Proceedings of the Fourth Workshop on

Evolutionary Algorithms for Dynamic Optimization Problems (EvoDOP-2005)

held in conjunction with the

2005 Genetic and Evolutionary Computation Conference (GECCO-2005)

26 June 2005, Washington DC, USA

edited by

Shengxiang Yang, University of Leicester, United Kingdom

Jürgen Branke, University of Karlsruhe, Germany

Evolutionary Algorithms for Dynamic Optimization Problems: Workshop Preface

Shengxiang Yang
Department of Computer Science
University of Leicester
University Road, Leicester LE1 7RH, U.K.
s.yang@mcs.le.ac.uk

Jürgen Branke
Institute AIFB
University of Karlsruhe
76128 Karlsruhe, Germany
branke@aifb.uni-karlsruhe.de

Categories and Subject Descriptors

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

General Terms: Algorithms

Keywords

Evolutionary algorithms, dynamic optimization problems

1. INTRODUCTION

Evolutionary algorithms (EAs) have been widely applied to solve stationary optimization problems. However, many real-world optimization problems are actually dynamic. For example, new jobs are to be added to the schedule, the quality of the raw material may be changing, and new orders have to be included into the vehicle routing problem etc. In such cases, when the problem changes over the course of the optimization, the purpose of the optimization algorithm changes from finding an optimal solution to being able to continuously track the movement of the optimum over time. This seriously challenges traditional EAs since they cannot adapt well to the changing environment once converged.

However, since in a sense natural evolution is a process of continuous adaptation and evolutionary algorithms are inspired from principles of natural evolution (e.g., selection and variation), it seems straightforward to consider evolutionary algorithms with proper enhancement as appropriate candidates for dynamic optimization problems (DOPs).

In recent years, there has been a growing interest in studying EAs for dynamic problems since many real world problems are known to be dynamic [1]. And the number of papers published in this area is rising continuously (see e.g. the online repository on the topic [8]). Most of these publications can be grouped into one of the following basic categories [4]:

- Identify the occourence of a change in the environment and then deliberately increase diversity in the population, e.g. by means of increased mutation [5, 11];
- Try to avoid convergence all the time, e.g. by including new random individuals in the population in every generation [7, 15];
- Supply the EA with a memory, e.g. by using diploidy [6, 9, 10, 12] or an explicit memory [2, 13, 16], so that the EA can recall useful information from past generations;
- Using multiple populations to cover several promising areas of the search space simultaneously [3, 14].

The purpose of the workshop is to foster interest in the important subject of evolutionary algorithms for dynamic optimization problems, get together the researchers working on the topic, provide an overview on the field, and discuss recent trends and future directions in the area.

The EvoDOP-2005 workshop, held as a part of GECCO-2005, is the fourth of a successful series of bi-annual workshops on "Evolutionary Algorithms for Dynamic Optimization Problems". The past three EvoDOP workshops have been held at GECCO-1999, GECCO-2001, and GECCO-2003 respectively with 60-100 participants each.

2. EVODOP-2005 PROGRAM

For the EvoODP-2005 workshop, six papers of high quality have been accepted for presentation. Younes et al. propose a method for constructing general benchmark dynamic combinatorial optimization problems, which is an important topic for performance comparisons of EAs. Rand and Riolo describe a set of measures to examine the behaviour of genetic algorithms (GAs) in dynamic environments and use these measures to examine the GA behaviour with a dynamic test suite, called the *shaky ladder hyperplane-defined functions*. Bosman tangles the time-linkage problem (i.e., decisions taken now may influence the score in the future) and shows how such time-linkage can deceive an optimizer. A means of predicting the future by learning from the past is proposed and formalized in an algorithmic framework to address the time-linkage problem. Boumaza studies the relationship between the dynamics of the environment and the self-adaptation of the mutation steps of evolutionary strategies and shows through experimentation that the nature of the movements of the optimum is reflected in the self-adaptive mutation step. The paper by Dudy et al. presents a study on inverse robust evolutionary design in the presence of uncertainty based on the concept of multi-objective optimization. For complex real-world problems, small populations for EAs are very desirable due to computational

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

cost. However, small population can dramatically reduce the performance of EAs. Jin et al. suggest a method to find the optimal search accuracy for evolutionary strategies with a small population[1].

The workshop concludes with a panel discussion of relevant topics, as shown below.

3. TOPICS FOR DISCUSSIONS

The EvoODP-2005 workshop is open to all registered attendees of the GECCO-2005 conference. We are open for topics that should be discussed during the panel discussion. Some preliminary topics for discussion are listed as follows:

- What constitutes a good benchmark DOP?
- What factors contribute to the difficulty of EAs for dynamic optimization problems?
- How should one measure "adaptability"?
- What makes a DOP different from a static problem?
- What is the difference between a dynamic optimization problem and a control problem?
- What are the deficits of current approaches?
- What properties should one pursue when analysing EAs for dynamic optimization problems?
- What tools are available to analyse EAs for DOPs?

The topics discussed in EvoDOP-2005 will surely lead to interesting future directions for evolutionary algorithms for dynamic optimization problems.

4. PROGRAMME COMMITTEE

The programme committee for the EvoDOP-2005 workshop reviewed the papers and will also lead the panel discussion into interesting future directions for evolutionary algorithms for dynamic optimization problems.

- Shengxiang Yang (Co-chair, Univ. of Leicester, UK)
- Jürgen Branke (Co-chair, Univ. of Karlsruhe, Germany)
- Hussein A. Abbass (University of New South Wales, Australia)
- Tim Blackwell (University College London, UK)
- Ernesto Costa (University of Coimbra, Portugal)
- Kenneth A. De Jong (George Mason University, USA)
- Daniel Merkle (University of Karlsruhe, Germany)
- Ron Morrison (Mitretek Systems, Inc., USA)
- William Rand (University of Michigan, USA)
- Karsten Weicker (University of Stuttgart, Germany)
- Sima Uyar (Istanbul Technical University, Turkey)

We would like to thank all who have helped making the workshop a success, especially the programme committee members, and wish all participants enjoy the workshop.

[1]On the request of the authors, the work by Jin et al. will be presented at EvoDOP-2005 and included in the CD-ROM entitled "Workshop Proceedings, Tutorials, and Late-Breaking Papers at the 2005 Genetic and Evolutionary Computation Conference" as a late-breaking paper instead of in the workshop proceedings and the ACM digital library.

5. REFERENCES

[1] T. Bäck. On the behaviour of evolutionary algorithms in dynamic fitness landscape. In *Proc. of the 1998 IEEE Int. Conf. on Evolutionary Computation*, pages 446–451, 1998.

[2] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proc. of the 1999 Congress on Evolutionary Computation*, volume 3, pages 1875–1882, 1999.

[3] J. Branke, T. Kaußler, C. Schmidth, and H. Schmeck. A multi-population approach to dynamic optimization problems. *Adaptive Computing in Design and Manufacturing*, pages 299–308, 2000.

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

[5] H. G. Cobb and J. J. Grefenstette. Genetic algorithms for tracking changing environments. In *Proc. of the 5th Int. Conf. on Genetic Algorithms*, pages 523–530, 1993.

[6] D. E. Goldberg and R. E. Smith. Nonstationary function optimization using genetic algorithms with dominance and diploidy. In *Proc. of the 2nd Int. Conf. on Genetic Algorithms*, pages 59–68, 1987.

[7] J. J. Grefenstette. Genetic algorithms for changing environments. In *Proc. of the 2nd Int. Conf. on Parallel Problem Solving from Nature*, pages 137–144, 1992.

[8] Internet repository on "evolutionary algorithms for dynamic optimization problems," online, `http://www.aifb.uni-karlsruhe.de/~jbr/EvoDOP`.

[9] E. H. J. Lewis and G. Ritchie. A comparison of dominance mechanisms and simple mutation on non-stationary problems. In *Proc. of the 5th Int. Conf. on Parallel Problem Solving from Nature*, pages 139–148, 1998.

[10] N. Mori, H. Kita and Y. Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In *Proc. of the 7th Int. Conf. on Genetic Algorithms*, pages 299–306. Morgan Kaufmann Publishers, 1997.

[11] R. W. Morrison and K. A. De Jong. Triggered hypermutation revisited. In *Proc. of the 2000 Congress on Evol. Comput.*, pages 1025–1032, 2000.

[12] K. P. Ng and K. C. Wong. A new diploid scheme and dominance change mechanism for non-stationary function optimisation. In *Proc. of the 6th Int. Conf. on Genetic Algorithms*, 1997.

[13] C. L. Ramsey and J. J. Greffenstette. Case-based initializtion of genetic algorithms. In *Proc. of the 5th Int. Conf. on Genetic Algorithms*, 1993.

[14] R. K. Ursem. Multinational GAs: Multimodal optimization techniques in dynamic environments. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 19–26, 2000.

[15] S. Yang. Memory-based immigrants for genetic algorithms in dynamic environments. In *Proc. of the 2005 Congress on Evolutionary Computation*, 2005.

[16] S. Yang. Population-based incremental learning with memory scheme for changing environment. In *Proc. of the 2005 Congress on Evolutionary Computation*, 2005.

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.

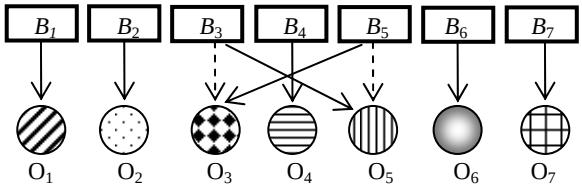


Figure 1. swapping in mapping function.

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.

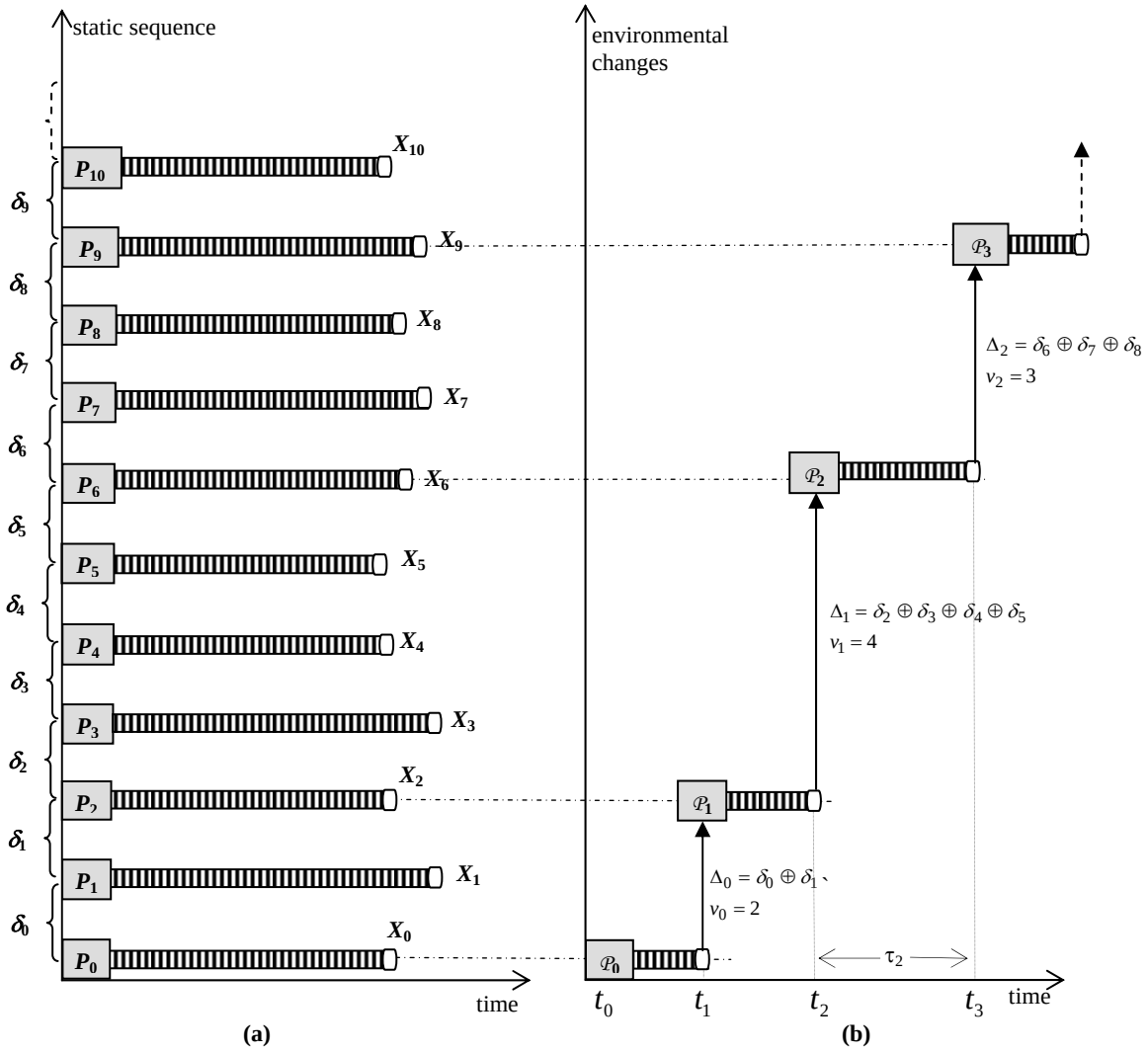


Figure 2 Generalized Benchmark Generation

(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.

Measurements for Understanding the Behavior of the Genetic Algorithm in Dynamic Environments

A Case Study using the Shaky Ladder Hyperplane-Defined Functions

William Rand and Rick Riolo
Center for the Study of Complex Systems
University of Michigan
4485 Randall Lab
Ann Arbor, MI, USA, 48109-1120
wrand@umich.edu

ABSTRACT

We describe a set of measures to examine the behavior of the Genetic Algorithm (GA) in dynamic environments. We describe how to use both average and best measures to look at performance, satisficability, robustness, and diversity. We use these measures to examine GA behavior with a recently devised dynamic test suite, the Shaky Ladder Hyperplane-Defined Functions (sl-hdf's). This test suite can generate random problems with similar levels of difficulty and provides a platform allowing systematic controlled observations of the GA in dynamic environments. We examine the results of these measures in two different versions of the sl-hdf's, one static and one regularly-changing. We provide explanations for the observations in these two different environments, and give suggestions as to future work.

Categories and Subject Descriptors

F.2.m [**Analysis of Algorithms**]: Misc.; I.2.8 [**Artificial Intelligence**]: Search

General Terms

Algorithms, Measurement

Keywords

Genetic Algorithms, Measurement, Dynamic Environments, Hyperplane-Defined Functions, Genetic Algorithms

1. INTRODUCTION

The Genetic Algorithm (GA) has been shown to work successfully in many dynamic environments [1] [3], and while work has been done on trying to understand how the GA works in these environments, the behavior of the GA in dynamic environments is still not well understood. Part of the problem is that when examining a GA on a particular search problem often researchers are interested in the performance of the GA and not necessarily why the GA works and thus they only report performance measures. This does not always give a good indication of the overall behavior of the GA. Thus in this paper we present a suite of measures that are useful to measure beyond the standard performance metrics. By examining all of these measures together we hope to more fully understand the behavior of the GA in dynamic environments.

In a related area, Branke et al, among others, are interested in characterizing and measuring the dynamic landscape that evolutionary algorithms (EAs) are operating within [4]. While that work is related, that work does not directly address the goal we are addressing here, i.e., understanding the behavior of the EAs themselves, not the landscape they operate upon. Moreover, many researchers have developed benchmark dynamic landscapes, like the moving peaks function, that allow the comparison of different EAs to evaluate their performance [2] [13]. This work is also related but tangential to the work described here, since we do not want to compare different EAs but instead we are concerned with understanding the behavior of the simple GA.

In order to systematically examine the behavior of the GA we require a test suite of functions. The test suite presented here is similar to the dynamic bit matching functions utilized by Stanhope and Daida [17] among others. The difference between the test suite in this paper and other dynamic test functions is that the underlying representation of this test suite is schemata, which make it easier to examine how the GA is operating. By utilizing a test suite that reflects the way the GA searches, the performance of the GA can be easily observed. This test suite is a subset of John Holland's hyperplane-defined functions (hdfs) [9]. We have extended the hdfs to dynamic environments and we call this test suite, the Shaky Ladder Hyperplane-Defined Functions (sl-hdf's) [15].

In the rest of this paper, we examine the measures that we are interested in exploring, briefly describe the sl-hdf's, and finally present some results of these measures on the sl-hdf's in two different environments: a static and a regularly-changing environment. We conclude by discussing these re-

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

sults, providing some preliminary explanations and describing future work.

2. MEASURES

Though the performance of the GA is an important thing to measure, there may be other measures that can be useful along with performance in order to give a better description of how the system is behaving. This is especially true in dynamic environments where the behavior of the system over time can vary dramatically as the environment changes. Other measures may provide valuable explanations and allow researchers to better describe and characterize the dynamics of populations being changed by a GA and an associated (exogenous or endogenous) mechanism that assigns fitness ratings to the individuals.

In general researchers view the behavior of the GA from two different perspectives. Some are concerned with understanding extreme behaviors of the system, particularly the best that the system can do. These tend to be the measures preferred by application practitioners, who want to know what is the best result the system can possibly obtain. These measures tend to be of more interest to people who have particular problems that they are trying to solve. Other measures which may be useful characterize the population as a whole, i.e. average, standard deviations, distributions. Population distribution measures are often important to researchers trying to understand GAs as representations of evolutionary systems. They utilize the GA to model these systems and thus the behavior of the best individual in the system is not as important to them. Given that loose dichotomy, we examine all of the measures below through the context of both a best and an average measure.

Since a population evolving under the influence of a GA is, in general, a complex adaptive system, there are inherent stochastic and path-dependent processes such that each "run" of the GA, even given the same starting population, will almost always result in dynamics that differ in the details, and in some cases, the results will differ qualitatively from run to run as well. When the primary goal is to find a high performance solution for a real world problem, it often is most useful to track the best fitness individuals from each of a set of runs and then focus on the best of those best-of-run individuals. On the other hand, in other situations it is important to include measures that reflect the distribution of population histories, since that gives more information about both the complexity of the solution space and the dynamical properties of populations evolving in that problem space.

Standard measures of performance fall into a category we call *fitness-related* measures, which describe the system by examining the fitness ratings of the individuals in the population. Fitness in an evolutionary algorithm (EA) is defined to be the score that influences selection and hence determines the ability of the individual to replicate [8]. Performance, on the other hand, is a score that the individual receives on an objective function. Often fitness is performance in EAs; however, there are cases where other factors are included in the fitness score. In this paper the fitness score of an individual is defined as the performance score. Besides the standard measure which we call *performance*, we describe two additional fitness-related measures: *satisficability*, and *robustness*. Another category of measures is *composition-related* measures which attempt to describe the behavior of the system by describing the components that currently make up the system. In this paper we chose to look at only one composition-related measure, *diversity*.

We describe these four measures in detail below, but briefly, performance is the standard measure of how well the system performs the task presented it, satisficability is the ability of the system to maintain a certain level of fitness and to avoid egregious errors, robustness is a measure of how the system responds to changes in the environment, and finally diversity is a measure of how different the members of the population are. We do not think of this as a definitive list of all the possible measurements of a GA's behavior, but rather as a representative list of measures that are interesting to both those with a design or engineering perspective and those with a biological perspective.

2.1 Performance

Performance is the standard measure of how well the system solves an objective function. As mentioned, in many cases, this is the fitness function, but some GAs such as co-evolving systems, and systems with implicit fitness functions, do not have an absolute fitness function. In this paper we use the fitness function as a measure of performance. One use of performance is in evaluating the suitability of a GA-discovered solution to solving a design problem. The GA has proved to be particularly useful if there are epistatic interactions between variables. Examining the performance of the best individual in a population gives an estimate of the best solution the GA can find for a problem, given a set of GA parameters and the resulting total computational effort expended, in terms of the total number of fitness evaluations required [11]. For biological modelers it is more important to understand the performance of the whole population. The average fitness of the population provides a first-order representation of the overall fitness distribution, and thus is a good first measure. In both cases it usually makes sense when describing how the system works to aggregate the results of multiple runs by averaging the results across runs, since the average gives a good indication of how the system is expected to do on any given run.

In the experiments described below, we define *Best Performance* to be the average across n runs of the fitness of the best individual of the current generation in the population. One could also examine the best performance of any individual in any run, but we leave investigations of that measure for the future. *Average Performance*, on the other hand, is the average across n runs of the average fitness of the population. Since we know the optimal value a priori given the sl-hdf construction scheme, we express these measures as a fraction of the optimal fitness possible. This allows us to compare results where the optimal value changes.

2.2 Satisficability

Satisficability is a measure of when the system is able to achieve a predetermined criteria. The notion was first introduced by Simon who claimed that often humans do not optimize the solution to a problem but instead simply come up with a solution that satisfices some criteria they have previously defined [16]. In this paper, satisficability measures how well the system is able to maintain a certain level of fitness and not drop below a pre-set threshold. One application of this measure would be autonomous agent control. If a GA is in charge of steering a robot from one location to

another, it maybe not be necessary to get there as quickly as possible but the design specifications may require that it will get there in a reasonable amount of time.

Whether we are interested in the average satisficability or the best satisficability we set a threshold and count how many times the system is able to exceed that threshold. In the experiments described below, we define *Best Satisficability* to be the fraction of runs out of n that the GA's best solution in that generation exceeds $s = o\theta$ which is expressed as a fraction θ of the optimal o. This gives us a clear indication of how many times the system will return a result that is at least as good as our threshold. The fraction, θ (between 0 and 1), is a parameter to the measurement. *Average Satisficability* is defined to be the average across n runs of the fraction of individuals in the current population that exceed s. Of course it would be simple to extend this measure to a problem where the optimal was not known, in which case o could be set to 1 and $s = \theta$, which means that the measures would be based on the fraction of individuals or runs to exceed an absolute threshold.

2.3 Robustness

Of all the measures listed here, robustness is probably the most complicated and has the most varied definitions. There are many different notions of robustness [10], and thus defining it must always be done within the context of a particular question. Here we specifically address the idea of robustness as a measure of how a system's output changes in response to environmental changes. We want to know how much the fitness of the next generation of the GA can drop, given the current generation's fitness. The idea is that the performance of the system should never dramatically decrease since a dramatic decrease may upset other elements if the system is not isolated. In order to make this measure more useful in real-world systems it is important to define it so that it is not necessary to know when the environment changed, because changes are not always easily observable. One application of such a measure would be managing assets in a stock market portfolio. For instance a GA could be used to specify which stocks to hold at each time step, with the goal of never suffering a dramatic decreases in the total value of the portfolio. As long as the overall net value is increasing the owner is earning money, but it is important to make sure that this portfolio is robust to dramatic changes.

In the experiments below, we define *Best Robustness* to be the fitness of the best individual in the current generation divided by the fitness of the best individual in the previous generation. If this score is greater than 1 we set it equal to 1. For simplicity we set the robustness score of the first generation equal to 1. We then average this measure across n runs. We define *Average Robustness* to be the same measure but for the average fitness of the population.

2.4 Diversity

Diversity is a measure of the variance in the genomes in the population of solutions. Diversity captures the notion of how much of the search space the GA is currently exploring. Moreover having a diversity of solutions may mean the system is more able to adapt to changes in the environment. The diversity within the run of an EA has been studied many times before, including attempts to use it as an objective in a multi-objective EA fitness function [18]. A diversity of solutions is often needed to avoid premature convergence, which can cause an EA to get stuck at a local optima. In fact a whole variety of techniques have been used to maintain diversity throughout a run, like niching through fitness sharing [6]. Moreover similar techniques have been used to try and maintain diversity in dynamic environments throughout a run [7] as well as after a change in the environment[1] [5]. Besides the study of diversity within GAs for the purposes of studying GAs, it is also important to look at diversity within the context of biological modeling. Very often biological diversity is considered important to the success of a species and thus studying how different parameters of the GA affect the overall diversity of the system could be interesting to biological modelers [20].

The above concept of diversity is concerned with measuring how diverse solutions are within a run. Another measure of interest is how diverse solutions are across runs. This provides an idea of how different the various results of the GA will be between, as opposed to within, runs. In some cases diversity among runs would be a good thing, since it would indicate the system is able to find very different parts of the search space that may not have been obvious as potential areas for fruitful exploration. In other cases diversity may be a bad thing because it indicates that the system is very dependent on initial conditions.

In order to examine diversity we chose to use Hamming distance because it measures how many single bit mutations would be required to move from one string to another within the population. Likewise Hamming distance is also the Manhattan distance between two vertexes in an n-dimensional hypercube. Thus Hamming distance is a good approximation of how far apart two solutions are in the sl-hdf solution space.

In the experiments discussed below, we define *Best Diversity* to be the average Hamming distance between the genomes of the best individuals of each generation found in each of the n runs. We can also observe the diversity within a run and thus, we define *Average Diversity* to be the average Hamming distance between every member of the population averaged over n runs. To allow for comparisons of diversity where GA string lengths differ, we normalize these measures to the length of the string.

However, we do not present the results from the Best Diversity of the experiments. The reason is that in the case of the sl-hdf's the random number seed used to generate the population of the GA is also used to generate the particular problem to be solved. The result of this is that in every run the GA is facing a different problem, and thus the Hamming distance between different runs of the GA is always 0.5 since they are optimizing toward different fitness peaks.

3. SHAKY LADDER HYPERPLANE-DEFINED FUNCTIONS

The functions that we will be utilizing to explore the GA in dynamic environments are a subset of the hdfs [9]. The hdfs are designed to represent the way the GA searches by combining building blocks (through the use of schemata) hence they are appropriate for understanding the behavior of the GA. The hdfs were constructed to meet a set of criteria specified by Whitley [19]. The problem with the hdfs in the dynamic case is that the optimal set of strings is not easily known given the definition of the function, and thus the

[1]For a more thorough review please consult Bränke [3]

absolute performance can not easily be measured. Moreover, there is no way to take one hdf and create another that is similar to it, which would be useful when exploring dynamic environments.

Thus we impose three conditions on the full suite of hdfs and use a simple algorithm to these functions which solves these two problems. The process described below is more thoroughly explained in previous work [15] [14]. The first condition is the *Unique Position Condition* (UPC). It requires that all elementary schemata contain no conflicting bits. The second condition we call the *Unified Solution Condition* (USC). This condition guarantees that all of the specified bits in the elementary level schemata must be present in the highest level schema. The third condition is the *Limited Pothole Cost Condition* (LPCC), which states that the fitness contribution of any pothole plus the sum of the fitness contributions of all the building blocks in conflict with that pothole must be greater than zero. These three conditions guarantee that any string which matches the highest level schema must be a string with optimal fitness. By knowing the optimal set of strings we solve one of the problems with Holland's original hdfs.

Assuming these conditions, once a set of elementary schemata have been established we already know the highest level schema. If we hold the elementary and highest level schemata constant, we can generate new similar hdfs by creating new intermediate schemata, we call this *Shaking the Ladder*. Thus we have an easy way to create similar but different hdfs randomly, and solve the second problem with Holland's original hdfs.

For the experiments discussed below, we used sl-hdf's with a length of 500. There are 50 elementary schemata of order 8, 5 intermediate levels of schemata, and 1 highest level schemata. The length of the schemata is unknown since the location of the fixed bits are chosen randomly. Moreover the order of the higher level schemata is also unknown since the elementary schemata can "share" fixed loci. However we can place upper limits on the order. If all elementary schemata are disjoint, the maximum order of the highest level schemata is 400, which means there will always be at least 100 wildcards present in the highest level schemata.

4. EXPERIMENTS AND RESULTS

The basic setup for our experiments is a simple GA using the sl-hdf as its fitness function. The base GA presented here uses one-point crossover, per bit mutation, full population replacement, and is similar to the one described by Mitchell [12]. In this set of experiments we only change one variable, t_δ, which specifies the number of generations between shakes of the ladder. We only examine two values for t_δ, 1801 and 100. $t_\delta = 1801$ represents a static environment because the time between changes exceeds the run of the GA. $t_\delta = 100$ represents a regularly changing environment. The optimal value achievable by the sl-hdf is 1.0. All results below are presented at 10 generation increments in order to make the graphs easier to read.

Figure 1 illustrates both the fitness of the best individual in the population (Best Performance) and the average fitness of the whole population (Average Performance) for both t_δ values, averaged across 30 runs. These results have been normalized to 0 to 1; the non-normalized optimal value of these sl-hdf's is 191. Figure 2 illustrates (for both t_δ values) how many best of generation individuals out of 30 runs

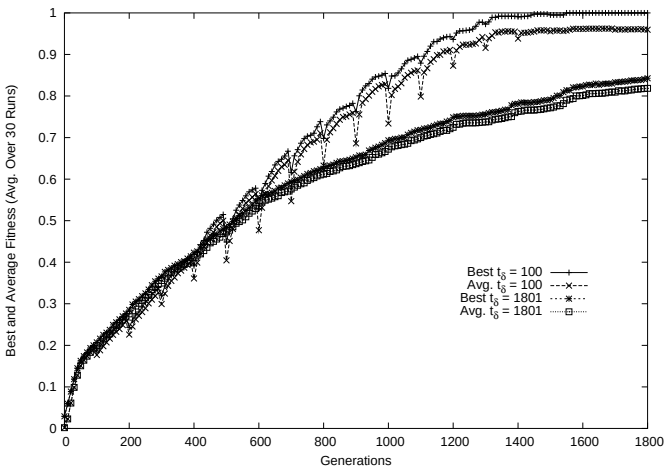


Figure 1: Performance Results

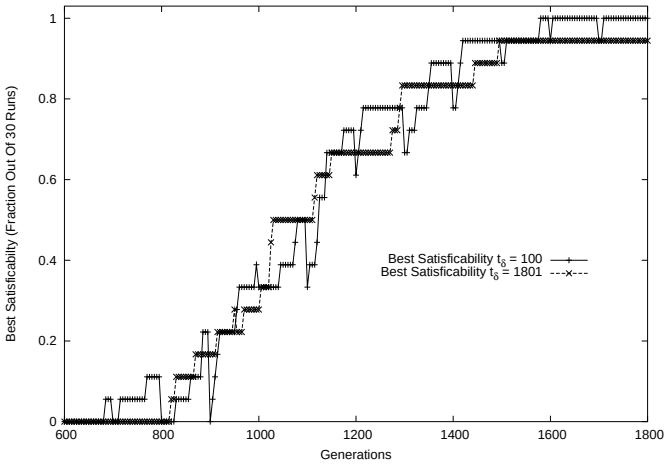


Figure 2: Satisficability Results

(Best Satisficability) were able to satisfice a goal of achieving $\theta = 0.5$, where $o = 191$, and thus $s = 0.5 \times 191 = 95.5$. The actual value of θ that is chosen in this case is arbitrary since the problem is abstract. Instead what is interesting is observing the dynamics of the measure over time which will be explored below. In Figure 2 only the last 1200 generations are presented to increase the resolution of the data. Figure 3 displays the robustness of the best individual in the population (Best Robustness) and the robustness of the average fitness of the population (Average Robustness) for $t_\delta = 100$ across the entire run (averaged across 30 runs). Figure 4 displays the average scaled hamming distance of the population (Average Diversity) for the last 1300 generations for both t_δ values.

5. DISCUSSION

The performance of the system has been more thoroughly

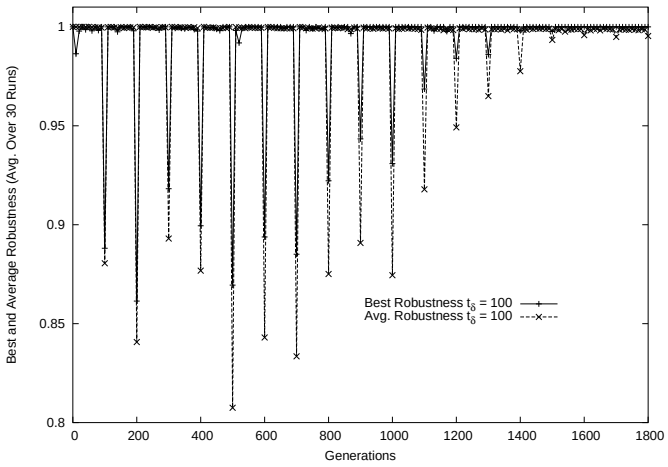


Figure 3: Robustness Results

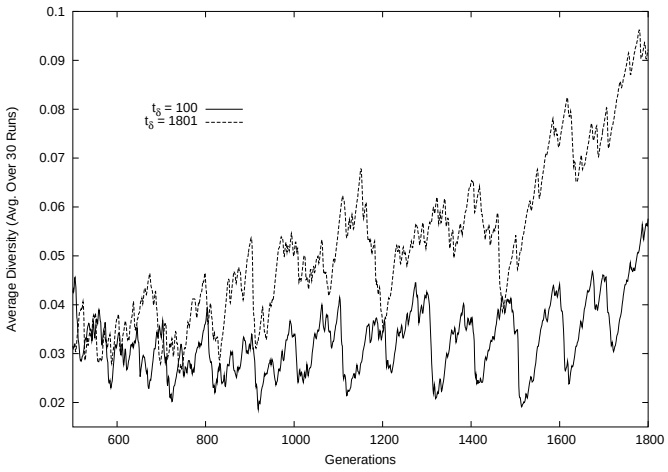


Figure 4: Diversity Results

explored in a previous paper [15]. However, it is clear from the results here that the regularly changing environment is able to outperform the static environment in the long run. Initially the dynamic environment under-performs the static environment but before halfway through the run the dynamic environment has achieved a superior fitness in both the best individual and the average fitness. We believe this is because the regularly changing environment prevents the GA from being locked into particular building blocks and forces it to explore a wider range of intermediate schemata. It is also interesting to note that when the ladder is shaken in the regularly changing environment, the Average Performance of the system falls farther than the Best Performance of the system (see Figure 3 for additional clarification). This makes sense– when the ladder is shaken many of the individuals that were being rewarded before lose those rewards and hence their fitness falls greatly; however it is reasonable to suppose that there are some individuals immediately after a shake that have a higher performance (the new best individuals) than they did before the shake and thus they mitigate the fall of the best performance.

Given the performance results the satisficability results are interesting as well. Since the satisficing results are measures of individuals above a certain threshold, it would make sense that the dynamic environment, which outperforms the static environment, would also have a higher level of satisficability. However, both the static and dynamic environments behave in a similar fashion. The satisficing threshold is set at 0.5 and the elementary building blocks constitute almost half of the fitness function reward. Thus, one hypothesis to explain why both systems are able to achieve similar levels of satisficability is that both the dynamic and static environments are finding basic building blocks at roughly the same rate. However the dynamic environment outperforms the static environment because it is better at finding the intermediate building blocks. The dynamic environment is getting rewarded for different intermediate building blocks and thus has a higher selection pressure to find them, whereas the static environment is mainly under pressure to find the elementary building blocks and the particular intermediate schemata that are rewarded in its sl-hdf instance. We do not present the results of Average Satisficability but it closely mirrors the Best Satisficability, which is interesting since it is a measure within a run instead of across runs, and thus there is no guarantee that it would be the same.

The robustness results are also interesting. The static environment results are not presented since it is almost always able to maintain its robustness, which means that its fitness is constantly improving. The one exception to this is that occasionally the best individual suffers some degradation, probably due to a deleterious mutation. However, for $t_\delta = 100$, every 100 generations the robustness decreases substantially, but then immediately recovers. Moreover, the robustness score at each shake changes as the run goes on. Basically there are three phases to the robustness score, early on (Generation 0 to Generation 400) the decreases are small, in the middle generations (Generation 400 to Generation 1000) they are larger, and in the final generations (Generation 1000 to Generation 1800) they are small again. The first phase is because the population has little fitness value to lose. The population is dominated by individuals who have a few elementary schemata and maybe one or two intermediate schemata, thus the ladder shakes have little ef-

fect on them since it only changes intermediate schemata. At the end of the run the GA has found most of the intermediate schemata, but has not assembled them into one individual, and thus it is not affected much by shakes of the ladder since there is some individual in the population that has the new intermediate schemata. In the middle is when the GA has the largest decreases in robustness, and this is because at this phase the GA has devoted lots of resources to exploring particular intermediate schemata. Also, the Best Robustness score never falls as much as the Average Robustness score. This is explained above when discussing the performance results.

The most interesting result is the diversity results. Our initial hypothesis was that the overall diversity of the static system would decrease as time went on, indicating that the population was converging, and that diversity in the regularly changing environment would increase immediately after a change but then decrease again. However that is not what happens. Instead it appears that for $t_\delta = 1801$ diversity (minus some noise) always increases, whereas for $t_\delta = 100$ diversity varies around some average, but decreases immediately after a ladder shake and then increases until the ladder is shaken again. Our new hypothesis is that the static environment's population is dominated by a few strong individuals but over time these individuals gain additional mutations that are neutral. This results in diversity increasing. In the regularly changing environment on the other hand, when the ladder is shaken, there are just a few individuals that contain the proper bit values to work well in the new environment, and thus these individuals quickly dominate the population. This results in the loss of diversity of all the previous explorations that were going on, and a very sharp founder's effect [9] which results in a quick convergence of the population around these new intermediate building blocks.

This explanation may seem to contradict the earlier statements that the dynamic environment prevents the premature convergence (loss of diversity) of the GA on particular intermediate building blocks. However the diversity measure that we present here is based on bits, not building blocks. There are some bits that never matter even in optimal strings, since they are wildcards in the highest level schema, we will call these highest level wildcards (as mentioned above the lower limit on this is 100 bits). There are other bits that are not as important given the current elementary schemata present in the population and the current intermediate schemata being rewarded, we will call these currently quasi-neutral bits. The problem with these bits is that only if an entire new elementary schemata (8 bits out of 500) is discovered and the proper combination of it with a few other elementary schemata occurs, is there enough of a selection pressure to allow these bits to go to fixation. Thus, diversity increases in the static environment because both highest level wildcards and the currently quasi-neutral bits can mutate without greatly affecting the fitness of an individual. In the dynamic environment diversity is kept lower because the shaking of the ladder causes a strong selection pressure that forces the population to move to a new set of intermediate schemata, and only after that can the currently quasi-neutral bits mutate. However the above explanation does not say anything about the diversity of schemata (building blocks). The dynamic environment has a larger diversity of schemata and that is how it is able to outperform the static environment, whereas the the static environment quickly converges on a group of intermediate schemata and does not search for additional combinations.

6. CONCLUSION

The overall goal of our project is to better understand how the GA works in dynamic environments. We have presented a set of measures that we feel will help us in better understanding the overall behavior of the GA in dynamic environments. Our observations and results here are on the sl-hdf's but we hope to show that these measurements are useful in a wide variety of environments. Though many of these measures have been presented before in different formats, by viewing them as a suite we are able to gain a deeper understanding of how the GA performs in dynamic environments. We plan to continue to conduct systematic controlled observations of the GA. We feel that this allows us to contribute to theory by providing a series of regular observations and to contribute to practice by providing suggestions for a rich set of environments.

Acknowledgments
We would like to thank the University of Michigan's Center for the Study of Complex Systems for the computer resources that they have provided to us without which this paper would not have been possible.

7. REFERENCES

[1] Branke, J. Evolutionary algorithms for dynamic optimization problems: A survey. Tech. Rep. 387, Institute AIFB, University of Karlsruhe, February 1999.

[2] Branke, J. Memory enhanced evolutionary algorithms for changing optimization problems. In *Congress on Evolutionary Computation CEC99* (1999), vol. 3, IEEE, pp. 1875–82.

[3] Branke, J. *Evolutionary Optimization in Dynamic Environments*. Kluwer Academic Publishers, 2001.

[4] Branke, J., Salihoğlu, E., and Uyar, S. Towards an analysis of dynamic environments. In *To appear in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)* (2005).

[5] Cobb, H. G. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Tech. Rep. AIC-90-001, Naval Research Laboratory, Washington, D.C., 1990.

[6] Goldberg, D. E., and Richardson, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms* (Hillsdale, New Jersey, 1987), J. J. Grefenstette, Ed., Lawrence Erlbaum Associates, pp. 41–9.

[7] Grefenstette, J. J. Genetic algorithms for changing environments. In *Parallel Problem Solving from Nature 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels 1992)* (Amsterdam, 1992), R. Männer and B. Manderick, Eds., Elsevier, pp. 137–144.

[8] Holland, J. *Adpatation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[9] Holland, J. H. Building blocks, cohort genetic algorithms, and hyperplane-defined functions. *Evolutionary Computation 8*, 4 (2000), 373–391.

[10] Jen, E. Stable or robust? what's the difference? *Complexity 8*, 3 (January / February 2003), 12–18.

[11] Koza, J. R. *Genetic Programming*. MIT Press, Cambridge, Massachusetts, 1992, ch. 8.

[12] Mitchell, M. *An Introduction To Genetic Algorithms*. MIT Press, Cambridge, Massachusetts, 1997, ch. 1.

[13] Morrison, R. W., and DeJob, K. A. A test problem generator for non-stationary environments. In *Congress on Evolutionary Computation CEC99* (1999), vol. 3, IEEE, pp. 2047–53.

[14] Rand, W., and Riolo, R. The problem with a self-adaptative mutation rate in some environments: A case study using the shaky ladder hyperplane-defined functions. In *To appear in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)* (2005).

[15] Rand, W., and Riolo, R. Shaky ladders, hyperplane-defined functions and genetic algorithms: Systematic controlled observation in dynamic environments. In *EvoWorkshops 2005 Proceedings* (2005), R. et al., Ed., Lecture Notes In Computer Science, Springer. In Press.

[16] Simon, H. *Models of Man*. Wiley, New York, 1957.

[17] Stanhope, S. A., and Daida, J. M. Optimal mutation and crossover rates for a genetic algorithm operating in a dynamic environment. In *Evolutionary Programming VII* (1998), no. 1447 in LNCS, Springer, pp. 693–702.

[18] Toffolo, A., and Benini, E. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation 11*, 2 (2003), 151–67.

[19] Whitley, D., Rana, S. B., Dzubera, J., and Mathias, K. E. Evaluating evolutionary algorithms. *Artificial Intelligence 85*, 1-2 (1996), 245–276.

[20] Wilson, E. O. The current state of biological diversity. In *Biodiversity*, E. O. Wilson and F. M. Peter, Eds. National Academy Press, Washington, 1988, pp. 3–18.

Learning, Anticipation and Time–Deception in Evolutionary Online Dynamic Optimization

Peter A.N. Bosman
Centre for Mathematics and Computer Science
P.O. Box 94079
1090 GB Amsterdam
The Netherlands
Peter.Bosman@cwi.nl

ABSTRACT

In this paper we focus on an important source of problem–difficulty in (online) dynamic optimization problems that has so far received significantly less attention than the traditional shifting of optima. Intuitively put, decisions taken now (i.e. setting the problem variables to certain values) may influence the score that can be obtained in the future. We indicate how such time–linkage can deceive an optimizer and cause it to find a suboptimal solution trajectory. We then propose a means to address time–linkage: predict the future by learning from the past. We formalize this means in an algorithmic framework. Also, we indicate why evolutionary algorithms are specifically of interest in this framework. We have performed experiments with two new benchmark problems that contain time–linkage. The results show, as a proof of principle, that in the presence of time–linkage EAs based upon this framework can obtain better results than classic EAs that do not predict the future.

Categories and Subject Descriptors

F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Online Computation*; G.1 [**Numerical Analysis**]: Optimization; I.2 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Performance, Experimentation

Keywords

Evolutionary Algorithms, Dynamic Optimization, Online Optimization, Learning, Predicting

1. INTRODUCTION

The majority of the literature on dynamic optimization [11] involves the tracking of optima as the search space transforms over time. If evolutionary algorithms (EAs) [14] are used to achieve this goal, issues such as maintaining diversity around (sub)optima and continuously searching for new regions of interest that may appear over time are the most important [1, 2, 3, 4, 7, 8, 9, 13, 15, 16, 19, 24, 27, 31]. The shifting of optima in dynamic optimization problems is important to study and to (re)design EAs for. However, there is another feature of dynamic optimization problems that is common in real–world problems such as scheduling [10] and vehicle routing [21, 22, 29] that has received less attention in the literature. We will call this feature *time–linkage*.

Intuitively put, the presence of time–linkage in a dynamic optimization problem causes decisions that are made now, which are often made on the basis of maximizing a certain score right now, may influence the maximum score that can be obtained in the future. This in turn decreases the overall score obtained in the long run. A typical and illustrative example is the case of dynamic vehicle routing where the locations to visit are announced over time. If locations are clustered, but the clusters themselves are far apart, routing on the basis of the currently available locations will likely lead to oscillatory behavior of the vehicles if the announced locations oscillate between the clusters. More efficient routes could be formed by keeping vehicles inside clusters and only occasionally letting them move to another cluster. In addition, quality of service (e.g. being on time) as determined by the routing, influences future customer demand. Poor performance will likely not result in repeated orders. Hence, the revenue of a company over time will be determined by the current performance, but also by the impact the current way of routing has on future events.

Time–linkage in dynamic optimization problems is to a certain extent related to the temporal credit assignment problem in reinforcement learning [25]. In the temporal credit assignment problem, the feedback from the system for an action taken now is returned after a certain delay. Hence it is uncertain when taking an action what its eventual effect is on the environment. The problem then is how to assign credit to an action taken in the past based upon feedback received now. Ultimately, the goal in reinforcement learning is to learn a policy for navigating a state space optimally and the same action may be taken starting from the same state multiple times while performing reinforcement learning. The difficulty in temporal credit assignment is that actions and their corresponding rewards are not synchronized, which makes reinforcement learning more difficult. In essence, learning a policy is a static optimization problem where policies are solutions and the decisions taken during the process serve to aid in determining a good policy. This is fundamentally different from the dynamic optimization problems that we study in this paper where

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

the decisions themselves must be made optimally and they can be made only once. What these two problems have in common is that in both cases the use of predictions of consequences of actions taken now can help to obtain better results [23]. Whereas this is already known for the temporal credit assignment problem in reinforcement learning, in this paper we show that this is also the case for time–linkage in dynamic optimization problems.

It is important to note that such dependencies between decisions over time requires their explicit processing in an algorithm to ensure the best performance in the long run. Any approach that does not explicitly anticipate these dependencies and instead only solves the problem for the current time will never obtain an optimal result.

In this paper, we present an algorithmic framework for solving dynamic optimization problems that is specifically equipped with the possibility of processing time–linkage. To this end, we propose the incorporation of learning (e.g. statistical [30] or machine [20]) with the explicit task of predicting the future to prevent being deceived over time. An evolutionary approach in which the future is predicted for dynamic optimization has been proposed before [28]. However, the cited approach only predicts the future for a single discrete timestep. As a result, the algorithm cannot process longer, arbitrary sized, time–linkage intervals. Moreover, the approach was only tested on a problem that doesn't contain the time–linkage aspect. As a result, no significant difference was observed in using either a good predictor or a bad predictor. In this paper, we present two new benchmark problems that contain time–linkage and show, as a proof of principle, how they can be solved using an instance of our proposed framework.

It should be noted that it is not our goal in this paper to propose a new state–of–the–art EA for dynamic optimization. Instead, we want to point out the influence that time–linkage can have and how, in a general manner, EAs can be equipped with tools to cope with time–linkage.

The remainder of this paper is organized as follows. In Section 2 we characterize online dynamic optimization problems. In Section 3 we discuss solving online dynamic optimization problems by only taking into account the current situation. Section 4 describes the advantages of solving online dynamic optimization problems by also taking into account future implications of decisions. In Section 5 we describe our algorithmic framework and in Section 6 we present results of running experiments with EAs based on this framework. Finally, possible directions for future research as well as conclusions are presented in Section 7.

2. ONLINE DYNAMIC OPTIMIZATION

2.1 Dynamic optimization

In general, optimization problems can be defined as:

$$\max_{\boldsymbol{\zeta}\in\mathbb{P}} \{\mathfrak{F}_{\boldsymbol{\gamma}}(\boldsymbol{\zeta})\} \textbf{ subject to } \mathfrak{C}_{\boldsymbol{\gamma}}(\boldsymbol{\zeta}) = \textit{feasible} \qquad (1)$$

where $\mathfrak{F}_{\boldsymbol{\gamma}} : \mathbb{P} \rightarrow \mathbb{O}$ is the optimization function, $\mathbb{P}$ is the parameter space, $\mathbb{O} = \mathbb{R}^{n_o}$ is the n_o–dimensional objective space, $\mathfrak{C}_{\boldsymbol{\gamma}} : \mathbb{P} \rightarrow \{\textit{feasible}, \textit{infeasible}\}$ is the constraint function and $\boldsymbol{\gamma} \in \mathbb{G}$ are problem–specific parameters.

In dynamic optimization the optimization function and the constraint function are functionals where the function space to optimize over consists of functions $\boldsymbol{\zeta}(t)$ of the time variable $t \in \mathbb{T} = [0, t^{\text{end}}]$, $t^{\text{end}} > 0$:

$$\mathfrak{F}_{\boldsymbol{\gamma}}(\boldsymbol{\zeta}(t)) = \int_0^{t^{\text{end}}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t)}(\boldsymbol{\zeta}(t))\, dt \qquad (2)$$

$$\mathfrak{C}_{\boldsymbol{\gamma}}(\boldsymbol{\zeta}(t)) = \begin{cases} \textit{feasible} & \text{if } \forall t \in [0, t_{\text{end}}]\text{: } \mathfrak{C}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t)}(\boldsymbol{\zeta}(t)) = \textit{feasible} \\ \textit{infeasible} & \text{otherwise} \end{cases}$$

where the convention that $\int_a^b (f_0(x), \ldots, f_{n_o-1}(x))\, dx = \left(\int_a^b f_0(x)dx, \ldots, \int_a^b f_{n_o-1}(x)dx\right)$ is used. Note that the dynamic variants of the optimization– and constraint function have parameters $\boldsymbol{\gamma}^{\text{dyn}}(t)$ that may change over time. Note that they may be dependent on earlier decisions by using the time variable indirectly, i.e. through a function of $\boldsymbol{\zeta}(t)$.

In the above time is assumed to be continuous. However, the parameters do not have to change continuously over time. In that case, the integral can be written as a discrete sum and the dynamic problem is said to be discrete.

2.2 The online case

Equation 2 can still be solved in an offline manner by searching for the best function $\boldsymbol{\zeta}(t)$ either in parametric or time–discretized form. It is the online variant or treatment of dynamic optimization that is the most interesting and the most practical, but unfortunately also the most difficult.

In the online case the dynamic optimization problem must be solved as time goes by. In other words, solutions cannot be evaluated for any future time $t > t^{\text{now}}$. Hence, decisions on what values to use for the variables at the current time t^{now} have to be made continuously. The only thing that can be evaluated is how well the algorithm has done so far, i.e. the goodness of the history and the present:

$$\mathfrak{H}^{\text{dyn}}(t^{\text{now}}, \boldsymbol{\zeta}(t)) = \int_0^{t^{\text{now}}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t)}(\boldsymbol{\zeta}(t))\, dt \qquad (3)$$

2.3 System influence and control influence

We distinguish between two types of influence that cause the dynamic optimization problem to change with time:

1. *System influence.* This is the type of influence that the problem solver has no control over. It is the part of the dynamic system that changes over time, regardless of choices made for the problem variables. It is the inherent reason why the optimization problem is dynamic and hence why the optimization function parameters $\boldsymbol{\gamma}^{\text{dyn}}(t)$ are a function of time.
2. *Control influence.* This type of influence is the response of the dynamic system at time t^{now} to the choices of the problem variables made in the past, i.e. the trajectory $\boldsymbol{\zeta}(t)$ with $t \in [0, t^{\text{now}})$.

Most EAs designed for dynamic optimization problems studied in the literature are specifically designed to cope with system influences. Without taking into account the (possible) presence of control influence however, the online dynamic optimizer risks falling victim to time–deception.

3. OPTIMIZING PRESENT: VICTIM TO TIME–DECEPTION

3.1 The approach

An often–used approach to solving online dynamic optimization problems is to optimize $\mathfrak{H}^{\text{dyn}}(t^{\text{now}}, \boldsymbol{\zeta})$ continuously or whenever an event resulting from system influence takes

place. Since we cannot change the past, we can only vary the settings of the variables at t^{now}. Hence, the optimization problem to solve at time t^{now} using this approach is actually static: optimize the value of the dynamic optimization function at time t^{now}. To cope with a variety of system influences when using EAs, diversity preserving mechanisms are often used to prevent complete convergence as are other techniques such as detecting (major) changes in the landscape to trigger a restart or forking off multiple sub–populations from a general optimizer to search various parts of the search space more closely as they become more interesting over time.

3.2 How bad can it be?

Unfortunately, the answer is *arbitrarily bad*. The most important reason for this is the presence of control influence. Of course system influence could make the problem change in a random way, clearly already making the problem arbitrarily difficult. However, even if system influence is smooth and the landscape is not–complex, optimizing only the current situation can lead to arbitrarily bad results. Consider for instance the following unconstrained l–dimensional dynamic optimization problem; a simple adaptation of the sphere problem that shifts with time:

$$\max_{\boldsymbol{\zeta}(t)} \left\{ \int_0^{t^{\text{end}}} \varphi(\boldsymbol{\zeta}(t), t) dt \right\} \tag{4}$$

where

$$\varphi(\boldsymbol{\zeta}(t), t) = \begin{cases} -\sum_{i=0}^{l-1} \left(\boldsymbol{\zeta}(t)_i - t\right)^2 & \text{if } 0 \leq t < 1 \\ -\sum_{i=0}^{l-1} \left(\boldsymbol{\zeta}(t)_i - t\right)^2 + \psi\left(\left|\boldsymbol{\zeta}(t-1)_i\right|\right) & \text{otherwise} \end{cases}$$

Now, when optimizing only the present in an online setting, a value for $\boldsymbol{\zeta}(t^{\text{now}})$ is chosen by maximizing $\varphi(\boldsymbol{\zeta}, t^{\text{now}})$. But for any t, $\varphi(\boldsymbol{\zeta}, t)$ is just a hyperparabola with a unique maximum for $\boldsymbol{\zeta}(t)_i = t$. For $0 \leq t < 1$ the associated value is 0 and for $t \geq 1$ this value is $-\sum_{i=0}^{l-1} \psi\left(\left|\boldsymbol{\zeta}(t-1)_i\right|\right)$. It is this construction that deceives an approach in which only the present is optimized because then the actual value of function $\psi(\cdot)$ is not taken into account although it may decrease at an arbitrary rate, depending on its form.

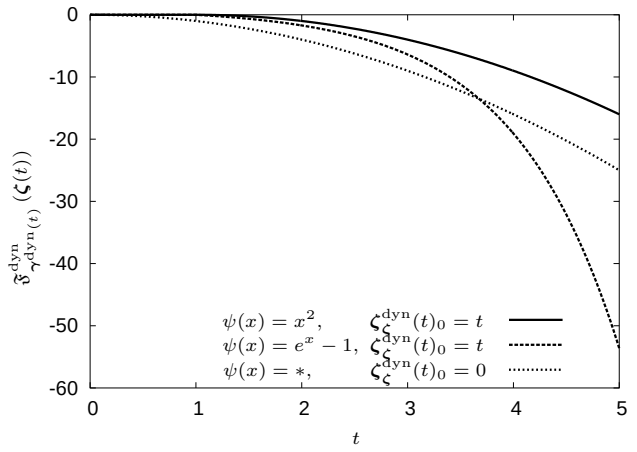


Figure 1: Illustration of the optimization values obtained for different variable trajectories and different forms of $\psi(\cdot)$ in the case of $l = 1$ and $t^{\text{end}} = 5$.

If however $\boldsymbol{\zeta}(t)_i = 0$ is simply always chosen, then, assuming that $\psi(0) = 0$, the optimization value that is reached is $l \int_0^{t^{\text{end}}} -t^2 dt = -\frac{l}{3}\left(t^{\text{end}}\right)^3$, regardless of function $\psi(\cdot)$.

Now if for instance $\psi(x) = x^2$, the result is better if only the present is optimized than if just $\boldsymbol{\zeta}(t)_i = 0$ is chosen. Although a better result can still be obtained because $\boldsymbol{\zeta}(t)_i = 0$ is not the optimal solution, the penalty of time–deception is only small. But if $\psi(\cdot)$ is a higher–order increasing function, such as for instance $\psi(x) = e^x - 1$, a (much) worse optimization value will be obtained. A graphical illustration of the difference in obtained optimization values for different variable trajectories for $l = 1$ is given in Figure 1.

Since in the online case the behavior of the optimization function in the future is not known, optimizing only the present can thus significantly reduce overall solution quality. Hence, optimizing only the present is not a good approach unless it the problem is provably not time–deceptive.

4. OPTIMIZING PRESENT AND FUTURE: LEARN TO AVOID TIME–DECEPTION

4.1 The approach

The approach of optimizing only the present is deceived over time because the true problem definition (i.e. equation 2) is not used. Future changes that occur as a result of decisions made earlier are neglected. To remedy this, optimization over future choices is required. In the online case however, an evaluable future is absent. Hence the only option is to *predict* the future. The available information to base that prediction upon besides problem–specific information is information collected in the past. The better the prediction, the closer the algorithm can get to optimality.

Summarizing, the optimization problem to be solved using this approach at any time t^{now} is based on an approximation of the value of the dynamic optimization function over a future timespan of length t^{plen}:

$$\max_{\boldsymbol{\zeta}(t)} \left\{ \int_{t^{\text{now}}}^{\min\left\{t^{\text{now}}+t^{\text{plen}}, t^{\text{end}}\right\}} \hat{\mathfrak{F}}^{\text{dyn}}_{\boldsymbol{\alpha}}\left(t, \boldsymbol{\zeta}(t)\right) dt \right\} \tag{5}$$

$$\textbf{s.t. } \forall t \in \left[t^{\text{now}}, \min\left\{t^{\text{now}}+t^{\text{plen}}, t^{\text{end}}\right\}\right] : \hat{\mathfrak{C}}^{\text{dyn}}_{\boldsymbol{\alpha}}\left(t, \boldsymbol{\zeta}(t)\right) = \textit{feasible}$$

where $\hat{\mathfrak{F}}^{\text{dyn}}_{\boldsymbol{\alpha}}\left(t^{\text{now}}, \boldsymbol{\zeta}(t^{\text{now}})\right) = \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t^{\text{now}})}\left(\boldsymbol{\zeta}(t^{\text{now}})\right)$

4.1.1 Prediction in the complete BBO case

The complete BBO (Black–Box Optimization) case is the most general case. No prior knowledge on the problem to be solved is assumed other than the number of variables and their types. Additional knowledge can only be gained by evaluating solutions. Since nothing is known about the optimization function, only a a very general form of induction can be performed to predict future function values.

We assume that the number of variables and their semantics do not change. To predict the (expected) value of the dynamic optimization function, an approximation based on previously evaluated solutions can be used. Computing this approximation is a (statistical) learning problem. The available data in the learning problem is:

$$\bigcup_{i=0}^{n_{\text{data}}-1} \left\{ \left(\left(t^i, \boldsymbol{\zeta}^i\right), \boldsymbol{y}^i \right) \right\} \tag{6}$$

where t^i, $0 < t^i \leq t^{\text{now}}$ is the time–component of the i–th pattern in the data set, $\boldsymbol{\zeta}^i$ is the variable–value–component and $\boldsymbol{y}^i$ contains the value of the dynamic optimization function for $\boldsymbol{\zeta}^i$ at time t^i. Note that the use of an EA can greatly add to the availability of data and can hence increase the accuracy of the predictions because a (diverse) population is used. Each population member can serve as a pattern.

The actually chosen trajectory $\boldsymbol{\zeta}(t), t \in [0, t^{\text{now}})$ is of course also available to the predictor. Parts of this history–trajectory can be integrated into the dataset to be able to process time–linkage, i.e. dependencies of the dynamic optimization function on values actually chosen for the problem variables in the past.

The goal of learning is to estimate the value of the dynamic optimization function for future times (assuming that the constraint function does not need to be estimated) by minimizing the generalization error over the timespan that contains the data to learn from. In the single–objective case:

$$\min_{\boldsymbol{\alpha} \in \mathbb{A}} \left\{ \int_{t^{\min}}^{t^{\max}} \left(\mathfrak{F}^{\text{dyn}}_{\gamma^{\text{dyn}}(t))} \left(\boldsymbol{\zeta}(t) \right) - \hat{\mathfrak{F}}^{\text{dyn}}_{\boldsymbol{\alpha}} \left(\boldsymbol{\zeta}(t) \right) \right)^2 dt \right\} \quad (7)$$

where

$$\begin{cases} t^{\min} &= \min_{i \in \{0,1,\ldots,n_{\text{train}}-1\}} \{t^i\} \\ t^{\max} &= \max_{i \in \{0,1,\ldots,n_{\text{train}}-1\}} \{t^i\} \end{cases}$$

and $\boldsymbol{\alpha} \in \mathbb{A}$ are the parameters of the function class from which to choose the approximation.

4.1.2 Prediction in the partial BBO case

In the presence of problem–specific information, the learning task may be less involved which may improve the reliability of the predictions. A typical case is when the function can be evaluated for any $0 \leq t < t^{\text{end}}$, as long as the required parameters are set. Then, if we are able to predict the values for the parameters accurately, we automatically get an accurate function evaluation. The less parameters to estimate, the better the hope of obtaining good approximations.

4.1.3 Prediction length and prediction base

Two key issues are how far into the future predictions should be made (prediction length, denoted t^{plen}) and information from how far in the past should be used to base the prediction upon (prediction base, denoted t^{pbase} indicating generally collected data and history length, denoted t^{hlen} indicating the history of the actually chosen trajectory). A proper choice for the prediction length and the history length depends on the time–linkage timespan, i.e. how far into the future do current choices have a significant influence? Certainly this is also the minimal choice for the prediction base. However, larger values for prediction length, prediction base and history length may be required to look beyond the deception and observe the general dynamics of the optimization problem.

Another issue that influences the proper choice for the prediction length is the reliability of predictions. As predictions are made further into the future, they are bound to become less reliable, giving a trade–off between the required prediction length as a result of time–linkage and the feasible prediction length as a result of reliability issues.

4.2 How good can it be?

Fortunately, the answer is *arbitrarily good*. However, although it is intuitively clear that the optimum is attainable, this does require *perfect* predictions. Then, the problem can be solved to optimality by optimizing at any time the integral over the predictions with $t^{\text{plen}} = t^{\text{end}} - t^{\text{now}}$.

The strength of the optimization method (with respect to the problem at hand) is still a key component to success. However, the success of the approach now also heavily depends on the strength of the prediction method. Bad predictions may even lead to worse results than are obtained by optimizing only the present. Hence, careful design and performance assessment of methods that predict the future are certainly called for. In the following section we present a general framework for solving dynamic optimization problems by incorporating learning techniques as described above.

5. ALGORITHMIC FRAMEWORK

5.1 Components

5.1.1 Solver

The solver, denoted S, is an optimization algorithm, possibly equipped with tools to allow for adaptability as time changes. The function to be optimized is provided by the function component discussed in Section 5.1.3.

5.1.2 Predictor

The predictor, denoted P, is a learning algorithm that approximates either the optimization function directly or several of its parameters. The data set from which to estimate a function is provided by the database component discussed in Section 5.1.4. When called upon, the predictor returns either the predicted function value directly or predicted values for parameters.

5.1.3 Function

The function, denoted F, is the optimization function to be maximized by the solver. If the future is not to be taken into account, this function is just the dynamic optimization function. The trajectory of the variables in the predicted future represents the variables to be optimized by the solver. To be able to compute the optimization value of such a future trajectory, a solution for each possible time between t^{now} and $\min\{t^{\text{now}} + t^{\text{plen}}, t^{\text{end}}\}$ is needed. It is convenient to divide the trajectory–future interval as well as the trajectory–history interval into non–zero sub–intervals of length t^{pint} and t^{hint} respectively. The optimization value then is a discrete approximation of the integral over the future interval where future predictions are in addition to other data based on a discretized past trajectory. The number of variables that the solver needs to optimize over is the union of all sets of variables that pertain to the dynamic optimization function at the beginning of the sub–intervals. The dynamic optimization function is used to compute the optimization value that pertains to the current time and the predictor is used to predict the future.

5.1.4 Database

The database, denoted D, is a collection of patterns upon which the predictor bases its predictions. Patterns are added either by the function component (i.e. in the complete BBO case whenever a new solution is evaluated) or by the system

whenever an event occurs that is related to the parameters of interest (i.e. in the partial BBO case). All patterns are time–stamped. The database only contains patterns with a timestamp t for which $t^{\text{now}} - t^{\text{pbase}} \leq t \leq t^{\text{now}}$ holds.

5.1.5 Timer

The timer, denoted T, can provide the current time t^{now}.

5.2 Dividing resources

Clearly, optimization becomes more involved if we also want to take into account predictions of the future. Not only does the number of variables to optimize over increase (at least if we regard the complete BBO case), but also additional time is required for learning to make predictions.

It is important to note that there is a trade-off between how much time should be spent on running the solver and how much time should be spent on running the predictor. To allow for a scheme that implements this trade–off we propose to implement the solver and the predictor components as threads. This allows both for a scheme in which the solution component and the predictor component run simultaneously as well as a scheme where the predictor and solver are run sequentially by synchronization using signals. An example of the second scheme is when the solver sends a signal when a certain number of generations have passed and subsequently awaits completion of the learning task before continuing.

5.3 Definition

To complete the framework, in this section we provide an algorithmic description of how the components are used together to solve online dynamic optimization problems. First, the trajectory is made empty and all the components are initialized. Then, the solver and the predictor are started and the actual optimization begins. Although the solver may store a solution into the trajectory at any time (e.g. at the end of each generation for an EA), we want to ensure that at least a few solutions are stored in the trajectory. To this end, the solver is requested for a solution at regular intervals of length t^{sint}. These requests are issued until t^{end} is reached. Then, the solver and the predictor are halted and the resulting trajectory is returned:

FRAMEWORK($S, P, F, D, T, t^{\text{end}}, t^{\text{pbase}}, t^{\text{plen}}, t^{\text{sint}}, t^{\text{pint}}$)
1 $\boldsymbol{Z} \leftarrow ()$
2 S.INITIALIZE($S, P, F, \ldots, t^{\text{pint}}$)
3 P.INITIALIZE($S, P, F, \ldots, t^{\text{pint}}$)
4 F.INITIALIZE($S, P, F, \ldots, t^{\text{pint}}$)
5 D.INITIALIZE($S, P, F, \ldots, t^{\text{pint}}$)
6 T.INITIALIZE($S, P, F, \ldots, t^{\text{pint}}$)
7 S.START()
8 P.START()
9 **do**
  9.1 $t^{\text{now}} \leftarrow T$.GETTIME()
  9.2 $\boldsymbol{\zeta} \leftarrow S$.REQUESTSOLUTION()
  9.3 $\boldsymbol{Z} \leftarrow \boldsymbol{Z} \sqcup ((\boldsymbol{\zeta}, t^{\text{now}}))$
  9.4 $t^{\text{next}} \leftarrow \min\{t^{\text{next}} + t^{\text{sint}}, t^{\text{end}}\}$
  9.5 AWAITTIME(t^{next})
  while $t^{\text{now}} \leq t^{\text{end}}$
10 S.STOP()
11 P.STOP()
12 **return**($\boldsymbol{Z}$)

The final part of the framework that is of a specific form is the way in which a solution in the form of a future trajectory is evaluated. It is here that the prediction component can influence the way in which the solver searches for a solution at t^{now} because the predictor is used to evaluate all parts of the trajectory that pertain to future times:

F.EVALUATE($(\boldsymbol{\zeta}^0, \boldsymbol{\zeta}^1, \ldots, \boldsymbol{\zeta}^{\lceil t^{\text{plen}}/t^{\text{pint}} \rceil - 1})$)
1 $t^{\text{now}} \leftarrow T$.GETTIME()
2 $\boldsymbol{y} \leftarrow t^{\text{pint}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t^{\text{now}}, Z(t^{\text{now}}, \boldsymbol{\zeta}))}(\boldsymbol{\zeta}^0)$
3 **if** COMPLETEBBOCASE() **then**
  3.1 D.ADDPATTERN($((\boldsymbol{\zeta}^0, t^{\text{now}}), \boldsymbol{y})$)
  3.2 **for** $i \leftarrow 1$ **to** $\lceil t^{\text{plen}}/t^{\text{pint}} \rceil - 1$ **do**
    3.2.1 $\boldsymbol{y} \leftarrow \boldsymbol{y} + t^{\text{pint}} P$.PREDICT($\boldsymbol{\zeta}^i, t^{\text{now}} + i \cdot t^{\text{pint}}$)
4 **else**
  4.1 **for** $i \leftarrow 1$ **to** $\lceil t^{\text{plen}}/t^{\text{pint}} \rceil - 1$ **do**
    4.1.1 $\boldsymbol{\gamma}^{\text{predicted}} \leftarrow P$.PREDICT($\boldsymbol{\zeta}^i, t^{\text{now}} + i \cdot t^{\text{pint}}$)
    4.1.2 $\boldsymbol{y} \leftarrow \boldsymbol{y} + t^{\text{pint}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{predicted}}}(\boldsymbol{\zeta}^i)$
5 **return**($\boldsymbol{y}$)

6. EXPERIMENTS

6.1 EA

The optimization problems that we use are real–valued, but at any point in time not very daunting as the most important thing we focus on is time–linkage. Therefore, we opt for a simple and fast real–valued EA. We use an EDA (Estimation–of–Distribution Algorithm) for real–valued optimization [5, 18] without learning dependencies between problem variables. The main difference with traditional EAs is that in EDAs a probabilistic model is learned using the selected solutions. The probabilistic model can capture various properties of the optimization problem. By drawing new solutions from the probabilistic model these properties can be exploited to obtain more efficient optimization.

In this paper we performed experiments with a real–valued EDA based on the normal distribution in which each variable is taken to be independent of all the other variables. Such an EDA is also known as the naive IDEA (Iterated Density–Estimation Evolutionary Algorithm) [6]. In the naive variant the mean and standard deviation of a one–dimensional normal distribution are estimated from the selected solutions for each variable separately. A new solution is constructed by sampling one value per variable from the associated one–dimensional normal distribution. Since the optimization problem is dynamic, we prevented total premature convergence by bounding the estimated variance for each variable to a minimum of 0.1. Finally, all results were averaged over 100 independent runs.

6.2 BBO: Time–deceptive numerical problem

6.2.1 The problem

We first investigate the real–valued time–deceptive problem introduced in Section 3, Equation 4. We regard two variants by setting $\psi(x) = x^2$ and $\psi(x) = e^x - 1$. Moreover, we have used a dimensionality of $l = 1$.

6.2.2 Instantiating the framework

We used three different predictor instances. In this problem the goal of the predictor is to predict the value of the optimization function directly. The first instance is optimal, i.e. it is the true value of the dynamic optimization function. The second instance estimates a linear function and the third instance estimates a quadratic function. The latter two estimates are computed using a least–squares approximation. For clarity: the linear function for example is estimated from a set of patterns with timestamps at most t^{pbase} time ago. Each pattern $((t^i, \boldsymbol{\zeta}^i), \boldsymbol{y}^i)$ is complemented with the actually chosen trajectory in the past up to time $t^i - t^{\text{hlen}}$ in steps of t^{hint}, i.e. if $t^{\text{hlen}} = t^{\text{hint}} = 1$, the pattern is transformed to $((t^i, \boldsymbol{\zeta}^i, \boldsymbol{\zeta}(t^i - 1)), \boldsymbol{y}^i)$. The linear estimator now is constructed from the transformed dataset.

Only the function class used by the quadratic estimator contains the target function for the case of $\psi(x) = x^2$. Hence, effective future predictions are possible with proper estimations in this case, preventing time–deception. For the case of $\psi(x) = e^x - 1$, neither of the estimators can represent the target function. However, the quadratic estimator should be capable of far better approximations.

Since we present a proof–of–principle, we do not investigate the selection of t^{plen} during optimization. Instead, we fix it to either 0 that corresponds to the traditional approach of not looking into the future or to the optimal value of 1. Moreover, we set $t^{\text{hlen}} = t^{\text{hint}} = t^{\text{pbase}} = t^{\text{pint}} = t^{\text{plen}}$.

6.2.3 Results

A population size of 25 was experimentally found to be adequate for solving the optimization problem in each time step. We set $t^{\text{end}} = 10$ and advanced time by a timestep of 0.001 every 25 evaluations (i.e.every generation). Since the database contains all patterns over a timespan of length 1 and the timesteps are of size 0.001, the size of the database can become quite large. Although this allows for a higher precision of estimations, it also results in large time requirements for the learning task. Learning was performed after a predefined number of generations had passed. To investigate the impact on the overall quality of optimization, we performed experiments with various values for the number of generations between learning phases: $1, 10, 100$ and 1000.

The average trajectories obtained for the quadratic and the exponential time–deceptive numerical problem are shown in Figures 2 and 3 respectively. The overall result (i.e. the integrated function over $t \in [0, 10]$) is tabulated in Table 1.

Theoretically, under the assumption that the length of the time–linkage is known, the optimal trajectory can be obtained if the target function is in the function class used by the learner and the learner is competent in that it will indeed find that target when learning. In the case of the quadratic time–deceptive numerical problem this is experimentally verified by the results. The use of the quadratic estimator leads to results that are very close to optimality (i.e. when the future is known). The discrepancy is explained by the startup time the learner needs before being able to construct a model based upon previously encountered data. Moreover, results improve if learning is performed more frequently because the model is then constructed earlier.

In the case of the exponential time–deceptive numerical problem, neither the linear nor the quadratic estimator provide a function class that contains the target exponential function. However, for the time–linkage in this problem that depends only on a single point in the past over a distance of 1, a quadratic function can quite closely approximate an exponential function. For this reason the use of the quadratic estimator leads to good results here as well, albeit not optimal. Small deviations from the optimal trajectory as a result of a small learner error can indeed be seen in Figure 3. The linear estimator is not capable of approximating a quadratic function well. For the exponential function, linear estimation is even worse. The use of the linear estimator therefore leads to far worse results. An even more important point to note is that the results using the linear estimator can be even worse than when prediction is not used because of the large errors in the predictions. Hence, another important issue in using learning for online dynamic optimization is the assessment of the reliability of predictions and the use of predictions only if this reliability is large enough.

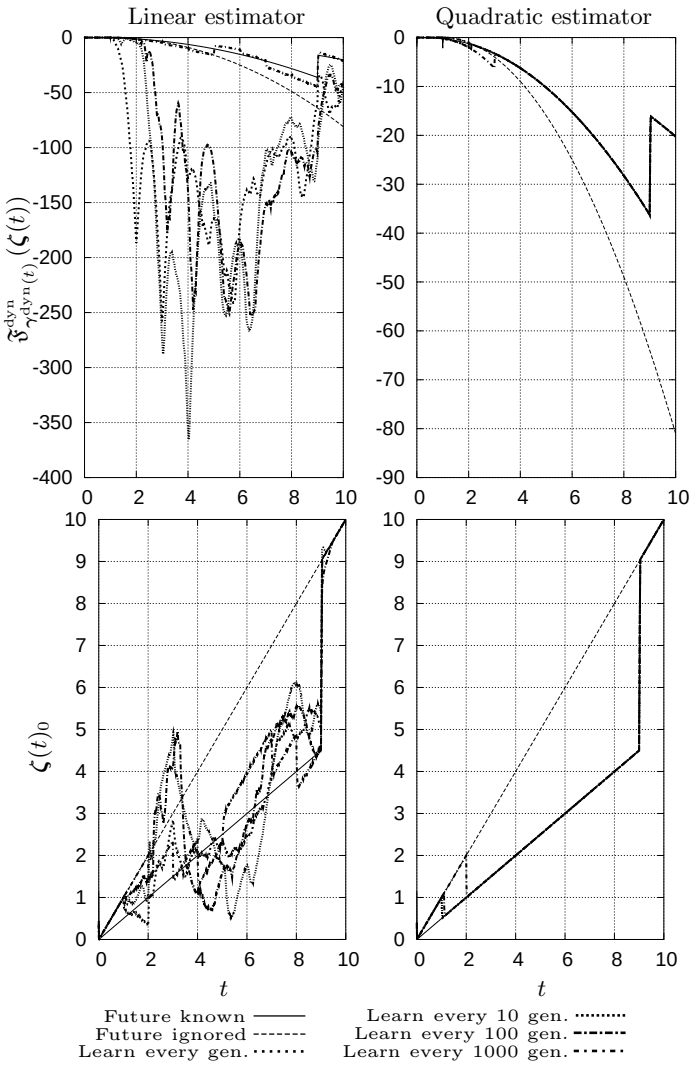


Figure 2: Results averaged over 100 runs on the time–deceptive numerical problem with $\psi(x) = x^2$.

The results lead to the expected conclusion that competent learners are called for and that reliability of predictions is a major issue. The competence of the learner in the BBO case depends on general/overall competence which is very hard to obtain. In the problem–specific case however, achieving learner competence may be easier because the shape of the model to be learned (i.e. parametric learning) is known from domain knowledge, ensuring that the target function is in the function class used by the learner.

6.3 Partial BBO: dynamic pickup problem

6.3.1 The problem

The second problem that we investigate is a discrete partial BBO problem. Although it is based on a very simple model, the time–linkage in the problem is large: any decision made now influences the result of the dynamic optimization function for all future timesteps. The intuitive description is that at timestep t a truck is located at $\boldsymbol{x}^{\text{truck}}(t)$ and a package appears at location $\boldsymbol{x}^{\text{package}}(t)$. It must now be decided whether to send the truck to go and pick up the package or to drive elsewhere. If the package is not picked up, it disappears. Picking up the package pays a value of 1, but

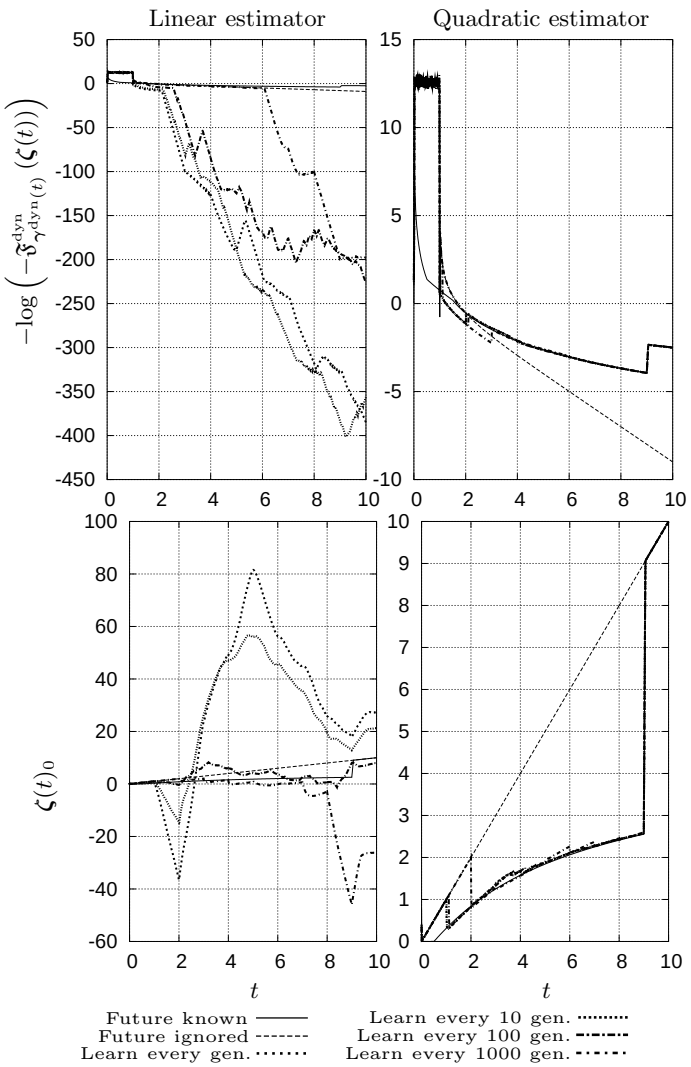


Figure 3: Results averaged over 100 runs on the time–deceptive numerical problem with $\psi(x) = e^x - 1$.

driving costs a value equal to the Euclidean distance traveled. The number of packages is $n_{\text{packages}} = t^{\text{end}} + 1$, i.e. the timesteps are of size 1. A solution at time t now is a tuple $\boldsymbol{\zeta}(t) = (b(t), \boldsymbol{x}^{\text{alternative}}(t))$ where $b \in \{0, 1\}$ indicates whether the package at time t should be picked up ($b(t) = 1$) and $\boldsymbol{x}^{\text{alternative}}(t)$ is the location to drive to if the package is not to be picked up ($b(t) = 0$). Mathematically:

$$\mathfrak{F}^{\mathrm{dyn}}_{\boldsymbol{\gamma}^{\mathrm{dyn}}(t)}\left((b(t), \boldsymbol{x}^{\text{alternative}}(t))\right) = \tag{8}$$

$$\begin{cases} 1 - \parallel \boldsymbol{x}^{\text{package}}(t) - \boldsymbol{x}^{\text{truck}}(t) \parallel & \text{if } b(t) = 1 \\ 0 - \parallel \boldsymbol{x}^{\text{alternative}}(t) - \boldsymbol{x}^{\text{truck}}(t) \parallel & \text{otherwise} \end{cases}$$

where

$$\boldsymbol{x}^{\text{truck}}(t) = \begin{cases} \sim \prod_{i=0}^{l-1} \mathcal{N}(0,1) & \text{if } t = 0 \\ \boldsymbol{x}^{\text{package}}(t-1) & \text{if } t = 1 \text{ and } b(t-1) = 1 \\ \boldsymbol{x}^{\text{alternative}}(t-1) & \text{otherwise} \end{cases}$$

For simplicity, the model used to generate new package locations is a univariately factorized normal distribution with zero mean and unit variance, i.e. $\boldsymbol{x}^{\text{package}}(t) \sim \prod_{i=0}^{l-1} \mathcal{N}(0,1)$.

		$\psi(x) = x^2$	$\psi(x) = e^x - 1$
	Future known	$-1.21846 \cdot 10^2$	$-1.55430 \cdot 10^2$
	Future ignored	$-2.42940 \cdot 10^2$	$-8.08692 \cdot 10^3$
Linear	**Learn every gen.**	$-1.09434 \cdot 10^3$	$-2.04922 \cdot 10^{65}$
Linear	**Learn every 10 gen.**	$-1.18946 \cdot 10^3$	$-7.93853 \cdot 10^{172}$
Linear	**Learn every 100 gen.**	$-9.98665 \cdot 10^2$	$-3.02553 \cdot 10^{96}$
Linear	**Learn every 1000 gen.**	$-1.38907 \cdot 10^2$	$-1.22634 \cdot 10^{86}$
Quadratic	**Learn every gen.**	$\mathbf{-1.22010 \cdot 10^2}$	$\mathbf{-1.55966 \cdot 10^2}$
Quadratic	**Learn every 10 gen.**	$-1.22013 \cdot 10^2$	$-1.55969 \cdot 10^2$
Quadratic	**Learn every 100 gen.**	$-1.22062 \cdot 10^2$	$-1.56069 \cdot 10^2$
Quadratic	**Learn every 1000 gen.**	$-1.23178 \cdot 10^2$	$-1.58092 \cdot 10^2$

Table 1: Overall results (i.e. $\int_0^{t^{\text{end}}} \mathfrak{F}^{\mathrm{dyn}}_{\boldsymbol{\gamma}^{\mathrm{dyn}}(t)}(\boldsymbol{\zeta}(t))\, dt$) on the time–deceptive problem.

6.3.2 *Instantiating the framework*

A very simple strategy is given by a hillclimber. The decision taken at each timestep is to move only to pick up a package and moreover only to do so if the distance to the package is less than 1. In other words, a negative score is never accepted.

We have compared the hillclimber with an EA instance of the general framework. Since the optimization function is completely known with the exception of $\boldsymbol{x}^{\text{package}}(t)$, the problem is only partially a BBO problem. Hence, we can restrict the prediction task to predicting future values for $\boldsymbol{x}^{\text{package}}(t)$. We have performed experiments where we assumed the distribution of $\boldsymbol{x}^{\text{package}}(t)$ to be known and where we estimated this distribution from data, assuming only that the data is indeed normally distributed.

In theory, the influence of any decision at time t influences the outcome of the dynamic optimization function at any time $t' > t$. However, the larger $t' - t$, the smaller the remaining impact on the situation at time t'. Although in theory it would be optimal to set t^{plen} to t^{end} with $t^{\text{pint}} = 1$, such a choice gives rise to two practical problems. First, a large t^{plen} gives to extremely large trajectories to optimize. This drastically increases the resources required by the EA to solve the problem. Second, since the future is inherently stochastic, a proper estimation of the expected future profits requires averaging evaluation over multiple calls. Moreover, the variability of these outcomes increases as t^{plen} increases because more uncertainty is introduced. Hence, unless an infinite number of calls is used, a smaller value for t^{tplen} is expected to be optimal in practice.

To prevent large trajectories to be subject to evolution in the EA, we choose a simplified approach by choosing a very special form for the predictor. The predictor predicts the result of the dynamic stochastic optimization function up to a timespan of t^{plen} into the future by using the hillclimber instead of explicit solutions for each timestep provided by the EA. The EA only provides a solution for the current time. Although better results may be obtained by allowing the EA also to evolve future solutions, using the hillclimber to predict the future can already give good solutions. The reason is that using the hillclimber can already give a good impression of the quality of a certain starting point. Since the dynamic optimization function is stochastic, it should be noted that multiple calls are required to estimate the expected future payoff even when evaluating the future using the hillclimber. To reduce the number of statistical errors, the best evolved decision is compared to the default choice of doing nothing, i.e. $b(t^{\text{now}}) = 0$ and $\boldsymbol{x}^{\text{alternative}}(t^{\text{now}}) =$

$\boldsymbol{x}^{\text{truck}}(t^{\text{now}})$. Only if the mean fitness of the best evolved decision averaged over 100 calls to the dynamic optimization function is statistically significantly larger than the mean fitness of the default decision, the evolved decision is used. The statistical hypothesis test used to this end is the Aspin–Welch–Satterthwaite (AWS) T–tests at a significance level of $\alpha = 0.05$. The AWS T–test is a statistical hypothesis test for the equality of means in which the equality of variances is not assumed [17].

6.3.3 Results

A population size of 100 was experimentally found to be adequate for solving the optimization problem in each time step. We set $t^{\text{end}} = 100$ and advanced time by a timestep of 1 every 5000 evaluations (i.e. every 50 generations). Since only one pattern was added to the dataset each timestep, and only a normal distribution is estimated from data, learning can be done very fast for this problem. Therefore learning was performed whenever time was advanced. The final results (i.e. the integral of the dynamic optimization function over $[0, 100]$) are shown in Figure 4 for different values of the prediction length t^{plen}. Indeed as expected and motivated earlier in the previous subsection, the best value for t^{plen} is not the maximum length of t^{end}, but a smaller length, even if the model is fully known.

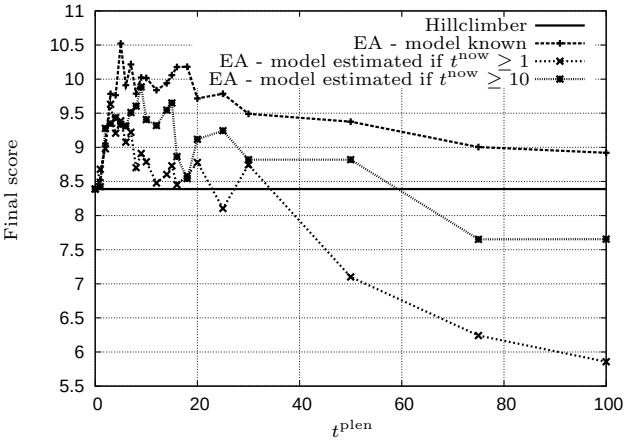


Figure 4: Final score (i.e. $\int_0^{t^{\text{end}}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t)}(\boldsymbol{\zeta}(t))\,dt$) averaged over 100 runs on the dynamic pickup problem as a function of the prediction length.

The trajectory of the cumulative fitness for the best value and maximum value of t^{plen} are shown in Figure 5. This figure also reveals why the use of information about the future results in a better result in the end. All algorithms other than the hillclimber are willing to accept negative scores in a single turn if the prospect on future gains is larger. This of course happens if the truck moves more towards the origin as the density of the normal distribution is the highest there. The better strategy adopted by the system is thus to initially move towards the region close to the origin and never move too far away from it even if a profitable pickup can be made in a single turn by doing so.

Finally, it is again interesting to note that postponing the use of learning until a higher reliability is obtained leads to better results, indicating again the importance of reliable predictions in the proposed approach.

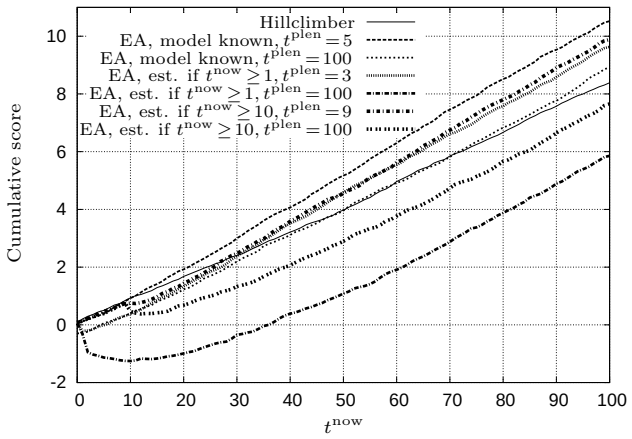


Figure 5: Cumulative score (i.e. $\mathfrak{H}^{\text{dyn}}(t^{\text{now}}, \boldsymbol{\zeta})$) averaged over 100 runs on the dynamic pickup problem.

7. DISCUSSION AND CONCLUSIONS

In this paper we have highlighted a specific source of difficulty in online dynamic optimization problems. We have labeled the difficulty time–linkage. In the worst case time–linkage can lead to time–deception. In that case any optimization algorithm is mislead and finds suboptimal results unless future implications of current decisions are taken into account. To tackle problems exhibiting this type of problem difficulty, we have proposed a framework that learns to predict the future and optimizes not only the current situation but also future predicted situations. We have proposed and used two new benchmark problems, but a larger suite of problems containing time–linkage is called for and should become a standard in dynamic optimization research.

In our experiments, we have fixed the future prediction timespan as well as the history data timespan. An interesting question is whether the timespans required to prevent deception can be measured during optimization. This calls for techniques for time–linkage identification in a similar sense as gene–linkage identification techniques are required in standard GAs to prevent deception as a result of dependencies between a problem's variables [12, 26].

Another important and related issue is how quickly the reliability of prediction degrades into the future. Even if we know how far into the future we must predict, it is hardly of any use to use these predictions if they are unreliable. The prediction reliability is influenced mostly by the difficulty of the function to predict (i.e. relatively steady or heavily fluctuating) and by the availability of data.

Ultimately, the expansion of dynamic EAs to process time–linkage information should be integrated with current state–of–the–art dynamic EAs that are capable of tackling other important problem difficulties that arise in dynamic optimization such as the overtaking of the optima by other local optima as time goes by. An EA that is capable of efficiently tackling both sources of problem difficulty is likely to be robust and well–suited to be used in practice and hence to be tested in real–world scenario's. To that end however, a further expansion that makes the approach well–suited for the multi–objective case is also likely to be crucial.

8. REFERENCES

[1] M. Andrews and A. Tuson. Diversity does not necessarily imply adaptability. In J. Branke, editor, *Proceedings of the Workshop on Evolutionary Algorithms for Dynamic Optimization Problems at the Genetic and Evolutionary Comp. Conference – GECCO 2003*, pages 24–28, 2003.

[2] P. J. Angeline. Tracking extrema in dynamic environments. In P. J. Angeline et al., editors, *Sixth Int. Conf. on Evol. Programming*, pages 335–345, Berlin, 1997. Springer Verlag.

[3] D. V. Arnold and H.-G. Beyer. Random dynamics optimum tracking with evolution strategies. In J.J. Merelo et al., editors, *Parallel Problem Solving from Nature – PPSN VII*, pages 3–12, Berlin, 2002. Springer Verlag.

[4] T. M. Blackwell. Particle swarms and population diversity II: Experiments. In J. Branke, editor, *Proceedings of the Workshop on Evolutionary Algorithms for Dynamic Optimization Problems at the Genetic and Evolutionary Comp. Conference – GECCO 2003*, pages 14–18, 2003.

[5] P. A. N. Bosman and D. Thierens. Advancing continuous ideas with mixture distributions and factorization selection metrics. In M. Pelikan and K. Sastry, editors, *Proc. of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Comp. Conference – GECCO 2001*, pages 208–212, 2001.

[6] P. A. N. Bosman and D. Thierens. The naive $\mathbb{MIDEA}$: a baseline multi–objective EA. In C. A. Coello Coello et al., editors, *Evolutionary Multi–Criterion Optimization – EMO'05*, pages 428–442, Berlin, 2005. Springer–Verlag.

[7] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the 99 Congress on Evolutionary Computation – CEC 99*, pages 1875–1882, Piscataway, New Jersey, 1999. IEEE Press.

[8] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer, Norwell, Massachusetts, 2001.

[9] J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck. A multi–population approach to dynamic optimization problems. In I. C. Parmee, editor, *Adaptive Computing in Design and Manufacture – ACDM 2000*, pages 299–308, Berlin, 2000. Springer Verlag.

[10] J. Branke and D. Mattfeld. Anticipation in dynamic optimization: The scheduling case. In M. Schoenauer et al., editors, *Parallel Prob. Solving from Nature – PPSN VI*, pages 253–262, Berlin, 2000. Springer Verlag.

[11] M. R. Caputo. *Foundations of Dynamic Economic Analysis*. Cambridge University Press, Cambridge, 2005.

[12] K. Deb and D. E. Goldberg. Sufficient conditions for deception in arbitrary binary functions. *Annals of Mathematics and Artificial Intelligence*, 10:385–408, 1994.

[13] S. M. Garrett and J. H. Walker. Genetic algorithms: Combining evolutionary and 'non'–evolutionary methods in tracking dynamic global optima. In W. B. Langdon et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2002*, pages 359–366. Morgan Kaufmann, 2002.

[14] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learing*. Addison Wesley, Reading, Massachusetts, 1989.

[15] J. Grefenstette. Evolvability in dynamic fitness landscapes: a genetic algorithm approach. In *Proceedings of the 99 Congress on Evolutionary Computation – CEC 99*, pages 2031–2038, Piscataway, New Jersey, 1999. IEEE Press.

[16] K. De Jong. Evolving in a changing world. In Z. W. Ras and A. Skowron, editors, *Foundations of Intelligent Systems*, pages 512–519, Berlin, 1999. Springer Verlag.

[17] M.G. Kendall and A. Stuart. *The Advanced Theory Of Statistics, Volume 2, Inference And Relationship*. Charles Griffin & Company Limited, 1967.

[18] P. Larrañaga, R. Etxeberria, J. A. Lozano, and J. M. Peña. Optimization in continuous domains by learning and simulation of Gaussian networks. In M. Pelikan et al., editors, *Proceedings of the Optimization by Building and Using Probabilistic Models OBUPM Workshop at the Genetic and Evolutionary Computation Conference – GECCO 2000*, pages 201–204, 2000.

[19] A. M. L. Liekens, H. M. M. ten Eikelder, and P. A. J. Hilbers. Finite population models of dynamic optimization with alternating fitness functions. In J. Branke, editor, *Proc. of the Workshop on Evolutionary Algorithms for Dynamic Optimization Problems at the Genetic and Evol. Comp. Conference – GECCO 2003*, pages 19–23, 2003.

[20] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, New York, 1997.

[21] W. B. Powell. Algorithms for the dynamic vehicle allocation problem. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 249–292. Elsevier Science, Amsterdam, 1988.

[22] H. N. Psaraftis. Dynamic vehicle routing problems. In B. L. Golden and A. A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 223–248. Elsevier Sc., Amsterdam, 1988.

[23] B. Ravindran and S. S. Keerthi. C3: Reinforcement learning. In E. Fiesler and R. Beale, editors, *Handbook Of Neural Computation*. Oxford Univ. Press, Oxford, 1996.

[24] L. Schöneman. On the influence of population sizes in evolution strategies in dynamic environments. In J. Branke, editor, *Proceedings of the Workshop on Evolutionary Algorithms for Dynamic Optimization Problems at the Genetic and Evolutionary Computation Conference – GECCO 2003*, pages 29–33, 2003.

[25] R. S. Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts, Amherst, Massachusetts, 1984.

[26] D. Thierens. Scalability problems of simple genetic algorithms. *Evolutionary computation*, 7:331–352, 1999.

[27] R. K. Ursem. Multinational gas: Multimodal optimization techniques in dynamic environments. In D. Whitley et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2000*, pages 19–26. Morgan Kaufmann, 2000.

[28] J. I. van Hemert, C. Van Hoyweghen, E. Lukschandl, and K Verbeeck. A "futurist" approach to dynamic environments. In J. Branke and T. Bäck, editors, *Proceedings of the Workshop on Evolutionary Algorithms for Dynamic Optimization Problems at the Genetic and Evolutionary Computation Conference – GECCO 2001*, pages 35–38, 2001.

[29] J. I. van Hemert and J. A. La Poutré. Dynamic routing problems with fruitful regions: models and evolutionary computation. In X. Yao et al., editors, *Parallel Problem Solving from Nature – PPSN VIII*, pages 692–701, Berlin, 2004. Springer Verlag.

[30] V. Vapnik. *Statistical learning theory*. Wiley, New York, New York, 1998.

[31] M. Wineberg and F. Oppacher. Enhancing the ga's ability to cope with dynamic environments. In D. Whitley et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2000*, pages 3–10. Morgan Kaufmann, 2000.

Learning Environment Dynamics From Self-Adaptation

A preliminary investigation

Amine Boumaza
LORIA - MaIA
Campus Scientifique B.P. 239
54506 Vandoeuvre-les-Nancy, France
amine.boumaza@loria.fr

ABSTRACT

We present an experimental study that shows a relationship between the dynamics of the environment and the adaptation of strategy parameters. Experiments conducted on two adaptive evolutionary strategies SA-ES and CMA-ES on the dynamic sphere function, show that the nature of the movements of the function's optimum are reflected in the evolution of the mutation steps. Three types of movements are presented: constant, linear and quadratic velocity, in all, the evolution of mutation steps during adaptation reflect distinctly the nature of the movements. Furthermore with CMA-ES, the direction of movement of the optimum can be extracted.

Categories and Subject Descriptors

I.2.6 [**Learning**]: Parameter learning; G.1.6 [**Optimization**]: Stochastic programming

General Terms

Algorithms, Design, Experimentation

Keywords

Evolutionary computation, Self-adaptation, Dynamic environments

1. INTRODUCTION

Self-adaptation has become a very important property in evolutionary computation (EC). The idea of self-tuning strategy parameters by the algorithm during the search has proved very powerful and very successful on a wide range of problems [6, 5, 3]. A substantial body of work regarding self-adaptation in evolutionary computation exists, for a comprehensive overview see [12].

Although adaptation can take place at any stage of the evolution [12], the best-known examples are self-adaptation of the mutation steps. Self-adaptive mutations were originally introduced in evolutionary strategies by H.P. Schwefel [14], however extensions to other areas of evolutionary computation as evolutionary programming [9] and genetic algorithms [4] exist. In this study, we address the case of evolutionary strategies trough the examination of two self-adaptive algorithms.

In dynamic optimization problems self-adaptation plays an important role mainly because it controls the exploration and exploitation phases. In such problems, the goal is not only to find the optimum, but also track it over time [7]. Therefore, the algorithm must adjust its exploration capacity when it looses the optimum once this one moves; self-adaptation is one way of doing it. There exist many studies on self-adaption in dynamical environments see for example [7, 8, 1], however to our knowledge, there are no results that link the self-adaptation process and the dynamics of the fitness landscape.

The aim of this article is to present experimental results on the relationship between the evolution of the mutation steps and the dynamics of the search problem. Section 2 presents briefly the algorithms used in this article, whereas section 3 presents the experiments conducted and comments the results. Finally section 4 presents the discussion and conclusions.

2. SELF-ADAPTIVE MUTATIONS

The idea underlying self-adaptive mutations, is to evolve the parameters of the mutation operator to adjust the behavior of the algorithm to the environment. In most cases, these parameters represent the parameters of the normal distribution used to sample offspring. Different methods of adaptation exist depending on the type of the normal distribution. The simplest case is when this distribution is chosen isotropic (hyper-sphere), here only one step size is adapted. This model can be extended to non-isotropic distributions (ellipsoid), where each coordinate posses its own step size. Finally a further generalization can be made, where the mutation normal distribution has a full covariance matrix. In this article we will only address the case of the last two types of adaptation through two implementation the $(\mu + \lambda)$SA-ES [14] and the $(\mu/\mu, \lambda)$CMA-ES [11].

2.1 SA-ES

In this algorithm, to each coordinate corresponds a mutation step, which is usually encoded within the genotype of the individual. Steps are adapted independently in each

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

dimension of the search space and mutation takes place in two steps. At first the parameter vector $\overrightarrow{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is mutated using a log-normal rule:

$$\tilde{\sigma}_i = \sigma_i \exp\left(\tau N(0,1) + \tau' N_i(0,1)\right) \quad \forall i \in \{0, \cdots, n\} \quad (1)$$

here, N_i are n independent realizations of a normal random variable, whereas N is a single realization common to all σ_i. τ and τ' are fixed parameters representing learning rates. In the second step, the individual $\overrightarrow{x} = (x_1, \ldots, x_n)$ is mutated by adding it to a normal vector whose components have $(\tilde{\sigma_1}, \ldots, \tilde{\sigma_n})$ as variances.

$$\tilde{x_i} = x_i + N_i\left(0, \tilde{\sigma_i}^2\right) \quad \forall i \in \{0, \cdots, n\} \quad (2)$$

2.2 CMA-ES

The CMA-ES algorithm introduced by N. Hansen [10, 11] is to this day one of the best performing algorithms based on covariance matrix adaptation. The main advantage of this algorithm compared to SA-ES is its invariance against rotations. The axes of the normal distribution are independent from the coordinate system which allows invariance against rotations of the search space.

The mathematical details of the algorithm were intentionally left out for the purpose of this article, the reader is encouraged to consult the referenced work for full details.

In CMA-ES, offspring at generation (g) are sampled from the following normal distribution:

$$X_k^{(g+1)} \sim \mathcal{N}\left(\langle x \rangle_\mu^{(g)}, {\sigma^{(g)}}^2, C^{(g)}\right) \quad (3)$$

in which $\langle x \rangle_\mu^{(g)}$ is a weighted average of the μ best individuals at generation (g), C is a positive definite matrix representing the covariance matrix, and σ is the mutation step size, note that in this case there is only one step size.

Adaptation in CMA-ES takes place in two steps: the adaptation of the covariance matrix C and the adaptation of the mutation step σ. These steps need not occur at the same time, in fact, they usually occur at different time scales.

The step size is adapted using cumulative step size adaptation (CSA) [13] which in a nutshell adapts it using information on the evolution path (a sequence of mutation steps). If consecutive mutation steps have correlated directions then fewer larger steps could have been applied to cover the same distance. Conversely, uncorrelated mutation steps are the result of large steps that make the algorithm oscillate back and forth. In a similar fashion, the covariance matrix is adapted using the evolution path and the successive differences between the mean population vectors at generations g and $g+1$.

Unlike SA-ES whose distribution ellipsoid is dependent of the coordinate system, CMA-ES allows the realization of any normal distribution. The ellipsoid has its own orthogonal basis described by the eigenvectors of the covariance matrix.

3. EXPERIMENTAL STUDY

3.1 Description of the scenario

We are interested in studying the evolution of the mutation steps over time, for that we have conducted experiments using the above described algorithms on a dynamic sphere function:

$$\mathcal{F}(X) = min \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \quad (4)$$

were $\hat{X} = (\hat{x}_1, \ldots, \hat{x}_n)$ represents the moving optimum over time. Table 1 shows the different parameters used in the experiments in which "period" represents the number of generations between the movements of the optimum. In the case of CMA-ES, we kept the default parameters suggested by N. Hansen [11]. The results illustrate the evolution of the mutation steps over time and the error to the optimum, in the case of the CMA-ES we also present the direction of the search i.e. the direction of the eigenvectors of the covariance matrix. Due to the stochastic nature of the algorithms, each experiment was conducted 1000 times and the figures represent the averaged results with their respective standard deviations. The algorithms were left evolving throughout the changes with no explicit actions taken when changes occur.

Table 1: Parameter settings for the presented experiments.

Parameters	Algorithm	
	SA-ES	CMA-ES
Dimension n	2	2
Population size μ	5	3
Offspring λ	10	6
Initial step size	5.0	0.5
τ and τ'	0.3 and 0.4	n.a.
Period (gen)	50	50

In this study, we were interested in three types of movements of the optimum: movements with constant velocity, movements with linearly increasing velocity and movements with quadratically increasing velocity.

3.2 Results and discussion

Experiments with SA-ES

Figures 1 trough 7 represent the evolution of the mutation steps and the error of the best individual over time in the case of SA-ES. In all experiments the optimum moves every 50 generations. Figure 1 represents the fluctuations of σ_0 when the optimum moved with a constant velocity in one dimension (severity of 1.0). This is the effect of self adaptation, the mutation step increases to favor exploration when the algorithm is no longer at the optimum, and then gradually decreases when it starts converging toward the optimum. Figure 2 shows the error to the optimum for that case. The interesting part about this experiment, is that the maximum value σ_0 takes remains constant throughout the movements of the optimum.

In the case where the optimum moves with linearly increasing velocity, the mutation steps behave differently, figure 3 shows the fluctuations of σ_0. In this experiment the optimum moves with linearly increasing severity i.e. the distance traveled increases linearly. Here the results show that at each movement of the optimum the mutation steps follow a similar behavior and attain a maximum value that increases linearly over time. This values reflects the search distance or the width of the normal distribution for the mutation needed to find the optimum. Figure 4 shows the error of the best individual.

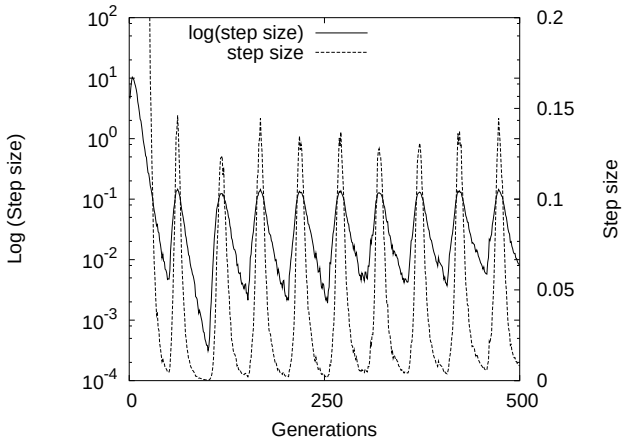


Figure 1: Evolution of the mutation step (both in log and regular scale) in the case of SA-ES when the optimum moves with constant velocity in the dimension x_0

In the third experiment conducted on the SA-ES, the optimum moved with a quadratically increasing velocity in other words the distance (severity) traveled by the optimum increased quadratically. Here again the mutation steps behave in a similar fashion with the movements of the optimum, and attain a maximal value that increase quadratically overtime. Figure 5 shows the evolution of σ_0 and figure 6 shows the error of the best individual to the optimum.

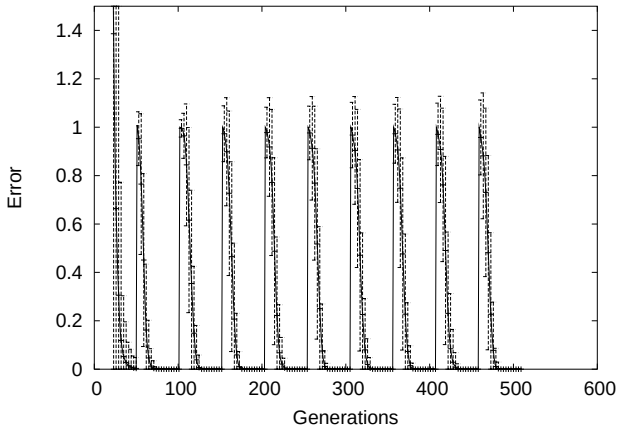


Figure 2: Evolution of the error in the case of SA-ES when the optimum moves with constant velocity in the dimension x_0

Figure 7 show a case where the optimum moves with constant speed (severity 1.0) in the first dimension and then shifts direction at generation 250 to move in both dimensions at once with a 60 degrees heading. In the first phase of this experiment, only σ_0 adapts to the changing environment, σ_1 keeps low values during this phase (no exploration needed in this dimension). However when the optimum changes direction, σ_1 starts evolving and attain higher values than σ_0 due to the fact that with a 60 degrees the optimum moves more in the second dimension than in the first one. However since the axes of the mutation distribution are linked to the coordinates system, the heading of the optimum cannot be known directly. In the next set of experiments with the CMA-ES we show that the direction of movement could be easily known.

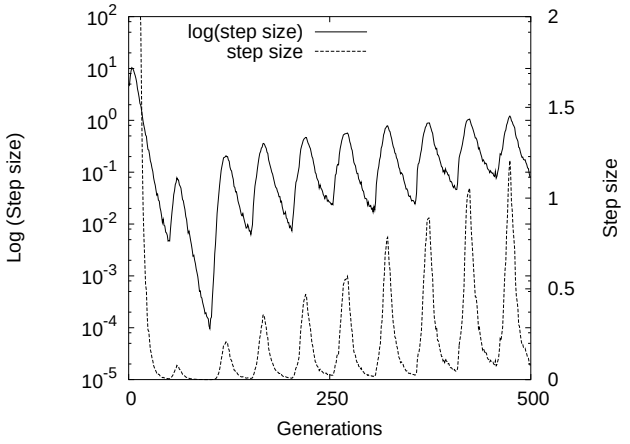


Figure 3: Evolution of the mutation step (both in log and regular scale) in the case of SA-ES when the optimum moves with linearly increasing velocity in the dimension x_0

Experiments with CMA-ES

In this part, we present the experiments conducted using the CMA-ES algorithm. In addition to the mutation steps, we show the direction of the dominant eigenvector. The dominant eigenvector is the largest vector in the eigenvector basis of the covariance matrix. These experiments show that not only the mutation step reflect the amplitude of the optimum's movements, the eigenvector follows the direction of the optimum.

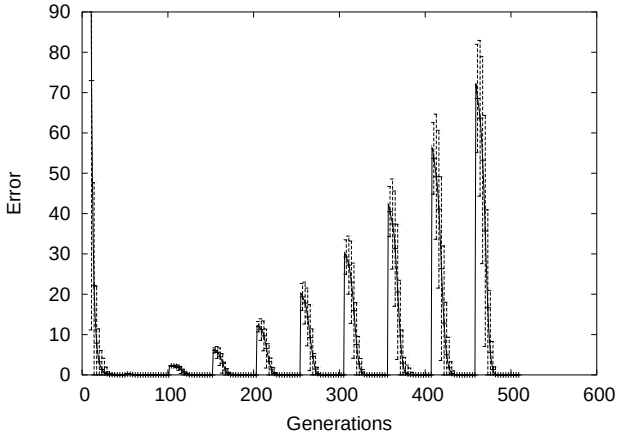


Figure 4: Evolution of the error in the case of SA-ES when the optimum moves with linearly increasing velocity in the dimension x_0

In the first experiment of this series, the optimum moves with constant velocity with severity one in the first dimension. Figure 8 illustrates the direction of the eigenvector when the optimum moves in the first dimension. It shows

that the angle of the eigenvector maintains a constant direction at angle zero throughout the experiment; this corresponds to the direction in which the optimum moves. However from figure 8, we can't describe the nature of movement, we will have to inspect the evolution of the mutation step (figure 9) which behave in a similar fashion as in the case described by figure 1, where step sizes attain a constant maximum value.

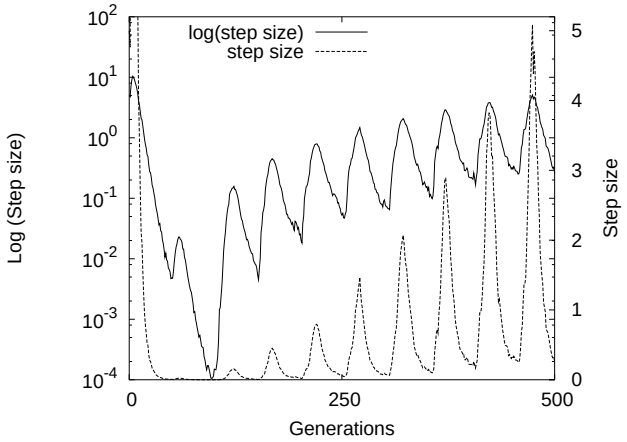


Figure 5: Evolution of the mutation step (both in log and regular scale) in the case of SA-ES when the optimum moves with quadratically increasing velocity in the dimension x_0

In the second experiment, the optimum changes direction during the evolution. In the first phase, the optimum moves on the first dimension as in the experiment above. Afterward it shifts direction by 45 degrees during the second phase. In phase three it moves back in the direction of the first dimension then sifts again in phase four to -45 degrees. Throughout all phases, the optimum maintains a constant velocity. The evolution of the direction the eigenvector takes is shown on figure 10, in which we can clearly distinguish the different phases of the movement.

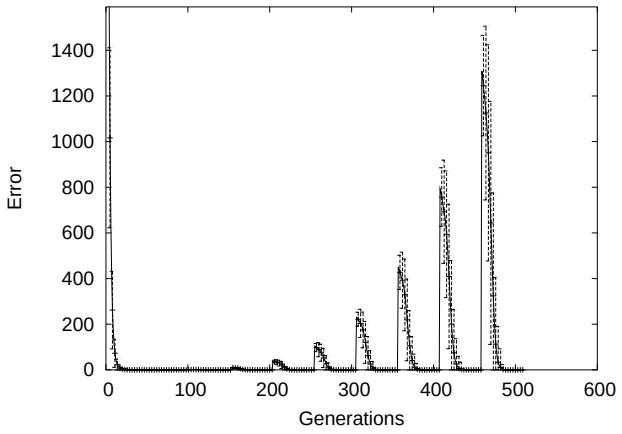


Figure 6: Evolution of the error in the case of SA-ES when the optimum moves with quadratically increasing velocity in the dimension x_0

Figures 12 and 13 illustrate the case where the optimum moves with linearly increasing velocity in both dimensions at once (45 degrees heading). They correspond respectively to the direction of the eigenvector and the evolution of the mutation step. In this case we also note that the mutation step increases linearly over time. Similarly, figures 14 and 15 show the same entities in the case were the optimum moves with quadratically increasing velocity. In this experiment the optimum moved in the first dimension.

3.3 Discussion

The aim of mutation adaptation in evolutionary computing is to control the step size in order to adapt the algorithm to the fitness landscape. This adaptation process is generally expressed in the evolution of the step sizes; this evolution is characterized by two phases: exploration and exploitation. The exploration phase represents the period during which the algorithm is searching for better performance, most mutations during this period are unsuccessful and the step sizes increase. When the algorithm starts to gain performance, it enters the exploitation phase, mutations become more successful and the step size reduces as the algorithm approaches the optimum. Mutation steps increase during exploration and decrease during exploitation.

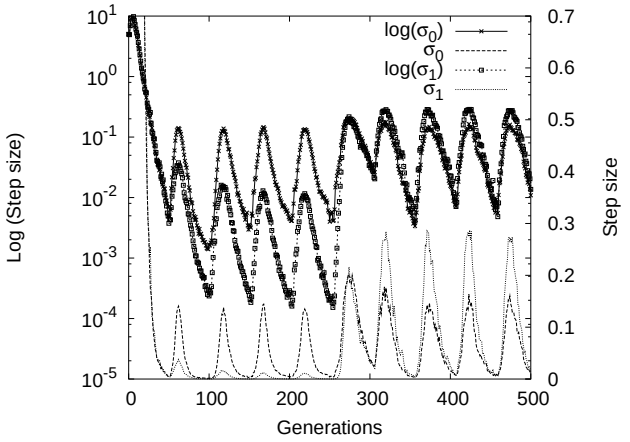


Figure 7: Evolution of the mutation steps in the case of SA-ES when the optimum moves with constant velocity with a change of direction at generation 250

In dynamical environments the algorithm is constantly switching between exploration and exploitation, and step sizes oscillate between high and low values. Once an optimum is lost, the adaptation process increases mutation steps to raise exploration until it finds it again.

In the case where the optimum moves with constant velocity, the step sizes attain a constant hight value after the exploration phase and then drop during the exploitation phase. Furthermore, in the case where the optimum moves with linearly increasing velocity, the maximum values attained by the mutation steps are not constant but increase linearly over time, which is in concordance with the movements of the optimum. Finally, when the velocity of the optimum increases quadratically, the maximum values attained by the mutation steps increase quadratically also. These results

show that there exist a correspondence between the nature of the dynamics in the environment and the evolution of the step sizes. This correspondence exists with both adaptation techniques.

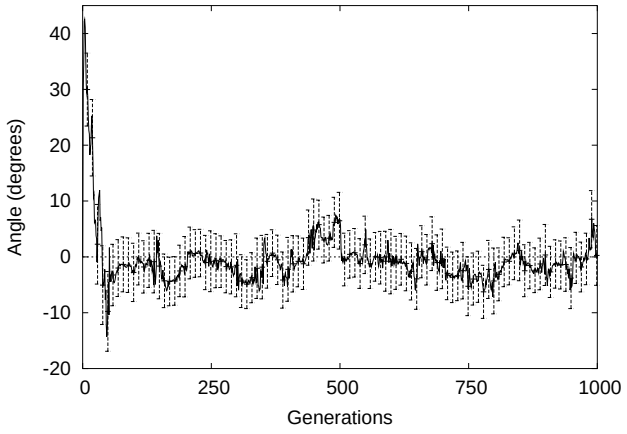


Figure 8: Direction of the dominant eigenvector in the covariance matrix over time when the optimum moves in the dimension x_0 with constant speed.

It is conceivable to exploit the information given by the step size evolution, in order to for example detect the dynamic of the environment and further adapt the algorithms to the different movements. Even though our experiments were conducted on a simple unimodal function, we believe that in multi-modal environments the evolution steps should behave similarly. In that case we expect to track the evolution of different optima at once.

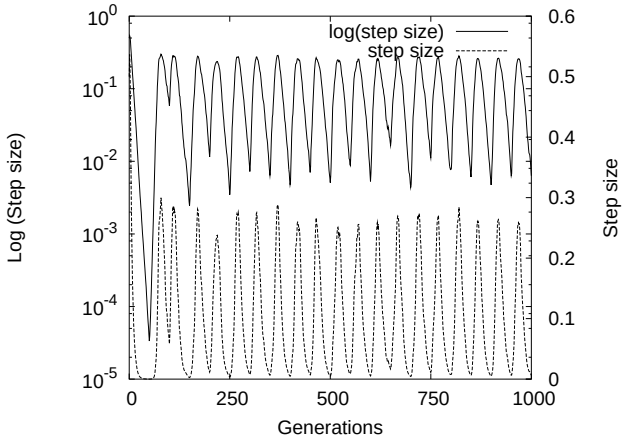


Figure 9: Mutation steps (both in log and regular scale) over time when the optimum moves in the dimension x_0 with constant speed in the case of CMA-ES.

We have also noticed in our experiments that using high severity for the optimum's movements, as in our case, the algorithm needs several generations to adapt its parameters. Experiments not presented here with 10 generations between changes in the environment, showed that the algorithm was not able to adapt its mutation steps before the change occurred. However that was not the case when the severity was low, with a small number of generations the algorithm could adapt its steps. This observation implies that, in order to extract useful results on the dynamics of the environment, we have to give the algorithm enough time to adapt. However most of the times we don't have such a possibility, since we don't control the dynamics.

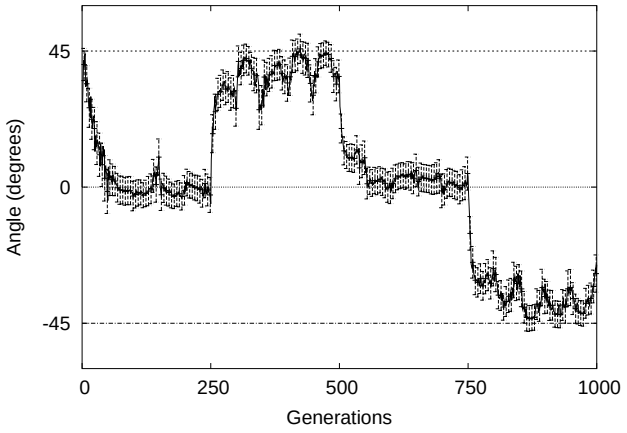


Figure 10: Direction of the dominant eigenvector in the covariance matrix over time when the optimum changes direction.

Finally, our study lacks the investigation concerning the link between the values of the mutations steps and the velocity values of the movements. We believe, that the amplitude of the step sizes could indicate the distance traveled by the optimum. In figure 7 where both mutation steps are shown, we notice that since the movement in the second dimension is greater than the movement in the first one, the steps have different amplitudes. It would be interesting to make a link between the value of the step size and the displacement.

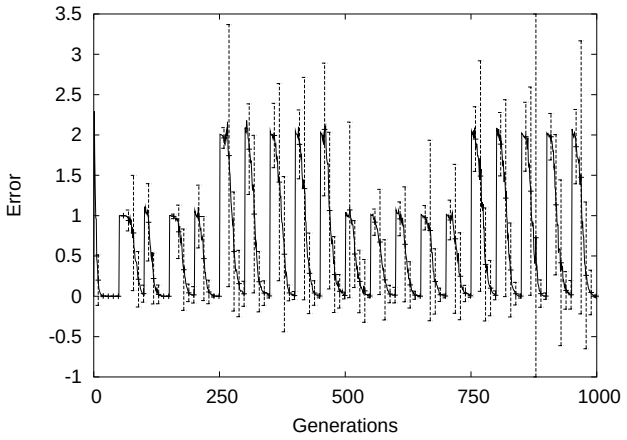


Figure 11: Evolution of the error when the optimum changes directions in the case of CMA-ES.

In experiments where the optimum shifts direction, we show how the different steps evolve. In the case of CMA-ES, the direction of the optimum's movements is learned

be the covariance matrix and characterized by the principal axis of the mutation ellipsoid. This is due to the invariance of CMA-ES against rotations. This is not the case with SA-ES, since it is dependent of the coordinate system. The results show that if the optimum moves further in one dimension than in the others, the step size on that dimension attain higher values during adaptation. We believe though not shown here, that the heading can by inferred from the difference in proportions of the mutation steps. For example if the σ_0 is twice the value of σ_1, we could say that the optimum moves roughly at a 25 degrees heading.

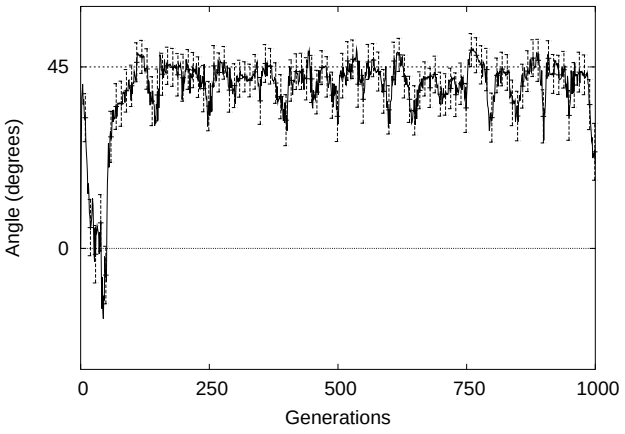


Figure 12: Direction of the dominant eigenvector in the covariance matrix over time when the optimum moves with linearly increasing velocity in both dimensions.

Theoretical results [2] showed that the convergence of the step sizes is linear in log scale, this property is demonstrated here, the step size in all experiments converge lineally. Although the theoretical results apply to the $(1, \lambda)$ strategy, it is interesting to notice that they are verified in the case where the population size is greater than one.

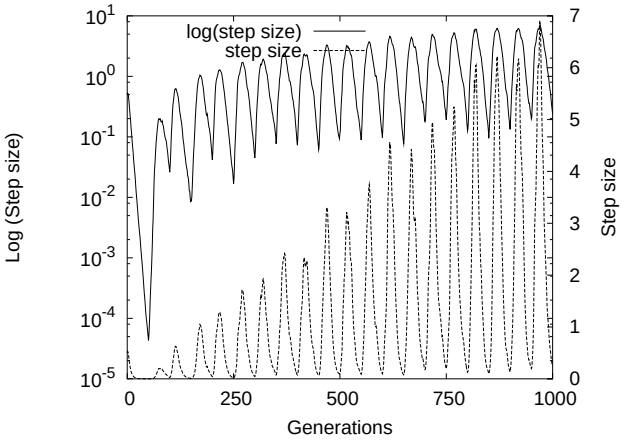


Figure 13: Evolution of the step size (both in log and regular scale) over time in CMA-ES when the optimum moves with linearly increasing velocity in both dimensions.

4. CONCLUSIONS

In this article we showed through experimentation that there exists a relationship between the movements of the dynamic sphere model and the evolution of the self-adapted mutation steps. Experiments on both SA-ES and CMA-ES algorithms show that the nature of the movement of the optimum is reflected in the evolution of the mutation steps. In the case of CMA-ES we have shown that not only the nature of the movements is reflected in the mutation step, the direction of movement of the optimum is learned by the covariance matrix. The dominant eigenvector has the same direction as the moving optimum.

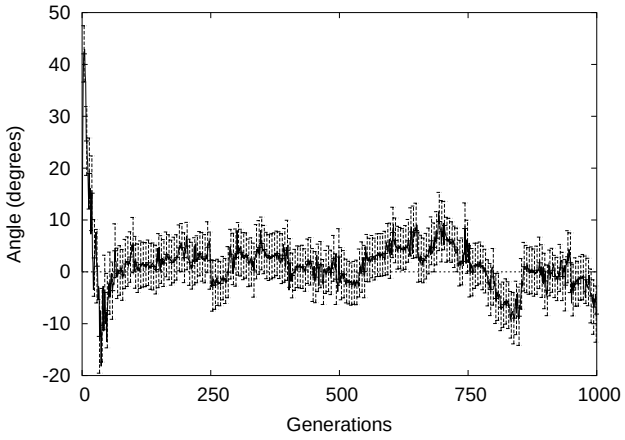


Figure 14: Direction of the dominant eigenvector in the covariance matrix over time when the optimum moves with quadratically increasing velocity.

This, as stated in the introduction, is only a preliminary study, further experiments should be conducted on different fitness models and higher dimensions than presented here. It is also important to experiment on the multi-modal case, where several moving optima need to be detected. Finally, it is interesting to see such relationships between the dynamics of the search space and the evolution of the mutation parameters; however what would be more interesting and by the same means challenging, is to use the information learned from the adaptation in a meaningful manner, say for example privilege certain directions of the search space, or tune the internal parameters to the specific dynamics of the problem.

5. ACKNOWLEDGMENTS

I would like to warmly thank Nikolaus Hansen for providing me with an implementation of the CMA-ES algorithm and for his useful comments.

6. REFERENCES

[1] P. J. Angeline. Tracking extrema in dynamic environments. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors, *Proceedings of the 6th International Conference on Evolutionary Programming*, volume 1213 of *Lecture Notes in Computer Science*, Indianapolis, Indiana, USA, April 13-16 1997. Springer Verlag.

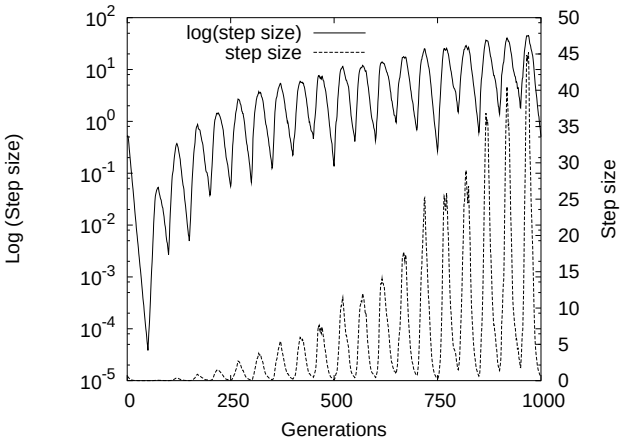


Figure 15: Evolution of the step size (both in log and regular scale) over time when the optimum moves with quadratically increasing velocity in the case of CMA-ES.

[2] A. Auger. Convergence results for (1,λ)-SA-ES using the theory of ϕ-irreducible markov chains. *Theoretical Computer Science*, 2004. In press.

[3] T. Bäck and H. P. Schwefel. Evolution strategies. In J. Périaux and G. Winter, editors, *Genetic Algorithms in Engineering and Computer Science*, chapter 6-7. John Wiley and Sons, 1995.

[4] T. Bäck. Self-adaptation in genetic algorithms. In F. J. Varela and P. Bourgine, editors, *Proc. of the 1st European Conf. on Artificial Life*, pages 227–235, Cambridge, MA, 1992. MIT Press.

[5] H.-G. Beyer. Evolution strategies: A comprehensive introduction. *Natural Computing*, 1(1):3–52, 2002.

[6] H.-G. Beyer and K. Deb. On self-adaptive features in real-parameter evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 5(3):250–270, 2001.

[7] J. Branke. *Evolutionary Optimization in Dynamic Environments*. Kluwer, 2001.

[8] J. Branke and W. Wang. Theoretical analysis of simple evolution strategies in quickly changing environment. In E. Cantu-Paz et al., editor, *Proceedings of the Genetic and Evolutionary Conference*, pages 537–548. Springer, 2003.

[9] L.J. Fogel, P.J. Angeline, and D.B. Fogel. An evolutionary programming approach to self-adaptation on finite state machines. In J. McDonnell, R. Reynolds, and D. Fogel, editors, *In Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming.*, pages 355–366. MIT Press, 1995.

[10] N. Hansen, S.D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[11] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[12] R. Hinterding, Z.Michalewicz, and A.E. Eiben. Adaptation in evolutionary computation: A survey. In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, pages 65–69, 1997.

[13] A. Ostermeier, A. Gawelczyk, and N. Hansen. A derandomized approach to self-adaptation of evolution strategies. *Evolutionary Computation*, 2(4):369–38, 1994.

[14] H. P. Schwefel. *Numerical Optimisation of Computer Model*. Wiley, 1981.

Inverse Multi-Objective Robust Evolutionary Design Optimization in the Presence of Uncertainty

Dudy Lim Yew-Soon Ong Bu-Sung Lee
School of Computer Engineering
Nanyang Technological University
Nanyang Avenue, Singapore 639798
{dlim, asysong, ebslee}@ntu.edu.sg

ABSTRACT
In many real-world design problems, uncertainties are often present and practically impossible to avoid. Many existing works on Evolutionary Algorithm (EA) for handling uncertainty have emphasized on introducing some prior structure of the uncertainty or noise to the variable domain and conducting sensitivity analysis based on the assumed information. In this paper, we present an evolutionary design optimization that handles the presence of uncertainty with respect to the desired robust performance in mind, which we call an inverse robust design. The scheme, unlike others developed to represent uncertainty does not assume any structure of the uncertainty involved; hence it is particularly useful when there is very little information about the uncertainties available. In our formulation, we model the clustering of uncertain events in families of nested sets using a multi-level optimization searches within the multi-objective evolutionary search. Empirical studies were conducted on synthetic functions to demonstrate that our algorithm converges to a set of designs with non-dominated nominal performances and robustness to the presence of uncertainties.

Categories and Subject Descriptors
G.1.6 [**Numerical Analysis**]: Optimization – *global optimization.*

General Terms
Algorithms.

Keywords
Evolutionary Algorithms, robust design optimization, design optimization in the presence of uncertainty.

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

1. INTRODUCTION
Uncertainties are often present and practically impossible to avoid in many real world engineering design problems. For instance, if a design is very sensitive to small geometric variations, which may arise either due to manufacturing processes, and/or in-service degradation due to erosion processes and foreign object damage, and/or drifts in operating conditions, it may not be desirable to use this design. Hence optimization without taking uncertainty into consideration generally leads to designs that should not be labeled as optimal but rather potentially high risk designs that are likely to perform badly when put to practical use. Faced with high sensitivities to uncertainties, traditional Evolutionary Algorithms (EAs) [1] tend to display sign of over-searching since they naturally favor designs with a larger fitness value. However, in practice, the preferable design solution is probably one that may not be the globally optimum solution, but one that has a high tolerance or robustness to uncertainties. Solutions whose performances do not change much in the presence of uncertainties are often referred to as *robust designs*.

In recent years, a number of approaches have been proposed in the literature to attain *robust designs*. These include the One-at-a-Time Experiments, Taguchi Orthogonal Arrays, bounds-based, fuzzy and probabilistic methods [2]. In EA, a number of prominent new studies on handling the presence of uncertainty in engineering designs have also been made over the recent years. A noisy phenotype scheme was introduced in [3] where a probabilistic noise vector is added to the genotype before fitness evaluation. In biological terms, this means that part of the phenotypic features of an individual is determined by the decoding process of the genotypic code of genes in the chromosomes. In [4], the study of an (1+1)-Evolutionary Strategy (ES) with isotropic normal mutations using the noisy phenotype scheme has also been reported. In [5-7], uncertainty was regarded in the form of a dynamic environment where the landscape of the problem is perceived to be changing dynamically. In their work, approaches using multi-populations to facilitate exploration and exploitation were also considered. A multi-objective approach to handling uncertainty in EA was also studied in [8] where the trade-off between robustness and nominal performance of a solution was discussed. A strategy to attain robust designs with minimum variations in noise was also presented in [9] on realistic mechanical design problem.

In most of these schemes, often some prior knowledge about the structure of the uncertainties, such as the distribution property involved were assumed to be available. For instance, the uncertainty that exists in the environmental conditions and design parameters of evolutionary search are often assumed to be uniformly or normally distributed, with certain range or standard deviation. However, it is worth noting that the quality of the solution is generally attainable only when the assumptions made on the structure of the uncertainty reflect the actual uncertainty flawlessly.

In many real world engineering design problems, it is often the case that very little knowledge about the structure of the uncertainty involved is available. Making assumptions about the uncertainty that are not backed up by strong evidence in evolutionary design optimization can possibly lead to erroneous designs that could have catastrophic consequences. Thus, it would be wiser for one to avoid making assumptions about the structure in the formulation of the optimization search process. In this paper, instead of using sensitivity analysis, i.e., analyzing the changes in performance of a design with respect to variability in the key design variables, we present evolutionary design optimization that handles the presence of uncertainty in view of the desired robust performance, which we call the inverse robust design. From the desired performance, we search for solutions that guarantee a certain degree of maximum uncertainty and at the same time satisfy the desired nominal performance of the final design solution. For this purpose, we conduct series of nested multi-point local searches using the Sequential Quadratic Programming method [10] within the Non-dominated Sorting Genetic Algorithm (NSGA) [11].

The remaining of this paper is organized as follows: Section 2 provides a brief discussion on evolutionary design optimization in the presence of uncertainties. The proposed algorithm for inverse evolutionary robust design optimization in the presence of uncertainties is presented subsequently in section 3. Section 4 summarizes our empirical study on synthetically generated benchmark functions before section 5 finally concludes this paper.

2. EVOLUTIONARY DESIGN OPTIMIZATION IN THE PRESENCE OF UNCERTAINTY

In this section, we present a brief overview on some fundamentals of robust evolutionary design optimization in the presence of uncertainties. In particular, we consider the general bound constrained nonlinear programming problem of the form:

$$\textbf{Maximize}: \quad f(x)$$
$$\textbf{Subject to}: \quad x_l \le x \le x_u \qquad (1)$$

where $f(x)$ is a scalar-valued objective function, $x \in \Re^d$ is the vector of design variables, while x_l and x_u are vectors of lower and upper bounds for the design variables.

Further, it is noted that the present focus is on EAs for robust engineering design optimization under uncertainties that arise in:

i) design vector x

$$F(x) = f(x+\delta) \qquad (2)$$

where $\delta = (\delta_1, \delta_2, ..., \delta_k)$, is the noise in the design vector where some distribution about the uncertainty is assumed and $F(x)$ is the effective fitness of design vector x.

ii) operating/environmental conditions

$$F(x) = f(x, c+\xi) \qquad (3)$$

where $c = (c_1, c_2, ..., c_n)$, is the nominal value of the environmental parameters and ξ is a random vector used to model the variability in the operating conditions. Referring to [8], both forms of uncertainties may be treated equivalently. Hence in this study, we do not differentiate the uncertainties between design variables and operating conditions. Rather the reader is referred to [8] for greater details on the issue. However, as far as this paper is concerned, we consider uncertainties in the design variables.

The core mechanism in many existing robust schemes for solving this type of uncertainty is driven based on the effective fitness $F(x)$ of the design solutions. Mathematically, the effective evaluation function $F(x)$ is generally defined as:

$$F(x) = \int_{-\infty}^{\infty} f(x+\delta) q(\delta) d\delta \qquad (4)$$

where $q(\delta)$ is a continuous density function of noise δ which is often assumed to be known *a priori*, usually a Gaussian or uniform distribution.

To locate a robust design solution in the presence of uncertainties in the design vector, one may consider using the *Noisy Phenotype Scheme* proposed in [3] which is outlined in Figure 1.

BEGIN EA (for maximization problem)

- Generate a population of designs

while(termination condition is not satisfied)

for(each individual *i* in the population)

for(*j=1* to *m*)

- Perturb individual to arrive at x_{ij}
- Evaluate $F(x_{ij}) = f(x_{ij} + \delta_j)$

end for

- Determine effective fitness, $F(x_i)$ of individual *i*

$$F(x_i) = \frac{1}{m} \sum_{j=1}^{m} F(x_{ij})$$

end for

- apply mutation and crossover to create new population
- perform selection of individuals.

end while

END EA

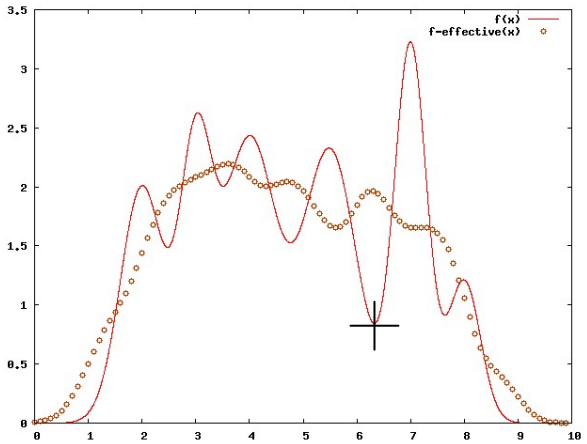
Figure 1. Noisy Phenotype Scheme.

Consider the one-dimensional function depicted in Figure 2 and defined by

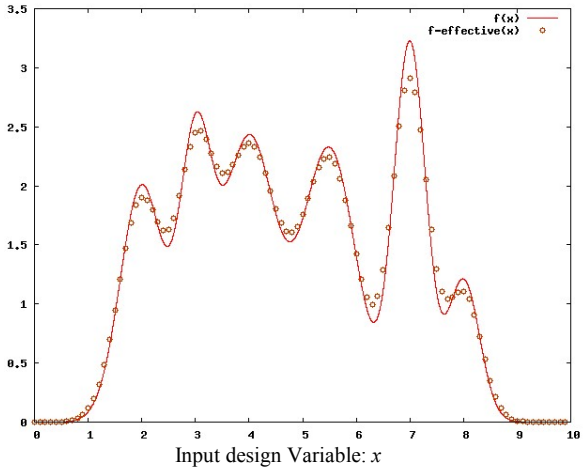
$$f(x) = 2e^{-(x-2)^2/0.32} + 2.2e^{-(x-3)^2/0.18} + 2.4e^{-(x-4)^2/0.5}$$
$$+ 2.3e^{-(x-5.5)^2/0.5} + 3.2e^{-(x-7)^2/0.18} + 1.2e^{-(x-8)^2/0.18}$$

where $-1 \leq x \leq 10$ (5)

This represents a multimodal function with a nominal global optimum located at sharp peak $x^* \in [6.5,7.8]$ and has many other ocal optima located elsewhere[1]. The robust solution that the noisy phenotype scheme in figure 1 converges to depends on the perturbation assumed on δ, *i.e.*, the assumption on the structure of the uncertainty, δ. By making assumption on the distribution of δ in $f(x)$, one may easily derive the respective effective fitness function, $F(x_i)$. For instance, figures 2(a) and (b) illustrates the effective fitness functions for the one-dimensional function defined in equation (5) assuming a uniform distribution for δ with σ set to ± 1.0 and ± 0.25, respectively. Note that σ defines the range or bound of the uncertainty that is assumed about δ. When the range for σ is configured to be ± 1.0, the global robust optimum[2] may be easily found to be located in the region $x^\wedge \in [3.0,4.0]$. On the other hand, if σ is configured to be ± 0.25, the global robust optimum approaches that of the nominal fitness function $f(x)$. For a complete explanation of how these may be arrived at, the reader is referred to [3, 6].



(a) Range of the uncertainty, σ= ± 1.0



(b) Range of the uncertainty, σ= ± 0.25

Figure 2. Effective fitness F(x) of the function defined in equation (5) assuming a uniform distribution for δ

In most cases, the algorithm described in Figure 1 is capable of converging appropriately to the robust design solution defined by the effective fitness as long as the assumption made about the uncertainty, δ, including the type of distribution and the respective range or deviation, are known precisely. In contrast, it is often the case in many realistic problems that very little knowledge about the structure of the uncertainty involved is available *a prior*. Besides, a major problem with many existing robust schemes in the literature is that the nominal fitness of the final design is often neglected [3, 5, 12]. These schemes generally optimize the robustness of the final design, at the expense of nominal performance of the final design. For instance, it may be observed from figure 2(a) that the design point x at 6.3 possess very good robustness, *i.e.*, a high effective fitness of around 1.4, but have a very poor nominal fitness of only 0.84. This implies that it is crucial to consider both the nominal performance and robustness in the design optimization search. A straightforward manner to solve this problem is to reformulate the robustness scheme as a constraint problem with $f(x) \geq c$ and c is the minimum acceptable nominal performance for the final design [13]. This way, any individuals that fail the constraint gets heavily penalized in the robust EA search. This approach however may not be practical since information about the perceived minimum performance may not always be available.

A more promising solution to handle the trade-off between the robustness and nominal fitness is to consider a multi-objective optimization approach [8] where a pareto front of robustness and nominal fitness can be attained. Motivated by this work, we present here an inverse robust solution based on a multi-objective evolutionary approach. In particular, we consider two objective functions, namely the *robustness* and *nominal fitness* of the design.

[1] Note that x^* represents the nominal global optimum.

[2] Note that $x^\wedge$ represents the global effective optimum.

3. INVERSE MULTI-OBJECTIVE ROBUST EVOLUTIONARY DESIGN OPTIMIZATION

To mitigate the problems identified in section 2, we present in this section an inverse multi-objective robust evolutionary design optimization strategy for locating designs with non-dominated nominal performances and robustness in the presence of uncertainties.

In many real world engineering design problems [12, 14-16], it is often the case that very little knowledge about the structure of the uncertainty involved is available. Hence, instead of focusing on making any probably unjustifiable mathematical model out of the uncertainty, we focus here on how a design may deteriorate in the presence of uncertainties. While it is common that designers may not possess the necessary expertise or have sufficient knowledge to identify suitable bounds of the uncertainties involve. On the contrary, it is more viable that designers have practical knowledge about the robust performance of the final design it desires.

Here, we present the proposed algorithm for Inverse Multi-Objective Robust Evolutionary design optimization (IMORE) in the presence of uncertainty. The basic steps of the proposed algorithm are outlined in Figure 3. In the first step, the maximum degradation tolerable for the final design, d_t and step size Δ used to conduct nested searches are defined and initialized. Within the initialization phase, a population of designs is also created either randomly or using design of experiments techniques such as Latin hypercube sampling or minimum discrepancy sequences [17]. Each individual in the population is first evaluated to determine its nominal fitness. Subsequently, each individual then undergoes a sequence of nested searches across a family of nested search regions parameterized by the uncertainty vector in the spirit of Info-Gap theory [18-20]. The aim of the nested searches is to determine the maximum robustness that a particular design can be guaranteed to handle under the permitted maximum performance degradation of d_t defined. More specifically, during the inner search for each chromosome in an IMORE generation, we solve a sequence of bound constrained optimization subproblems of the form:

$$\textbf{Maximize: } d(x) = f(x_i) - f^k(x)$$

$$\textbf{subject to: } x_l^k \le x \le x_u^k \qquad (6)$$

where x_l^k and x_u^k are the appropriate bounds on the design variables, which is updated at each k iteration based on the step size defined, Δ.

For each optimization subproblem (or during each k iteration), the optimal solution of the k^{th} subproblem is sought. The objective of each subproblem search is to find the worst-case fitness function value by solving a bound constrained maximization problem. After each iteration, the design variables search bounds, x_l^k and x_u^k are updated using the step size Δ which is given by

$$x_l^k = x_i - k\Delta$$

$$x_u^k = x_i + k\Delta \qquad (7)$$

BEGIN IMORE (Consider a maximizing problem)

Initialization Phase:

- Initialize Maximum degradation tolerable for the final design, d_t
- Initialize the step size Δ for local search
- Generate a population of design vectors

Search Phase:

While (termination condition is not satisfied)

For (each individual i in the population)

- Evaluate $f(x_i)$

Repeat

- **Maximize:** $d(x) = f(x_i) - f^k(x)$

  subject to: $x_l^k \le x \le x_u^k$

  $x_l^k = x_i - k\Delta$, $x_u^k = x_i + k\Delta$
- Obtain x_{opt}^k and $d\left(x_{opt}^k\right)$
- Store $d\left(x_{opt}^k\right)$ and associate it with $k\Delta$

until $\left\{d^k(x) = f(x_i) - f(x_{opt}^k)\right\} > d_t$

- Estimate maximum uncertainty δ_{max}^i using linear interpolation from $d\left(x_{opt}^k\right)$ for different $k\Delta$
- Nominal fitness$(x_i) = f(x_i)$
- Maximum uncertainty$(x_i) = \delta_{max}^i$

end For

- Apply standard MOEA operators to create a new population

end While

END IMORE

Figure 3. Inverse Multi-Objective Robust Evolutionary Design Algorithm in the presence of uncertainty.

It is worth noting that by conducting a sequence of local searches across a family of ascending nested bounds parameterized by the uncertainty vector, we arrive at a monotonic increasing function of performance degradation versus uncertainty as illustrated in Figure 4 such that

$$x_l^{k+1} \le x_l^k, x_u^k \le x_u^{k+1} \rightarrow d\left(x_{opt}^k\right) \le d\left(x_{opt}^{k+1}\right) \qquad (8)$$

where x_{opt}^k denotes the optimum at the k^{th} iteration and $d\left(x_{opt}^k\right) = f(x_i) - f\left(x_{opt}^k\right)$ is the corresponding maximum performance degradation obtained for $x_l^k \le x \le x_u^k$. In addition, the $d\left(x_{opt}^k\right)$ found and associated $k\Delta$ for each search iteration is then stored to create a database of uncertainties and corresponding performance degradations. For example, consider a design point with x_i=4 in Figure 4, labelled points A, B and C correspond to $(x_{opt}^k, f(x_{opt}^k))$ for k=1, 2 and 3 respectively and Δ set to 1.

For each chromosome, the iterative searches are terminated when the optimal solution of the k^{th} subproblem exceeds the maximum degradation defined, i.e.

$$\left\{d^k(x) = f(x_i) - f(x^k_{opt})\right\} > d_t \qquad (9)$$

At the end of the sequences of searches for a chromosome, the maximum uncertainty δ_{max} that a design may handle given a maximum performance degradation of d_t permitted can be determined by interpolating from the database of previous uncertainties and maximum performance degradations, *i.e.*, $k\Delta$ and $d^k(x)$. This is also illustrated in Figure 4 where D represents the point where a maximum performance degradation of d_t is reached and δ_{max} is the corresponding maximum uncertainty that the design guarantees to handle. The IMORE search then proceeds with the standard multi-objective operators to create a new population and terminates upon convergence.

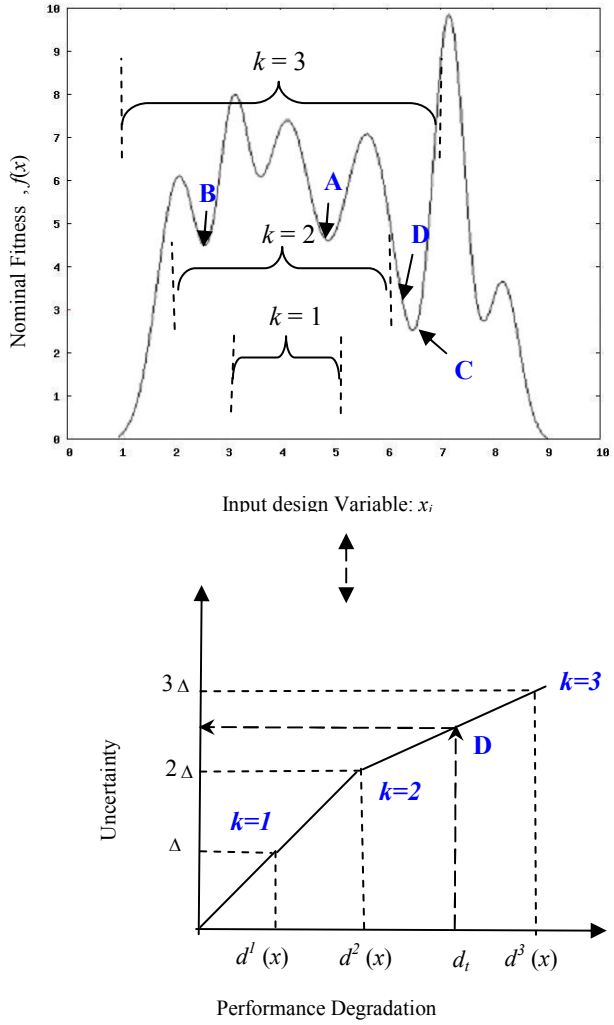


Figure 4. Steps of IMORE for x_i=4 and Δ =1.

4. EMPIRICAL STUDY

To illustrate the utility of the IMORE algorithm described in section 3, we present here an empirical study based on two synthetic one-dimensional multimodal functions. The EA was run for 100 generations with 16 bit binary-coded, linear ranking selection, mutation probability of 0.01, crossover probability of 0.9 and a population size of 100. Further, we consider here a sequence of *multi-start local* bound constrained optimization subproblems to locate the maximum certainty δ_{max} in our IMORE algorithm. In this study, we employ the Feasible Sequential Quadratic Programming (FSQP) as the local search strategy.

Test Function 1. The first test function we consider here is a one-dimensional multimodal function which is an aggregation of multiple one-dimensional Gaussian functions given in equation (10) and depicted in Figure 5. A unique property of this test function is that it contains a mixed of many sharp peaks or noisy near-global optimum solutions and rounded robust peaks in the regions $x \in [0, 4]$ and $x \in [10, 12]$, respectively. Hence it is not a simple task to identify a robust solution among them. This function is defined as:

$$\begin{aligned} f(x) &= e^{-(x-1)^2/0.5} + 2e^{-(x-1.25)^2/0.045} + 0.5e^{-(x-1.5)^2/0.0128} \\ &+ 2e^{-(x-1.6)^2/0.005} + 2.5e^{-(x-1.8)^2/0.02} + 2.5e^{-(x-2.2)^2/0.02} \\ &+ 2e^{-(x-2.4)^2/0.005} + 2e^{-(x-2.75)^2/0.045} + e^{-(x-3)^2/0.5} \\ &+ 2e^{-(x-6)^2/0.32} + 2.2e^{(x-7)^2/0.18} + 2.4e^{-(x-8)^2/0.5} \\ &+ 2.3e^{-(x-9.5)^2/0.5} + 3.2e^{-(x-11)^2/0.18} + 1.2e^{-(x-12)^2/0.18} \end{aligned}$$

where $-1 \le x \le 13$ (10)

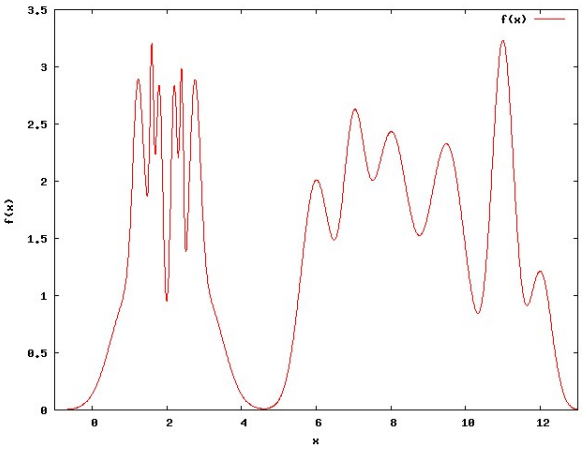


Figure 5. Test function 1.

Test Function 2. The second multimodal test function is based on the one-dimensional "Michalewicz 2" function. This test function contains a mixture of a flat and robust region with moderate

nominal fitness for $x \in [-0.5, 0.5]$ and noisy peaks with good nominal fitness for $x \in [0.5, 3]$ as depicted in Figure 6 and is defined as:

$$f(x) = \sum_{i=1}^{10} \left(sin(x) sin^{10}\left(\frac{ix^2}{\pi}\right) \right), \; -1.5 \le x \le 3 \qquad (11)$$

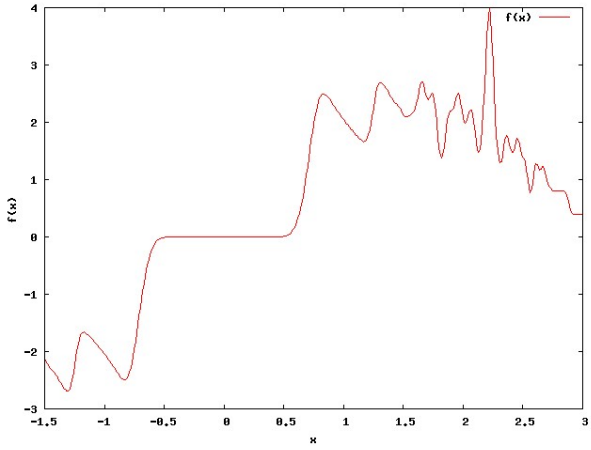


Figure 6. **Test function 2.**

In our IMORE algorithm outlined in Figure 3, it can be observed that besides the standard EA control parameters, we have introduced two additional parameters. These include 1) the maximum degradation permitted, d_t and 2) the step size for local search, Δ. d_t is user-specific and depends on the degree of robustness desired by the designer in the final design. Hence, this leaves us only with the Δ value to consider. To define a suitable value of Δ, we conduct an empirical study on the effect of IMORE for different Δs on the two test functions. In our experimental study, d_t is kept fixed at 1.0. The results obtained from the study are tabulated in Table 1. Here Δ is defined as a percentage of the search bound, *i.e.*, $x_u - x_l$. The average approximated robustness may then be defined by equation (12).

$$\frac{1}{n} \sum_{i=1}^{n} \frac{\delta_{max}^{i}}{x_u - x_l} \times 100\% \qquad (12)$$

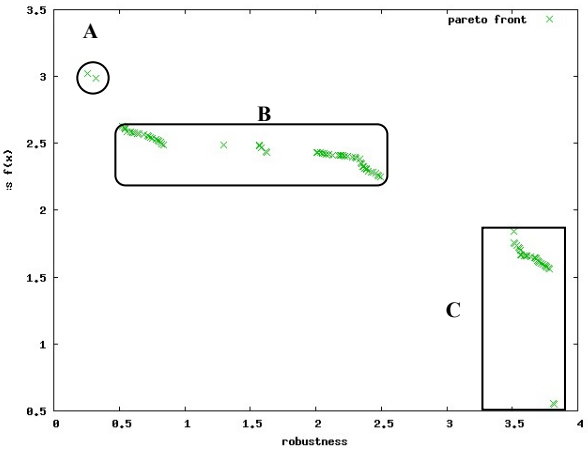
The average exact robustness is defined using the same equation (12), except that δ_{max}^{i} is now the exact robustness. From the results, it is shown that generally the average error increases with the step size, *i.e.*, the accuracy decreases with a larger step size. This makes good sense since a larger step size translates to a larger interpolation error. Like all algorithms, it is crucial to balance the accuracy desired and the computational cost incurred by the nested searches. Since a smaller step size translates to greater iterations of nested searches, *i.e.*, k, as a result, more function evaluations are also required. In our IMORE algorithm, it is possible to show that the computational cost as $O(kl)$, if l is the average number of function evaluations incurred in a single multi-start local search. Further, we consider the use of Δ =3% in all experimental studies from here onwards since this value offers good accuracy, see Table 1, the % error is lower than 0.25%.

Table 1. Effect of step size Δ in IMORE on test functions 1 and 2.

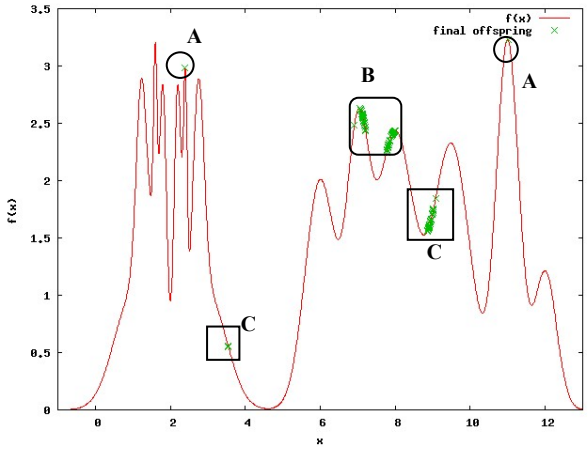
	Step Size Δ (%)	Average Approximated Robustness (%)	Average Exact Robustness (%)	Average Error (%)
Test Function 1	1	16.78	16.79	0.01
	3	*14.11*	*14.20*	*0.09*
	5	16.03	16.46	0.43
	10	14.91	15.36	0.45
Test Function 2	1	5.59	5.60	0.01
	3	*7.16*	*6.95*	*0.21*
	5	5.83	4.91	0.92
	10	9.33	6.13	3.2

Next, we consider the IMORE algorithm for optimization of functions 1 and 2. The pareto fronts obtained from the simulation runs are presented in Figures 7 and 8 for test function 1 and 2, respectively.

The solution in the pareto fronts represents a diverse set of designs having non-dominated nominal performances and robustness to the presence of uncertainties. To explain the results presented in these figures, we cluster the solutions into three separate groups in each pareto front. Group A consists of solutions having excellent nominal fitness at the expense of poor robustness. On the other hand, group B consists of solutions that are a balance trade-off between nominal fitness and robustness, while the solution members of group C have poor nominal fitness but excellent robustness measure.

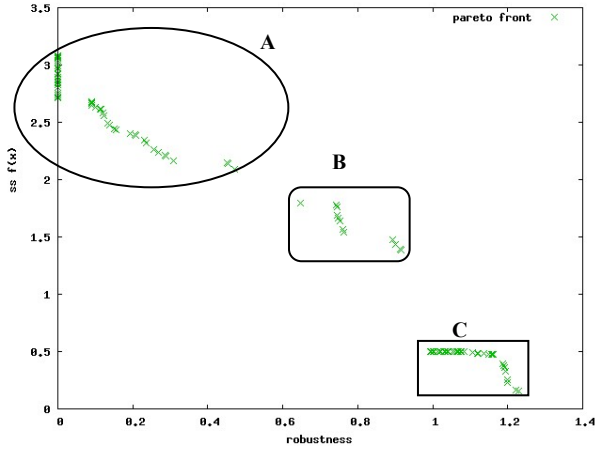


(a)

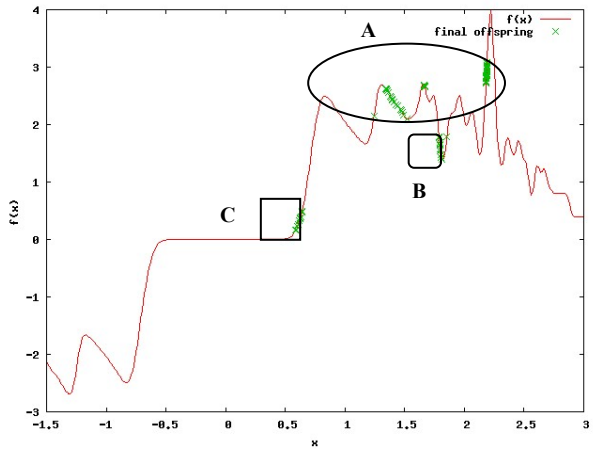


(b)

Figure 7. (a) Pareto front at generation 100, (b) Corresponding offspring in (a) for test function 1.



(a)



(b)

Figure 8. (a). Pareto front at generation 100, (b). Corresponding offspring in (a) for function 2.

5. CONCLUSION AND FUTURE WORK

In this paper, we have presented a study on inverse multi-objective robust evolutionary design optimization in the presence of uncertainty. Using *a prior* information on the desire robustness of the final design, the algorithm was shown capable of converging to a set of solutions that gives good nominal performances while handling maximum robustness in the presence of uncertainties when applied on two synthetic functions. Most importantly, these solutions were discovered without any requirement to make possible untrue assumptions about the structure of the uncertainties involved.

In evolutionary algorithms, many thousands of calls to the objective function are often required to locate a near optimal solution. While the IMORE algorithm proposed offers an effective approach to modeling of uncertainty in engineering design, a compelling limitation of the theory is the massive computational efforts incurred in the nested evolutionary design search. The computational efforts incurred would be even more devastating if the objective function is computationally expensive which is very common in complex engineering design problems [21-22]. Nevertheless, it is worth noting here that a promising and intuitive way to reduce the search time incurred in solving the sequences of bound constrained subproblems is to replace as much as possible the computationally expensive high-fidelity analysis solvers with lower-fidelity models that are computationally less expensive. The reader is referred to [21, 22] for greater details on the algorithm available to achieve this cost savings.

6. ACKNOWLEDGMENTS

The authors would like to thank the Parallel and Distributed Computing Centre at the School of Computer Engineering, Nanyang Technological University for providing support and computing resources to this work.

7. REFERENCES

[1] Goldberg D.E., "Genetic Algorithms in Search, Optimization and Machine Learning", 1989.

[2] Huyse L., "Solving Problems of Optimization Under Uncertainty as Statistical Decision Problems", *AIAA-2002-1519*, 2001.

[3] Tsutsui S. and Ghosh A., "Genetic Algorithms with a Robust Solution Searching Scheme", *IEEE Transaction on Evolutionary Computation*, Vol. 1, No. 3, pp. 201-208, 1997.

[4] Arnold D. V. and Beyer H. G., "Local Performance of the (1+1)-ES in a Noisy Environment", *IEEE Trans. Evolutionary Computation*, Vol. 6, No. 1. , pp 30-41, 2002.

[5] Branke J., "Creating Robust Solutions by Means of Evolutionary Algorithms", Springer-Verlag Berlin Heidelberg, 1998.

[6] Branke J., "Evolutionary Optimization in Dynamic Environments", Kluwer Academic Publishers, 2002.

[7] Branke J., Kaußler T., Schmidt C., and Schmeck H., "A Multi-Population Approach to Dynamic Optimization

Problems", *Adaptive Computing in Design and Manufacturing*, Springer, 2000.

[8] Jin Y. and Sendhoff B., "Trade-Off between Performance and Robustness: An Evolutionary Multiobjective Approach", *Proceedings of Second International Conference on Evolutionary Multi-criteria Optimization*. LNCS 2632, Springer, pp.237-251, Faro, 2003

[9] Chen W., Allen J.K., Tsui K.L., and Mistree F., "A Procedure for Robust Design: Minimizing Variations caused by Noise Factors and Control Factors", *ASME Journal of Mechanical Design*, 118:478-485, 1996.

[10] Lawrence C.T. and Tits A.L., "A Computationally Efficient Feasible Sequential Quadratic Programming Algorithm", *Society for Industrial and Applied Mathematics*, Vol. 11, No. 4, pp. 1092-1118, 2001.

[11] N. Srinivas and K. Deb. Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221-248, 1994.

[12] Anthony D.K. and Keane A.J., "Robust Optimal Design of a Lightweight Space Structure Using a Genetic Algorithm", *AIAA Journal* 41(8), pp. 1601-1604, 2003.

[13] Michalewicz Z., Dasgupta D., Le Riche R.G., and Schoenauer M., "Evolutionary Algorithms for Constrained Engineering Problems", *Computers & Industrial Engineering Journal*, Vol.30, No.4, pp. 851-870, 1996.

[14] Wiesmann D., Hammel U. and Back T., "Robust Design of A Multilayer Optical Coating by Means of Evolutionary Algorithms", *IEEE Transaction on Evolutionary Computation*, Vol. 2, No. 4, pp. 162-167, 1998.

[15] Huyse L. and Lewis R.M., "Aerodynamic Shape Optimization of Two-dimensional Airfoils Under Uncertain Operating Conditions", Hampton, Virginia: ICASE NASA Langley Research Centre, 2001.

[16] Padula S.L. and Li W., "Robust Airfoil Optimization in High Resolution Design Space", Hampton, Virginia: ICASE NASA Langley Research Centre, 2002.

[17] Fang K.T., Ma C.X., and Winker P., "Centered L2-Discrepancy of Random Sampling and Latin Hypercube Design, and Construction of Uniform Designs", *Mathematics of Computation*, Vol. 71, No. 237, pp. 275-296, 2000.

[18] Ben-Haim Y., "Information Gap Decision Theory", California: Academic Press, 2001.

[19] Ben-Haim Y., "Uncertainty, Probability, and Information-Gaps", *Reliability Engineering and System Safety 85*, pp. 249-266, 2004.

[20] Ben-Haim Y., "Robust Reliability in Mechanical Sciences", Springer-Verlag, Berlin, 1996.

[21] Ong Y.S., Lum K.Y., Nair P.B., Shi D.M. and Zhang Z.K., "Global Convergence of Unconstrained and Bound Constrained Surrogate-Assisted Evolutionary Search in Aerodynamic Shape Design Solvers", *IEEE Congress on Evolutionary Computation*, Special Session on Design Optimization with Evolutionary Computation", 2003.

[22] Ong Y.S., Nair P.B. and Keane A.J., "Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modeling", *AIAA Journal,* Vol. 41, No. 4, pp 687-696, 2003.

Finding the Optimal Search Dimension for Evolution Strategies with A Small Population

Yaochu Jin
Honda Research Institute Europe
Carl-Legien-Str. 30
63073 Offenbach am Main
yaochu.jin@honda-ri.de

Markus Olhofer
Honda Research Institute Europe
Carl-Legien-Str. 30
63073 Offenbach am Main

Bernhard Sendhoff
Honda Research Institute Europe
Carl-Legien-Str. 30
63073 Offenbach am Main

ABSTRACT

Small populations are very desirable for reducing the required computational resources in evolutionary optimization of complex real-world problems. Unfortunately, the search performance of small populations often reduces dramatically in a large search space. To address this problem, a method to find an optimal search dimension for small populations is suggested in this paper. The basic idea is that the evolutionary algorithm starts with a small search dimension and then the search dimension is increased during the optimization. The search dimension will continue to increase if an increase in the search dimension improves the search performance. Otherwise, the search dimension will be decreased and then kept constant. Through empirical studies on a test problem with an infinite search dimension, we show that the proposed algorithm is able to find the search dimension that is the most efficient for the given population size.

Categories and Subject Descriptors

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

General Terms

Algorithms

Keywords

Evolution strategy, optimal search dimension, dynamic problems

1. INTRODUCTION

To reduce the computational time in solving expensive optimization problems using evolutionary algorithms, a commonly adopted approach is to parallelize the fitness evaluation process so that each individual is evaluated on a separate machine. In this case, use of a relatively small population size will be very helpful in reducing the computational cost for the evolutionary optimization.

Unfortunately, we are left in a dilemma when we use small populations for solving complex real-world problems. On the one hand, many real-world optimization problems, e.g., design optimization where splines are used to describe the geometry of a structure [6], have a very large number of design parameters. On the other hand, the search efficiency decreases seriously when small populations are used to optimize problems with a high search dimension.

Two approaches could be employed to alleviate, if not solve, the difficulty mentioned above. One method is to develop an efficient evolutionary algorithm with a small population size whose performance is less sensitive to the search dimension. One good example is the derandomized evolution strategy with covariance matrix adaptation (CMA-ES) [4], which has shown to be robust on various unimodal test functions. Nevertheless, the search efficiency of the CMA-ES still greatly depends on the population size. A conclusion from empirical studies is that the population size should be scaled between linear and cubic with the problem dimension to locate the global optimum [1].

Another method is to adapt the search dimension to the population size in use. To this end, an adaptive coding scheme has been suggested where the CMA-ES is employed in aerodynamic shape optimization [7]. The basic idea is to encode the number of parameters to be optimized (the search dimension) in the chromosome and to mutate during the optimization. One issue that arises in the adaptive coding scheme is that the self-adaptation of the evolution strategy can be disturbed due to the mutation in the search dimension, which is harmful to the search performance. One measure to address this problem is to ensure that the mutations are neutral, i.e., the shape of the geometry will be kept the same before and after a new point is inserted in the spline representation.

In this paper, we will explicitly monitor the performance change after the search dimension is increased. If the increase in the search dimension is beneficial, the search dimension will be further increased. Otherwise, the search dimension will be decreased and then will be kept constant until the end of the optimization. To minimize the disturbance on the self-adaptation mechanism, the dimension is increased only by 1 in each change in dimension. Through simulations on various population sizes, it is shown that our

Genetic and Evolutionary Computation Conference, June 25-29, Washington DC, USA
.

method is able to find an optimal or nearly optimal search dimension for the given population size on a test problem with an infinite search dimension.

The test problem used in this study will be briefly described in Section 2. The search capacity of the CMA-ES with regard to the population size on the test problem are investigated empirically in Section 3. The algorithm to find the optimal search dimension is given in Section 4 and a number of simulations are conducted in Section 5, where we show that the algorithm is able to find the optimal or sub-optimal search dimension for different population sizes. Conclusion and further research topics are discussed in Section 6.

2. TEST PROBLEM

The test problem used in this study is very simple. However, it serves our purpose well where an infinitely large search dimension is needed theoretically. We consider the approximation of a one-dimensional function using a Taylor series. If a function $f(x)$ has continuous derivatives, then this function can be expanded as follows:

$$\begin{aligned} f(x) &= f(a) + f'(a)(x-a) + \frac{f''(a)(x-a)^2}{2!} + \cdots \\ &\quad + \frac{f^{(n)}(a)(x-a)^n}{n!} + R_n, \end{aligned} \tag{1}$$

where R_n is the remainder after $n+1$ terms defined by:

$$\begin{aligned} R_n &= \int_a^x f^{(n+1)}(u)\frac{(x-u)^n}{n!}du \\ &= \frac{f^{(n+1)}(\xi)(x-a)^{n+1}}{(n+1)!}, \end{aligned} \tag{2}$$

where $a < \xi < x$. When this expansion converges over a certain range of x, i.e., $lim_{n\rightarrow}R_n = 0$, then the expansion is known as em Taylor Series of function $f(x)$ about a. For example, the Taylor expansion of sine function is as follows:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, -\infty < x < \infty. \tag{3}$$

The optimization problem is to find the coefficients of the Taylor series by minimizing the squared approximation error:

$$E(x) = (\sum_{i=0}^{n} a_i x^i - \sin(x))^2, \tag{4}$$

where x is the point about which the Taylor series is expanded, $a_i, i = 0, 1, 2, \cdots, n$ is the number of terms of the Taylor series. Theoretically, an infinite search dimension is needed to realize a perfect approximation of a sinusoidal function using Taylor series. To estimate the approximation error reliably, we sample 100 points uniformly within the range of $0 \leq x \leq 1$:

$$E = \sum_{j=1}^{100} E(x_j). \tag{5}$$

An interesting fact in the above test function is that the influence of each term on the function value decreases as the order increases. Thus, terms in the Taylor expansion are added in the search algorithm from lower orders to higher ones. This is reasonable because in optimization of real-world problems, we try to account for at first the most important factors and then try to include those with minor influence.

3. SEARCH EFFICIENCY OF SMALL EAS

As we mentioned in the Introduction, the derandomized evolution strategy with covariance matrix adaptation (CMA-ES) proposed in [3] was designed for small populations. It has shown to be efficient on a large number of unimodal optimization problems, particularly on ill-conditioned and non-separable problems [4]. In the (μ, λ)-CMA-ES without recombination, the λ offspring of generation $g+1$ is generated as follows:

$$\begin{aligned} \mathbf{x}_k^{(g+1)} &= \mathbf{x}_j^{(g)} + \sigma^{(g)}\mathbf{B}^{(g)}\mathbf{D}^{g)}\mathbf{z}_k^{(g+1)}, \\ &\quad j = 1, \cdots, \mu;\ k = 1, \cdots, \lambda, \end{aligned} \tag{6}$$

where k is randomly chosen from the μ selected parents, $\mathbf{z}$ is an n-dimensional (n is the search dimension) vector of normally distributed random numbers with expectation zero and identity covariance matrix, $\mathbf{BD}\left(\mathbf{BD}\right)^T = \mathbf{C}$ is the covariance matrix. During the evolution, the covariance matrix is updated as follows:

$$\mathbf{C}^{(g+1)} = (1 - c_{\text{cov}})\mathbf{C}^{(g)} + c_{\text{cov}}\mathbf{p}_{\text{C}}^{(\mathbf{g+1})}\left(\mathbf{p}_{\text{C}}^{(\mathbf{g+1})}\right)^T, \tag{7}$$

where $\mathbf{p}_{\text{C}}^{(\mathbf{g+1})}$ is known as the evolution path calculated by:

$$\mathbf{p}_{\text{C}}^{(g+1)} = (1 - c_{\text{C}})\mathbf{p}_{\text{C}}^{(g)} + \sqrt{c_{\text{C}} \cdot (2 - c_{\text{C}})}\mathbf{B}^{(g)}\mathbf{D}^{(g)}\mathbf{z}_k^{(g)}. \tag{8}$$

The adaptation of the global step-size $\sigma^{(g+1)}$ is calculated by:

$$\sigma^{(g+1)} = \sigma^{(g)}\exp\left(\frac{1}{d_\sigma}\frac{||\mathbf{p}_\sigma^{(g)}|| - c\hat{h}i_n}{\hat{\chi}_n}\right), \tag{9}$$

where $\hat{\chi}_n$ is the expected length of a ($\mathbf{0}$, $\mathbf{I}$)-normally distributed random vector and can be approximated by $\sqrt{n}(1 - \frac{1}{4n} - \frac{1}{21n^2})$, d_σ is a damping coefficient, and $\mathbf{p}_\sigma^{(g+1)}$ is a "conjugate" evolution path:

$$\mathbf{p}_\sigma^{(g+1)} = (1 - c_\sigma)\mathbf{p}_\sigma^{(g)} + \sqrt{c_\sigma \cdot (2 - c_\sigma)}\mathbf{B}^{(g)}\mathbf{z}_k^{(g)}. \tag{10}$$

The default parameter setting suggested in [4] is as follows:

$$c_{\text{C}} = \frac{4}{n+4},\ c_{\text{cov}} = \frac{2}{(n+\sqrt{2})^2},\ c_\sigma = \frac{4}{n+4}, d_\sigma = c_\sigma^{-1} + 1. \tag{11}$$

In this study, a slightly modified variant of the algorithm presented in [3] has been adopted, where a separate covariance matrix is maintained for each parent individual. Though the CMA-ES is designed for small populations, recent studies have found that CMA-ESs with a large population can improve the search performance significantly [2, 1].

However, little work has been reported on what is the optimal search dimension for a CMA-ES with a small population size when the theoretic search dimension is very large or even infinite. In the following, we investigate the search performance of CMA-ES with small populations on the test

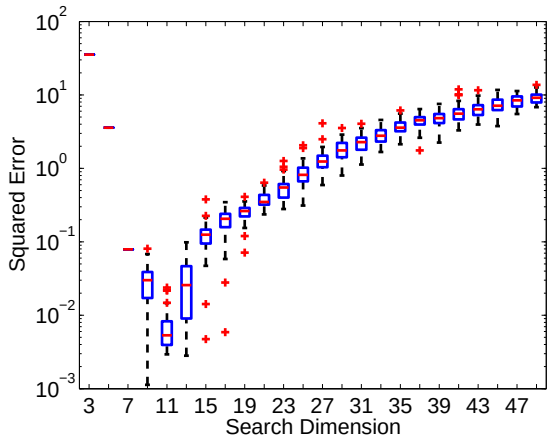


Figure 1: Search performance of the (1,4)-CMA-ES for search dimensions ranging from 3 to 49. Results averaged over 50 runs.

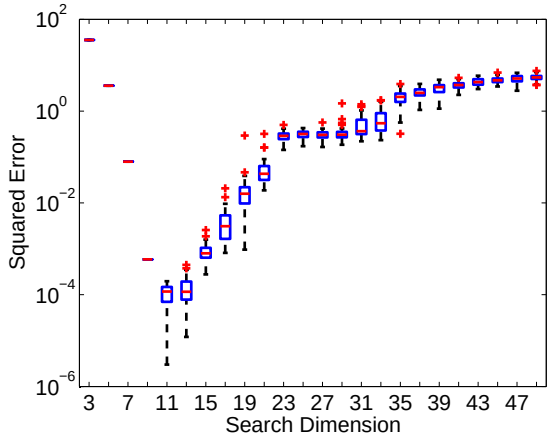


Figure 2: Search performance of the (2, 10)-CMA-ES for search dimensions ranging from 3 to 49. Results averaged over 50 runs.

problem described in Section 2. In our simulations, the CMA-ES without recombination has been adopted and a maximum of 2000 generations are run for search dimensions $5, 7, \cdots, 47, 49$. For each search dimension, the results are averaged over 50 independent runs. The results from a (1, 4)-CMA-ES and a (2, 10)-CMA-ES are presented in Figures 1 and 2, respectively.

From Fig. 1, we can see that the search performance of the (1, 4)-CMA-ES heavily depends on the search dimension. For a search dimension smaller than 7, the approximation error is quite large due to the limited number of free parameters. The minimal approximation error (0.002936) is achieved when the search dimension is 11, where the approximation error is mostly smaller than 0.01. When the search dimension further increases, the search performance degrades seriously due to the limited search capacity of the (1,4)-CMA-ES.

Similar simulations are carried out for the (2, 10)-CMA-ES. The minimal approximation error (0.000003) is achieved when the search dimension is 11. This implies that the (1, 4)-CMA-ES failed to locate the global optimum for an eleven-dimensional optimization problem in 50 runs. Even the (2, 10)-CMA-ES is able to locate the best found solution only once in the 50 runs. These results indicate that the search efficiency of CMA-ES with small populations is limited even for a relatively low dimensional problem. Meanwhile, as in the (1, 4)-CMA-ES case, the search performance becomes worse when the search dimension increases, though not as serious as the (1, 4)-CMA-ES. Again, there is an optimal search dimension where the (2, 10)-CMA-ES achieves the best performance and the search performance is acceptable when the search dimension is from 9 up to 17 (approximation error smaller than 0.01).

4. ADAPTATION OF SEARCH DIMENSION

It can be seen from the results in the previous section that there is an optimal search dimension for a given population size that is able to achieve the minimal approximation error. The optimal search dimension is unknown beforehand and is presumably dependent on the population size and the problem at hand.

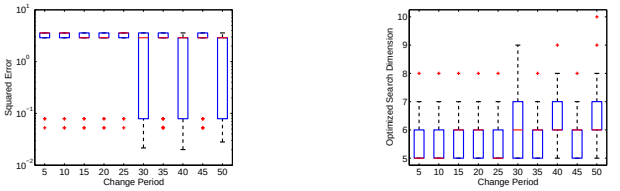
In this section, we suggest a simple approach to address this problem by adapting the search dimension during the optimization to find an approximately optimal search dimension for a given population size. The basic idea is to start the optimization from a relatively low search dimension and let the search dimension increase in every k generations during the optimization. k is called *change period*. To determine whether an increase in search dimension is beneficial, we compare the best fitness values before and after dimension increase. Assume the best (minimal in this work) fitness values before and after an increase in search dimension are $PBest$ and $CBest$, respectively. Note that $CBest$ is the best fitness value after k generations with an increased search dimension. The increase in search dimension is considered to be beneficial if $CBest$ is smaller than $Pbest$ for minimization problems. If an increase in dimension is regarded as beneficial, then the search dimension will be further increased by one. Otherwise, the search dimension will be decreased by one and fixed until the end of the optimization.

To implement the above idea, the change period k needs to be determined. We conduct simulations to investigate the influence of this parameter on the adaptation performance. Another parameter to be determined is the initial search dimension. This parameter should depend on the problem at hand. In our simulations, the initial dimension is set to 5.

When the search dimension is increased, we have the following three alternatives:

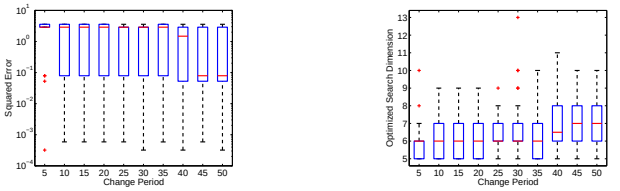
- Re-initialize all design parameters randomly;
- Inherit the value for existing design parameters and initialize new design parameter randomly;
- Inherit the value for existing design parameters and set the new parameter to zero, so that the fitness function does not change after the inclusion of the new dimension.

We test the performance of the suggested algorithm for 10 change periods, i.e., $k = 5, 10, 15, 20, 25, 30, 35, 40, 45, 50$.



(a) (b)

Figure 3: Adaptation of search dimension for (1, 4)-CMA-ES with various change periods. The design parameters are randomly re-initialized during dimension changes. (a) The best fitness value, and (b) the optimized search dimension. Results averaged over 50 runs.



(a) (b)

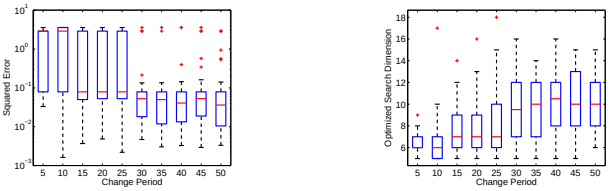
Figure 4: Adaptation of search dimension for (2, 10)-CMA-ES with various change periods. The design parameters are randomly re-initialized during dimension changes. (a) The best fitness value, and (b) the optimized search dimension. Results averaged over 50 runs.

The results where all design parameters are randomly initialized are presented in Figures 7 and 8, respectively. Again, 50 runs are conducted for each k.

From Fig. 3 and Fig. 4, we see that the neither the (1, 4)-CMA-ES nor the (2, 10)-CMA-ES shows acceptable performance. The search dimension is largely underestimated for all tested change periods. A much larger change period is not practical, since an overly large period will unfavorably increase the needed computational time. Thus, we conclude that randomly re-initialize the design parameters is undesirable in adopting an adaptive search dimension.

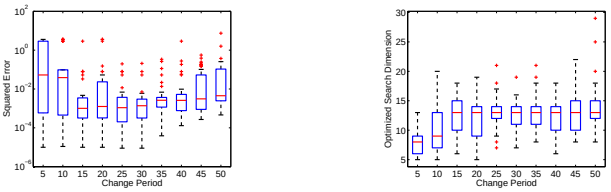
The next idea to try out is to inherit the value for each existing design parameter and then initialize the newly added design variable randomly. The simulation results are shown in Figures 5 and 6, respectively. We notice that the performance has been improved significantly. For the (1, 4)-CMA-ES, the performance is quite good when the change period is between 30 and 50, where the optimized search dimension is between 9 and 11 on average, which are optimal or sub-optimal if we refer to the empirical results shown in Fig. 1. Similar conclusion can be made to the results obtained for the (2, 10)-CMA-ES. However, the performance of the algorithm seems more robust against the change period in that satisfying performance has been achieved when the change period varies from 15 to 50, where the estimated optimal search dimension is 13 on average, which is one of the optimal search dimension as shown in Fig. 2.

Finally, we investigate the performance of the algorithm when we initialize the newly added design parameter to 0, which in this example makes the inclusion of the new search



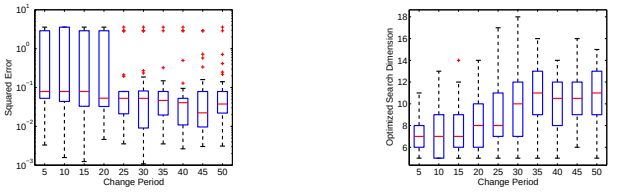
(a) (b)

Figure 5: Adaptation of search dimension for (1, 4)-CMA-ES with various change periods. The value of the existing design parameters are inherited and the new one is randomly initialized during dimension change. (a) The best fitness value, and (b) the optimized search dimension. Results averaged over 50 runs.



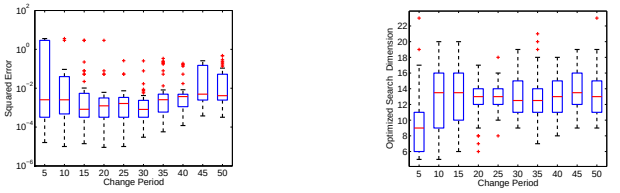
(a) (b)

Figure 6: Adaptation of search dimension for (2, 10)-CMA-ES with various change periods. The value of the existing design parameters are inherited and the new one is randomly initialized during dimension changes. (a) The best fitness value, and (b) the optimized search dimension. Results averaged over 50 runs.



(a) (b)

Figure 7: Adaptation of search dimension for (1, 4)-CMA-ES with various change periods. The value of the existing design parameters are inherited and the new one is set to zero during dimension changes. (a) The best fitness value, and (b) the optimized search dimension. Results averaged over 50 runs.



(a) (b)

Figure 8: Adaptation of search dimension for (2, 10)-CMA-ES with various change periods. The value of the existing design parameters are inherited and the new one is set to zero during dimension changes. (a) The best fitness value, and (b) the optimized search dimension. Results averaged over 50 runs.

dimension neutral to the fitness value. Such neutral mutations have shown to be essential to the success of adaptive coding when splines are used for geometry description in design optimization [7]. Comparing the results in Fig. 5 and those in Fig. 7 regarding the (1, 4)-CMA-ES, we see that minor improvements have been achieved, particularly when the change period is small.

5. CONCLUSIONS

Evolutionary optimization of large problems with evolutionary algorithms with a small population is a challenging topic. To efficiently optimize possibly infinite large problems using small populations, a method to adapt the search dimension has been suggested in this paper. The basic idea is that for small populations, we should start from a relatively low search dimension and then increase it gradually during the optimization. The increase in search dimension should continue until performance improvement cannot be achieved in a number of generations after the dimension increase. In this case, the search dimension is decreased by one and and kept constant till the end of the optimization. From our empirical studies, the change period should be between 20 to 50 generations. A too small change period is not desirable because the algorithm needs some time to find the potential improvement after an increase in dimension. Neither is a large change period preferred because a larger change period tends to increase the computational time rapidly.

It is found essential for the success of our algorithm that the value of the existing design parameters should be inherited after an increase in search dimension. This result is consistent with the findings reported in the literature that *a priori* knowledge is beneficial in enhancing the performance of evolutionary algorithms [5].

The strategy parameters are randomly re-initialized during dimension changes in this work, which may not be optimal for evolution strategies. It will be one of our future work to investigate the influence of re-initialization of strategy parameters on the performance of our algorithm using a dynamic search dimension.

6. REFERENCES

[1] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In *Parallel Problem Solving from Nature*, volume 3242 of *LNCS*, pages 282–291. Springer, 2004.

[2] N. Hansen, S. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution startegy with covariance matrix adaptation (CMA-ES. *Evolutionary Computation*, 11:1–18, 2003.

[3] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *IEEE Conference on Evolutionary Computation*, pages 312–317, 1996.

[4] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[5] Y. Jin, editor. *Knowledge Incorporation in Evolutionary Computation*. Springer, Berlin Heidelberg, 2005.

[6] Y. Jin, M. Olhofer, and B. Sendhoff. A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation*, 6(5):481–494, 2002.

[7] M. Olhofer, Y. Jin, and B. Sendhoff. Adaptive encoding for aerodynamic shape optimization using evolution strategies. In *Congress on Evolutionary Computation*, volume 1, pages 576–583, 2001.