# Benchmark Generator for the IEEE WCCI-2014 Competition on Evolutionary Computation for Dynamic Optimization Problems

**Dynamic Travelling Salesman Problem Benchmark Generator**

Michalis Mavrovouniotis[1], Changhe Li[2], Shengxiang Yang[1], Xin Yao[3]

October 17, 2013

[1]Centre for Computational Intelligence (CCI), School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, UK
[2]School of Computer Science, China University of Geosciences, Wuhan 430074, China
[3]CERCIA, School of Computer Science, University of Birmingham Edgbaston, Birmingham B15 2TT, UK

**mmavrovouniotis@dmu.ac.uk, changhe.lw@gmail.com, syang@dmu.ac.uk, x.yao@cs.bham.ac.uk**

The field of dynamic optimization is related to the applications of nature-inspired algorithms [1]. The area is rapidly growing on strategies to enhance the performance of algorithms, but still there is limited theoretical work, due to the complexity of nature-inspired algorithms and the difficulty to analyze them in the dynamic domain. Therefore, the development of BGs to evaluate the algorithms in dynamic optimization problems (DOPs) is appreciated by the evolutionary computation community. Such tools are not only useful to evaluate algorithms but also essential for the development of new algorithms.

The exclusive-or (XOR) DOP generator [5] is the only general benchmark for the combinatorial space that constructs a dynamic environment from any static binary-encoded function $f(x(t))$, where $x(t) \in \{0,1\}^n$, by a bitwise XOR operator. XOR DOP shifts the population of individuals into a different location in the fitness landscape. Hence, the global optimum is known during the environmental changes. In the case of permutation-encoded problems, e.g., the travelling salesman problem (TSP) where $x(t)$ is a set of numbers that represent a position in a sequence, the BGs used change the fitness landscape. For example, the dynamic TSP (DTSP) with exchangeable cities [2]

and DTSP with traffic factors [4] where the global optimum value is unknown during the environmental changes.

In this report, the dynamic BG for permutation-encoded problems for the TSP (DBG-PTSP) proposed in [3] is used to convert any static TSP benchmark problem to a DOP, by modifying the encoding of the instance instead of the fitness landscape. The rest of the paper is organized as follows. Section 1 gives a description of the TSP. Section 2 describes the framework of DBGPTSP. Section 3 gives the performance measurement and Section 4 gives an example of results obtained from an algorithm.

# 1 Dynamic Travelling Salesman Problem

The TSP is one of the most fundamental $\mathcal{NP}$-complete combinatorial optimization problems. It can be described as follows: Given a collection of cities, we need to find the shortest path that starts from one city and visits each of the other cities once and only once before returning to the starting city. Usually, the problem is represented by a fully connected weighted graph $G = (N, A)$, where $N = \{0, \ldots, n\}$ is a set of nodes and $A = \{(i, j) : i \neq j\}$ is a set of arcs. The collection of cities is represented by the set $N$ and the connections between them by the set $A$. Each connection $(i, j)$ is associated with a non-negative value $d_{ij}$ which represents the distance between cities $i$ and $j$.

Formally, the TSP can be described as follows:

$$f(x) = min \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} \psi_{ij}, \tag{1}$$

subject to:

$$\psi_{ij} = \begin{cases} 1, & \text{if } (i, j) \text{ is used in the tour,} \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

where $\psi_{ij} \in \{0, 1\}$, $n$ is the number of cities, and $d_{ij}$ is the distance between city $i$ and $j$.

This report gives a benchmark generator to generate dynamic test cases of the TSP, i.e., the dynamic benchmark generator for permutation encoded problems for TSP (DBGPTSP) [3].

# 2 Framework of the DBGPTSP

Considering the description of the TSP above, each city $i \in N$ has a location defined by $(x, y)$ and each connection $(i, j) \in A$ is associated with a non-negative distance $d_{ij}$. Usually, the distance matrix of a problem instance is defined as $\vec{D} = (d_{ij})_{n \times n}$.

Every $f$ algorithmic evaluations a random vector $\vec{V}(T)$ is generated that contains exactly $m \times n$ objects of the TSP problem instance, where $T = \lceil t/f \rceil$ is the index of the period of change, $t$ is the iteration count of the algorithm, $f$ determines the speed of change, and $m$ determines the degree of change.

Table 1: Details of the TSP problem instances

| name | no. cities | optimum |
|---|---|---|
| eil51 | 51 | **426** |
| kroA100 | 100 | **21282** |
| kroA150 | 150 | **26524** |
| kroA200 | 200 | **29368** |
| lin318 | 318 | **42029** |
| pr439 | 439 | **107217** |
| rat783 | 783 | **8806** |

More precisely, $m \in [0.0, 1.0]$ defines the degree of change, in which only the first $m \times n$ of $\vec{V}(T)$ object locations are swapped. Then a randomly re-ordered vector $\vec{U}(T)$ is generated that contains the objects of $\vec{V}(T)$. Therefore, exactly $m \times n$ pairwise swaps are performed in $\vec{D}$ using the two random vectors $(\vec{V}(T) \otimes \vec{U}(T))$, where $\otimes$ denotes the swap operator.

With DBGPTSP the encoding of the problem changes but the fitness landscape remains unchanged. Hence the known optimum of the benchmark instances remains the same during the execution whereas the algorithm is biased to search to a different location in the fitness landscape on every dynamic change. The implementation of DBGPTSP integrated with an ant colony optimization (ACO) algorithm as an example, see section 4, is available is available from the competition website[1]

## 3 Evaluation Criteria

- **Problem Instances and Dimensions**: The seven problem instances obtained from the TSPLIB are described in Table 1.
- **Dynamic Test Environments**: The algorithm should be tested in the following test cases for all problem instances:
  - Case1: $f = 1000$, $m = 0.1$
  - Case2: $f = 1000$, $m = 0.25$
  - Case3: $f = 1000$, $m = 0.5$
  - Case4: $f = 1000$, $m = 0.75$
  - Case5: $f = 10000$, $m = 0.1$
  - Case6: $f = 10000$, $m = 0.25$

[1] `http://cs.cug.edu.cn/teacherweb/lichanghe/pages/organization/competition/ECDOP-Competition14.html`

- Case7: $f = 10000$, $m = 0.5$
- Case8: $f = 10000$, $m = 0.75$

The eight test cases for all seven problem instances are already implemented in a shell, i.e., Run. Therefore, when the algorithm is implemented, then the Run file should be executed.

- **Runs**: 30 (already set in the code).
- **Algorithmic Evaluations**: 100000 (already set in the code).
- **Termination Condition**: When the algorithm reaches the maximum number of algorithmic evaluations (already set in the code).
- **Detection of Change**: Algorithms should be informed when a dynamic change occurs.
- **Measurement**: Record the best-so-far solution for every 100 fitness evaluations and calculate the modified offline performance [1] as follows:

$$\bar{P}_{offline} = \frac{1}{S}\sum_{i=1}^{S}\left(\frac{1}{R}\sum_{j=1}^{R}P_{ij}^{*}\right), \tag{3}$$

where $S = 1000$ is the size of samples (Algorithmic Evaluation / 100), $R = 30$ is the number of independent runs, and $P_{ij}^{*}$ is the fitness of the best solution of the $i-th$ sample in run $j$ after the last dynamic change. NOTE: The $\bar{P}_{offline}$ is already implemented in the benchmark generators and saved in text files. The error percentage that should be given in Table 2 is defined as follows:

$$Error = \frac{(\bar{P}_{offline} - Opt)}{Opt}100 \tag{4}$$

where $\bar{P}_{offline}$ is as defined in Eq. 3 (result from the algorithm) and $Opt$ is given in Table 1 for each problem instance. Note that, the less $Error$ obtained the better the performance.

## 4 Example

Algorithm Name: Ant System (AS)
Parameters Used: $\alpha = 1$, $\beta = 5$, $\rho = 0.8$, $popsize = 50$ ants.

Table 2: Example of the results obtained from the AS algorithm

| problem instance | Case1 | Case2 | Case3 | Case4 | Case5 | Case6 | Case7 | Case8 | Total |
|---|---|---|---|---|---|---|---|---|---|
| eil51 | 10 | 14 | 17 | 19 | 6 | 6 | 7 | 7 | 11.2 |
| kroA100 | 17 | 25 | 32 | 33 | 9 | 10 | 11 | 11 | 18.9 |
| kroA150 | 19 | 28 | 36 | 38 | 10 | 13 | 14 | 14 | 22.1 |
| kroA200 | 25 | 34 | 41 | 43 | 14 | 16 | 18 | 18 | 26.4 |
| lin318 | 28 | 39 | 48 | 50 | 15 | 20 | 23 | 25 | 31.5 |
| pr439 | 37 | 51 | 60 | 63 | 21 | 28 | 36 | 38 | 42.2 |
| rat783 | 51 | 69 | 81 | 86 | 32 | 44 | 59 | 66 | 61.4 |
| Average Percentage | | | | | | | | | **30.5** |

## References


[1] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments - a survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[2] Guntsch, M., Middendorf, M.: Applying population based ACO to dynamic optimization problems. In: Proc. of the 3rd Int. Workshop on Ant Algorithms, LNCS 2463, pp. 111–122 (2002)

[3] M. Mavrovouniotis, S. Yang and X. Yao "A benchmark generator for dynamic permutation-encoded problems," in *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature*, ser. LNCS, vol. 7492. Springer-Verlag, 2012, pp. 508–517.

[4] Mavrovouniotis, M., Yang, S.: Memory-based immigrants for ant colony optimization in changing environments. In: EvoApplications 2011, LNCS 6624, pp. 324–333 (2011)

[5] Yang, S.: Non-stationary problem optimization using the primal-dual genetic algorithm. In: Proc. of the 2003 IEEE Congr. on Evol. Comput., pp. 2246–2253 (2003)