Use of Explicit Memory in the Dynamic Traveling Salesman Problem

Renato Tinós
Department of Computing and Mathematics
University of São Paulo
Ribeirão Preto, SP, Brazil
rtinos@ffclrp.usp.br

Darrell Whitley
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
whitley@cs.colostate.edu

Adele Howe
Department of Computer Science
Colorado State University
Fort Collins, CO, USA
howe@cs.colostate.edu

ABSTRACT

In the dynamic traveling salesman problem (DTSP), the weights and vertices of the graph representing the TSP are allowed to change during the optimization. This work first discusses some issues related to the use of evolutionary algorithms in the DTSP. When efficient algorithms used for the static TSP are applied with restart in the DTSP, we observe that only some edges are generally inserted in and removed from the best solutions after the changes. This result indicates a possible beneficial use of memory approaches, usually employed in cyclic dynamic environments. We propose a memory approach and a hybrid approach that combines our memory approach with the elitism-based immigrants genetic algorithm (EIGA). We compare these two algorithms to four existing algorithms and show that memory approaches can be beneficial for the DTSP with random changes.

Categories and Subject Descriptors

1.2.8 [**Artificial Intelligence**]: [Problem Solving, Control Methods, and Search]

Keywords

Dynamic Environments, Traveling Salesman Problem, Memory Schemes, Evolutionary Algorithms

1. INTRODUCTION

In the last decade, interest has been increasing in the area of evolutionary dynamic optimization (EDO) (Cruz et al. [4] and Nguyen et al. [14] offer excellent surveys). In a dynamic optimization problem (DOP), the fitness landscape may be modified at intervals, and as a consequence, the dynamic problem can be viewed as a sequence of stationary instances of an optimization problem [14].

One of the benchmark problems for EDO algorithms is a dynamic version of the traveling salesman problem (TSP). An instance of the static TSP is defined by a complete weighted graph $G(V,E)$, where $V = \{v_1, v_2, \dots, v_n\}$ is a set of n vertices (representing cities in the classic TSP) and E contains edges between every pair of vertices in V. An undirected graph defines a symmetric TSP, while a directed graph defines an asymmetric TSP. Each edge $e_{i,j} \in E$ between vertices $v_i, v_j \in V$ is associated with a weight $w_{i,j} \in \mathbb{R}^+$, indicating, in the classic TSP, the distance or travel cost between two cities. Only Hamiltonian cycles in G are allowed as solutions for the TSP. Considering v_{x_1} as the fixed start and final vertex, then the evaluation of a particular solution $\mathbf{x} = [x_1, x_2, \dots, x_n]^{\mathrm{T}} \in X$, specifying a permutation on $n-1$ vertices, is given by:

$$f(\mathbf{x}) = w_{x_n,x_1} + \sum_{i=1}^{n-1} w_{x_i,x_{i+1}} \qquad (1)$$

The objective of optimization for the static TSP is to find $\mathbf{x} \in X$ such that $f(\mathbf{x})$ is minimized.

In the dynamic TSP (DTSP), the set of vertices V and/or the weight matrix $\mathbf{W}$ can be modified while optimization is ongoing. In some research (e.g., [5, 10, 17, 13, 12]), only $\mathbf{W}$ is modified. For example, in [5], $w_{i,j}$ indicates the automotive travel time between cities v_i and v_j and the changes in $\mathbf{W}$ are caused by traffic. Simões and Costa [17] present a DTSP based on the scheme proposed in [21], where an edge of the best current solution is increased and then decreased by a given factor. In this case, the DOP has the time-linkage property, i.e., knowing the current best solution influences the future dynamics of the problem [14]. In [10], the DTSP arises from a network routing configuration problem with $n-1$ land-based stations and one artificial satellite with known periodic orbit. In this case, $w_{i,j}$ represents the distance between points v_i and v_j, which can be a land-based station or the artificial satellite, and the dynamism of the problem is caused by changing the position of the satellite, i.e., only the edges related to the vertex representing the satellite are changed during the optimization procedure.

Other research (e.g., [7, 21, 11]) allows the set of vertices V also to be modified. For example, in [11], the number of vertices or the weights between two vertices can change with a fixed probability.

Because changes in the DTSP are detectable, we ask: Can we consider the DTSP as a series of unrelated static instances and use efficient algorithms already developed for the static TSP? In other words, because the DTSP can be viewed as a sequence of static instances of the TSP (Figure 1), we ask if it is better to restart the optimization process after each change instead of using the knowledge

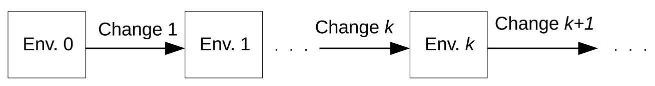


Figure 1: The DTSP as a sequence of static instances of the TSP, where each environment is created by changing the previous environment.

obtained before the change, which is usually done by most of the EDO algorithms. Section 2 explores this issue.

In Section 3, we investigate two types of changes in the DTSP. From simulations of a state-of-art heuristic for the static TSP applied to DTSP for both types of changes, we observe an interesting behavior: only some edges, from all possible edges, are inserted in and removed from the best solutions after the changes.

We hypothesize that explicit memory, which is used often by algorithms developed for dynamic environments in the case of cyclic changes, can be useful for the DTSP even in the case of random changes. A memory approach for the DTSP is then proposed in Section 4. A related issue is how to store and retrieve past solutions when the vertices of the TSP change. We propose a mechanism for this issue in Section 4. Section 5 presents an experimental comparison of memory approaches and other approaches for DOPs.

2. THE DYNAMIC TSP (DTSP)

In the DTSP, if the change is considerable, we can treat the landscapes before and after the change as two unrelated static instances of the TSP. However, if the change is small, the two graphs representing the problem before and after the change are similar and the knowledge obtained before the change can be employed to search for the solutions of the problem after the change.

The interval between changes in the DTSP leads to the following trade-off: Is it better to restart a new optimization process after the change in the problem or to continue the optimization process with the knowledge previously obtained? To address our question, it is important to check if the interval between the changes is adequate to find a good solution and whether information about good solutions can profitably be carried over to the changed problem.

If the application is the *classic TSP* (the lowest cost circuit for a vehicle around a set of cities or physically disparate locations), the restart approach with methods currently used for the static TSP is the best option. The reasons are: i) Current exact methods to solve TSPs, e.g., Concorde [3], are capable of finding a global optimum for instances of the symmetric TSP with hundreds of cities *in seconds*. In fact, some GPS devices incorporate exact solvers for the TSP with a few points [3]. ii) The change interval is generally large, when compared to the optimization time, for the classic TSP. For example, Google Maps updates traffic data generally every 15 minutes, which is clearly less than is needed by Concorde to find the new best solution even for problems with hundreds of cities. iii) Real-world classic TSPs like pick-ups and deliveries problems are small-scale TSPs [3]. Thus, EDO seems to be more appropriate for other dynamic vehicle routing problems (VRPs) where routes for more than one vehicle should be found.

However, DTSP can also encompass applications, such as robotics and telecommunication, where the change interval is short relative to the computational time required by exact solvers. For example, in [6], TSP algorithms are used to optimize the order in which a robotic manipulator inspects points in an object. The *atex* instances of the asymmetric TSP [2] derive from robotic motion planning. In [1], methods to solve the TSP are applied to minimize communication costs and energy consumption among sensors where each edge of the graph represents the average number of transmissions required to successfully complete a delivery. TSPLIB [15] also presents asymmetric TSPs which could have rapidly changing dynamic versions, such as scheduling and sequencing problems. Soler et al. [18] present a polynomial transformation of the VRP with time windows and time-dependent weights into an asymmetric TSP; an exact solver is then used to find the optimal solutions, which requires hours or days of computation for some instances.

We consider two types of the DTSP. In both, because the weight matrix $\mathbf{W}$ is assumed to be available, we can detect when and where the change occurs and identify the solutions that are affected by the changes. This important property does not hold in several of the DOPs presented in the literature.

For example, Mavrovouniotis et al. [13] present a benchmark problem generator that is used in the *IEEE WCCI-2014 Competition on Evolutionary Computation for Dynamic Optimization Problems*. In the DTSP produced by this generator, some vertices of the TSP are swapped at every change point, producing a permutation of the weight matrix $\mathbf{W}$. In this way, the generator supports setting the change frequency (by the number of fitness evaluations between changes) and severity (by the percentage of swapped cities); also the global optimum is known during the optimization process, what is very useful for comparing EDO algorithms. However, the authors concede that the dynamism produced by this benchmark generator does not necessarily reflect situations in real-world TSPs [13]. If one knows how the weight matrix $\mathbf{W}$ is permuted when the problem changes, it is not difficult to develop algorithms that efficiently exploit this property by correctly permuting past best solutions.

In a real situation, the changes in the weight matrix are known because $\mathbf{W}$ is used to compute the fitness of the solutions, and this knowledge can be explored in the analysis of the fitness landscape changes of the problem and when developing algorithms for the DTSP.

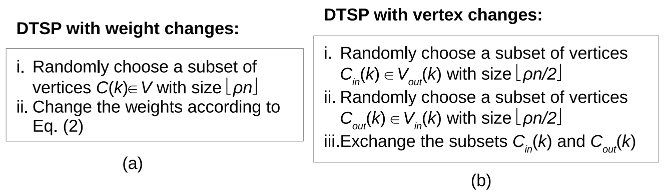
3. DTSP TYPES

Two components of the graph representing the TSP can change: the weights on the edges and the set of vertices.

3.1 DTSP with Weight Changes

Def. 1: DTSP with Weight Changes incurs at least one modification to $\mathbf{W}$ during the optimization process.

As depicted in Figure 2(a), all the weights of a subset of $\lfloor \rho n \rfloor$ vertices are randomly modified at each change point. Thus, the parameter ρ controls the change severity by defining the percentage of weights modified each time. At each change point, a subset of edges is randomly chosen according



DTSP with weight changes:

i. Randomly choose a subset of vertices $C(k) \in V$ with size $\lfloor \rho n \rfloor$
ii. Change the weights according to Eq. (2)

(a)

DTSP with vertex changes:

i. Randomly choose a subset of vertices $C_{in}(k) \in V_{out}(k)$ with size $\lfloor \rho n/2 \rfloor$
ii. Randomly choose a subset of vertices $C_{out}(k) \in V_{in}(k)$ with size $\lfloor \rho n/2 \rfloor$
iii. Exchange the subsets $C_{in}(k)$ and $C_{out}(k)$

(b)

Figure 2: Types of DTSP: a) DTSP with weight changes: for each change k, the weights of the edges of a random subset of vertices $C(k) \in V$ are modified according to Eq. 2. b) DTSP with vertex changes: in environment k, the set of vertices V is split into two subsets ($V_{in}(k)$ and $V_{out}(k)$) with equal size. The graph for environment k is defined only by the vertices in $V_{in}(k)$. Each change k removes a random subset of vertices $C_{out}(k)$ from $V_{in}(k)$ and inserts another random subset of vertices $C_{in}(k) \in V_{out}$.

to a uniform distribution, while the weights of this subset of vertices are randomly modified according to a normal distribution. In this way, the fitness of all solutions of the search space is modified. In the DTSP with weight changes, it is possible that the best solution may remain the same. However, as more edges are modified, it becomes less likely that the same solution will be best before and after the changes.

The changes in the weights are given by:

$$w_{ij}(k+1) = \begin{cases} w_{ij}(0) + p_{ij}(k), & \text{if } v_i \in C(k) \\ w_{ij}(k), & \text{otherwise} \end{cases} \tag{2}$$

where $p_{ij}(k)$ is a random number generated from a normal distribution with zero mean and standard deviation equal to $0.2 * w_{ij}(0)$ at change k, $w_{ij}(0)$ is the weight between vertices v_i and v_j for the static instance of the TSP used to generate the DTSP (i.e, in environment 0), and the subset $C(k)$ contains $\lfloor \rho n \rfloor$ vertices randomly chosen at change k.

If the change frequency is not high, i.e., the time between changes is not small, state-of-the-art static TSP heuristics can be effectively applied to each environment of the DTSP (Figure 1). To test this conjecture, we applied the well known LKH heuristic [8] to environments generated by following the process in Figure 2. LKH is a state-of-art heuristic for the TSP, capable of producing very impressive results both for symmetric and asymmetric TSPs [8, 9]. The environments were generated by 500 consecutive applications of changes to the asymmetric static instance *atex5* [2] in environment 0 with $\rho = 0.1$; *atex5* has $n = 72$ vertices.

To show how the best solutions change with the weight changes, Figure 3 presents the number of times that each possible edge of the TSP was added to the best solution found by LKH after a change, i.e., the edge was not present in the best solution for environment $k-1$, but was present in the best solution for environment k (the results were sorted from the edge that was included the most times to the edge that was included the fewest times).

From Figure 3, we can observe that only a small part of all possible edges were added to the best solutions after the changes. Some edges were included (after being removed) in the best solution more than 50 times. Most of the possible edges were not added to the best solution after any change. This occurs because some expensive edges overly degrade the evaluation of the solutions and so, are never (or rarely) found

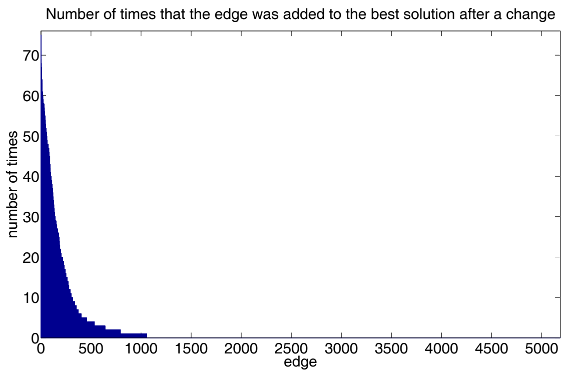


Figure 3: Modifications in the best solution found by LKH after changes in the weights of static instance *atex5* for $\rho = 0.1$.

in the local and global optima. Very similar results were reached in symmetric TSP instances using Concorde (this result is not shown in this paper due to space limitations).

Based on these observations, we hypothesize that past best solutions can effectively inform solutions in the current environment, i.e., in the fitness landscape after a change. Thus, memory approaches, which are used often by algorithms developed for dynamic environments with cyclic changes, can be applied to the DTSP even with random changes. This hypothesis is tested in Section 5.

3.2 DTSP with Vertex Changes

Def. 2: DTSP with vertex changes inserts or deletes at least one vertex during the optimization process.

As in [12], we consider the case where, after a change, $\lfloor \rho n/2 \rfloor$ of the vertices are removed, while $\lfloor \rho n/2 \rfloor$ other vertices not previously present are inserted into the TSP. For the k-environment, the set of vertices V of the static instance of the TSP is split into two subsets with equal size ($n/2$): one subset ($V_{in}(k) \in V$) with vertices that are present in the graph of the current environment and one subset ($V_{outn}(k) \in V$) with the remaining vertices. When a change occurs, $\lfloor \rho n/2 \rfloor$ randomly chosen vertices of the current graph are replaced by the same number of vertices randomly chosen from the subset of vertices currently not used. Thus, the size of the graph is equal for all environments, and ρ controls the severity of the change. Figure 2(b) summarizes the process.

When LKH is applied to the DTSP generated from changing the vertices of static instance *atex5* for $\rho = 0.05$, similar results are obtained (Figure 4), i.e., only a small part of all possible edges were added to the best solutions after the changes. In this case, some edges of the best solution are always removed after the changes.

The results presented in Figure 4 reinforces our hypothesis that, in DTSPs, knowing solutions previously found in past environments can be useful to find the solution in the current environment. However, in this case, solutions from the memory (i.e., past solutions) cannot be directly inserted into the new environment because some of the vertices of the problem are different. A strategy to solve this problem is presented in Section 4.

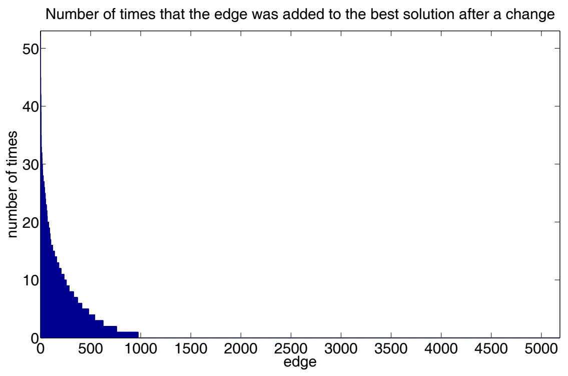


Figure 4: Modifications in the best solution found by LKH after changes in the vertices of static instance *atex5* for $\rho = 0.05$.

4. MEMORY APPROACH

One of the common approaches to EDO is the use of implicit or explicit memory where past solutions (or parts of them) are stored and retrieved according to different strategies. The basic idea is that solutions found in the past can be useful in subsequent environments. This is particularly true in cyclic DOPs, where past environments often reappear, sometimes slightly modified, during the optimization.

One of these memory approaches is the memory-enhanced genetic algorithm (MEGA) [20], where some of the best solutions found are explicitly stored in a memory M with size N_m, i.e., the memory contains N_m individuals found in past generations (in general $N_m < N$, where N is the size of P). When the problem changes, the current population (P) and the memory (M) are re-evaluated, and the best individuals among them form the new population P.

In this paper, we propose a different approach. Instead of directly inserting the individuals of memory M into population P as in MEGA, the individuals of M (or parts of them) can only be inserted into P after crossover. Thus, parts of the solutions in memory can be reintroduced, using crossover, in any generation of the optimization process.

The pseudo-code for the proposed approach for the DTSP with weight changes is presented in Algorithm 1. Individuals can be selected from the current population P or the memory M for crossover. In this way, the crossover can occur between one solution in the memory and another in the current population, or between two solutions in memory, or between two solutions in the current population. Crossover supports recombination of parts of past solutions with current solutions of the TSP, which may prove to be useful even after the changes in the problem. As in MEGA, when the memory is updated, one solution of M is removed before inserting the current best solution, $\mathbf{x_b}$, of P. The replacement strategy is: i) find the solution $\mathbf{x_s} \in M$ that is most similar to $\mathbf{x_b}$; ii) if the fitness of $\mathbf{x_b}$ is better than the fitness of $\mathbf{x_s}$, then $\mathbf{x_b}$ replaces $\mathbf{x_s}$ in the memory.

For the DTSP with vertex changes, the solutions must be modified when a change occurs because some vertices are removed and others are added to the subset of vertices used to evaluate the solutions. Also, as the subsets of vertices in different environments are not the same, a solution from memory cannot be directly recombined with solutions from the population as in Algorithm 1. Thus, two memories are employed. The first (M_1), which is similar to the memory used in Algorithm 1, contains solutions created in different environments, i.e., solutions with different encodings. The second (M_2) is created by repairing the individuals of M_1 according to the subset of vertices used in the current environment. As M_2 contains solutions mapped to the current environment, these solutions can be recombined with the solutions in the current population. Algorithm 2 shows the proposed memory approach for the DTSP with vertex changes.

Algorithm 1 Memory approach for the DTSP with weight changes

```
initialize population $P$ and memory $M$;
while termination condition is not satisfied do
    if change detected then
        $M$=updateMemory($P$,$M$);
        re-evaluate the individuals in $P$ and $M$
    end if
    $Q$=selectForReproduction($P$,$M$);
    $Q$=Reproduction($Q$);
    $P$=$Q$;
end while
```

COMMENTS:
i) M=updateMemory(P,M): *insert* $\mathbf{x_b}$ *(the current best solution of P) in M if* $\mathbf{x_b}$ *is better than the solution in M that is most similar to* $\mathbf{x_b}$
ii) Q=selectForReproduction(P,M): *each individual in Q is obtained by selecting an individual from M or P*

The repair process for M_2 occurs in two steps. First, the indices of the solutions in the original environment are mapped to the new environment. This is made possible by recording information about the subset of vertices used in the current environment and in the environment where the solution was generated. When the solution is repaired, the recorded subset of vertices are used to map the solution in the new codification according to the same order of vertices in the original solution. Since some vertices are removed, the new intermediate solution has less vertices than the original solution. Then, the new vertices are inserted in the solution, i.e., the vertices that are not present in the subset of vertices in the original environment but are now present.

The insertion of the vertices occurs according to the insertion heuristic presented in [16], where the edge (x_i, x_{i+1}) in the current solution that minimizes $w_{x_i,m} + w_{m,x_{i+1}} - w_{x_i,x_{i+1}}$ is removed and the edges (x_i, m) and (m, x_{i+1}) are inserted, where v_m is the vertex that should be inserted in the solution. This process is repeated for all vertices that were not present in the subset of vertices of the original environment, but are now present after the change.

5. EVALUATION

To evaluate our approaches, we compare the performance of the following Genetic Algorithms (GAs):

Alg. 1: Standard GA (SGA)

Alg. 2: Random immigrants GA (RIGA), where in every generation rN randomly chosen individuals of the current population are replaced by randomly generated individuals. The parameter r ($0 \leq r \leq 1$) controls the percentage of the population to be replaced by random immigrants;

Algorithm 2 Memory approach for the DTSP vertex changes

```
initialize population $P$ and memories $M_1$ and $M_2$;
while termination condition is not satisfied do
    if change detected then
        $M_1$=updateMemory($P$,$M_1$);
        $P$=repairSolutions($P$);
        $M_2$=repairSolutions($M_1$);
        re-evaluate the individuals in $P$ and $M_2$
    end if
    $Q$=selectForReproduction($P$,$M_2$);
    $Q$=Reproduction($Q$);
    $P$=$Q$;
end while

COMMENT:
i) $M_2$=repairSolutions($M_1$): for each individual in $M_1$, the vertices that were present in the environment where the individual was created, but are not present in the current environment, are removed from the solutions, while the vertices that were not present in the environment where the individual was created, but are now present, are inserted according to the heuristic presented in [16].
```

Alg. 3: Elitism-based immigrants GA (EIGA) [20], where in every generation rN randomly chosen individuals of the current population are replaced by individuals generated by mutation of the current best solution of the population;

Alg. 4: Memory-enhanced GA (MEGA) [20];

Alg. 5: Proposed memory approach presented in Section 4;

Alg. 6: Hybrid approach combining the proposed memory approach (Alg. 5) with EIGA. The algorithm is the same as Alg. 5, but rN randomly chosen individuals of the current population are replaced by individuals generated by mutation of the current best solution of the population, as in Alg. 3.

RIGA is a traditional algorithm employed for DOPs, while EIGA and MEGA are state-of-art EDO algorithms. While the first three algorithms (SGA, RIGA, and EIGA) are not based on memory, algorithms 4, 5, and 6 explicitly store best individuals found in past generations.

For our DTSP comparison, changes occur after a fixed number of fitness evaluations (τ). Thus, τ controls the interval of the changes. Besides the evaluations of the new individuals generated by crossover and mutation, evaluations are also counted each time an individual is re-evaluated after a change. The follow additional evaluations are counted: i) for RIGA, $r * N$ evaluations for the new random individuals each generation; ii) for EIGA, evaluations resulting in the mutation of $r * N$ individuals (immigrants) in each generation; iii) for MEGA and Alg. 5, N_m evaluations resulting in the re-evaluation of the individuals in the memory after each change; iii) for Alg. 6, N_m evaluations resulting in the re-evaluation of the individuals in the memory after each change and evaluations resulting in the mutation of $r * N$ individuals each generation.

For each DTSP generated from a static instance of an asymmetric TSP, 50 changes occur during each run. To test a range of DTSPs, we vary the change type (weights or vertices), change frequency (τ = 2000 or 6000), and change severity (ρ = 0.1 or 0.3 for the DTSP with weight changes and 0.05 or 0.1 for the DTSP with vertex changes). The results of 25 runs for each combination are presented here. The reduced number of evaluations between changes (τ) considered here does not allow the application of the restart approach with LKH employed in the experiments presented in Section 3; LKH needs many more evaluations for each environment created in the DTSPs.

For all algorithms, tournament selection (where the best solution between two random solutions are chosen with probability 0.8) and elitism are used. The Generalized Asymmetric Partition Crossover (GAPX) [19] is applied to the solutions selected by tournament selection, and, if the solution is not improved, mutation is applied. When a solution is selected for mutation, it is modified by 3opt in 30% of the cases, by 2opt in 10% of the cases, and by local search using 3opt and "don't look bits", as in [19] but limited to a total of 2 evaluations per individual, in the remaining cases. While 3opt keeps the order of the edges in the original solution, 2opt allows the inversion of sub-paths of the tour given by the solution. To compute the fitness of the individuals generated by 2opt and 3opt, we do not need to evaluate the entire individual. For example, each time 2opt is applied, only 2 edges of the solution are replaced (in this way, the mutated individual has fitness equal to the fitness of the original individual plus the difference caused by exchanging 2 edges). The proportional values of the evaluations are added to the total number of evaluations when 2opt and 3opt are applied. The initial population in each run is generated by local search using 3opt [19].

As in [20], the population size is $N = 100$ individuals, the parameter r is equal to 0.2 when the immigrants approaches are utilized and the memory size is equal to 20% of the population size (i.e., $N_m = 0.2N$). The static asymmetric instances used to generated the DTSPs are instances with less than 400 vertices of TSPLIB and the *dc* and *atex* instances from [2]. These asymmetric static instances do not represent the classic TSP (i.e., with vehicles) and were used in the *8th DIMACS TSP Challenge*.

As the performance metric, the mean best fitness in each environment is recorded for each run, i.e.:

$$F(run) = \frac{1}{n_k} \sum_{k=1}^{n_k} f^*(k, run) \quad (3)$$

where n_k is the number of changes in a run and $f^*(k, run)$ is the best fitness found in environment k for the run.

Tables 1 and 2 show the results for each of the two DTSP types. Each column compares a pair of algorithms, e.g., the first column is between Alg. 5 and Alg. 3 [1]. To test statistical significance, the Wilcoxon signed-rank test with significance level 0.01 was used. The comparison between algorithms 5 and 3 contrasts the memory approach (Alg. 5) against the non-memory approach with the best performance (Alg. 3).

For the DTSP with weight changes, the memory approach performed better for the instances *atex*, *ft* and *p43*. In such instances, only a small part of all possible edges are added to the best solution after the changes, as observed in Section 3.1. As a consequence, the insertion of parts of good solutions (subpaths of the tours) found in past environments is shown to be beneficial to performance, i.e., finding good solutions faster.

[1]Tables with the results of average $F(run)$ and the source codes are available as supplementary material.

However, the memory approach performed worse in most of the cases for instances *dc*, *rbg*, and *td* in Table 1. Figure 5 shows a simulation of the DTSP generated from the asymmetric static instance *dc112* where 500 consecutive changes occur with $\rho = 0.1$ and LKH is used to find the best solutions after each change (as in the simulation presented in Section 3). When compared to Figure 3, we can observe that more edges are used in the best solutions for *dc112*. This is likely because many solutions in the fitness space for instances *dc* (and for the instances *rbg* and *td*) present close (or even equal) evaluation. Thus, when the problem changes, the algorithm finds a different solution for the new environment, but with similar values of fitness (this can also be observed when we analyze the weight matrices $\mathbf{W}$). In such cases, the memory approaches do not perform best because new edges (not found in the past solutions) need to be explored. We did not observe this effect in the DTSP with vertex changes (Table 2), except for the instances *rbg*, because the redundancy of the solutions is partially reduced by the fact that only 50% of the vertices are present in each environment each time. The simulation presented in Figure 6 indicates that a smaller subset of the possible edges are inserted in the best solutions when LKH is used to find the best solutions after the changes in the vertices for instance *dc112*. The memory approach performed better than EIGA (Alg. 3) for the DTSP with vertex changes, with the exception of instances *rbg*.

The hybrid approach (Alg. 6) exploits both memory (Alg. 5) and elitism-based immigrants (Alg. 3). Even when its performance is inferior, it behaves similarly to Alg. 5 when this is the best approach, and to Alg. 3 when this is the best approach. This behavior can be seen in Figure 7, which shows the averaged fitness in each environment for the DTSP with weight changes generated from instance *dc112* for $\tau = 2000$ and $\rho = 0.1$. In this DTSP, Alg. 3 presents the best results (Table 1). However, in Figure 7, the results for algorithms 6 and 3 present similar behavior, clearly different from the behavior of the other algorithms.

Alg. 6 showed very good performance, when compared to the other algorithms, for both types of DTSP (tables 1 and 2). The averaged results for $F(run)$ differ when the frequency (τ) and severity (ρ) of the changes are modified (tables in the supplementary material). However, the observations concerning the comparison between Alg. 6 and the other algorithms are practically the same for different combinations of τ and ρ.

6. CONCLUSIONS

The DTSP has been used by many authors to test EDO algorithms. In this paper, we argue that developing new EDO algorithms for the dynamic version of the *classic TSP* is not relevant from a practical point of view. However, instances of the DTSP in domains where the changes can be fast when compared to the optimization time required by the solver exist, as in robotics and telecommunication, and in such cases, EDO algorithms present an interesting approach.

From simulations of the LKH heuristic applied to environments generated by the DTSP, an interesting behavior is observed: only some edges, from all possible edges, are inserted and removed from the best solutions after the changes. This phenomenon occurs because some edges increase the fitness of the solution too much and, then, are never (or rarely) found in the best solutions. Thus, memory approaches can be effective for the DTSP. In this paper, a memory approach

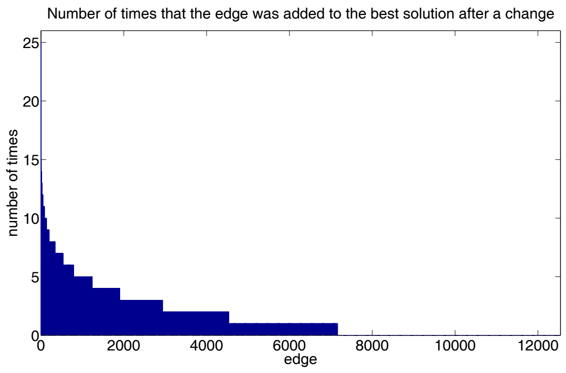


Figure 5: Modifications in the best solution found by LKH after changes in the weights of static instance *dc112* for $\rho = 0.1$.

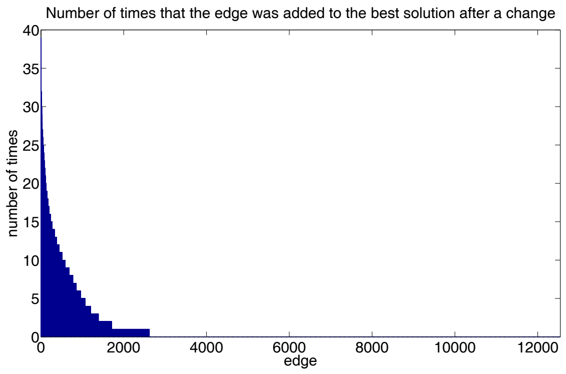


Figure 6: Modifications in the best solution found by LKH after changes in the vertices of static instance *dc112* for $\rho = 0.05$.

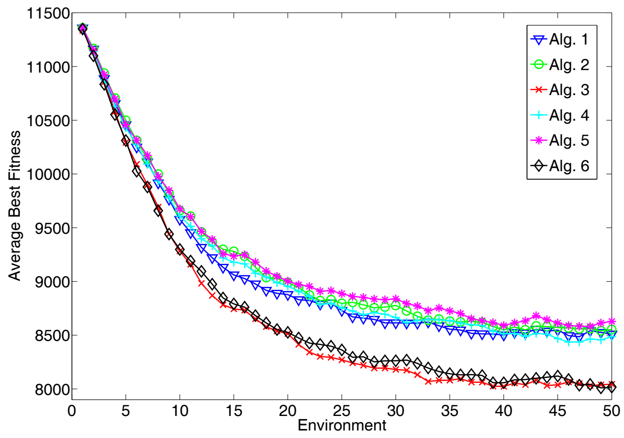


Figure 7: Averaged results (for 25 runs) of the best fitness in each change cycle for the DTSP with weight changes for instance *dc112* for $\tau = 2000$ and $\rho = 0.1$.

for the DTSP is proposed, as well as a method to store and retrieve past solutions for the case where the set of vertices of the DTSP changes.

The experiments support our hypothesis that memory of past solutions can be beneficial for the DTSP with random changes in the weight matrix or in the set of vertices. In instances of DTSP generated from static instances of TSP with redundancy in the solutions, an elitism-based immigrants approach produces better performance. A hybrid approach using memory and elitism-based immigrants produces the best overall performance on our generated problems, when compared to the other algorithms.

The DTSP instances employed here were generated from static instances of the asymmetric TSP and are not real-world instances of the DTSP. Thus, the obvious next step is to investigate how our observations generalize to real-world DTSPs. Investigating the adaptation of the memory size during the run and the effects of the insertion order for the new vertices in the proposed algorithms are also possible future directions.

7. ACKNOWLEDGMENTS

Renato Tinós was a visiting researcher in the Department of Computer Science at CSU. The visit of the researcher was made possible by the support provided by FAPESP (under grant 2012/22200-9), USP, and CSU. This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF (under grant FA9550-11-1-0088). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

8. REFERENCES

[1] D. Apiletti, E. Baralis, and T. Cerquitelli. Energy-saving models for wireless sensor networks. *Knowledge and Inf. Systems*, 28(3):615–644, 2011.

[2] J. Cirasella, D. S. Johnson, L. A. McGeoch, and W. Zhang. The asymmetric TSP: algorithms, instance generators, and tests. In *Algorithm Engineering and Experimentation*, pages 32–59. Springer, 2001.

[3] W. Cook. *In pursuit of the traveling salesman: mathematics at the limits of computation*. Princeton University Press, 2011.

[4] C. Cruz, J. González, and D. Pelta. Optimization in dynamic environments: a survey on problems, methods and measures. *Soft Computing*, 15:1427–1448, 2011.

[5] C. J. Eyckelhof, M. Snoek, and M. Vof. Ant systems for a dynamic TSP: Ants caught in a traffic jam. In *Ant Algorithms : Third International Workshop (LNCC2463)*, pages 88–99. Springer Verlag, 2002.

[6] L. B. Gueta, R. Chiba, J. Ota, T. Arai, and T. Ueyama. A practical and integrated method to optimize a manipulator-based inspection system. In *Proc. of the IEEE Int. Conf. on Robotics and Biomimetics*, pages 1911–1918, 2007.

[7] M. Guntsch, M. Middendorf, and H. Schmeck. An ant colony optimization approach to dynamic TSP. In *Proc. GECCO'2001*, pages 860–867, 2001.

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

[9] K. Helsgaun. LKH results for Soler's ATSP instances. http://www.akira.ruc.dk/ keld/research/LKH/, April 2012.

[10] L. Kang, A. Zhou, B. McKay, Y. Li, and Z. Kang. Benchmarking algorithms for dynamic travelling salesman problems. In *Proc. of the 2004 IEEE Cong. on Evolut. Comp.*, volume 2, pages 1286–1292, 2004.

[11] W. Li. A parallel multi-start search algorithm for dynamic traveling salesman problem. In *Experimental Algorithms (LNCC6630)*, pages 65–75. Springer Berlin Heidelberg, 2011.

[12] M. Mavrovouniotis and S. Yang. Ant colony optimization algorithms with immigrants schemes for the dynamic travelling salesman problem. In S. Yang and X. Yao, editors, *Evolutionary Computation for Dynamic Optimization Problems*, pages 317–341. Springer Berlin Heidelberg, 2013.

[13] M. Mavrovouniotis, S. Yang, and X. Yao. A benchmark generator for dynamic permutation-encoded problems. In *Parallel Problem Solving from Nature XII (LNCC7492)*, pages 508–517. Springer Berlin Heidelberg, 2012.

[14] T. T. Nguyen, S. Yang, and J. Branke. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, 6:1–24, 2012.

[15] G. Reinelt. TSPLIB 95. http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/, 1995. Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg.

[16] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977.

[17] A. Simões and E. Costa. CHC-based algorithms for the dynamic traveling salesman problem. In *Applications of Evolutionary Computation (LNCC6624)*, pages 354–363. Springer Berlin Heidelberg, 2011.

[18] D. Soler, E. Martínez, and J. C. Micó. A transformation for the mixed general routing problem with turn penalties. *Journal of the Operational Research Society*, 59(4):540–547, 2007.

[19] R. Tinós, D. Whitley, and G. Ochoa. Generalized asymmetric partition crossover (GAPX) for the asymmetric TSP. In *Proc. of GECCO'2014*, 2014.

[20] S. Yang. Genetic algorithms with memory-and elitism-based immigrants in dynamic environments. *Evolutionary Computation*, 16(3):385–416, 2008.

[21] A. Younes, O. Basir, and P. Calamai. A benchmark generator for dynamic optimization. In *Proc. of the 3rd Int. Conf. on Soft Computing, Opt., Simulation and Manufacturing Syst.*, 2003.

Table 1: Results for the DTSP with weight changes. Column $A - B$ compared algorithms A and B, where $+$ and $-$, respectively, indicate that the results of Alg. A were statistically better ($+$) or worse ($-$) than Alg. B, while $=$ indicates that the results were statistically similar. For each static instance of the TSP, four instances of the DTSP were generated for the four combinations of τ and ρ settings.

Problem	τ	ρ	5-3	6-1	6-2	6-3	6-4	6-5
	2000	0.1	+	+	+	+	+	=
atex1		0.3	+	+	+	+	+	=
n=16	6000	0.1	+	+	=	+	=	=
		0.3	+	+	+	+	+	=
	2000	0.1	+	+	+	+	=	=
atex3		0.3	+	+	+	+	+	=
n=32	6000	0.1	+	=	+	+	=	=
		0.3	+	+	+	+	+	=
	2000	0.1	+	+	+	+	=	=
atex4		0.3	+	+	+	+	+	=
n=48	6000	0.1	+	+	+	+	+	=
		0.3	+	+	+	+	+	=
	2000	0.1	+	=	+	+	=	-
atex5		0.3	+	+	+	+	+	=
n=72	6000	0.1	+	+	+	+	=	=
		0.3	+	+	+	+	+	=
	2000	0.1	-	+	+	-	+	+
dc112		0.3	-	+	+	-	+	+
n=112	6000	0.1	-	+	+	-	+	+
		0.3	-	+	+	-	+	+
	2000	0.1	-	+	+	=	+	+
dc126		0.3	-	+	+	-	+	+
n=126	6000	0.1	-	+	+	=	+	+
		0.3	-	+	+	-	+	+
	2000	0.1	-	+	+	=	+	+
dc134		0.3	-	+	+	=	+	+
n=134	6000	0.1	-	+	+	=	+	+
		0.3	-	+	+	-	+	+
	2000	0.1	-	+	+	=	+	+
dc176		0.3	-	+	+	-	+	+
n=176	6000	0.1	-	+	+	=	+	+
		0.3	-	+	+	-	+	+
	2000	0.1	-	+	+	=	+	+
dc188		0.3	-	+	+	-	+	+
n=188	6000	0.1	-	+	+	-	+	+
		0.3	-	+	+	-	+	+
	2000	0.1	+	+	+	+	=	=
ft53		0.3	+	+	+	+	=	-
n=53	6000	0.1	+	+	+	+	=	=
		0.3	+	+	+	+	=	=
	2000	0.1	+	+	+	+	+	=
ft70		0.3	+	+	+	+	+	+
n=70	6000	0.1	+	+	+	+	+	=
		0.3	+	+	+	+	+	+
	2000	0.1	+	+	+	=	=	=
p43		0.3	+	+	+	+	=	=
n=43	6000	0.1	+	=	=	=	=	=
		0.3	+	+	+	+	=	=
	2000	0.1	-	+	+	=	+	+
rbg323		0.3	+	+	+	+	+	+
n=323	6000	0.1	-	+	+	-	+	+
		0.3	-	+	+	=	+	+
	2000	0.1	-	+	+	=	+	+
rbg358		0.3	+	+	+	+	+	+
n=358	6000	0.1	-	+	+	=	+	+
		0.3	-	+	+	+	+	+
	2000	0.1	-	+	+	=	+	+
td100.1		0.3	-	+	+	=	+	+
n=101	6000	0.1	-	+	+	=	+	+
		0.3	-	+	+	=	+	+
	2000	0.1	-	+	+	=	+	+
td316.10		0.3	+	+	+	+	+	+
n=317	6000	0.1	-	+	+	-	+	+
		0.3	-	+	+	+	+	+

Table 2: Results for the DTSP with vertex changes. Here, ρ indicates the percentage of vertices modified by each change.

Problem	τ	ρ	5-3	6-1	6-2	6-3	6-4	6-5
	2000	0.05	+	+	+	+	-	=
atex1		0.1	+	+	+	+	-	=
	6000	0.05	+	+	-	+	=	=
		0.1	+	+	-	+	=	=
	2000	0.05	+	+	+	+	=	=
atex3		0.1	+	+	+	+	=	=
	6000	0.05	+	+	+	+	=	=
		0.1	+	+	+	+	=	=
	2000	0.05	+	+	+	+	=	=
atex4		0.1	+	+	+	+	+	-
	6000	0.05	+	+	+	+	+	=
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	=	=
atex5		0.1	+	+	+	+	+	-
	6000	0.05	+	+	+	+	+	=
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	+	=
dc112		0.1	+	+	+	+	+	=
	6000	0.05	+	+	+	+	=	=
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	+	+
dc126		0.1	+	+	+	+	+	=
	6000	0.05	+	+	+	+	-	+
		0.1	+	+	+	+	-	=
	2000	0.05	+	+	+	+	-	=
dc134		0.1	+	+	+	+	-	=
	6000	0.05	+	+	+	+	+	=
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	=	=
dc176		0.1	+	+	+	+	=	=
	6000	0.05	+	+	+	+	-	=
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	=	=
dc188		0.1	+	+	+	+	=	=
	6000	0.05	+	+	+	+	+	+
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	=	-
ft53		0.1	+	+	+	+	+	=
	6000	0.05	+	+	+	+	+	=
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	=	=
ft70		0.1	+	+	+	+	+	=
	6000	0.05	+	+	+	+	=	=
		0.1	+	+	+	+	+	=
	2000	0.05	+	+	+	+	=	=
p43		0.1	+	+	+	+	=	=
	6000	0.05	+	+	+	+	+	=
		0.1	+	+	+	+	+	=
	2000	0.05	-	+	+	=	+	+
rbg323		0.1	=	+	+	+	+	+
	6000	0.05	-	+	+	=	+	+
		0.1	-	+	+	=	+	+
	2000	0.05	-	+	+	+	+	+
rbg358		0.1	+	+	+	+	+	+
	6000	0.05	-	+	+	=	+	+
		0.1	-	+	+	+	+	+
	2000	0.05	+	+	+	+	=	=
		0.1	+	+	+	+	+	-
td100.1	6000	0.05	+	+	+	+	+	=
		0.1	+	+	+	+	+	-
	2000	0.05	+	+	+	+	+	+
td316.10		0.1	+	+	+	+	+	+
	6000	0.05	=	+	+	+	+	+
		0.1	=	+	+	+	+	+