Learning and Anticipation in Online Dynamic Optimization

Peter A.N. Bosman

Centre for Mathematics and Computer Science (CWI), P.O. Box 94079, 1090 GB Amsterdam, The Netherlands, Peter.Bosman@cwi.nl

Summary. In this chapter we focus on the importance of the use of learning and anticipation in (online) dynamic optimization. To this end we point out an important source of problem–difficulty 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 (i.e. anticipation) by learning from the past. We formalize this means in an algorithmic framework and indicate why evolutionary algorithms (EAs) are specifically of interest in this framework. We have performed experiments with two benchmark problems that feature time–linkage. The results show, as a proof of principle, that in the presence of time–linkage EAs based on this framework can obtain better results than classic EAs that do not predict the future.

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, 23, 25, 29]. 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, 27] 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 optimizing a certain score right now, to influence the optimal 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 for a specific customer will likely not result in repeated orders from that customer. Hence, the revenue of a company over time is determined by the current performance, but also by the impact the current way of routing has on future events.

The most important contribution made in this chapter is that we will show that dependencies between decisions over time requires their explicit processing during optimization to ensure the best performance in the long run. Any approach that does not explicitly process these dependencies and instead only solves the problem for the current time will never obtain an optimal result.

In this chapter we also present an algorithmic framework for solving dynamic optimization problems. The algorithmic framework is specifically equipped with the possibility of processing time–linkage. To this end, we propose the incorporation of learning (e.g. statistical [28] 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 [26]. However, the cited approach only predicts the future for a single discrete time step. As a result, the algorithm cannot process longer, arbitrarily sized, time–linkage intervals. Moreover, the approach was only tested on a problem that doesn't contain time–linkage. As a result, no significant difference was observed in using either a good predictor or a bad predictor. In this chapter, 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 chapter 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 chapter is organized as follows. In Section 2 we characterize online dynamic optimization problems. Next, in Section 3 we discuss solving these problems. In Section 4 we describe our algorithmic framework and in Section 5 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 6.

2 Defining online dynamic optimization problems

In order to formally define the class of online dynamic optimization problems, we first give a general definition of optimization problems. Without loss of generality we assume that the goal is maximization.

Definition 1 (Optimization problem). *An optimization problem is defined as follows:*

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

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

2.1 Static versus dynamic

An optimization problem can either be dynamic or not. If it is not dynamic, it is said to be static (instead of non–dynamic).

Definition 2 (Static optimization problem). *An optimization problem is said to be static if it cannot be written as a dynamic optimization problem.*

Definition 3 (Dynamic optimization problem). *An optimization problem is said to be dynamic if the optimization function has the specific form of a functional. Moreover, the function space to optimize over consists of functions of a single variable* $t \in \mathbb{T} = [0, t^{\text{end}}]$*,* $t^{\text{end}} > 0$*. Variable* t *is commonly referred to as time. An optimization problem is said to be dynamic if the optimization function and the constraint function can be written as below in Equation 2 and there are no equivalent definitions of* $\mathfrak{F}_{\boldsymbol{\gamma}}$ *and* $\mathfrak{C}_{\boldsymbol{\gamma}}$ *that do not involve the time variable.*

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

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

where $\mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t, Z(t,\boldsymbol{\zeta}))} : \mathbb{P}^{\text{dyn}} \to \mathbb{O}^{\text{dyn}}$ *is the dynamic optimization function,* $\mathbb{P}^{\text{dyn}}$ *is the dynamic parameter space,* $\mathbb{O}^{\text{dyn}} = \mathbb{O} = \mathbb{R}^{n_o}$ *is the* n_o*–dimensional dynamic objective space,* $\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}} : \mathbb{T} \to \mathbb{P}^{\text{dyn}}$ *is the dynamic variable function that for any time* $t \in \mathbb{T}$ *returns the settings for the variables of function* $\mathfrak{F}^{\text{dyn}}$*,* $\boldsymbol{\zeta} \in \mathbb{P}$ *are the parameters of function* $\boldsymbol{\zeta}^{\text{dyn}}$*,* $Z : \mathbb{T} \times \mathbb{P} \to$

$\{\mathbb{T} \times \mathbb{P}^{\text{dyn}}\}$ is a function that for any time $t \in \mathbb{T}$ and any setting $\boldsymbol{\zeta} \in \mathbb{P}$ of the parameters of function $\boldsymbol{\zeta}^{\text{dyn}}$ returns a set that contains the settings of the parameters of function $\boldsymbol{\zeta}^{\text{dyn}}$ for all times before t, i.e. $Z(t, \boldsymbol{\zeta}) = \bigcup_{0 \leq t' < t} \left\{ \left(t', \boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t') \right) \right\}$, $\boldsymbol{\gamma}^{\text{dyn}} : (\mathbb{T}, \{(\mathbb{T} \times \mathbb{P}^{\text{dyn}})\}) \to \mathbb{G}^{\text{dyn}}$ is a function that for any time $t \in \mathbb{T}$ and the dynamic variable function restricted to all earlier times $t' < t$ returns the problem–specific parameters, $\mathfrak{C}_{\boldsymbol{\gamma}^{\text{dyn}}(t, Z(t,\boldsymbol{\zeta}))} : \mathbb{P}^{\text{dyn}} \to \{\textit{feasible}, \textit{infeasible}\}$ is the dynamic constraint function and $t^{\text{end}} > 0$ is the horizon of the dynamic optimization problem. Moreover, we use the convention that $\int_a^b (f_0(x), \ldots, f_{n_o-1}(x))\, dx = \left(\int_a^b f_0(x)dx, \ldots, \int_a^b f_{n_o-1}(x)dx \right)$ Concordantly, the optimization problem to solve can now be written as:

$$\max_{\boldsymbol{\zeta} \in \mathbb{P}} \left\{ \int_0^{t^{\text{end}}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t, Z(t,\boldsymbol{\zeta}))} \left(\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t) \right) dt \right\} \tag{3}$$

$$\textbf{s.t. } \forall t \in [0, t^{\text{end}}] : \mathfrak{C}_{\boldsymbol{\gamma}^{\text{dyn}}(t, Z(t,\boldsymbol{\zeta}))} \left(\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t) \right) = \textit{feasible}$$

In the dynamic optimization problem as defined above it is assumed that time is continuous. However, this does not always have to be the case. If time is not continuous, the integral can be written as a discrete sum and the dynamic optimization problem is said to be discrete.

2.2 Offline versus online

A dynamic optimization problem can either be offline or not. If it is not offline it is said to be online (instead of non–offline).

Definition 4 (Offline dynamic optimization problem). *A dynamic optimization problem is said to be offline if the dynamic optimization function can be evaluated completely.*

From definition 4 we have that in offline dynamic optimization problems the optimal dynamic variable function can be found by constructing complete dynamic variable functions and subsequently evaluating their corresponding optimization values by integrating over all time. The advantage of solving offline dynamic optimization problems is that with the exception of a practical one, there is no time–limit in solving the problem. In a sense, this type of problem is very close to the traditional definition of optimization problems in that we have a problem that must be optimized and we can evaluate the optimization function. The only thing that we know additionally here is that the optimization problem has the specific form of a dynamic optimization problem.

Definition 5 (Online dynamic optimization problem). *A dynamic optimization problem is said to be online if the dynamic optimization function* *cannot be evaluated for all future times $t > t^{now}$ and the dynamic optimization problem must therefore be solved as time goes by.*

Online dynamic optimization is by far the most practical variant of dynamic optimization. It must continually be decided what values to use for the dynamic variables from one moment to the next. The only thing that can be evaluated is how well the algorithm has done so far, which we call the history function.

Definition 6 (History function). *The history function is defined as follows:*

$$\mathfrak{H}^{\mathrm{dyn}}(t^{\mathrm{now}}, \boldsymbol{\zeta}) = \int_{0}^{t^{\mathrm{now}}} \mathfrak{F}^{\mathrm{dyn}}_{\boldsymbol{\gamma}^{\mathrm{dyn}}(t, Z(t,\boldsymbol{\zeta}))} \left(\boldsymbol{\zeta}^{\mathrm{dyn}}_{\boldsymbol{\zeta}}(t) \right) dt \tag{4}$$

Like many other optimization problems, online dynamic optimization problems can be constructed artificially. Since the problem is then fully known, there is in design no difference with the offline case. The difference only resides in the fact that the problem is *treated* as being online. Online optimization problems are however typically real–world problems in which the problem–specific parameters actually change over time in an unknown fashion.

3 Solving online dynamic optimization problems

First we identify two important and commonly distinguished sources of problem–difficulty in dynamic optimization problems in Section 3.1. In Section 3.2 we then discuss solving online dynamic optimization problems by only taking into account the current situation. Finally, in Section 3.3 we describe the advantages of solving online dynamic optimization problems by also taking into account future implications of decisions.

3.1 System influence and control influence

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

Definition 7 (System influence). *System influence 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 $\boldsymbol{\gamma}^{\mathrm{dyn}}(t)$ are a function of time.*

Definition 8 (Control influence). *Control influence is the response of the dynamic system at time t^{now} to the choices for the problem variables made in the past by the problem solver, i.e. the trajectory $\boldsymbol{\zeta}^{\mathrm{dyn}}_{\boldsymbol{\zeta}}(t)$ with $t \in [0, t^{\mathrm{now}})$.*

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

3.2 Optimizing the present: falling victim to time–deception

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 the optimization of 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 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.

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 due to the presence of time–linkage.

Definition 9 (Time–linkage). *A dynamic optimization problem is said to contain time–linkage if and only if there exists at least one time* $0 \leq t \leq t^{\text{end}}$ *for which the dynamic optimization value at time* t *is dependent on at least one earlier solution* $\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t')$, $0 \leq t' < t$.

As an example, consider the following unconstrained l–dimensional dynamic optimization problem; a simple adaptation of the sphere problem that shifts with time:

$$\max_{\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t)} \left\{ \int_{0}^{t^{\text{end}}} \varphi\left(\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t), t\right) dt \right\} \tag{5}$$

where

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

Now, when optimizing only the present in an online setting, a value for $\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^{\text{now}})$ is chosen by maximizing $\varphi(\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}, t^{\text{now}})$. For any t, $\varphi(\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}, t)$ is just a hyperparabola with a unique maximum for $\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t)_i = t$. For $0 \leq t < 1$ the associated optimization value is 0 and for $t \geq 1$ this value is $-\sum_{i=0}^{l-1}\psi\left(\left|\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t-1)_i\right|\right)$. 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. However, if the simple choice of $\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t)_i = 0$ is always made, 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 $\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t)_i = 0$ is chosen. Although a better result can still be obtained because $\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t)_i = 0$ is not the optimal solution, the penalty of disregarding time–linkage is only small. But if $\psi(\cdot)$ is a higher–order increasing function, such as $\psi(x) = e^x - 1$, a (much) worse optimization value will be obtained and the price to pay for not taking into account time–linkage is (much) higher. A graphical illustration of the difference in obtained optimization values for the case of $l = 1$ 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 the problem provably does not contain time–linkage. Otherwise, the problem is time–deceptive for this approach to solving online dynamic optimization problems.

Definition 10 (Time–deception). *A dynamic optimization problem is said to be time–deceptive for an optimizer if the problem contains time–linkage and the optimizer has no means to efficiently take this time–linkage into account during optimization and therefore cannot find the optimal solution trajectory.*

3.3 Optimizing the present and the future: learning to avoid time–deception

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.

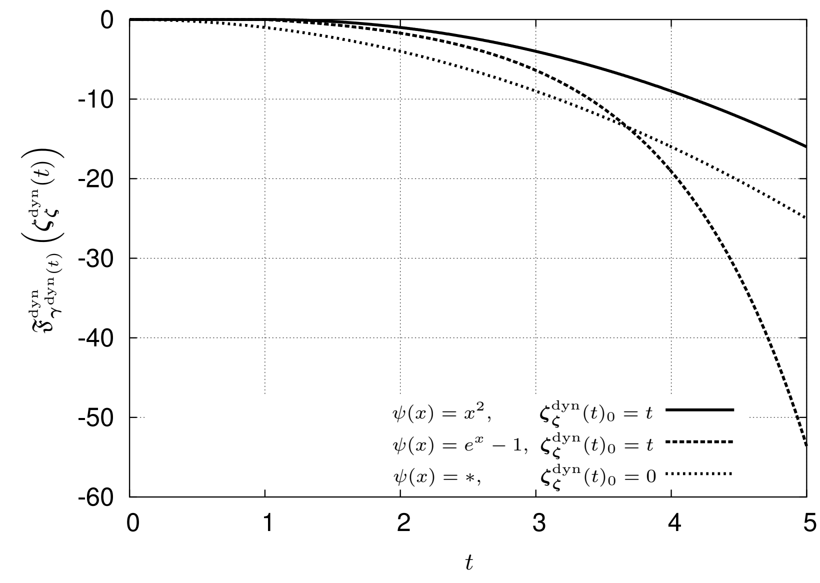

Fig. 1. Illustration of the optimization values obtained for different variable trajectories and different forms of $\psi(\cdot)$ in the case of $l = 1$ and $t^{\text{end}} = 5$. $\psi(x) = *$ is any function such that $\psi(0) = 0$.

The better the prediction, the closer the algorithm can get to optimality. The available information to base the prediction upon besides problem–specific information is information that was collected in the past.

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

$$\max_{\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t)} \left\{ \int_{t^{\text{now}}}^{\min\{t^{\text{now}}+t^{\text{plen}},t^{\text{end}}\}} \hat{\mathfrak{F}}^{\text{dyn}}_{\boldsymbol{\alpha}}\left(t, \boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t)\right) dt \right\} \tag{6}$$

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

where

$$\begin{cases} \hat{\mathfrak{F}}^{\text{dyn}}_{\boldsymbol{\alpha}}\left(t^{\text{now}}, \boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t^{\text{now}})\right) = \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t^{\text{now}}, Z(t^{\text{now}}, \boldsymbol{\zeta}))}\left(\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t^{\text{now}})\right) \\ \hat{\mathfrak{C}}^{\text{dyn}}_{\boldsymbol{\alpha}}\left(t^{\text{now}}, \boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t^{\text{now}})\right) = \mathfrak{C}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t^{\text{now}}, Z(t^{\text{now}}, \boldsymbol{\zeta}))}\left(\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t^{\text{now}})\right) \end{cases}$$

Prediction in the complete BBO case

The complete BBO (Black–Box Optimization) case is the most general case. No prior knowledge is assumed on the problem to be solved 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 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:

$$\bigcup_{i=0}^{n_{\text{data}}-1}\left\{\left(\left(t^i, \boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^i), Z(t^i,\boldsymbol{\zeta})\right), \boldsymbol{y}^i\right)\right\} \tag{7}$$

where on the input–side of the pattern we have the time t^i, $0 < t^i \leq t^{\text{now}}$, the value of the variables $\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^i)$ at time t^i and the history of the variable–value–trajectory $Z(t^i,\boldsymbol{\zeta})$ up to time t^i and on the output–side of the pattern we have the value of the dynamic optimization function $\boldsymbol{y}^i$ for solution $\boldsymbol{\zeta}^{\text{dyn},i}$ 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. Note that the integration of the history of the solution–trajectory into the data set is essential for the processing of time–linkage.

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 time span that contains the data to learn from. In the single–objective case the learning problem can be formalized as follows:

$$\min_{\boldsymbol{\alpha}\in\mathbb{A}}\left\{\int_{t^{\min}}^{t^{\max}}\left(\mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t,Z(t,\boldsymbol{\zeta}))}\left(\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t)\right) - \hat{\mathfrak{F}}^{\text{dyn}}_{\boldsymbol{\alpha}}\left(\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t)\right)\right)^2 dt\right\} \tag{8}$$

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 approximated optimization function.

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 be estimated, the better the hope is of obtaining good approximations.

Prediction length, prediction base and history length

In addition to the choice of approximation class and learning mechanism, there are two key issues of importance in using predictions in dynamic optimization:

1. **How far into the future should predictions be made?**
   (we call this prediction length, denoted t^{plen})
2. **From how far into the past should information be used to base the prediction upon?**
   This question is actually twofold:
   a) **How far into the past should t^i go in Equation 7?**
      (we call this prediction base, denoted t^{pbase})
   b) **For each pattern, how far into the past, starting from the time of that pattern, should we take into account the history of the variable trajectory?**
      (we call this history length, denoted t^{hlen})

A proper choice for the prediction length and the history length depends on the time–linkage time span, 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 linkage 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.

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.

4 An algorithmic framework

4.1 Components

Solver

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

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. When called upon, the predictor returns either the predicted function value directly or predicted values for parameters.

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. Otherwise, the dynamic optimization function is used to compute the optimization value that pertains to the current time and the predictor is used to predict future optimization values. The trajectory of the variables in the predicted future is to be set by the solver. To specify this trajectory a dynamic variable function is needed that can supply a solution for each possible time between t^{now} and $\min\{t^{\text{now}} + t^{\text{plen}}, t^{\text{end}}\}$. It is computationally 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. If the dynamic optimization function is not stochastic, a list of solutions can be used such that there is one solution for each sub–interval. If the dynamic optimization function is stochastic however, a predefined future list of actions may not be optimal. The reason for this is that different actions may be optimal in different situations. Because the future is stochastic, the actual future situation is unknown. Hence, a responsive strategy is then required instead to be able to obtain optimality. The solution to optimize by the solver then is thus not a list of assignments to the problem variables, but a strategy in the form of a function of time that returns a solution conditioned on the actual future situation. Note that the list of assignments in the non–stochastic case is a specific form of the strategy.

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 time stamp t such that $t^{\text{now}} - t^{\text{pbase}} \leq t \leq t^{\text{now}}$.

Timer

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

4.2 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 computation 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. 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 given by the scenario in which the solver sends a signal when a certain number of generations have passed and then awaits completion of the learning task before continuing.

4.3 Definition

To complete the framework, 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. Pseudo–code for the framework is given in Figure 2.

An important 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

```
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($t^{\text{next}}$)
  while $t^{\text{now}} \leq t^{\text{end}}$
10 $S$.STOP()
11 $P$.STOP()
12 return($\boldsymbol{Z}$)
```

Fig. 2. Pseudo–code for the algorithmic framework.

for a solution at time t^{now} because the predictor is used to evaluate all parts of the trajectory that pertain to future times. Pseudo–code for this specific part of the framework is given in Figure 3.

```
$F$.EVALUATE($\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}$)
1 $t^{\text{now}} \leftarrow T$.GETTIME()
2 $\boldsymbol{y} \leftarrow t^{\text{pint}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t^{\text{now}}, Z(t^{\text{now}}, \boldsymbol{\zeta}))}(\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^{\text{now}}))$
3 if COMPLETEBBOCASE() then
  3.1 $D$.ADDPATTERN(((\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^{\text{now}}), 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($\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}, t^{\text{now}} + i \cdot t^{\text{pint}}$)
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($\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}, t^{\text{now}} + i \cdot t^{\text{pint}}$)
      4.1.2 $\boldsymbol{y} \leftarrow \boldsymbol{y} + t^{\text{pint}} \mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{predicted}}}(\boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^{\text{now}} + i \cdot t^{\text{pint}}))$
5 return($\boldsymbol{y}$)
```

Fig. 3. Pseudo–code for the evaluation of future trajectories.

5 Experiments

5.1 EA

The optimization problems that we perform experiments with are real–valued, but at any point in time not very daunting because the most important thing we focus on in this chapter 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 chapter 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.

5.2 BBO: Time–linkage numerical problem

The problem

We first investigate the real–valued time–linkage problem introduced in Section 3.2, Equation 5. We regard two variants by setting $\psi(x) = x^2$ and $\psi(x) = e^x - 1$. Moreover, we have used a dimensionality of $l = 1$.

Instantiating the framework

In this problem the goal of the predictor is to predict the value of the optimization function directly. For clarity we point out that the predicted functions are estimated from a set of patterns with timestamps at most t^{pbase} time ago. The patterns in the data set contain the actually chosen trajectory in the past up to time $t^i - t^{\text{hlen}}$ in steps of t^{hint}, i.e. if $t^{\text{hlen}} = t^{\text{hint}} = 1$, a pattern is given by $((t^i, \boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^i), \boldsymbol{\zeta}_{\boldsymbol{\zeta}}^{\text{dyn}}(t^i - 1)), \boldsymbol{y}^i)$. We used three different predictor instances.

1. **Optimal**
   Returns the true value of the dynamic optimization function.

2. **Linear estimator**
   Returns the value of a linear function that was estimated using the least–squares technique.
3. **Quadratic estimator**
   Returns the value of a quadratic function that was estimated using the least–squares technique.

For the case of $\psi(x) = x^2$ only the function class used by the quadratic estimator contains the target function. Hence, effective future predictions are possible with proper estimations in this case, which should ensure the prevention of 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 than the linear estimator.

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, corresponding to the traditional approach of not looking into the future, or to the optimal value of 1. Moreover, we set $t^{\text{hlen}} = t^{\text{hint}} = t^{\text{pbase}} = t^{\text{pint}} = t^{\text{plen}}$.

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 time step of 0.001 every generation. Since the database contains all patterns over a time span of length 1 and the time steps 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. 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–linkage numerical problem are shown in Figures 4 and 5 respectively. The overall results (i.e. the integral over $t \in [0, 10]$) are 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–linkage 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–linkage numerical problem, neither the linear nor the quadratic estimator have a function class that contains the target exponential function. However, for the time–linkage in this problem that

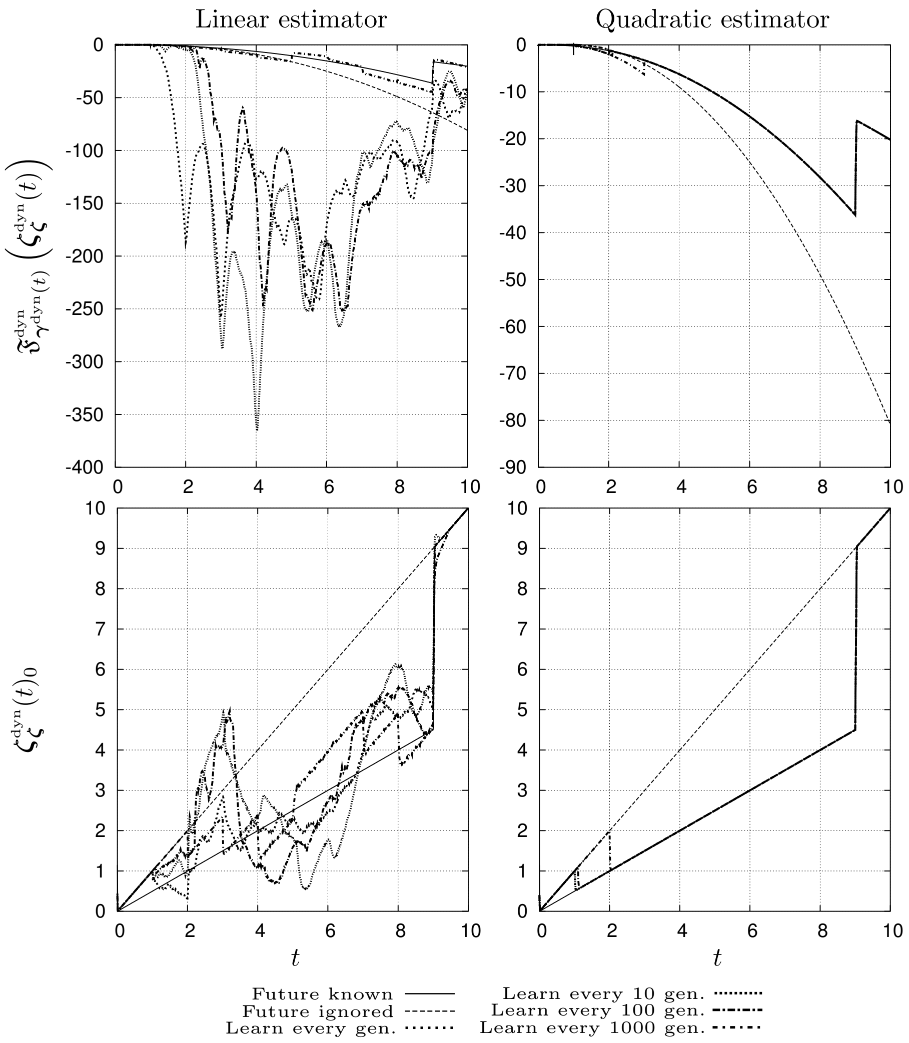

Fig. 4. Results averaged over 100 runs on the time–linkage numerical problem with $\psi(x) = x^2$.

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 5. 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.

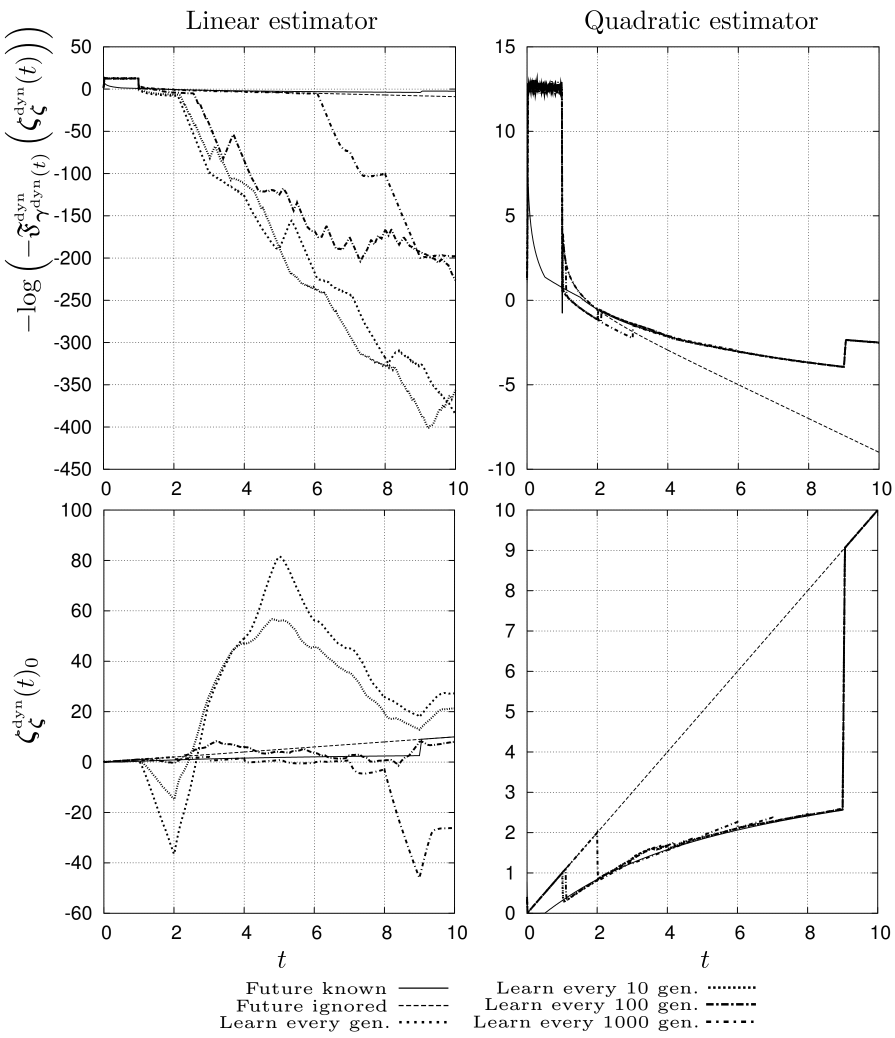

Fig. 5. Results averaged over 100 runs on the time–linkage numerical problem with $\psi(x) = e^x - 1$.

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

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

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

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.

5.3 Partial BBO: dynamic pick–up problem

The problem

The second problem that we investigate is a discrete partial BBO problem. Although it is based on a 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 time steps. The intuitive description is that at time step 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 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 time steps are of size 1. A solution at time t now is a tuple $\boldsymbol{\zeta}^{\text{dyn}}_{\boldsymbol{\zeta}}(t) = (b(t), \boldsymbol{x}^{\text{alternative}}(t))$ 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:

$$\mathfrak{F}^{\text{dyn}}_{\boldsymbol{\gamma}^{\text{dyn}}(t)}\left(\left(b(t), \boldsymbol{x}^{\text{alternative}}(t)\right)\right) = \begin{cases} 1 - \| \boldsymbol{x}^{\text{package}}(t) - \boldsymbol{x}^{\text{truck}}(t) \| & \text{if } b(t) = 1 \\ 0 - \| \boldsymbol{x}^{\text{alternative}}(t) - \boldsymbol{x}^{\text{truck}}(t) \| & \text{otherwise} \end{cases} \tag{9}$$

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. $\boldsymbol{x}^{\text{package}}(t) \sim \prod_{i=0}^{l-1} \mathcal{N}(0,1)$.

Instantiating the framework

A simple strategy is given by a hillclimber. The decision taken at each time step 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 extremely large trajectories to optimize. This drastically increases the resources required by the EA to solve the problem. Second, since the future is stochastic in this problem, a proper estimation of the expected future profits requires averaging evaluations 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.

Because the problem is stochastic, we require the solution in the solver component to be a dynamic variable function. For the problem at hand this means that we need to evolve a decision strategy for where to move the truck, given a certain (predicted) situation. The strategy we choose here is a simple one. For the current situation, a decision is directly subject to evolution. For future, predicted, situations up to a time span of t^{plen}, the hillclimber strategy is used. Hence, the EA only provides a solution for the current time. Certainly, better results may be obtained by allowing the EA to evolve a more elaborate strategy. However, using the hillclimber already gives an impression of the quality of a certain starting point. This information, albeit an approximation of what can truly be achieved, can therefore still give additional insights into the quality of a decision for the current situation, i.e. where to move the truck to right now. Since the dynamic optimization function is stochastic, we point out again that multiple calls are required to estimate the expected future payoff even when using the hillclimber to evaluate the future. 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].

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 time step of 1 every 50 generations. Since only one pattern was added to the data set each time step, 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 scores (i.e. the integral of the dynamic optimization function over $[0, 100]$) are shown in Figure 6 as a function 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.

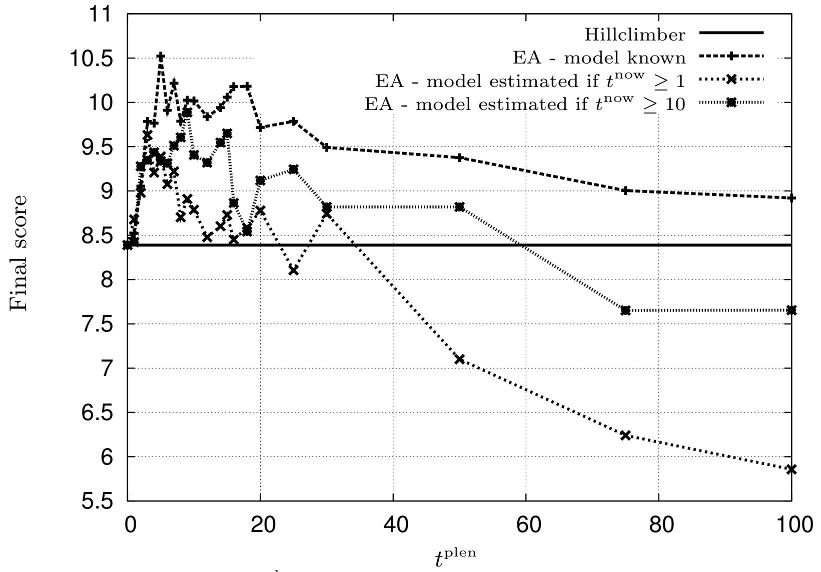

Fig. 6. Final score (i.e. $\int_0^{t^{\text{end}}} \mathfrak{F}^{\text{dyn}}_{\gamma^{\text{dyn}}(t,Z(t,\zeta))}(\zeta^{\text{dyn}}_{\zeta}(t)))$ averaged over 100 runs on the dynamic pick–up problem as a function of the prediction length.

The trajectory of the cumulative fitness for the best value and maximum value of t^{plen} are shown in Figure 7. This figure also reveals why the use of information about the future leads to 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 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 doing so means making a profitable pick–up.

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

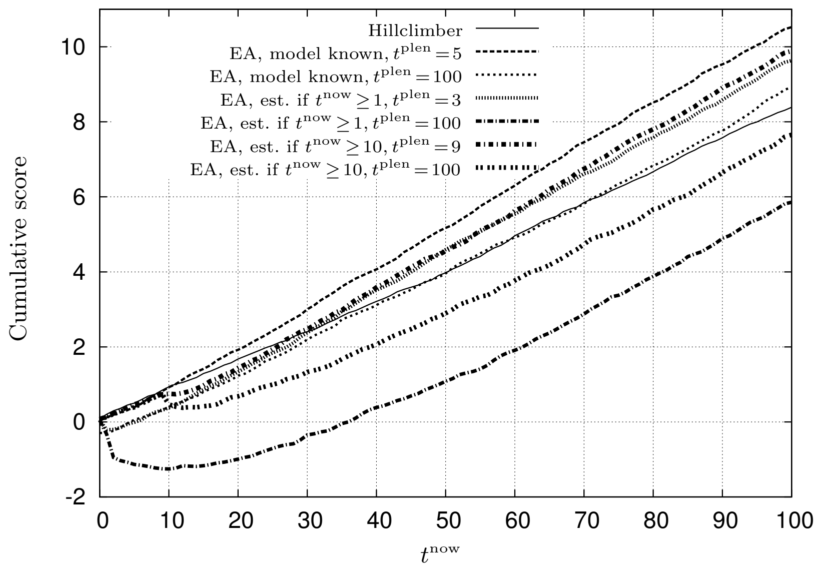

Fig. 7. Cumulative score (i.e. $\mathfrak{H}^{\text{dyn}}(t^{\text{now}}, \zeta)$) averaged over 100 runs on the dynamic pick–up problem as a function of time.

6 Discussion and conclusions

In this chapter 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 time span as well as the history data time span. An interesting question is whether the time spans 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, 24].

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, these predictions are hardly of any use 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.

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 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 Learning.* 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. 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.
24. D. Thierens. Scalability problems of simple genetic algorithms. *Evolutionary computation*, 7:331–352, 1999.
25. 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.
26. 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.
27. 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.
28. V. Vapnik. *Statistical learning theory.* Wiley, New York, New York, 1998.
29. 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.