is the vertex closest to the mean of the cluster. The vertices are taken in arbitrary order since the cycles will be optimized in the decoding procedure. Although the cost of this solution is usually not very good, it carries some geometric information about the distribution of the vertices and it is likely that some parts of these clusters will be part of the final solution. One can note that this approach can be used only in geometric instances. In [3], a similar approach is used to solve the Capacitated Location-Routing Problem.

The encoding of the chromosome from this solution is done in the following way: a vector $\mathbf{s}'$ is generated with n random keys, sorted in non-increasing order. The first k positions of the chromosome are associated with the size of the clusters taken in some arbitrary but fixed order, say $t_1, \ldots, t_k$. Using this same order, we associate each terminal t_i with a key of $\mathbf{s}'$. For each cluster O_{t_i}, we associate the vertices of this cluster with the remaining keys of $\mathbf{s}'$ in the order they appear. This ensures the compatibility and uniformity with the BRKGA framework.

6. EXPERIMENTAL RESULTS

We conducted several experiments with the goal of evaluating the effectiveness of the BRKGA to find good solutions on instances of varying sizes. By adjusting the BRKGA framework, we also implemented a simple Multi-Start Heuristic (MSH) using the same local search procedure as used by the BRKGA and compare the results obtained by both heuristics.

6.1 Computational environment

We implemented the algorithms in the C++ programming language using the GNU `g++` compiler version 4.4.2. Random numbers were generated by an implementation of the Mersenne-Twister [18]. We use the Chained Lin-Kernighan implementation of the Concorde TSP Solver [1] to do the local search. The experiments were conducted on a quad-core Intel Xeon 2.4 GHz CPU with 8 GBytes of RAM running GNU/Linux. Running times reported are UNIX real wall-clock times in seconds, excluding the time to read the instance. Each run was limited to 3,600 seconds.

6.2 Algorithm settings

The proposed algorithm was implemented on top of the BRKGA API [30], which implements all of the problem-independent components described in Section 4. We set the population size to $p = \min(10n, 100)$, where n is the number of vertices, the size of elite set of individuals to $p_e = \lceil 0.30p \rceil$,

and the number of mutants introduced at each generation to $p_m = \lfloor 0.15p \rfloor$. The probability of inheriting each allele from the elite parent was set to $\rho_e = 0.70$. We evolved $\Pi = 3$ populations simultaneously and once every 100 generations each population exchanged its two best solutions with the other populations. After 500 iterations without improvement, all three populations are reset to randomly generated individuals. This setup followed suggestions of [10].

To create the Multi-Start Heuristic (MSH), we set the size of the elite set to $p_e = 1$ and the number of mutants in each generation to $p_m = p - 1$. This setting disables any type of offspring generation and only keeps the best solution found throughout the optimization. We also disabled the initial heuristic leaving just the random chromosome generation.

At each iteration of the decoder, the Chained Lin-Kernighan heuristic was executed once for each cycle, using no time or tour length bounds. The stall counter was set to at most 1000 iterations, limiting the number of 4-swaps without progress. We use a random walk as the kicking strategy. This perturbs the Lin-Kernighan solution creating three independent random walks of a given length from a vertex v in the neighborhood of this solution. According to [8], this type of kicking strategy presents better results than other strategies suggested in [2].

In addition to stopping after 3,600 seconds, we also stop if 1,000 generations go by without improvement of the best solution. The algorithm decodes the chromosomes in parallel, using four cores. It was necessary to make slight modifications in the Concorde code to support multi-threading.

6.3 Instances

Our experiments use TSP instances from the TSPLIB repository [24] as base graphs. We build five scenarios based on different numbers of terminals k and cycle capacity C. We consider the 63 TSP instances with fewer than 700 vertices. We exclude instance `linhp318` due to problems in the loading routine of Concorde.

The first scenario varies the size of the inner cycle. We argued earlier that the extremal values $k = 1$ and $k = n$ reduce the k-IMDMTSP to the TSP (with capacity $C \geq n$). Therefore, we use other values of k in the experiments. We first set $k = \lceil 0.2n \rceil$ to build a small inner cycle. This is common, for instance, in transportation problems that deliver heavy truckloads to one or more depots. We then set $k = \lceil 0.5n \rceil$, promoting larger inner cycles. This occurs, for instance, when large optical fiber ring is used to maximize the throughput of a network. Our third variation is described below.

We also vary the capacity of the outer cycles looking for interesting instances. First, we set $C = \lceil n/k \rceil$, forcing each terminal to have a non-empty outer cycle. One can easily see that the sizes of these cycles differ by at most one. Second, we set $C = 2\lceil n/k \rceil$, allowing for the possibility of some empty or larger cycles. Note that $C \geq n$ leads to a solution with one outer cycle and one inner cycle.

Using the above settings, we built the following five scenarios for each of the 63 TSP graphs, generating 315 instances:

ST − small inner cycle: $k = \lceil 0.2n \rceil$ and tight outer cycles: $C = \lceil n/k \rceil$;

SL − small inner cycle and loose outer cycles: $C = 2\lceil n/k \rceil$;

LT − large inner cycle: $k = \lceil 0.5n \rceil$ and tight outer cycles;

LL − large inner cycle and loose outer cycles;

SQ − Inner and outer cycles of same size: $k = C = \sqrt{n}$.

It is important to note that, for the instance classes ST and SL, the capacities converge, respectively, to 5 and 10 for all $n \geq 17$, since the limits for those functions converge to these values. Similarly, for the instances in LT and LL, the capacities converge to 2 and 4, respectively. Therefore, we observe that the instance class LT corresponds to the "fur" instances exemplified in Figure 1(c).

6.4 Results and Discussion

For each instance, 30 independent runs of each algorithm were performed. This resulted in 1890 experiments per scenario, and 9450 experiments per algorithm. To compare the algorithms, we needed to scale the resulting costs since their magnitudes vary greatly from instance to instance. For each instance, we scaled the cost in the interval $[0, 1]$ using the minimum and maximum costs from both algorithms.

Figure 3 shows the distributions of the results for each scenario. The box plots show the smallest cost, the first quartile, the median cost, the third quartile and the largest cost. We can note that the medians and the quartiles of BRKGA results are systematically lower than the those of MSH, indicating better performance of BRKGA over MSH.

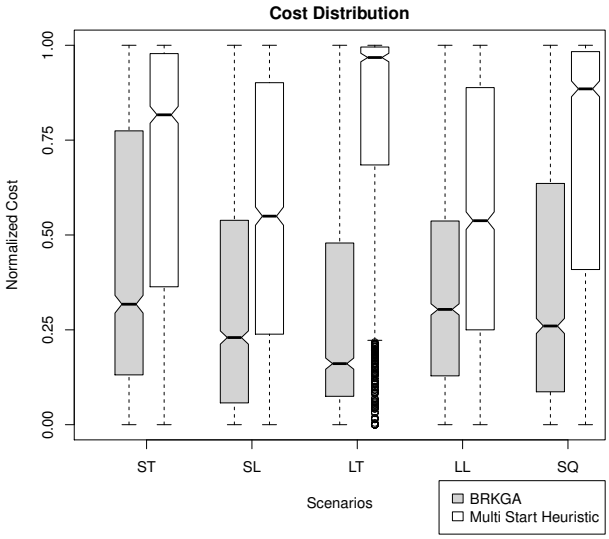

Figure 3: Boxplot of the medians and quartiles for each algorithm in each scenario.

To confirm these results, we tested the normality of these distributions using the Shapiro-Wilk test and we applied the Wilcoxon-Mann-Whitney test, considered more efficient than the t-test for distributions sufficiently far from normal and for sufficiently large sample sizes [6, 7]. For all tests, we assume a confidence interval of 95%. The Shapiro-Wilk tests revealed that no cost distribution fit normal distributions since the p-values for all tests are less than 2.5×10^{-28}. Therefore, we applied the Wilcoxon-Mann-Whitney which we show in Table 1. This test assumes as null hypothesis that the location statistics are equal in both distributions. The second column shows a difference in medians of the BRKGA and the MSH such that the medians of the BRKGA

Table 1: Median test for cost distributions.

Scen.	Diff. Loc.	Conf. Interv.	U	p-value
ST	-0.24	$[-0.27, -0.21]$	1.1×10^6	8.0×10^{-86}
SL	-0.23	$[-0.26, -0.21]$	1.1×10^6	2.9×10^{-82}
LT	-0.63	$[-0.66, -0.60]$	5.3×10^5	8.5×10^{-305}
LL	-0.19	$[-0.22, -0.17]$	1.2×10^6	1.1×10^{-67}
SQ	-0.36	$[-0.39, -0.33]$	9.4×10^5	6.1×10^{-140}
Overall	-0.32	$[-0.33, -0.31]$	2.4×10^7	0.00

Table 2: BRKGA means of best results using bootstrapping resampling (1000 replicates).

Dimension	Mean	Error	Conf. Interval	Min	Max
Iterations	2901.33	165.58	[2580, 3244]	199	17989
Init. Time	0.57	0.11	[0.34, 0.78]	0.00	23.34
Total Time	573.00	46.84	[480.50, 664.1]	1.30	3604.78

are less than those of the MSH. As all differences are in the respective confidence intervals and all p-values are much less than 0.05, we can conclude that the BRKGA presents significantly better results that does the MSH.[2]

Table 2 shows, for BRKGA over all instances, the mean number of iterations, the time taken by the initial heuristic, and the total time to reach the best solution. As suggested in [19], we use the bootstrapping method with resample size of 1000 and show the standard errors and 95% confidence intervals. Note the BRKGA uses an average of 2900 iterations to reach the best solution, showing that the genetic operations contribute to the convergence. We can also note that the minimum number of iterations is below 1,000, indicating that the maximum running time was used to stop the algorithm in certain large instances. In general, the initialization heuristic is very fast and does not add much in total running time.

Figure 4 shows the cumulative probability to reach a target cost over time. The target costs were set using the values of the best solution found for each instance. For each algorithm, we performed 30 runs for each instance (9,600 experiments per algorithm), limiting the maximum runtime to 1,800 seconds. The experiments with running time greater than or equal to 1,800 were pruned from the plot. Note that the cumulative probability curves for BRKGA are to the left and above those of MSH, indicating that it has a higher probability than MSH of reaching good solutions in a given time. Overall, BRKGA reached 53.19% of the target costs while MSH reach only 14.56%, with 1,000 seconds. Following the methodology proposed in [26], BRKGA has a 79.83% probability (with error $e = 0.001$) of finding a target solution in less time than MSN.

In both the LT and LL scenarios, both algorithms found few good solutions. We think that the main reason for this are the larger internal cycles required by these instances. In the other scenarios where the size of the cycles are more balanced, BRKGA reached more than 70% of the best solutions and outperformed MSH by a large margin. Another interesting observation is that on the SL and LL scenarios, the target values are found faster than on the other scenarios, indicating that the former are apparently easier than the latter.

[2]The full results for each scenario and instance can be found at `http://www.loco.ic.unicamp.br/results/kimdmtsp`.

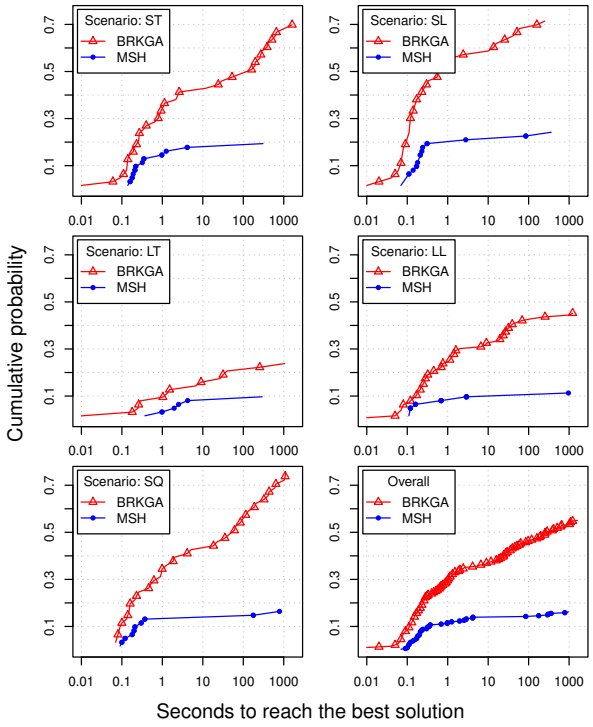

Figure 4: Time-to-target plots showing the cumulative probability to reach a determined cost in 1,800 seconds. The points corresponds to 0.2% of runs.

7. CONCLUDING REMARKS

In this paper, we introduced a new problem called the k-Interconnected Multi-Depot Multi-Traveling Salesmen Problem. The k-IMDMTSP is closely related to the Location Routing Problem and the 2-Echelon Vehicle Routing Problem. Unlike these problems, the k-IMDMTSP has the inherent difficulty of not having a fixed set of depots. To solve the k-IMDMTSP, we propose a biased random-key genetic algorithm (BRKGA) with a local search procedure in the decoding phase, using adaptations of classical procedures for the traveling salesman problem. We compare our algorithm with a multi-start heuristic (MSH) using the same local search algorithm, on five different scenarios with 63 instances each. We found that the BRKGA is significantly better than the MSH in our experiments, presenting a good performance that trades off solution quality with running time. Future research directions include extending our approach to address constraints like demands and flow limits and to compare it with other possible approaches like exact algorithms and their hybridizations.

Acknowledgments

Carlos E. Andrade is supported by São Paulo Research Foundation (FAPESP) grant 2010/05233-5. Flávio K. Miyazawa is supported by National Council for Scientific and Technological Development (CNPq).

8. REFERENCES

[1] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. Concorde TSP solver. http://www.tsp.gatech.edu/concorde.

[2] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The Traveling Salesman Problem:A Computational Study*. Princeton University Press, 2007.

[3] S. Barreto, C. Ferreira, J. Paixão, and B. S. Santos. Using clustering analysis in a capacitated location-routing problem. *European Journal of Operational Research*, 179(3):968–977, 2007.

[4] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal On Computing*, 2(6):154–160, 1994.

[5] J.-M. Belenguer, E. Benavent, C. Prins, C. Prodhon, and R. W. Calvo. A branch-and-cut method for the capacitated location-routing problem. *Computers & Operations Research*, 38(6):931–941, 2011.

[6] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, 2nd edition, 1980.

[7] M. P. Fay and M. A. Proschan. Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules. *Statistics Surveys*, 4:1–39, 2010.

[8] T. Fischer and P. Merz. Embedding a chained Lin-Kernighan algorithm into a distributed algorithm. In K. F. Doerner, M. Gendreau, P. Greistorfer, W. Gutjahr, R. F. Hartl, M. Reimann, R. Sharda, and S. Voß, editors, *Metaheuristics*, volume 39 of *Operations Research/Computer Science Interfaces Series*, pages 277–295. Springer US, 2007.

[9] J. F. Gonçalves, J. J. M. Mendes, and M. G. C. Resende. A random key based genetic algorithm for the resource constrained project scheduling problems. *Computers and Operations Research*, 36:92–109, 2009.

[10] J. F. Gonçalves and M. G. C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17:487–525, 2011. 10.1007/s10732-010-9143-1.

[11] J. F. Gonçalves and M. G. C. Resende. A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem. *Journal of Combinatorial Optimization*, 22:180–201, 2011.

[12] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[13] D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23(6):547–558, 1996.

[14] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973.

[15] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[16] W. Malik, S. Rathinam, and S. Darbha. An approximation algorithm for a symmetric generalized multiple depot, multiple travelling salesman problem. *Operations Research Letters*, 35(6):747–753, 2007.

[17] O. Martin, S. W. Otto, and E. W. Felten. Large-step markov chains for the tsp incorporating local search heuristics. *Operations Research Letters*, 11(4):219–224, 1992.

[18] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8:3–30, January 1998.

[19] C. C. McGeoch. *A Guide to Experimental Algorithmics*. Cambridge University Press, 2012.

[20] G. Nagy and S. Salhi. Location-routing: Issues, models and methods. *European Journal of Operational Research*, 177(2):649–672, 2007.

[21] P. Oberlin, S. Rathinam, and S. Darbha. A transformation for a heterogeneous, multiple depot, multiple traveling salesman problem. In *American Control Conference, ACC'09*, pages 1292–1297, June 2009.

[22] G. Perboli, R. Tadei, and D. Vigo. The two-echelon capacitated vehicle routing problem: Models and math-based heuristics. *Transportation Science*, 45(3):364–380, 2011.

[23] C. Prins, C. Prodhon, A. Ruiz, P. Soriano, and R. W. Calvo. Solving the capacitated location-routing problem by a cooperative lagrangean relaxation-granular tabu search heuristic. *Transportation Science*, 41(4):470–483, November 2007.

[24] G. Reinelt. TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95, 2008.

[25] M. G. C. Resende, R. F. Toso, J. F. Gonçalves, and R. M. A. Silva. A biased random-key genetic algorithm for the Steiner triple covering problem. *Optimization Letters*, pages 1–15, 2011.

[26] C. C. Ribeiro, I. Rosseti, and R. Vallejos. Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. *Journal of Global Optimization*, 54:405–429, 2012.

[27] F. Samanlioglu, M. B. Kurz, W. G. Ferrell, and S. Tangudu. A hybrid random-key genetic algorithm for a symmetric travelling salesman problem. *International Journal of Operational Research*, 2(1):47–63, 2006.

[28] L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, 174(1):38–53, 2006.

[29] W. M. Spears and K. A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

[30] R. F. Toso and M. G. C. Resende. A C++ application programming interface for biased random-key genetic algorithms. Technical report, AT&T Labs Research, 180 Park Avenue, Florham Park, NJ 07932 USA, 2011.

[31] S. Yadlapalli, W. Malik, S. Darbha, and M. Pachter. A lagrangian-based algorithm for a multiple depot, multiple travelling salesmen problem. In *American Control Conference, 2007. ACC '07*, pages 4027–4032, July 2007.