Generalized Benchmark Generation for Dynamic Combinatorial Problems

Abdunnaser Younes Paul Calamai Otman Basir
Systems Design Engineering
University of Waterloo
Waterloo, On N2L 3G1
Canada
ayounes@engmail.uwaterloo.ca

ABSTRACT
Several general purpose benchmark generators are now available in the literature. They are convenient tools in dynamic continuous optimization as they can produce test instances with controllable features. Yet, a parallel work in dynamic discrete optimization still lacks.

In constructing benchmarks for dynamic combinatorial problems, two issues should be addressed: first, test cases that can effectively test an algorithm ability to adapt can be difficult to create; second, it might be necessary to optimize several instances of an NP-hard problem. Hence, this paper proposes a method for generating benchmarks with known solutions without the need to re-optimize. Consequently, the method does not suffer the usual limitations on the problem size or the sequence length.

The paper also proposes a general framework for the generation of test problems. It aims to unify existing approaches and to form a basis for designing newer benchmarks. Such a framework can be more appreciated knowing that combinatorial problems tend to assume very distinct structures, and hence, relevant benchmarks are basically too specific to be of interest to the general reader.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics – *Performance measures;* I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Heuristic methods;* G.1.6 [**Numerical Analysis**]: Optimization – *Integer programming.*

General Terms
Algorithms, Performance, Design.

Keywords
Dynamic optimization, benchmarks, combinatorial optimization problems.

GECCO'05, June 25-29, 2005, Washington, DC, USA.

1. INTRODUCTION
As a new and still expanding field, dynamic optimization has many outstanding issues. Some of them are related to the generation of suitable benchmark problems. In this section, we give a brief introductory background on benchmarks, and some issues relevant to the generation of dynamic benchmarks for combinatorial optimization problems (COPs).

Benchmarks can be defined as standardized test problems designed to serve as bases for algorithm evaluation and comparison. The usual reason for running an algorithm on such problems is to obtain results that are comparable to studies on other algorithms and hence can attest to the superiority of the tested algorithm.

Test problems can be basically constructed from two types of data: randomly generated data, and real-life data. On the one hand, random data is easy to obtain and analyze and more importantly enables the drawing of general conclusions about the algorithm performance. On the other hand, a test problem from the second source would reflect a particular instance from the real-world as closely a possible. Thus, it can be used to attest to algorithm ability to fulfill the cause for which it was designed, i.e. solving a particular real-world problem. Thus, one would be inclined to include both types of data in the tests.

In dynamic optimization, however, a test problem is also characterized by a particular scenario, which postulates the sequence of events or environmental changes in the problem. Thus, if one intends to base a test problem on real life situations, one would be faced by the difficulty of identifying the instances that best represent the typical scenario(s). On the other hand, a general-purpose benchmark generator (BG) such as those mentioned in the next section would not have such a difficulty. These BGs use continuous functions with tunable parameters to produce wide varieties of scenarios. The user, thus, can precisely pre-determine particular courses of events for the test problems as deemed appropriate.

However, constructing general BGs for COPs can be more difficult. First, it might prove to be hard to generate a sequence of instances that can effectively test the adaptability of an algorithm, without explicitly solving each instance. Second, as different combinatorial problems take different structures, their test cases tend to be too problem specific. Hence the task of generalizing test problems is expected to be difficult.

The current paper aims to address the above mentioned issues. In the next section, the paper argues in favor of the importance of general-purpose BGs, as opposed to the use of problem specific test cases only. The rest of the paper focuses on dynamic COPs: Section 3 discusses difficulties of generating combinatorial benchmarks. Section 4 introduces a mapping-based scheme to generate benchmarks with known optima. Then, Section 5 proposes a general framework that unifies the approaches of benchmark generation for dynamic COPs.

2. GENERAL-PURPOSE BG's, ARE THEY REALLY NEEDED?

The need of having diverse test problems to evaluate and demonstrate the effectiveness of non-exact algorithms is widely appreciated. For dynamic optimization, test problems should also be able to cover wide ranges of environmental changes in order to pose as credible testers for a certain dynamic solver (DS). This opinion motivates several researchers [1, 5, 11, and 13] to work on the introduction of general purpose BGs to generate artificial dynamic landscapes with controllable features. Grefenstette [5], for example, specifies a dynamic landscape as a set of components. Each component consists of a single n-dimensional Gaussian peak characterized by three time-varying features: center, amplitude, and width. In a similar work, Branke [1] suggests a *moving peaks function*, which is basically a multimodal function with controllable height, width and center for each peak. The moving peaks function, however, offers an additional parameter λ, ranging between 0.0 and 1.0 to quantify "how much a peak's change in location depends on its previous move". Setting λ to 0.0 makes the peak's change completely random, while the other extreme, $\lambda = 1.0$, means the peak continues its shifting in the same direction. More in-depth discussion of the functions and the resultant landscapes produced by these generators can be found in [2].

Jin and Sendhoff [7] introduce a computationally efficient method to generate general dynamic test problems based on concepts from multi-objective optimization. They construct dynamic single objective and multi-objective test problems by aggregating different objectives of a multiple objective optimization problem and changing the weights dynamically.

Yang [15] uses a different approach to construct dynamic environments. In stead of explicitly defining time varying functions, he constructs the dynamic problem by continually introducing changes to a base stationary problem. He proposed using an exclusive-or (XOR) operator to introduce changes to a binary-encoded stationary problem. In [16], Yang and Yao use the XOR operator to generate a series of dynamic problems from a randomly generated stationary knapsack problem.

Unlike the examples above, many researchers depend solely on problem specific benchmarks [2], whereas others further downplay the usefulness of the general-purpose BGs. For example, Ursem et al. [14] note that these BGs do not reflect characteristic dynamics of real-world problems and hence are of little value for modeling realistic dynamic problems. They also argue that the use of BGs is pointless once a model is developed for the fitness landscape. However, one might find the notion that the typical real-world problem—which is supposed to be highly complex— can be modeled to the extent that the model alone is capable of testing the DS seems far fetched.

In this paper, the view is that, ideally any DS should be tested on randomly generated data in addition to real life data. The importance of general benchmarks problems, which use randomly generated instances, is evident in many aspects:

First, randomly generated data can be more effective than real life data in comparing algorithms. With random data, it is possible to introduce variations of different degrees to the elements of the optimization problem individually and in combination, whereas real life data is often too complex to evaluate easily.

Second, randomly generated data might be the only way to detect deficiencies (in an algorithm) that are not visible through the real-world data available at the time of evaluation.

Third, the issue of credibility of the tests suggests that test problems should be designed as independently as possible from the DS, and this is best achieved through general BGs; whereas for example the test case generator suggested in [14] obscures the line between the BG and DS. More generally, the less involved the DS designer is in the BG design, the less biased the results are expected to be and the more credible the DS— especially when it is largely maintained that techniques used by GAs are based on intuition.

Fourth, the use of general benchmarks promotes the portability of the ideas within an algorithm to solve other arbitrarily different problems. Indeed, the more general the testing problems are, the wider the applicability of the tested algorithm.

Furthermore, as it is well known that the use of GAs is often justified by their robustness, it seems unreasonable to confine test cases of a general algorithm to very problem-specific data.

In summary, the use of problem-specific test cases alone will at best confine the results of testing to the particular optimization problem under consideration; at worst, the results cannot even be generalized to problem instances other than those specifically used. In any case, problem specific tests do not encourage using algorithmic in other problems. Yet, most general benchmark generators available in the literature basically target continuous optimization. Thus, in their current state, these generators have little use in discrete optimization, except may be for the few cases discussed in the next section.

3. BENCHMAK PROBLEMS FOR DYNAMIC COPs

In this section, we discuss some issues related to the design of dynamic benchmarks in discrete optimization. First, we note that combinatorial problems tend to assume very distinct structures (e.g. vehicle routing versus job shop scheduling). This fact does not allow the testing of say a scheduling DS on a routing problem. Consequently, benchmark problems for COPs tend to be very specific to the application at hand. However, the time-varying knapsack problem and the dynamic traveling salesman problem (TSP) might be excluded, since their static counterparts are often considered representative of various combinatorial problems. There exist several publications related to such benchmarks. For instance, Goldberg and Smith [5] use a 17-object knapsack with a weight capacity oscillating between two values in their

benchmark. Other researchers [8, 9, and 10] increase the number of objects and make the weight change over several values. The main idea of dynamism in these benchmarks is to vary the allowable weight limit with time; which can make the current optimal solution infeasible if the knapsack capacity is sufficiently reduced.

More recent publications introduce benchmarks for the dynamic TSP. Guntsch et al. [6] solve the problem using an ant colony algorithm. They introduce 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 is kept constant but the cities themselves are changed. Eyckelhof and Snoek [3] present a new ants system approach to another version of the dynamic problem. They change edge length to imitate the appearance and the removal of traffic jams from roads. The pattern of change is limited to simple constant increment or decrement of the changing parameter. Younes et al. [17] introduce a more comprehensive dynamic TSP generator that can produce test problems with more complex dynamics.

The above mentioned benchmarks are limited both in the applications they address and in the dynamics they employ. There are several reasons behind these limitations. They are better addressed by first drawing a distinction between the generation of benchmarks for continuous optimization and that for discrete optimization.

In the continuous case, the generators use functions with adjustable parameters to simulate shifting landscapes. Basically, they introduce time as an additional independent variable in order to create dynamic landscapes in which optima shift through time. However, a similar approach will not work for discrete optimization, where even a static "landscape" cannot be defined without reference to the search algorithm. In fact, it is the notion of the continuity of the variables underlying the search space that makes it possible to define a unique landscape for a continuous optimization problem.

However, in the discrete case, the metaphor of landscape is an indistinct one, since the concepts of distance and relative positions depend on the optimizing algorithm as well. Actually, these concepts are induced by the particular operators employed by the algorithm to move from one solution to another together with what we call neighborhood structure, without which the metaphor of landscape does not make much sense, if any [12]. Thus in discrete optimization, we cannot define an algorithm-independent landscape that can be made time-dependent to simulate dynamic environments. A dynamic problem might have to be constructed as a time sequence of static problems, i.e. it should be thought of in terms of possible scenarios in which changes can happen over time. However, there can be an infinite number of such scenarios, which at the same time might prove to be hard to implement effectively and efficiently. These deficiencies are discussed in the following sections.

3.1 Environmental Effects

From a dynamic solver perspective, changes in a dynamic problem can be viewed as two categories: *dimensionality changes* and *non-dimensionality changes*.

Changes in the first category correspond to adding or dropping variables from the optimization problem. Such changes are applied to reflect for example the insertion and/or cancellation of assignments in a vehicle routing problem, orders in a job shop scheduling problem, cities in a TSP, and objects in a knapsack problem. These changes necessitate a corresponding alteration in solution representation. Hence, dynamic problems constructed in this way are generally harder to solve than those involving non-dimensionality changes.

In the second category, the non-dimensionality changes result from variations in the values of the parameters and coefficients of the problem constraints and objective function. As some of these values change from one instance to another, the optimal solution of a previous instance might lose quality relative to another solution that was inferior to it in the past. Examples are the changes in the capacity of the knapsack problem or in the weights or values of its objects. Other examples can affect the travel time on some roads in a vehicle routing problem, and the processing timing and ready dates of a scheduling problem. Such changes usually do not alter solution representation and hence are expected to be easier to solve than the first class.

However, benchmarks from the second category are harder to construct: While the construction of dimensionality benchmarks can be seen as basically a simple adding or deletion of variables, the construction of non-dimensionality benchmarks is not as simple. One reason for this difficulty that is not addressed specifically in the literature is what we will refer to by *significance of dynamism*.

3.2 Dynamically Significant Changes

When a new instance is generated by applying non-dimensionality changes to another instance, differences between both instances can inadvertently be made dynamically insignificant. In other words, the introduced changes are so trivial that any optimizing algorithm exhibits the same behavior with or without them.

Therefore, a dynamically insignificant change can be defined as an environmental change that does not alter the structure of the problem instance, i.e. one which keeps the number and relative positions and values of the peaks unchanged. In a knapsack problem, for example, increasing the weight of an object not in the optimal solution (or decreasing the weight of an object in the optimal solution) will not alter the optimal solution. Furthermore, reducing the weight of a non-optimal object (or increasing the weight of an object in the optimal solution) may not alter the optimal solution unless the changes in the weight are sufficiently large. In a similar manner, increasing and decreasing travel time on a road in a TSP may not be significant.

In order to further clarify how a DS exhibits the same behavior after a dynamically insignificant change, we borrow the following example from continuous optimization. Once a hill climber discovers a local maximum, it will consistently return the same solution if changes were confined to the height of the peak; no matter how much the change is, as long as the peak remains higher than its neighbors. This issue seems trivial since the BG can explicitly shift the location of the optima, and thereby making the environmental change dynamically significant. However, as one cannot identify a landscape to start with for a given COP instance, one would not have a clue to whether any induced non-dimensionality changes are significant or not. A minimum requirement to ensure significance of such changes is that the optimal solution of the current instance is known.

A dynamically insignificant change is worthless from a testing perspective. Furthermore, properties of dynamism such as severity and frequency of change of the underlying parameters may become misleading. In other words, patterns of optima shift can be considerably different from the patterns intended by the BG user.

This issue adds to the efforts of selecting the changing parameters and their corresponding values. It might even necessitate solving newer instances before actually adopting them in the benchmark.

3.3 The Challenge of NP-hardness

This issue arises from treating the dynamic problem as a sequence of static problems. Ideally, the optimal value for each problem in the sequence should be known in order to evaluate the effectiveness of some DS (by comparing its results with the known optimal values). Furthermore, in order to ensure that the change introduced to a problem is dynamically significant, the optimal solution is needed too. Therefore, in the course of constructing a dynamic test case, several static instances have to be solved to optimality: a non-trivial task if not impossible, especially when the problem in question belongs to the NP-hard class.

Two options are used to alleviate this difficulty. The first one uses small sequences with problems of limited size. Of course, using too small problem sizes may reduce the benchmark usefulness. At the same time, limiting the sequence length restricts the dynamism characteristics that can be modeled. The second option uses results of several dynamic solvers to compare with the DS under testing. This option has two disadvantages: first, evaluations are of a relative nature (to the quality of other algorithms). Second, as we do not have a wealth of results for other algorithms, the choice of the comparing algorithms and the way they are run can severely change the outcome of evaluations.

This issue motivated the authors to introduce a general scheme to generate benchmarks of arbitrary size and sequence length, as described in the next section.

4. MAPPING-BASED BGs

In a recent paper, Younes et al. [17] introduce a scheme to generate benchmarks for the dynamic TSP with arbitrarily long sequences and controllable characteristics of dynamics. In what follows, we generalize the underlying idea to other COPs.

The basic idea is to exploit the fact that GAs do not work directly on the solutions but rather on their encoding. Thus an environmental change can be applied at any time by modifying the mapping function, which encodes solutions to chromosomes. To illustrate this idea, let us consider a seven-object knapsack problem as an example, with its mapping function given in Figure1. In this setting, a candidate solution consisting of the objects O_1, O_4, O_3, O_2 and O_7 will be encoded as (B_1 B_4 B_3 B_2 B_7). Then if, for example, the object names associated with the labels B_3 and B_5 in the mapping function are swapped, the same chromosome (B_1 B_4 B_3 B_2 B_7) will represent a different individual, consisting of the objects O_1, O_4, O_5, O_2 and O_7.

If all the chromosomes in the population are treated in this way, they will point to different individuals. Then any re-evaluation of the population will reveal that it now consists of individuals which are actually different from their predecessors. Furthermore, some of the new individuals might even be infeasible. Hence, by repeatedly changing the encoding, a sequence of instances can be generated from a single problem. A dynamic solver will treat the sequence as a dynamic problem i.e. will try to adapt to changes in the problem. At the same time, the benchmark designer knows the values of the optimal solutions to all the generated instances, since they are actually the same. Thus, in using mapping-based scheme, one needs only optimize the initial instance of the dynamic problem.

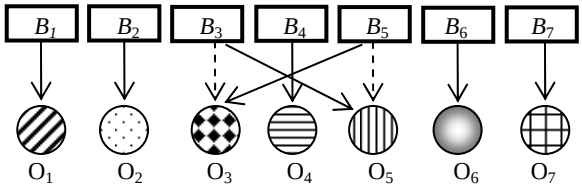

Figure 1. swapping in mapping function.

Before change, gene values B_3 and B_5 originally represent objects O_3 and O_5 respectively. After change, they represent O_5 and O_3 respectively.

The severity of change in the mapping-based benchmark can be expressed as the number of interchanges imposed on the mapping function a time; and the change frequency can be expressed in terms of the number of iterations or evaluations between changes.

Other COPs, can be treated in a similar way to produce dynamic versions. For instance, benchmarks for the dynamic TSP can be generated by swapping city labels in the mapping function. The scheme can also be applied to job scheduling or to flexible manufacturing systems by interchanging any of the labels of machines, parts or operations to create new instances. Similarly, a facility location problem can be made dynamic by interchanging locations labels or facility labels.

Although problems constructed via a mapping-based BG may not reflect real life situations, this technique serves the goal of generating dynamic test COPs with known optima without the usual limitations on the sequence length and instance size. Furthermore, complicated test problems that are more real-world oriented can be constructed by combining mapping-based procedures with dimensionality and/or non-dimensionality changes. In any case, a mapping-based BG offers a simple, quick and easy way to generate problems that can be used to test and analyze a dynamic algorithm running on almost any COP.

5. A GENERALIZED RAMEWORK FOR BENCHMARK GENERATION

The idea of having general BGs for COPs similar to those available for continuous optimization is tempting. However, different COPs in their static forms tend to take very distinct structures, which make the idea of a general framework for them in their dynamic states more challenging. This section aims to encompass dimensionality changes, non-dimensionality changes, and the proposed mapping-based changes in a general framework that can form a basis for the generation of benchmarks for dynamic COPs.

The general idea is to start with an initial static benchmark problem S_0 taken from the literature or from available real life data. Then, changes are repeatedly introduced to the problem in

order to generate a sequence of static problems, with some (exploitable) similarity between any two succeeding problems. The operation of the generator is divided into two stages: a *sequence generation stage* that creates a pool of $k_{\max}$ static problems and a *dynamism control stage* that selects problems from the pool to construct one dynamic problem with $m_{\max}$ instances, see Figure 2.

The sequence generation stage applies a limited amount of change in one element of the optimization problem P_k to create the next problem P_{k+1} in the sequence. We refer to the limited change as an *elementary step* δ_k , which have one of three forms:

(1) A dimensionality step, i.e. the addition or a deletion of a single variable.

(2) A non-dimensionality step, which corresponds to a change in the value of one of the parameters or the coefficients of the problem. In this case, it should be dynamically significant; and if it affects problem constraints, it also should be not too drastic to make the new problem infeasible.

(3) A mapping-based step, i.e. a single swap in the mapping function.

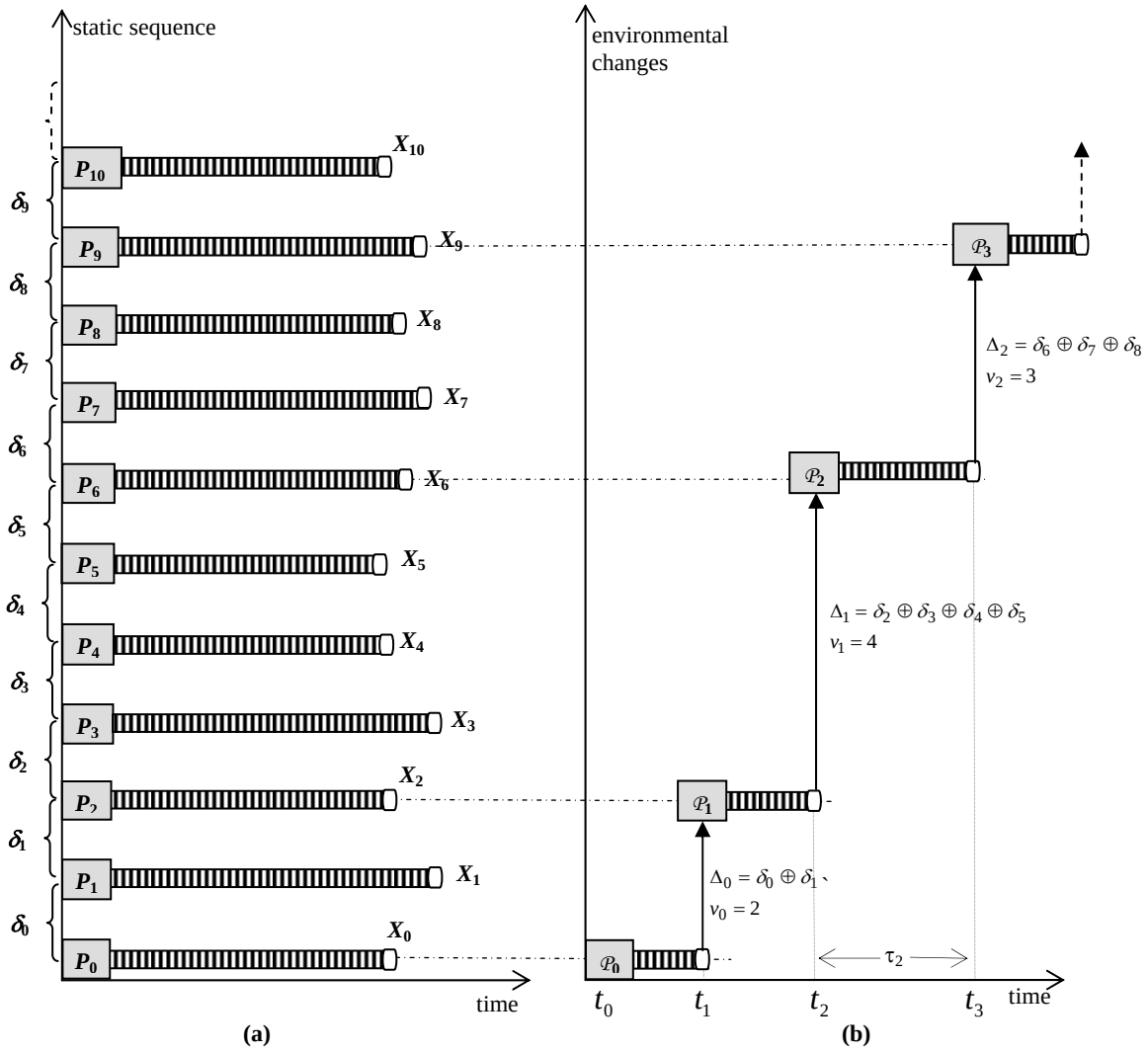

Figure 2 Generalized Benchmark Generation

(a) Sequence generation stage (b) Dynamism control stage

Although the figure may imply that there is a consistent forward progression of problem instances, actually the order of the instances in the figure does not reflect how they are close to each other. For example, if the change in elementary step $\boldsymbol{\delta_3}$ is the reverse of that in $\boldsymbol{\delta_2}$, static instances $\boldsymbol{P_2}$ and $\boldsymbol{P_4}$ will actually be identical. More generally, the sequence can be made to cycle from $\boldsymbol{P_3}$ back to $\boldsymbol{P_0}$ by repeatedly reversing the changes in $\boldsymbol{\delta_2}$, $\boldsymbol{\delta_1}$and $\boldsymbol{\delta_0}$ to create $\boldsymbol{\delta_3}$, $\boldsymbol{\delta_4}$and $\boldsymbol{\delta_5}$.

The above process of adding an elementary step can be written as:

$$P_{k+1} = P_k \oplus \delta_k \quad , k = 0,1,...,k_{\max} - 1 \qquad (1)$$

Then, each newly created static problem is solved independently of the others. Thus, the stage ends with a sequence $\mathbf{S}$ of static problems P_k and their corresponding optimal or near optimal solutions x_k. The sequence generation stage can be formally given as:

$$\begin{aligned} &\mathbf{S} = \{\mathbf{S}_k = (P_k, x_k), \quad k = 0,..,k_{\max}\} \\ &\text{where} \\ &P_k = P_0 \oplus \delta_0 \oplus \delta_1 \oplus ... \oplus \delta_{k-1} \\ &x_k = \text{optimize}(P_k) \end{aligned} \qquad (2)$$

In the second stage, a complete dynamic problem $\mathcal{P}$ is created by selecting some of the static problems in the sequence $\mathbf{S}$ to become instances of $\mathcal{P}$. The selection is done in such away that the resultant dynamic problem has the required properties of dynamism. For instance, by skipping more intermediate problems in the static sequence $\mathbf{S}$, severity of the change is increased; similarly the frequency of change can be specified by the number of evaluations/generations between successive instances. To further elaborate on this stage, let us first define an environmental *shift* Δ_m as the change applied to the mth instance of the dynamic problem to create the next instance, i.e.

$$\mathcal{P}_{m+1} = \mathcal{P}_m \oplus \Delta_m \qquad (3)$$

Then the change severity or v_m can be expressed as the number of elementary steps added to create Δ_m; and the *period of change* τ_m can be defined as the duration (number of generations or evaluations between successive shifts) of the m^{th} instance.

Once severity v_m and *period* τ_m are specified, the dynamic problem can be given as a sequence of problem instances $\mathcal{P}_m$, and their corresponding time t_m and target solutions y_m as :

$$\mathcal{P} = \{\mathcal{P}_m = (\mathcal{P}_m, t_m, y_m), \qquad m = 0,1,..,m_{\max}\} \qquad (4)$$

In which $\mathcal{P}_m$ and y_m are actually P_k and x_k in the sequence $\mathbf{S}$ respectively, where

$$k = \sum_{i=0}^{m-1} v_i \qquad (5)$$

, and each instance begins at

$$t_m = \sum_{i=0}^{m-1} \tau_i \qquad (6)$$

The target solutions y_m will be the basis of criteria that measure the success of any dynamic solver on the above benchmark.

Once a benchmark is generated according to the generalized form, additional test problems can be added by changing the static problem S_0, the elementary steps δ, and/or the values of severity v and period τ. As well, a second sequence of static problems can be added to the dynamic problem. The additional sequence is constructed by reversing the changes introduced to the first sequence. Thus, by repeatedly adding and reversing changes, cycling environments can be created. The three modes of the dynamic TSP benchmark generator introduced in [18] can be easily fitted in this framework, since it is a generalization of these three modes. Thus, we refer the interested reader to this paper to see an actual implementation of the proposed framework.

6. CONCLUSIONS

General purpose benchmark generators are necessary to compare non-exact algorithms. They enable more thorough analysis and encourages portability of the algorithm to other applications. Benchmarks for COPs are treated as sequences of static problems strung together. Thus, it may be necessary to solve each of them to optimality, which can be expensive. This difficulty can be further complicated if changes involve values of the problem parameters, since such changes might prove to be dynamically insignificant.

Therefore, this paper proposes a method for generating benchmarks for COPs that requires the solving of the initial instance only while solutions to all other instances can be determined from a changing mapping scheme. In this way, the method does not suffer the usual limitations on the problem size and the sequence length.

Problem specific benchmarks tend to repel general readers who might be interested in the ideas used in the benchmark generator and the dynamic solver. Hence, the paper proposes a general framework for the generation of test problems for COPs. It is hoped that such a frame work helps unify approaches in the literature and forms a basis for designing benchmarks.

Future work will aim enhance the proposed mapping benchmark and the generalized framework, as both are in need of further analysis and improvement.

7. ACKNOWLEDGEMENTS

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

The authors would like to thank the anonymous reviewers for many valuable suggestions and comments.

8. REFERENCES

[1] Branke, J. Memory enhanced evolutionary algorithms for changing optimization problems. In Congress on Evolutionary Computation CEC99, volume 3, pages 1875-1882. IEEE, 1999.

[2] Branke, J. Evolutionary Optimization in Dynamic Environments. Kluwer, 2002.

[3] Eyckelhof, C. J., Snoek, M. Ant Systems for a Dynamic TSP, In ANTS 2002: Brussels, Belgium, 88-99, 2002.

[4] Goldberg, D. E. and Smith, R. E Nonstationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette, editor, Second International Conference on Genetic Algorithms, pages 59-68. Lawrence Erlbaum Associates, 1987.

[5] Grefenstette, J. J. Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In Congress on Evolutionary Computation, volume 3, pages 2031-2038. IEEE, 1999.

[6] Guntsch, M., Middendorf, M., Schmeck, H. 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, 860-867, 2001.

[7] Jin, Y. and Sendhoff, B. Constructing dynamic optimization test problems using the multi-objective optimization concept, EvoWorkshops 2004, LNCS 3005, 525-536, 2004.

[8] Lewis, L., Hart, E., and Ritchie G. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, Parallel Problem Solving from Nature, number 1498 in LNCS, pages 139-148. Springer, 1998.

[9] Mori, N., Kita, H., and Nishikawa, Y. Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In H.-M. Voigt, editor, Parallel Problem Solving from Nature, number 1141 in LNCS, pages 513-522. Springer Verlag Berlin, 1996.

[10] Mori, N., Kita, H., and Nishikawa, Y. Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm. In A. E. Eiben, T. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, Parallel Problem Solving from Nature, number 1498 in LNCS, pages 149-158. Springer, 1998.

[11] Morrison, R. W. and DeJong, K. A. A test problem generator for non-stationary environments. In Congress on Evolutionary Computation, volume 3, pages 2047-2053. IEEE, 1999.

[12] Reeves, C. R. and Rowe, J. E. Genetic Algorithms: Principles and Perspectives. A Guide to GA Theory. Kluwer Academic Publishers,Boston (USA), 2002.

[13] Trojanowski, K. and Michalewicz, Z. “Searching for optima in non-stationary environments,” in Proceedings of the Congress of Evolutionary Computation, Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, Eds., Mayflower Hotel, Washington D.C., USA, 6-9 July 1999, vol. 3, pp. 1843–1850, IEEE Press.

[14] Ursem, R., K., Krink, T., Jensen, M., T., and Michalewicz, Z. Analysis and Modeling of Control Tasks in Dynamic Systems. IEEE Transactions on Evolutionary Computation, 2002.

[15] Yang, S. Non-stationary problem optimization using the primal-dual genetic algorithm. Proc. of the 2003 Congress on Evolutionary Computation, Vol. 3, pp. 2246-2253, 2003.

[16] Yang, S. and Yao, X. Dual population-based incremental learning for problem optimization in dynamic environments. Proc. of the 7th Asia Pacific Symposium on Intelligent and Evolutionary Systems, pp.49-56, 2003.

[17] Younes, A. A Hybridized Genetic Algorithm for Solving the Dynamic Vehicle Routing Problem, 2nd Annual McMaster Optimization Conference: Theory and Applications (MOPTA 02), Hamilton, Canada, 2002.

[18] Younes, A., Basir, O., and Calamai, P. A Benchmark Generator for Dynamic Optimization. Proceedings of the 3rd WSEAS International Conference on Soft Computing, Optimization, Simulation & Manufacturing Systems., Malta, 2003.