A Benchmark Generator for Dynamic Optimization

ABDUNNASER YOUNES and OTMAN BASIR and PAUL CALAMAI
Systems Design Engineering
University of Waterloo
Waterloo, Ontario, N2L 3G1
CANADA

Abstract: - Almost all real-world problems are dynamic and as such not all problem instances are known a priori. Many strategies to deal with dynamic problems exist in the literature, however they are not fully tested on combinatorial problems due to the deficiency of benchmarks. This paper presents a benchmark generator that uses Genetic Algorithms to produce benchmark problems for the dynamic traveling salesman problem. Various strategies of solving dynamic problems are compared on the new benchmarks. Generality of the Genetic Algorithm was retained so as it can be easily adapted for other kinds of combinatorial problems.

Key-Words: - dynamic optimization, traveling salesman problem, genetic algorithms, combinatorial problems.

1 Introduction

Optimization in dynamic environments is gaining increasing interest due to the simple fact that almost all real-world problems are dynamic to some degree or another. Metaheuristics that proved their effectiveness for static problems are being modified by different adaptation strategies for the use in dynamic environments. In addition, benchmark problems are generated to model the dynamic environments.

The current paper uses a Genetic Algorithm under different adaptation strategies to tackle the dynamic version of the Traveling Salesman Problem (TSP). It is expected that the GA, as an evolutionary technique, will work well with dynamic problems. Another contribution of this paper is a benchmark generator to create the dynamic instances necessary for testing and comparing these strategies.

TSP is considered here for its applications in science and engineering fields and, more importantly, because it is considered a representative of the larger class of combinatorial problems. It has often been the case that progress on the TSP has led to progress on other combinatorial problems.

Our benchmark generator produces a wide range of dynamism and in a comparatively small time. In addition, the approach was made very general by using a GA as the optimization algorithm.

Although the work in this paper focuses on the TSP, we believe that our algorithm and benchmark generator can be easily extended to other important dynamic combinatorial problems such as job shop scheduling, vehicle routing, and path planning.

The rest of the paper first discusses related work and some necessary back ground in Section(2), and then describes the benchmark generator and the dynamic strategies in sections (3) and (4). Section(5) presents our results. The paper concludes with some comments on the benefits of adaptation and our future work.

2 Background

Many real-world optimization problems are actually dynamic [1]. New jobs are to be added to the schedule, the composition of the raw material may be changing, new orders are received in the vehicle routing problem etc.

Solving a dynamic problem is usually harder than solving its static counterpart due to the uncertainty associated with dynamic problems. In addition, solutions of dynamic problems are to be found as time proceeds concurrently with the arriving information. Even the goal of the optimization changes from finding an optimal solution of the static problem, to continuously tracking the moving optimum through time in the dynamic problem.

2.1 Benchmarks for Dynamic Optimization

A crucial issue in dynamic optimization is the design of benchmark problems that can be used to represent different dynamic fitness landscapes.

There exist many test problems for dynamic optimization in the literature. Grefenstette [7] specifies his dynamic landscape as a set components,

Support of this work has been provided by the Natural Sciences and Engineering Research Council of Canada

each component consists of a single, time varying n-dimensional Gaussian peak. Each peak is characterized by three time varying features: its center, its amplitude, and its width.
In a similar work, Branke [4] suggests a "Moving Peaks Function". He introduces a multimodal function with controllable height, width and center for each peak. He also gives a detailed literature review of benchmarks for dynamic landscapes.
However, we note that the majority of these problems use real space, and hence are not suitable for combinatorial problems such as the TSP.

2.2 Dynamic Traveling Salesman Problem

Although the TSP finds applications in science and engineering, as in the manufacture of circuit boards, its real importance stems from the fact that it is typical of the larger class of problems known as combinatorial optimization problems.
The majority of the publications on TSP focus on the "traditional" static version in which all the problem instances are known and do not change with time. However, we believe that the dynamic version in which the complete picture is not known before hand due to the environmental shifting is more close to the real world and even harder to solve.
In part, our work continues a line of research about solving the dynamic TSP done by Guntsch et al. [8]. The authors solved the problem using an Ant Colony Optimization. They introduced dynamism by exchanging a number of cities between the actual problem and a spare pool of cities. The number of cities in the actual problem remains constant but the cities themselves do change.
Eyckelhof and Snoek [6] presented a new Ants System approach to the dynamic TSP. They introduced dynamism by changing travel times between the cities.
In our work, we make dynamism more general by removing the constraints on the number of cities, i.e. it will not be necessary to add and delete the same number of cities. Also, we add a second phase to the dynamic problem in which we reverse the changes introduced in the first phase. In this way, we can simulate cycling environments as well. In addition, we introduce a new simple, quick and effective way to create a dynamic TSP by interchanging city locations.

2.3 GA's for Dynamic Problems

In addition to the characteristics that make Genetic Algorithms effective solvers for a broad range of static problems [10], we identify other attributes that prompt using GA's for dynamic problems as well: their theory is based on natural evolution hence they are expected to be capable of adaptation to environmental changes; In addition, GA's have proved to be good for "noisy" environments [4, 9], and hence able to exploit previous or alternate solutions.
This paper uses GA's in both the benchmark generator and the dynamic solver.

3 The Benchmark Generator (BMG)

Benchmarks for dynamic continuous problems, use functions, with adjustable parameters, developed to simulate a shifting landscape. With combinatorial problems the task is much more difficult. We need to think of the dynamic problem in terms of possible scenarios in which changes to a particular problem can happen over time. There can be an infinite number of such scenarios and this might be a reason behind the deficiency in benchmarks for dynamic combinatorial problems
Before discussing our BMG, let us define some terms used to describe dynamic changes. The BMG constructs the dynamic problem from a sequence of static problems. Each static problem is generated by applying some changes or problem shifts to the preceding problem. A problem shift is a result of applying one or more elementary change steps simultaneously.
Thus, a change step is the smallest possible change that can be applied. Hence, the severity of a shift is calculated as the number of change steps in that shift. The number of the iterations or generation between successive shifts determines the period of change (?).
In our work we always start with a static problem of a moderate size taken from the TSP Library [11], introduce the required number of random shifts then remove the changes in the reverse order (last introduced first removed) until we end with exactly the same initial problem. In other words, the dynamic problem consists of two phases: the first phase introducing changes and the second phase removing those changes. In this way, it is also possible to investigate the algorithm behavior when some instances of the problem keep reoccurring.
The BMG works in three different modes: Edge change mode, insert/delete mode and city swap mode as follows.

3.1 Edge Change Mode (ECM)

This mode reflects one of the real-world scenarios, the traffic jam. Here, the distance between the cities is viewed as a time period or cost that may increase or

decrease with time, hence the introduction and the removal of a traffic jam, respectively, can be simulated by the increase or decrease in the distance between cities. The change step of the traffic jam is the increase in the cost of a single edge.
Our policy is as follows: If the edge cost is to be increased then that edge should be selected from the best tour, but if the cost were to be reduced then the selected edge should not be part of the best tour.
The BMG starts from one known instance and solves it to find the best or near best tour. Then an edge is selected randomly from the best tour, and its cost is increased by a user defined factor creating a new instance which will likely have a different best tour. The new instance is solved statically and again the best or near best tour is determined. After a pre-defined number of instances, the problem enters its second phase and the previously introduced jams are removed in the reverse order and the final instance will be exactly the same initial static problem as shown in Fig.1.
In this way, we can view the removals of the jams as new random events without the need to resolve them since all the instances were already solved during the introduction of the jams in the first phase.

3.2 Insert/Delete Mode (IDM):

IDM reflects the addition and deletion of new assignments (cities). This mode works similarly to the ECM mode. The step of the change in this mode is the addition or the deletion of a single city. This mode generated the most difficult problems to solve dynamically since they require variable representation to reflect the increase or decrease in the number of cities from one instance to another.

3.3 City Swap Mode (CSM):

CSM presents another way to create a dynamic TSP by interchanging city locations. Though it does not reflect direct real-world scenarios, it offers a simple, quick and easy way to test and analyze the dynamic algorithm. In CSM, the locations of two randomly selected cities are interchanged. The length of the optimal tour remains the same but the tour itself will be different. The change step is an interchange of costs between a single pair of cities.
Contrary to the previous modes, in CSM there is no need to work out the solution after each change, we only need to swap the cities of the current optimal solution to determine the optimum of the next instance.
The next section discusses how the dynamic problem is solved.

4 The Dynamic Solver:

One of goals of this paper is to compare adaptation strategies found in the literature on combinatorial problems. For this purpose, we employed a basically traditional GA hybridized with local search.
The algorithm is generational and uses tournament selection, and a two-point order crossover. Pair-wise interchange is used as a mutation operator. A straightforward path representation is used for the chromosomes.
The basic GA was modified in order to apply any of the following adaptation strategies to tackle dynamic problems.

4.1 Adapting Mutation Strategy:

The model based on this strategy uses a linearly changing mutation rate. When the environment changes, the rate of mutation is set equal to a fairly large value (P0) then it is decreased linearly with the number of generation until it reaches the base value (Pm) just prior to the next environmental change.

4.2 Random Immigrants Strategy (RIS):

Here, the population is partly replaced by randomly generated individuals whenever the environment

```
Read Initial Problem From TSP Library;
Do Optimization;
Repeat until half of the required instances // 1st phase
    Increase cost of one edge in the best found tour;
    Do Optimization;
Repeat until half of the required instances // 2nd.phase
    Remove Latest Introduced jam;
Output Best Tours for all instances;
```

Fig.1 Algorithm for bench mark generator

shifts. In this paper, RIS is adopted in two models: one model, RIS_10, replaces 10% of the population by random immigrants and the other model, RIS_20, replaces 20%.

4.3 Ignore Strategy:

GA's in their standard form suffer from the problem that once a population converges around an optimum it will not be able to further explore the search space if the environment shifts. Hence, the ignore strategy which does not apply any specific measures to tackle dynamism in the problem is expected to perform poorly. This strategy might produce good solutions if changes in the dynamic problem are small.

4.4 Restart Strategy:

This is the most straightforward strategy to handle dynamic problems. The population is regenerated randomly whenever the problem changes. This means that each change in the environment is regarded as a new problem to be solved independently of the previous instances.

This strategy depends totally on exploration to find new solutions and does not make use of any past knowledge.

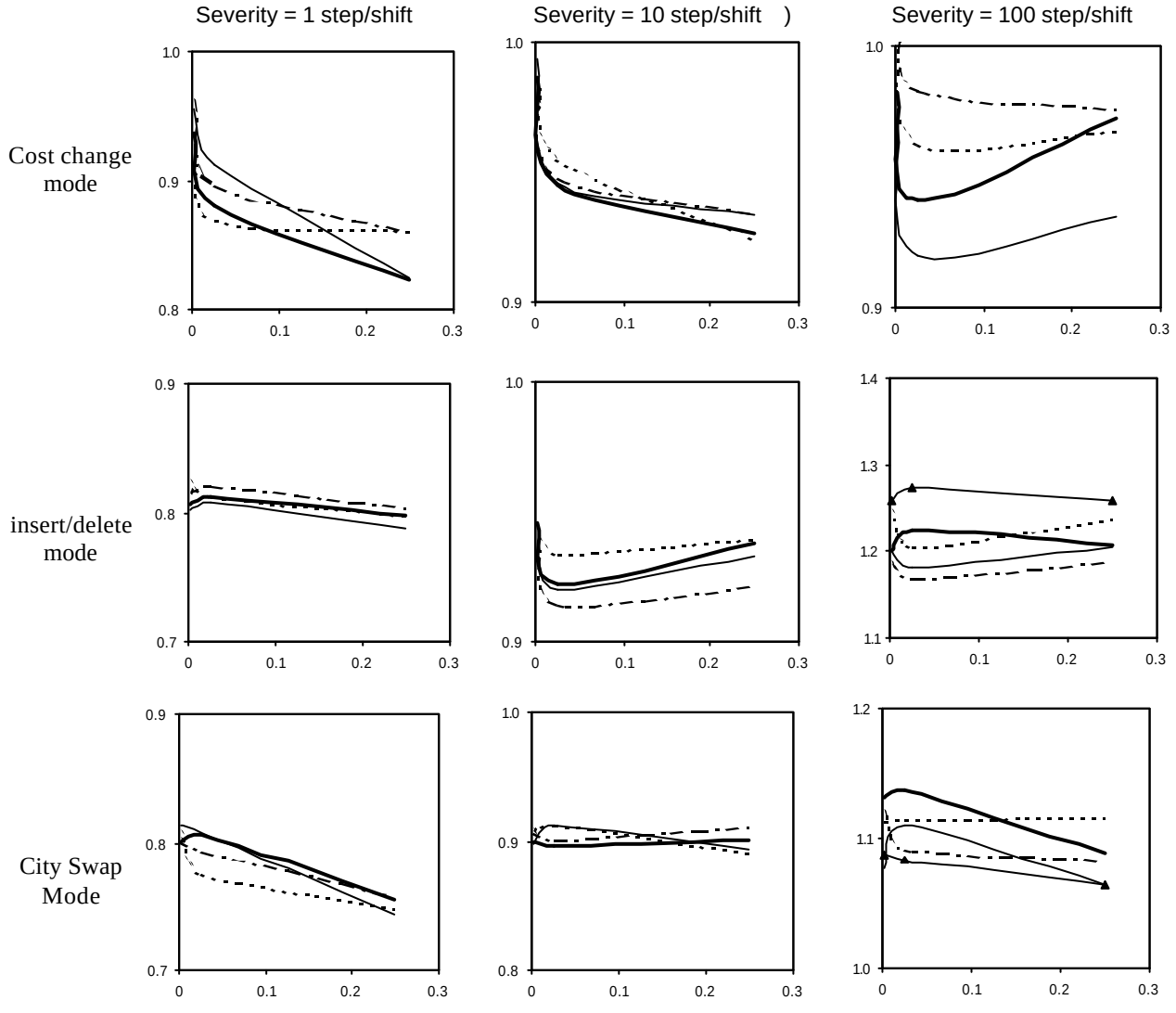

Fig.2 Population-best vs. mutation rate

Problem changes every 1000 generations
Note that the Results from applying Restart Model are of low quality (too large to appear with other models in most figures.

5 Computational Results.

The best solution found in the population "*Population-best*" is used as the criterion to compare different strategies. The goodness of a strategy is measured by how close the population-best is to the solutions given by the benchmark generator. Since the optimal solution is likely to vary with time, values of the population-best are reported as ratios to the base optimal solution given in the TSP library.

The experiments reported here use a 100-city problem, kroA100 from the TSP library, as the base static problem. The BMG is set up to apply 200 successive changes to the base problem. Thus, there will be a sequence of 200 static problems for each of the three modes of environmental shifts.

Each sequence of static problems will be translated into 9 dynamic benchmark problems, resulting from the combination of three degrees of severity (1, 10, 100 steps per shifts) and three periods of change (10,100, 1000 generations between shifts). The dynamic problems are used to test the performance of the five models representing the strategies of the previous section.

For each conducted experiment, the results are reported as the average from 10 runs using different random initial populations.

In every run, the environment is kept fixed at first then changes are applied after 10000 generations. This gives the GA sufficient time to reach initial convergence. A dynamic solver should be able to explore the new search space when the problem changes even after it has converged or nearly converged to some optimum.

A population size of 50 and crossover rate of 0.9 were used throughout.

Finally, since the five models depend in part on the underlying mutation rate, experiments were repeated for three values of the base mutation rate (0.0025,0.025 and 0.25).

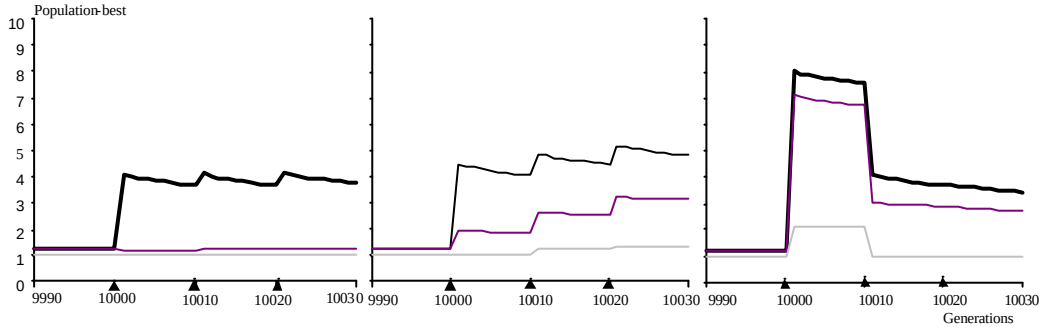

Fig.3 Population-best performance – fast change
Problem changes every 10 generations in City Insert/Delete benchmark

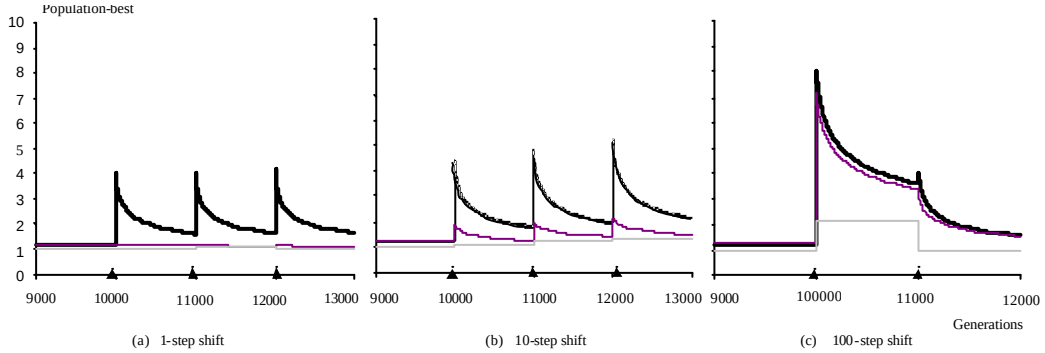

Fig.4 Population-best performance – slow change
Problem changes every 1000 generations in City Insert/Delete benchmark

Fig.2 summarizes our results. The Population-best reported in this figure is actually the average of the values obtained throughout the run. We note that adaptation strategies tend to be the closest to the goal: best solutions previously worked out by the BMG.
Also it seems to be a good idea to keep mutation rates in the vicinity of 0.1
Restart strategy shows the poorest performance, indeed, their values are so large they do not show in some slides. Restart does comparatively well only when changes in the problems are large.
Fig.3 details how the moving optimum is tracked while new cities are added to the assignment over time (number of generations) in a fast changing environment. The benefits of adapting old solutions are very clear in contrast with the Restart strategy.
Fig.4. also tracks the optimum over time but in a slow changing environment (frequency of change reduced to 1 change every 1000 generations). Again, it is clear that the Restart strategy should only be used when the changes are very large.

6 Conclusions

A comparison was made between several models of adaptation.
A benchmark (BM) generator was developed for a dynamic TSP. It can produce problems with known shifting optima with controllable qualities.
A dynamic solver was developed to track the moving optima in dynamic problems. It can run in different modes that apply strategies found in the literature to tackle dynamic problems.
The Restart strategy produced solutions of low quality suggesting that it is very important not to discard knowledge from past solutions.
We also observe that the best strategy to solve a dynamic problem depends on its dynamic characteristics e.g. severity and speed of change. This prompts us to measure the characteristics online and use the measurements to recommend the best strategy to the main algorithm.
In addition, our future work involves an enhancement to the variable mutation model utilizing a non-linear mutation model. We should also test a model with a variable crossover rate.
It should prove to be easy to extend our GA and the ideas of developing benchmarks in this paper to other types of combinatorial problems.

References:

[1] L. Bianchi, Notes on dynamic vehicle routing - the state of the art - Technical report *idsia* 05-01, Italy, 2000.

[2] T. Back, Self-adaptation. *In The Handbook of Evolutionary Computation*, IOP Publishing and Oxford University Press, 1997.

[3] P. Badeau, M. Gendreau, F. Guertin, J.Y. Potvin, D. Taillard, A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows, *Transportation Research* -C 5, 1997, 109-122.

[4] J. Branke, *Evolutionary Optimization in Dynamic Environments.* Kluwer Academic Publishers, 2002

[5] O. Bräysy, A New Algorithm for the Vehicle Routing Problem with Time Windows Based on the Hybridization of a Genetic Algorithm and Route Construction Heuristics. *Proceedings of the University of Vaasa, Research papers 227*, Vaasa, Finland, 1999.

[6] Casper Joost Eyckelhof, Marko Snoek, Ant Systems for a Dynamic TSP, ANTS 2002: Brussels, Belgium, 88-99

[7] J. Grefenstette, Evolvability in Dynamic Fitness Landscapes: A Genetic Algorithm Approach. *In Proc. 1999 Congress on Evolutionary Computation (CEC 99),* Washington, DC. IEEE Press, pp. 2031-2038

[8] M. Guntsch, M. Middendorf, H. Schmeck, An Ant Colony Optimization Approach to Dynamic TSP, In: L. Spector et al. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference*, San Francisco, CA: Morgan Kaufmann Publishers, 2001, 860-867.

[9] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, USA, 1996.

[10] D. T. Pham and D. Karaboga, *Intelligent Optimization Techniques*, Springer-Verlag, USA, 2000.

[11] Library of Traveling Salesman Problems, <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/> (20 December 2002).