Near-optimal Online Control of Dynamic Discrete-event Systems

Lenko Grigorov (lenko.grigorov@banica.org)
School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada

Karen Rudie (karen.rudie@queensu.ca)
Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario K7L 3N6, Canada

Abstract. A class of time-varying discrete-event systems, named dynamic discrete-event systems, is defined. The goal of this paper is to provide a method which is modular and can be applied in real solutions for the optimization of the online control of such systems. First, a simple control algorithm is presented, followed by illustrative examples of different issues that can arise if it is used. Afterward, a more complicated near-optimal online control algorithm with normalization of string values is proposed. The time variability of the systems is accounted for and the average computational time is drastically reduced. This is demonstrated with a set of simulations of the performance of the new algorithm.

1. Introduction

Discrete-event systems can be used as a model for many discrete processes or as convenient estimations of the function of real systems. An important research topic is the control of discrete-event systems (DESs). The work of Ramadge and Wonham (Ramadge and Wonham, 1987) introduced a paradigm for the study of the control of such systems. A *supervisor* of a DES is designed so that it ensures desired behavior of the system.

This approach to control proved to be very convenient for the purpose of research and resulted in many extensions. However, the construction of a supervisor requires complete knowledge of the controlled system and thus, it turns out, it is not applicable in many real-world situations. If the controlled system is very large or if it is time-varying, it might not be feasible to construct a complete supervisor.

The work of Chung, Lafortune and Lin (Chung et al., 1992) introduced a novel approach to the control of DESs: limited-lookahead online control. Instead of constructing a supervisor for the complete system, a *controller* is used which monitors and directs the system on the fly. After the occurrence of an event, the controller uses a *lookahead window* of the possible future behavior of the system to make a control decision. The lookahead window consists of a tree of strings of events that the system may execute and its depth is limited. The online method of control is suitable for large or time-varying systems, since making a control decision at each step is computationally less costly than creating a full-scale off-line supervisor.

Researchers have worked on various approaches to online control (Chung et al., 1994; Hadj-Alouane et al., 1994; Heymann and Lin, 1994; Hadj-Alouane et al., 1996; Kumar et al., 1998). These works propose different methods for how to produce valid control, including the use of online control in systems with partial observability. However, little attention has been devoted to the study of the application of online control to time-varying systems. Uncertainty in DESs has been addressed either by modifying the basic model by adding probabilities to the event transitions, (Kumar and Garg, 2001), or by modeling all possible variations of the system, (Lin, 1993; Young and Garg, 1995). None of these works explores the subject of using supervisory control for DESs which cannot be modeled *a priori* due to the lack of information on the possible system variations. Here we explore this topic. We propose the definition of a class of time-varying, dynamic discrete-event systems (DDESs), which consist of modules and which vary in that modules can appear or disappear with time. While each single module is well-defined and known once it becomes a part of the system, it is unknown what modules the system will consist of as it evolves in the future. Other ways to model how non-static DESs change have been examined before, however, they are limited

to a very small subset of possible perturbations of a DES. In (Chen et al., 1997), it is assumed that events can be refined as time advances. In (Gordon and Kiriakidis, 2000), systems are modeled as interconnected modules and a method to cope with the failure of modules is proposed. In our work, we investigate a much more general case when the system may evolve arbitrarily, within a modular framework. Examples of highly varying systems which can be modeled as DDESs are resource management systems such a dispatching centers.

Another topic of great interest is the improvement of the performance of a system under control. Optimization of supervisory control of DESs is well studied, (Brave and Heymann, 1993; Kumar and Garg, 1995; Sengupta and Lafortune, 1998; Mohanty et al., 1999), however, much less research has been done on the optimization of online control. In (Chen and Lin, 2001), Chen and Lin present a method which can be used to construct an optimal online controller which maximizes the performance of the system while minimizing the associated cost. While this and similar methods are immediately applicable to DDESs, there are some issues related to DDESs which arise from their time-varying nature and which are not considered in these works.

Dynamic discrete-event systems are time-varying and thus the following have to be taken into account:

- Using marked languages for such systems rarely makes sense: due to the limited lookahead window and due to the dynamics of the system the controller may frequently get into a position when the system cannot be guided to execute a marked string. However, it is not desirable to immediately terminate control with an error message, because in the next time period a missing module could suddenly appear and the controller might be able to lead the system to a desired state. Furthermore, if limited-lookahead control is employed, it is incorrect to denote the prefixes of legal strings as illegal, since the controller must not prevent their occurrence if a legal string is ever to be generated. Instead of using marked languages, a desirability measure on strings can be used to indicate what behavior of the system is most beneficial. Such a measure may assign more than two values, making it possible to distinguish between illegal strings, legal strings, and prefixes of legal strings.

- When static systems are taken into account, the greater the depth of the lookahead tree, the better are the predictions of the controller and the control is closer to optimal. However, when time-varying systems are considered, a large depth can lead to over-specialization of the prediction and the resulting control can become unrealistic.

- Control optimization can be either greedy or rely on long-term planning. For static systems the latter gives best results. However, for dynamic systems the greedy approach may offer better performance.

The contributions of this paper are as follows:

- a definition of a class of time-varying discrete-event systems,

- a discussion of the issues that arise when a naive optimal control algorithm is applied to such systems,

- the specification of a list of desirable properties that an optimizing control algorithm in such settings should have,

- the proposal of a framework for the modeling of the dynamic properties of these time-varying systems,

- the design of an algorithm which utilizes the model to provide near-optimal control of such systems and

– the presentation of performance results for the proposed algorithm.

The rest of the paper is organized as follows. The preliminaries are in Section 2. We present a simple optimal control algorithm for DDESs in Section 3 and in Section 4 we illustrate issues which arise with its use. Some of the properties of DDESs are discussed in Section 5. A new near-optimal control algorithm which uses normalization for risk is proposed in Section 6. The correctness and the complexity of this algorithm are discussed in Sections 7 and 8, respectively. In Section 9 we present the results we obtained through simulation of the performance of the new algorithm. The concluding remarks are in Section 10.

A preliminary, shorter version of this paper appeared in (Grigorov and Rudie, 2005).

2. Preliminaries

Discrete-event systems are systems where different events may occur. The occurrence of an event can be thought of as a transition between states of the system; thus, often directed graphs are used for the visualization of DESs. Event occurrences create sequences, or strings, of events. The set of all possible events in a system is denoted by Σ, and the set of all possible strings of events from Σ is denoted by Σ^*. Single events are usually denoted by σ_i, and elements of Σ^* are usually denoted by the lowercase letters s, t, $u\ldots$ The empty string (with zero events) is denoted ϵ. Each DES can generate a certain set of such strings, defined by the system behavior. Some of these strings might not be desirable (i.e., the system executes unwanted procedures) and thus a subset of the system behavior can be defined as legal (desirable sequences) and the rest of the sequences will be treated as illegal, or undesirable. The unrestricted set of event strings for the system G is denoted by $L(G)$, and the subset of legal sequences as $K \subseteq L(G)$. The prefix-closure of a language L is denoted by $\overline{L}$ and it contains all prefixes of all strings in L. By definition $L(G) = \overline{L(G)}$. In this paper we also assume that $K = \overline{K}$.

Supervision of DESs consists of ensuring that only legal sequences occur in the system, i.e., that whatever s is executed, $s \in K$. Events may be controllable or uncontrollable and the set Σ is partitioned in two (denoted by Σ_c and Σ_{uc}, respectively). The controllable events can be enabled or disabled during supervision, thus permitting control of the system. The uncontrollable events cannot be prevented from occurring.

The online controller, (Chung et al., 1994), can be defined as a function $\gamma : L(G) \rightarrow 2^{\Sigma}$. For each string which can occur in the system, γ returns the subset of events which should be enabled (all other events will be disabled) so that the system behavior is restricted to strings in K. Note that Σ_{uc} will always be contained in this subset. The behavior of the system G under the control of γ, denoted by $L(G, \gamma)$, is the set of all strings which the system can execute after the restrictions of γ are set in place.

The limited-lookahead controller operates by constructing a tree of all the possible continuations of the executed prefix, up to a certain depth, N. Then it recursively explores the branches of this tree to determine which nodes correspond to illegal sequences. The algorithm propagates the information back to the root and determines which events need to be disabled to prevent the occurrence of illegal strings. However, because of the limit to the tree depth, the controller might not have the information needed to make the correct decisions. This problem is discussed in (Chung et al., 1992; Chung et al., 1994) and related work.

2.1. Dynamic Discrete-Event Systems

Here we will consider the control of dynamic discrete-event systems. Dynamic discrete-event systems (DDESs) are a class of time-varying systems composed of DES modules. The use of modules is an approach similar in spirit to (de Queiroz and Cury, 2000; Minhas and Wonham, 2003; Brandin

et al., 2004). Time is considered to be discrete—it increases by one after each occurrence of an event. At each time period the system may consist of the composition of a different set of modules. Thus, the system varies in time in that old modules may disappear and new ones may appear. Modification of modules can be modeled as the disappearance of the old version of a module and the appearance of the new version.

Formally, a DDES can be modeled as follows. Let $M_i = \{M_{1i}, M_{2i}, \ldots, M_{ni}\}$ be some set of modules and let $\|M_i$ denote the composition of all elements of M_i, the modules available in time i. The specific composition used depends on the application domain. Then DDES $G = \{(\|M_i, i) \mid i \in \{0, 1, \ldots\}\}$ and the system at time i is denoted by $G_i = \|M_i$.

For the purpose of greater clarity in our examples, in the rest of this paper the composition is assumed to be parallel composition (Cassandras and Lafortune, 1999), also termed synchronous product.

2.2. Performance evaluation

In order to discuss improvements of the control of a system, some means of evaluation of the performance has to be defined. Traditionally, two methods are used to specify the desired behavior of a system: marking of states and a legal language. The first one tells the controller "this is desirable" and the second one tells what is acceptable and what is not. As discussed earlier, the use of marking with DDESs is not appropriate. The use of a legal language also has its limitations since it offers only a black-and-white view of the world. For example, it cannot be used to express the idea of "highly undesirable but still acceptable".

In this work we will use an approach which is widely used when optimization is considered: the definition of a measure of the behavior of a system. We define a function $v : L(G) \rightarrow \mathbf{R}$, called a *value function*, which evaluates the strings which the DES G may generate. The function can be any computable function which returns how valuable a string is for the user of the system. The greater the value of the value function for a given string, the more desirable it is; and, conversely, the smaller the value, the less desirable it is. In this paper we assume that the function returns $-\infty$ for all illegal strings and so it can also be used instead of specifying an explicit set of legal strings, K. A controller can use this function to choose how to guide the system. Thus, we can also use the value function as a measure of the performance of such a controller. This approach is similar to the use of a utility function to guide decision making in AI agents (Russell and Norvig, 2003).

The use of a value function provides many advantages when specifying the desired behavior of discrete-event systems, such as letting the user change the evaluation easily during system runtime (if, for example, the goals change). The disadvantage of the value function is in its increased complexity, compared to simple FSM-based approaches. It presumes that a more powerful computing device may need to sit at the steering wheel of the system control.

3. Control of DDES: simple approach

Before we proceed with the description of a more complex DDES control algorithm, we will describe a simple method which will be used to illustrate different issues with the optimal control of DDESs. The simple algorithm, shown in Fig. 1, is a slight modification of the original online control algorithm proposed in (Chung et al., 1994). The controller will explore the branches of the lookahead tree and compute the value function for each possible continuation. The values will be propagated back to the root using the following approach: if all events leading from a node are controllable, the maximal value from the children is taken (since the controller will be able to disable the unwanted events). Otherwise, if there are uncontrollable events leading from the node, the minimal of the values of the children via the uncontrollable events will be taken (since the controller will not be able to

prevent the most undesirable behavior through uncontrollable events). As well, the controller will not explore branches further on if they have a value of minus infinity.

```
1  benefit-to-go(x, h)
2  /* x is a node,
3  h is the string generated to reach this node */
4  Σ_out = events going out of x
5  if Σ_out ∩ Σ_uc = ∅
6      for each σ ∈ Σ_out
7          y = nodeVia(σ)
8          if hσ is illegal
9              v_σ = −∞
10         elseif y hits boundary
11             v_σ = v(hσ)
12         else
13             v_σ = benefit-to-go(y, hσ)
14     return max_{σ∈Σ_out}(v_σ)
15 else
16     for each σ ∈ (Σ_out ∩ Σ_uc)
17         y = nodeVia(σ)
18         if hσ is illegal
19             v_σ = −∞
20         elseif y hits boundary
21             v_σ = v(hσ)
22         else
23             v_σ = benefit-to-go(y, hσ)
24     return min_{σ∈(Σ_out∩Σ_uc)}(v_σ)
```

Figure 1. Simple DDES control algorithm

This algorithm can be used instead of the original online control algorithm. After obtaining the value of the different continuations from the root of the lookahead tree, the controller can choose the path with greatest value and thus the controlled behavior of the system will be optimized.

4. Issues with the simple approach to optimal control

The use of the simple algorithm improves control significantly, as illustrated in (Grigorov, 2004). However, its design is not very appropriate for the control of dynamic systems. A number of issues will be presented in this section using simple examples.

4.1. Description of the illustrative system

Before we present the examples, we will provide a brief description of the system which we will explore.

Let us consider that we have a company which needs supplies, for example wooden logs. The company uses the logistic services of a provider and the contract is such that we can rent one truck at a time. The provider has two types of trucks available—a small truck (ST) and a big truck (BT)—which can bring ten logs or either ten or twenty logs, respectively. The models are shown in Fig. 2(a) and Fig. 2(b).

(a) small truck ST_i (b) big truck BT_i

Figure 2. DES models of trucks

The system consists of the synchronous product of these modules. The number and type of trucks available at any given time is not known in advance. Thus the system is inherently dynamic. Let us consider that our company needs a resupply of exactly forty logs and we would like to have a controller which will guide the system so that we achieve our goal.

The legality specification for the system is given by the following two rules:

1. After one truck goes to fetch supplies, we cannot send another truck (Fig. 3).

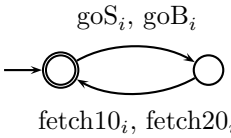


Figure 3. First restriction of the legal language, where $i \in \{1, 2, \ldots\}$.

2. The number of logs fetched is forty:

$$\forall s \in K : \#_{\text{fetch10}}(s) \times 10 \; + \; \#_{\text{fetch20}}(s) \times 20 = 40,$$

where $\#_\sigma(s)$ denotes the number of occurrences of σ in s.

The values of single events are as follows:

- $c(\text{goS}_i) = -100$
  (we pay \$100 to rent the small truck)
- $c(\text{goB}_i) = -150$
  (we pay \$150 to rent the big truck)
- $c(\text{fetch10}_i) = 500$
  (the revenue we get from every log is \$50)
- $c(\text{fetch20}_i) = 1000$

and the value function v will be computed as the summation of the values of single events:

$$\begin{array}{lll} v(s) & = & -\infty \qquad\qquad\qquad \text{if } s \notin \overline{K}, \\ v(s) & = & \sum_{s=\sigma_1\sigma_2\ldots\sigma_n} c(\sigma_i) \quad \text{otherwise.} \end{array} \tag{1}$$

All events in the system are controllable. Even though the proposed algorithm allows for uncontrollable events, the use of controllable events renders the illustrative examples clearer.

4.2. Examples

Example 1

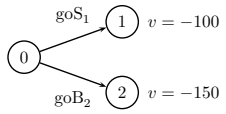


Figure 4. Time 0 (one small truck, one big truck)

In this example we will illustrate the effect of an overly limited lookahead capability. For this purpose, let us limit the prediction of the controller to just one step ahead. At the start the system will have a small truck and a big truck available.

At *time 0* (Fig. 4) the system controller has to make a decision about which events have to be enabled and which have to be disabled. Since the prediction capability is very limited, the tree is very shallow and simple. The cost of renting a big truck is greater than the cost of renting a small truck. Thus the controller, whose task is to optimize the behavior of the system, chooses to disable the event goB_2 and leave only the less costly goS_1.

Since our company needs forty logs, and since sending a small truck four times to bring the logs is more expensive than sending a big truck twice for the same number of logs, one would correctly observe that it is preferable to send the big truck. This judgment is based on the knowledge of what one expects to happen after a type of truck is dispatched. Unfortunately, the controller is much more limited—it can only foresee one step ahead. Without additional information, the controller will always prefer to send small trucks over big trucks. The fundamental problem illustrated here is that an online controller cannot provide optimal control if it cannot observe far enough along event sequences to compute both the relevant costs and the relevant payoffs.

Example 2

In this example we will illustrate the effect of overly lenient lookahead capability. For this purpose, the controller will be able to predict four steps ahead.

The system will be similar to the one in the previous example. At the beginning there will be a small and a big truck available. At *time 2* (after one round-trip) there will be only a small truck available. At *time 4* (after two round-trips) there will be only a big truck available.

As in the previous example, at *time 0* (Fig. 5), the system controller has to decide which events should be enabled or disabled. The tree is constructed, only this time it is much deeper and more complex, since the prediction capability is stronger. The branches $\mathrm{goS}_1\mathrm{goB}_2$ and $\mathrm{goB}_2\mathrm{goS}_1$ are not expanded further, because these sequences are illegal. The controller is able to look further into the future and it recognizes that sending the big truck is preferable, since it is cheaper to bring twenty logs at a time (the value function v examines sufficiently long portions of the event sequences). Thus goS_1 is disabled (the payoff through this branch is only 1250, while through the goB_2 branch it is 1700) and the big truck is dispatched.

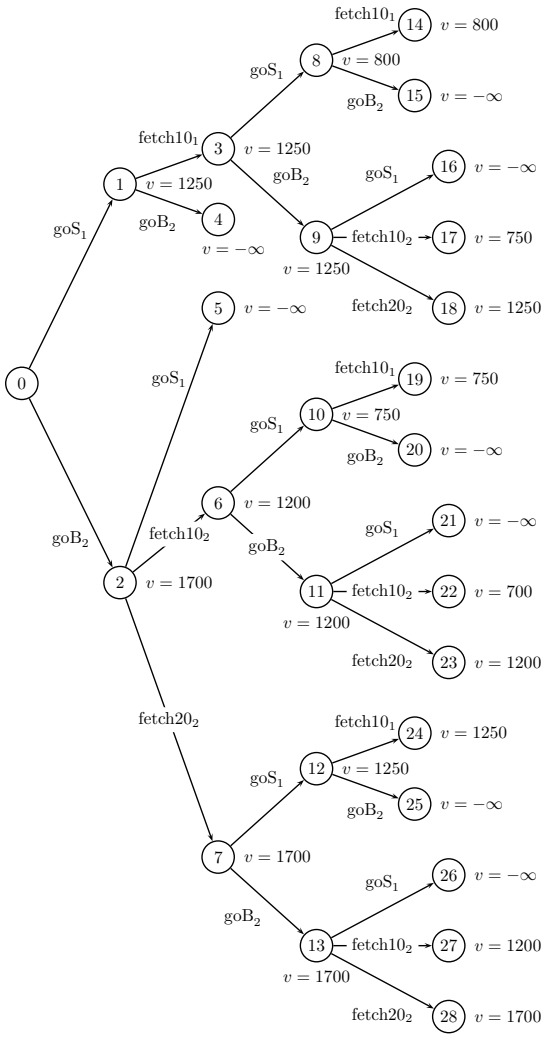
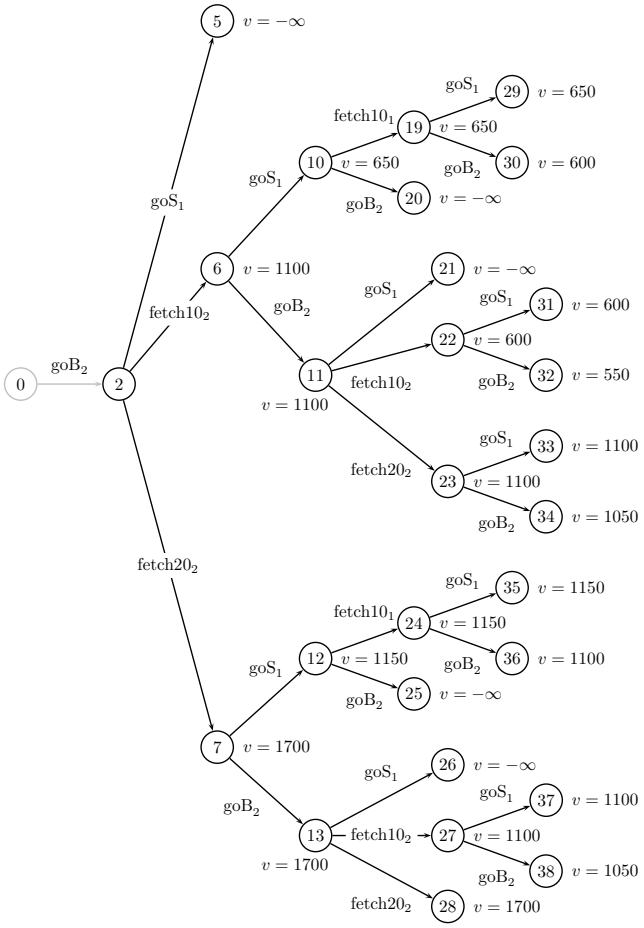
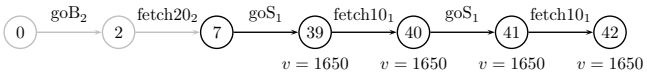


Figure 5. Lookahead tree for Example 2: *Time 0* (one small truck, one big truck)

At *time 2* (Fig. 6(b)), after one round-trip, only the small truck is available (for example, someone else might have rented the big truck). There is only one possible event which also turns out not to be illegal (the value function does not equal minus infinity) and the small truck is dispatched.

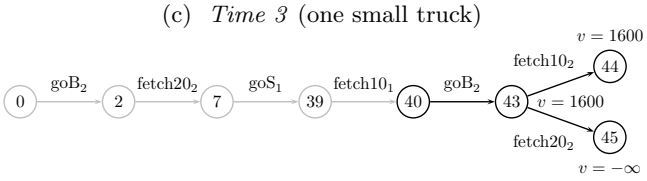


(a) *Time 1* (one small truck, one big truck)



(b) *Time 2* (one small truck)

(c) *Time 3* (one small truck)



(d) *Time 4* (one big truck)

Figure 6. Lookahead trees for Example 2

After the small truck fetches ten logs, the only available truck is the big truck (for example, the small truck might need maintenance). Thus at *time 4* (Fig. 6(d)), the controller has to enable goB_2 even though the truck will be used to fetch only ten logs (fetching more than forty logs in total is illegal).

After bringing the last ten logs, the goal is accomplished. As one can observe, however, the system ended up incurring a greater cost than necessary. Had we used the small truck at *time 0*, the payoff of fetching forty logs would have been

$$v(\mathrm{goS}_1\mathrm{fetch10}_1\mathrm{goS}_1\mathrm{fetch10}_1\mathrm{goB}_2\mathrm{fetch20}_2) = 1650,$$

while the payoff in this example is

$$v(\mathrm{goB}_2\mathrm{fetch20}_2\mathrm{goS}_1\mathrm{fetch10}_1\mathrm{goB}_2\mathrm{fetch10}_2) = 1600.$$

In other words, the solution produced by the controller is not optimal. Unlike the situation in example 1, the problem is not caused by the lack of information: the controller has sufficient lookahead capability. This time the cause is the availability of incorrect information. Since the system is dynamic, basing control decisions on predictions of a too distant future will most likely result in incorrect assumptions. In this example, at *time 0* the system assumes the big truck will be available at *time 2* and the computations of string values are based on this. The fundamental problem illustrated here is that the lack of a good model of the changes of a system renders far-reaching predictions inherently unreliable.

Example 3

In this example we will illustrate why an optimization approach with long-term planning may not be suitable for the control of dynamic systems.

Let us assume that the system will have a scheduler which can postpone the sending of a truck using a wait event, where $c(\mathrm{wait}) = 0$. There is a small truck available at *time 0* and it will not be available at *time 1* if it is not used. In this setting, at *time 0* (Fig. 7) the strings "$\mathrm{goS}_1\mathrm{fetch10}_1\mathrm{wait}$" and "$\mathrm{wait}\,\mathrm{goS}_1\mathrm{fetch10}_1$" have the same value and thus the controller will choose randomly between them. As a result, one out of two tries the system will miss the chance to utilize the available equipment. The fundamental problem illustrated here is that, due to the unreliability of dynamic

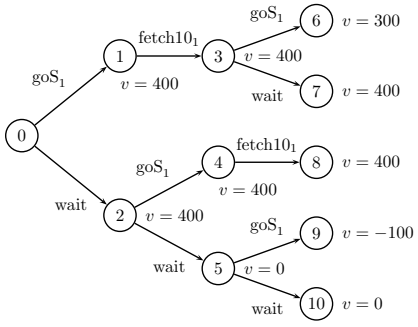


Figure 7. Time 0 (one small truck)

systems, a greedy approach where the systems attempts to execute short valuable strings may be preferable over well-planned strategies that reach far in the future.

5. Properties of DDESs

In this section we will discuss some of the properties of DDESs which are used in the design of the advanced control algorithm described in Section 6.

5.1. System reliability

As was discussed in the previous section, there are a number of issues that arise from the inherent uncertainty of dynamic systems and which are not present in static systems. If a controller has to operate in such circumstances, it has to take uncertainty into account. One way to approach the problem is to model system reliability.

The theory of probability (Barr and Zehna, 1983) can be used to model the reliability of a DDES over time. Different application-specific probability distributions can be utilized, depending on how precise one needs the model to be. In this paper we chose to use the complement of the geometric distribution, both for clarity and for its wide range of applicability. Given $p \in [0, 1]$, the probability that a change in the system will occur, the reliability of the system over time $t \in \mathbf{N}$ is

$$R(t) = (1 - p)^t. \tag{2}$$

The graph of this function for $p = 0.25$ is given in Fig. 8.

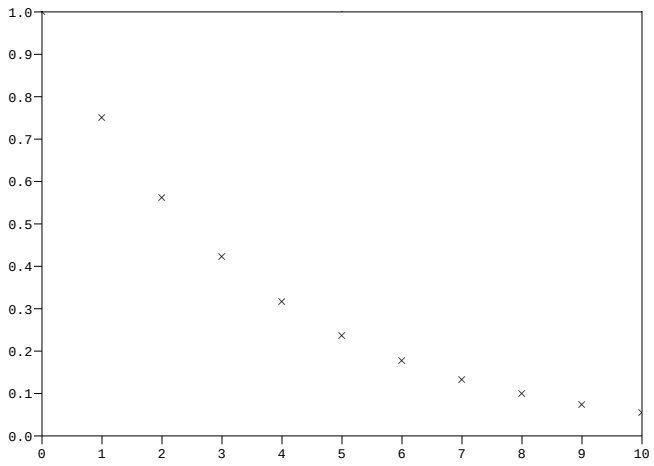


Figure 8. The reliability function $R(t)$ for $p = 0.25$ and $t \in \{0, \ldots, 10\}$.

Example

Let a system be comprised of two modules, A and B, and let the probabilities of individual changes in the system be $P(A \text{ disappears}) = 0.2$, $P(B \text{ disappears}) = 0.01$ and $P(\text{new module appears}) = 0.05$. Then

$$\begin{aligned} &P(A \text{ disappears} \vee B \text{ disappears} \vee \text{new module appears}) \\ &\quad = 1 - P(\overline{A \text{ disappears}})P(\overline{B \text{ disappears}})P(\overline{\text{new module appears}}) \\ &\quad = 1 - 0.8 \times 0.99 \times 0.95 \\ &\quad \approx 0.25\ . \end{aligned}$$

Thus, the overall probability that the system will change is $p = 0.25$ and the system reliability is $R(t) = (1 - 0.25)^t = 0.75^t$.

The probability of the occurrence of a string can be further refined if probabilities for the occurrence of single events are also taken into account (e.g., see (Kumar and Garg, 2001)). However, in our work we focus on another aspect of reliability—that of the overall system dynamics.

5.2. Normalization for risk

After the values of the strings in the lookahead tree are established, the controller may use this information directly and choose the path that would generate the most benefit to the user (as in (Chen and Lin, 2001)). This method is readily applicable to static or relatively stable systems. However, when a DDES is considered and it is highly time-varying, this method may result in an undesired bias toward very profitable long-term goals which in reality cannot be reached due to system permutations. Thus the user may want to adjust the value of strings in accordance with the reliability of the system (i.e., the confidence that the string can actually occur).

In AI research, *utility theory* is used to deal with a related problem (Russell and Norvig, 2003). The value of a state is dependent on the possible stochastic evolution of the system from this state. Furthermore, in Markov decision processes, one can use discounting to signify preferences for immediate rewards. The utility (value) of a Markov chain is discounted geometrically as a function of its length, using a constant coefficient. In the following, we borrow some of these ideas, however, our approach allows for a more flexible value adjustment than simple discounting.

A function $\mathcal{N} : [0; 1] \to [0; 1]$, called *normalization for risk*, can be defined and used to adjust string values according to the user's confidence in the actual occurrence of strings. If the value of a string is big but it is most likely that the system will change before the full string is executed, the user can use the normalization function to reduce the impact of this string in favor of strings with smaller values which are very likely to occur. The function receives as an input the reliability of the system for a string, $R(|s|)$, and outputs the normalization coefficient for this string value. For strings with negative values, the reciprocal of the coefficient is used. Thus, the *normalized value* of a string $s = ht$, where h is the substring already executed and t is the substring in the lookahead tree, would be

$$V(s) = v(s)\mathcal{N}(R(|t|))^{sgn(v(s))}, \tag{3}$$

where $sgn(x)$ equals 1 if $x \geq 0$ and -1 otherwise.

Normalization is important for a highly-dynamic environment because it is a means by which the problem of overspecialized predictions vs. lack of information can be addressed. The controller would consider beneficial strings far in the future only if they are very profitable: the same way people tend to create long-term plans only for the most important things.

Example

Let us choose a normalization function such that if the probability of the system to remain stable is 90% or over, we accept the original string value; if the reliability is less than 90% but at least 50%, we reduce the original value in half; and if the reliability is less than 50%, we multiply the original value by zero (i.e., it is too risky). The corresponding function is shown in Fig. 9. Let us consider the strings $s = \alpha\beta$ and $t = \beta\alpha\alpha$ which can occur in the system from the previous example (with $p = 0.25$) and let $v(s) = 10$ and $v(t) = 100$. We have $R(|s|) = R(2) = 0.75^2 \approx 0.56$ and $R(|t|) = R(3) = 0.75^3 \approx 0.42$. Thus, $V(s) = v(s)\mathcal{N}(R(|s|))^1 = 10 \times 0.5 = 5$ and $V(t) = v(t)\mathcal{N}(R(|t|))^1 = 100 \times 0 = 0$. As a consequence, the system controller will guide the system toward executing s rather than t, even though the string t has greater value before normalization.

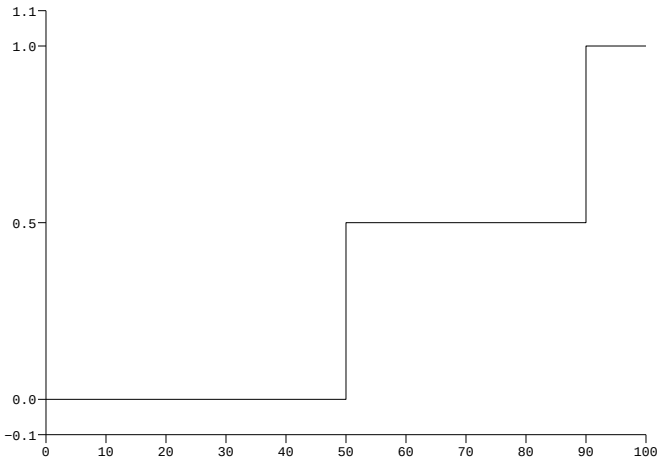


Figure 9. Example normalization function $\mathcal{N}(x)$.

5.3. String value potential

The normalized value function V as defined previously leads to an important observation. Let us assume that the single-step increase of the function v be bounded by a positive number $r \in \mathbf{R} > 0$, i.e.,

$$\exists r : \forall s\sigma \in L(G), v(s\sigma) - v(s) \leq r.$$

Then for each string in the lookahead tree we can compute the maximal value that a continuation of this string in the lookahead window can have. We call this value the *potential* of the string.

Let d be the distance between a node and the root of the lookahead tree, N be the limit to the depth of the lookahead tree, and $v = v(s)$ be the value of the string s leading to the node.

Lemma 1 (Potential)**.** *Let M denote the maximum of the function $(tr + v)\mathcal{N}(R(t + d))^{sgn(tr+v)}$ for $t \in \{1, \ldots, N - d\}$. Then for any string su in the lookahead tree, $V(su) \leq M$.*

Proof. If r is the single-step increase limit, then for all u, $v(su) \leq r|u| + v(s)$. We can substitute $r|u| + v(s)$ for $v(su)$ in the definition of $V(su)$ and the rest of the proof is trivial. □

It is safe to assume that for most practical choices of v, R and $\mathcal{N}$, the maximum M will exist and the potential of strings can be computed.

Example

It is not always necessary to compute the maximum M if it is simpler to obtain an upper bound, as illustrated next.

Let $R(t) = (1 - p)^t$ and $\mathcal{N}(x) = x$. For $p = 0.25$ and $v(s) > 0$, we can derive that the function $(tr + v(s))\mathcal{N}(R(t)) = (tr + v(s))(1 - p)^t$ has an upper bound at $t' = -1/\ln 0.75 - v(s)/r$. Thus, the potential for any string su in the subtree of s (the maximal possible evaluation $V(su)$) would not be more than $(t'r + v(s))(1 - p)^{t'}$.

The above result can be used to limit the exploration of the lookahead tree. If there is already a string that is evaluated higher than the potential of the currently explored string, then there is no need to explore its subtree.

6. Near-optimal online control algorithm with normalization

In this section we will present a more complex algorithm for the control of DDES. It utilizes the ideas discussed previously and the A* search algorithm (Hart et al., 1968) to augment the simple control algorithm (see Section 3) and provide optimization of the control of highly-dynamical systems. The goals behind the design of the algorithm are as follows:

- Legality constraints should be observed and the system should avoid executing illegal strings;
- From all legal strings, the controller should choose to execute the string that would have the most value to the user of the system;
- The controller should be conservative with risk-taking (i.e., worst-case should be assumed when uncontrollable strings are evaluated);
- The controller should be optimistic when there is no risk (i.e., the best case should be pursued when the strings are controllable);
- Exploration of subtrees of the lookahead tree should be limited only to nodes which may influence the control decision.

The above are standard requirements also for controllers of static systems. However, there is a need to consider two new goals:

- The controller should account for the variability of dynamic systems and should put less emphasis on strings too far in the future;
- The controller should attempt to utilize the available resources with a greedy attitude: it should prefer strings that benefit the user sooner rather than later.

The new algorithm is shown in figures 10 and 11. It starts with a call to `control-step`. All uncontrollable events are enabled by default. Then `benefit-to-go` for each child of the *root* is called. If all choices lead to illegal strings ($m = -\infty$) or if an illegal string can be executed via an uncontrollable event, the controller announces a *run-time error* (RTE). Otherwise, only the events leading to the best-evaluated nodes are enabled (in addition to the uncontrollable events).

The heart of the algorithm lies in the `benefit-to-go` function. ▷ If the function is called on nodes constructing an illegal string, it returns $-\infty$ (i.e., the worst possible evaluation of the node). ▷ If it is called on leaf nodes (they hit the boundary or they do not have events leading out), the function returns the normalized evaluation V of the string leading to the node. ▷ If the node has uncontrollable events leading out, safety has to be verified and the worst-case scenario is assumed. The function is called recursively on all descendants of the node and the minimal value is taken (which is identical to the behavior of the original control algorithm). ▷ If, however, the function is called on a node with only controllable events leading out, the algorithm switches to a dual-to-A* search of the subtree of this node. The goal is to find the node with *greatest* normalized value of all nodes that do not have uncontrollable events leading out. (If they have, safety has to be verified instead.) This is a goal dual to the goal in the A* algorithm where the node with the *minimal* value is of interest. Consequently, instead of using an underestimating predictor function, we need to use an overestimating predictor function. According to Lemma 1, the value potential of nodes is an overestimation and thus it is used to select the most promising branches of the subtree. The properties of the A* algorithm trivially hold in the dual setting, thus, in the rest of the paper, we refer to this implementation simply as the A* search. The list of nodes still not explored is *openNodes*. The operation '*maxNode*' can be applied to this list to select the node with greatest value potential. The list and maximal normalized value found so far are initialized

with the root of the system. Then, the maximal element in the list is explored in a loop until the potential of this element is lower than the maximal normalized value already found. Since the function `potential` is an overestimation, exploring all nodes with a value potential greater than the current maximal normalized value guarantees that all nodes with a normalized value greater than the current maximal normalized value will be explored as well. This exhausts all nodes that may influence the control decision and we can be sure that it is safe to skip all other nodes in the open list. ♢ For each explored node from the list, if it is illegal, it is skipped (this is safe to do because it is reachable through a controllable event). ♢ If it has uncontrollable events leading out, `benefit-to-go` is recursively called on it to do worst-case exploration of the its subtree. ♢ Otherwise, if the node contains only controllable events leading out, its value is compared to the maximal value determined so far. In this way, greediness is achieved: instead of considering only the values of the leaf nodes (as is done in the simple algorithm, Fig. 1), the algorithm considers the normalized values of all nodes in the tree. Thus, an inner node with a greater normalized value will be preferred over a leaf node with a smaller normalized value even though the leaf node may be a better long-term solution with a greater non-normalized value. If the node hits the border, its children are not examined. Otherwise, the potential of each child is examined and only children with a potential higher than the maximal value are included in the open list. At the end, when all legitimate nodes from the open list are examined, the function returns the maximal value from the subtree of nodes that do not have uncontrollable events leading out of them.

```
1  /* x.σ denotes the node reachable from node x via σ,
2  Σ(x) denotes the events leading out of node x */
3
4  control-step()
5  /* E is the set of enabled events for the next step */
6  E = Σ(root) ∩ Σ_uc
7  m = max_{σ∈Σ(root)}{benefit-to-go(root.σ)}
8  if m = −∞ or min_{σ∈Σ(root)∩Σ_uc}{benefit-to-go(root.σ)} = −∞
9      announce RTE
10 E = E ∪ {σ | σ ∈ Σ(root), benefit-to-go(root.σ) = m}
11 return E
12
13 potential(x)
14 /* r is the single-step increase limit for v, N is the depth limit of
15 the tree */
16 return the maximum of (tr + v(x))N(R(t + depth(x) − 1))^{sgn(tr+v(x))}
17        for t ∈ {0, ..., N − depth(x)}
```

continued in Fig. 11...

Figure 10. Near-optimal DDES control algorithm with normalization (1)

. . . continued from Fig. 10

```
benefit-to-go(x)
/* x is a node (also used to denote the string leading to it) */
if x is illegal
    return −∞
if x hits the boundary or Σ(x) = ∅
    return V(x)
if Σ(x) ∩ Σ_uc ≠ ∅
    for each σ ∈ (Σ(x) ∩ Σ_uc)
        v_σ = benefit-to-go(x.σ)
        if v_σ = −∞
            return −∞
    return min_{σ∈(Σ(x)∩Σ_uc)}(v_σ)
else /* Σ(x) ∩ Σ_uc = ∅ */
    maxValue = V(x)
    openList = {x}
/* 'maxNode' is an operation that returns the node with greatest
value potential */
    while openList ≠ ∅ ∧ potential(openList.maxNode) > maxValue
        y = openList.maxNode
        openList = openList \ {y}
        if y is illegal
            continue
        if Σ(y) ∩ Σ_uc ≠ ∅
            if benefit-to-go(y) > maxValue
                maxValue = benefit-to-go(y)
        else
            if V(y) > maxValue
                maxValue = V(y)
            if y hits the boundary
                continue
            for each σ ∈ Σ(y)
                z = y.σ
                if potential(z) > maxValue
                    openList = openList ∪ {z}
    return maxValue
```

Figure 11. Near-optimal DDES control algorithm with normalization (2)

This algorithm implements the desired properties because it switches between two different modes of operation depending on the type of node examined. For nodes with uncontrollable events leading out, it applies strict safety-verification, while for nodes with only controllable events leading out, the A* search is applied to find fastest the node with maximal value. Note that modern improvements to the A* search like IDA* (Korf, 1985) and others are not employed, since the only advantage they offer is in the amount of space used for the tree exploration at the expense of having to repeatedly call the evaluation functions. However, when very limited space for the algorithm is available, it can easily be modified to use other search methods.

7. Correctness

The near-optimal algorithm performs, in terms of safety, at least as well as the original online control algorithm with optimistic attitude. Furthermore, in terms of optimality of control, at each step it chooses the path that leads to the substring with greatest normalized value while avoiding costly strings reachable through uncontrollable events. The algorithm is also greedy by design. We formalize and prove these claims next.

Let $L(G)$ be the language generated by the uncontrolled system, $L(G, \gamma_n)$ be the language of the system controlled by the near-optimal algorithm, K be the legal language and N be the depth of the lookahead tree. Let $N_u(L)$ denote the length of the longest finite substring of uncontrollable events in the language L if it exists:

$$N_u(L) = max\{|v| \mid v \in \Sigma_{uc}^*, \text{ there exist } u, w \in \Sigma^*, uvw \in L\}$$

Theorem 1 (Safety)**.** *The language generated by the algorithm is at least as safe as the language generated by the original control algorithm with optimistic policy (*Chung et al., 1992*). It is safe under the same sufficiency requirements, i.e.,*

$$\textit{if } N > N_u(K) + 1 \textit{ or } N > N_u(L(G)), \textit{ then } L(G, \gamma_n) \subseteq K.$$

Proof. We will prove that if the left side of the implication holds, the algorithm will prevent the execution of illegal strings. There are two possible forms of an illegal string $s \in L(G)$: its suffix consists of controllable events or it consists of uncontrollable events.

(i) $s = s't, s' \in \Sigma^*, t \in \Sigma_c^+$:
Let $t = t'\tau$ for some $t' \in \Sigma_c^*$ and $\tau \in \Sigma_c$, and let us assume that the system executes the string $s't'$, i.e., it gets as close as possible to the illegal string s without actually executing it. In `control-step` (line 7), all child nodes are evaluated using a call to `benefit-to-go`. The function `benefit-to-go` will return $-\infty$ immediately after it is called for the child node via τ (lines 20 and 21). If the maximal evaluation of the children of the root node is $m = -\infty$, `control-step` will announce an RTE (lines 8 and 9). Otherwise, the set of enabled events, E, will be computed (line 10) and τ will not be included, since the evaluation of the node it leads to is $-\infty \neq m$. Thus, the execution of the illegal string s will be prevented. Note that in this case the left side of the implication is not needed.

(ii) $s = s'u, s' \in \Sigma^*, u \in \Sigma_{uc}^+$:
If s' is not the empty string, then, without loss of generality, we can assume that $s' \in K$ and that the last event in s' is a controllable event σ, i.e., $s' = s''\sigma, s'' \in \Sigma^*, \sigma \in \Sigma_c$. Let us assume that the system executes s'', i.e., it gets as close as possible to the string s' without actually executing it. In `control-step` (line 7), all child nodes are evaluated using a call to `benefit-to-go`. Since there are uncontrollable events following s', line 24 will hold and `benefit-to-go` will perform a recursive depth-first search across uncontrollable events, including the string u.

- If $N > N_u(L(G))$, then the string σu fits entirely in the lookahead tree and the function `benefit-to-go` will return $-\infty$ if it finishes exploring u (lines 20 and 21). Since $-\infty$ is less than any other value, it will propagate to the root of the lookahead tree (line 29). If `benefit-to-go` does not finish exploring u, the only reason for this can be that another illegal string was found earlier during the depth-first search and $-\infty$ was returned via line 28.
- If $N > N_u(K) + 1$, we will distinguish two cases. If $|u| \leq N_u(K)$, then the string σu fits entirely in the lookahead tree and, analogous to the discussion above, it follows that the evaluation of the string will be $-\infty$. If $|u| > N_u(K)$, then there is a substring v of u such that $|v| = N_u(K)+1$ and, by the definition of $N_u(K)$, $s'v \notin K$. The string σv fits entirely in the lookahead tree ($N \geq N_u(K) + 2 = 1 + |v|$) and we can again conclude that the evaluation will be $-\infty$.

In both cases, the node reachable through σ will be evaluated with $-\infty$. If the maximal evaluation of the children of the root node is $m = -\infty$, `control-step` will announce an RTE (lines 8 and 9). Otherwise, the set of enabled events, E, will be computed (line 10) and σ will not be included, since the evaluation of the node it leads to is $-\infty \neq m$. Thus, in all cases when $s' \neq \epsilon$, the controller will either announce an RTE or will disable the controllable event leading to the execution of u, preventing the execution of the illegal string s.

If s' is the empty string, we can follow reasoning analogous to the above to conclude that the controller will announce an RTE and will prevent the execution of s.

□

We continue by showing that the algorithm is conservative with risk-taking and optimistic otherwise. Let S be a subtree of the lookahead tree and let r be its root. We define $T_u(S)$ to be the largest subtree of S such that r is its root and all edges are transitions via uncontrollable events. In other words, $T_u(S)$ is the subtree S where all controllable transitions are removed and subsequently so are all isolated components. Similarly, we define $T_c(S)$ to be the largest subtree of S such that r is its root and all nodes which have children via uncontrollable events become leaves. In other words, $T_c(S)$ is the subtree S where all transitions leading out of nodes with children via uncontrollable events are removed and subsequently so are all isolated components. Notice that in the so defined subtrees, the leaf nodes have properties inverse to the properties of the inner nodes. In $T_u(S)$, the inner nodes have uncontrollable transitions, while the leaf nodes were chosen from the nodes without uncontrollable transitions. In $T_c(S)$, the inner nodes have only controllable transitions, while the leaf nodes were chosen from the nodes which have uncontrollable transitions. Thus, we can view the leaf nodes as the "bridges" between the two types of subtrees. As was discussed previously (see Section 6), the algorithm switches to a different mode of node exploration depending on the type of node (with or without uncontrollable transitions). Thus, for the purpose of binding different types of subtrees (T_c and T_u), we can assume the following. For leaf nodes x, the normalized value $V(x)$ represents the benefit derived from the exploration of the subtree of x using the corresponding inverse mode, rather than the normalized value of the string leading to x. Then, we can treat each type of subtree separately. Let

$$m(S) \;=\; \min_{x \in \text{leaves of } S} V(x) \tag{4}$$

be the minimum evaluation of the leaves of S; and

$$M(S) \;=\; \max_{x \in \text{nodes of } S} V(x) \tag{5}$$

be the maximum evaluation of all nodes in the tree S.

Proposition 1 (Conservatism)**.** *Let S be a subtree of the lookahead tree and r be its root. When* `benefit-to-go` *is called on r in $T_u(S)$, the return value equals $m(T_u(S))$.*

Proof. Since all edges in $T_u(S)$ are via uncontrollable events, $T_u(S)$ will be explored using a depth-first recursive search (lines 24 to 29). The stopping conditions are either that the explored node is illegal (lines 20 and 21), in which case $-\infty$ is returned, or that the explored node is a leaf (lines 22 and 23), in which case the adjusted value V is returned. The minimal value is returned for each subtree (line 29). The exploration of subtrees is omitted only if a node has been evaluated with $-\infty$ (lines 27 and 28). It follows that the value returned for the root r will be $m(T_u(S))$. □

Proposition 2 (Optimism)**.** *Let S be a subtree of the lookahead tree and r be its root. When* `benefit-to-go` *is called on r in $T_c(S)$, the return value equals $M(T_c(S))$.*

Proof. Since all nodes in the tree have edges only via controllable events, $T_c(S)$ will be explored using A* search (lines 30 to 52). The A* algorithm (Hart et al., 1968) guarantees that if the heuristic function never *underestimates* the evaluation of paths, the *maximal* value in the tree will be found. In our settings, we use `potential` to estimate the value of paths and, by definition, it never underestimates. Again, the exploration of a subtree is omitted only if a node has been evaluated with $-\infty$ (38 and 39). All children of such a node will also be evaluated with $-\infty$ and thus need not be considered. Thus, the value returned for the root r will be $M(T_c(S))$. □

Remark 1 (Greediness)**.** *A direct result from Proposition 2 is that the algorithm will prefer a string that generates a higher value in the short term even if all of its continuations have values lower than the long-term value of another string. If s is a string in some subtree $T_c(S)$, then s will be preferred over another string t in $T_c(S)$ if for all $u \in \Sigma^*, tu \in T_c(S)$ implies $V(s) > V(tu)$. This is true even if for all u there does not exist v such that $v \in \Sigma^+, sv \in T_c(S) : V(sv) > V(tu)$.*

Before we state the control optimality theorem, let us prove two helpful lemmas.

Lemma 2. *Let S be a subtree of the lookahead tree and r be its root. If* `benefit-to-go`$(r) = b$*, then there exists a node x in S such that $V(x) = b$.*

Proof. It is sufficient to examine the `return` statements of `benefit-to-go` in order to prove this lemma. All of them either return directly the normalized value V for a node, or return the result of a recursive call (sometimes intermittently stored in a variable). Thus, we can conclude that when the function returns a value, there is a node in the subtree which realizes this value. □

Lemma 3. *Let S be a subtree of the lookahead tree and r be its root. Then, for all nodes x in S:*

(i) $V(x) \leq$`benefit-to-go`(r) *or, if this does not hold, then*

(ii) there exists x' in S such that the strings leading to x and x' share a common prefix $u \in \Sigma^*$, the string leading to x' is of the form $u.\Sigma^+_{uc}$, and $V(x') \leq$`benefit-to-go`(r).

Proof. If r is an illegal node, the call `benefit-to-go`(r) will return $-\infty$ (lines 20 and 21). The value of all nodes in the subtree is by definition also $-\infty$ so the claim holds trivially. For the case when the root is not an illegal node, we will prove the claim by using induction on the depth of S. Let $b =$`benefit-to-go`(r).

Let r be the only node in the subtree (i.e., the depth is 0). Then, a call to `benefit-to-go` will return $V(r)$ (lines 22 and 23). Thus, $V(r) \leq b$ and the claim is trivially fulfilled. Now let us assume that the claim holds for subtrees of depth up to k, where $k \in \mathbf{N_0}$. We will show that this implies the claim holds also for subtrees of depth $k + 1$. We need to examine two cases: the root r has children via uncontrollable events or it has children only via controllable events.

If r has children via uncontrollable events, only such children will be examined (lines 24 to 29). Thus, the return value b will equal the return value of `benefit-to-go`(r') for some child, r', of the root reachable via an uncontrollable event. Let us suppose that there is a node x in the subtree of r

such that $V(x) > b$ (i.e., point (i) does not hold for x). We will show that this implies point (ii), i.e., there exists a node x' as described. Since $b =$`benefit-to-go`(r'), $V(x) >$`benefit-to-go`(r'). Let $V(r') \leq$`benefit-to-go`(r'). The strings leading to x and r' have the common prefix ϵ, the event leading from r to r' is uncontrollable, and $V(r') \leq b$. Thus, if we consider $x' = r'$, we have the node described in point (ii). If, on the other hand, $V(r') >$`benefit-to-go`(r'), the node r' violates point (i) for its subtree. Since the subtree of r' has a depth of at most k, we can use the induction hypothesis and conclude that point (ii) holds for the subtree of r'. In other words, there exists a node y in the subtree such that it shares a common prefix, ϵ, with r', it is reachable through uncontrollable events from r', and $V(y) \leq$`benefit-to-go`(r'). This implies that y is reachable via uncontrollable events also from r, since an uncontrollable event leads from r to r'. We can observe that x and y share the common prefix ϵ, the (non-empty) string leading to y consists of uncontrollable events, and $V(y) \leq b =$`benefit-to-go`(r'). Thus, if we consider $x' = y$, we have the node as described in point (ii).

Now we need to consider the case when there are only controllable events leading out of r. Thus, `benefit-to-go` will use the A* search for the subtree of r (lines 30 to 52). Let us suppose that there is a node x in the subtree of r such that $V(x) > b$ (i.e., point (i) does not hold for x). We will again show that this implies point (ii), i.e., there exists a node x' as described. If the string leading from r to x passes only through nodes which have only controllable transitions coming out, then we arrive at a contradiction since the A* algorithm will be applied to this string and will return the maximal normalized value encountered. Thus, we can conclude that x belongs to some subtree of r such that there are uncontrollable events leading out of the root, r'. Without loss of generality, we can assume that, on the path from r to x, r' is the first node with uncontrollable events. By definition, `potential`$(r') > V(x)$. Thus, `potential`$(r') > b$ and, from the properties of the A* algorithm, it follows that r' will be explored. Since there are uncontrollable events leading out of r', `benefit-to-go` will use a recursive call to evaluate the node (lines 40 to 42). Since the return value b cannot be smaller that the result of `benefit-to-go`(r') (guaranteed by lines 41 and 42) and since $V(x) > b$, we can conclude that $V(x) >$`benefit-to-go`(r'). The subtree of r' has a depth not greater than k, thus, using the induction hypothesis, we can conclude that there exists a node y in the subtree of r' such that the strings leading to x and y share a common prefix u, the string leading to from r' to y is of the form $u.\Sigma_{uc}^{+}$, and $V(y) \leq$`benefit-to-go`$(r') \leq b$. We can extend this result by prefixing u with the string leading from r to r'. Thus, if we consider $x' = y$, we have the node as described in point (ii). □

Now let us return to the theorem about optimality. Let T be a lookahead tree and $\Sigma(r)$ be the set of events leading out of the root. Let $T(s)$ denote the subtree of T whose root is reached via the string s and let $E(T)$ denote the set of enabled events for this lookahead tree.

Theorem 2 (Optimality)**.** *If $\Sigma(r) \cap \Sigma_c \neq \emptyset$ and there is no RTE, then for all $\sigma \in (E(T) \cap \Sigma_c)$, there is a node y in $T(\sigma)$ such that for all nodes x in T:*

(i) $V(x) \leq V(y)$ or, if this does not hold, then

(ii) there exists x' in T such that the strings leading to x and x' share a common prefix $u \in \Sigma^*$, the string leading to to x' is of the form $u.\Sigma_{uc}^{+}$, and $V(x') \leq V(y)$.

Proof. We can observe that the only place in the algorithm where controllable events are added to $E(T)$ is line 10. A controllable event σ is added only if `benefit-to-go`$(r.\sigma)$ equals the maximal evaluation obtained for all children of r (see line 7). According to Lemma 2, there exists a node y in $T(\sigma)$ such that $V(y) =$`benefit-to-go`$(r.\sigma)$. In order to prove the theorem, we can conveniently use the result of Lemma 3. We have that for all nodes x in $T(\sigma)$, either point (i) or (ii) holds. Thus, it remains to verify this for the nodes in the other subtrees. Let x be a node in a subtree $T(\tau)$ for some $\tau \in \Sigma(r), \tau \neq \sigma$. If $V(x) >$`benefit-to-go`$(r.\sigma)$, then also $V(x) >$`benefit-to-go`$(r.\tau)$ (since

$T(\sigma)$ is the best evaluated subtree). From Lemma 3, it follows that there is a node x' in $T(\tau)$ as described in point (ii) of the lemma. Since $V(x') \leq$`benefit-to-go`$(r.\tau)$ and `benefit-to-go`$(r.\tau) \leq V(y)$, we can conclude that $V(x') \leq V(y)$. As a result, x' also satisfies the conditions of point (ii) in the theorem. This completes the proof. □

8. Complexity

The worst-case time complexity of the near-optimal algorithm is exponential, $O(|\Sigma|^N)$, where Σ is the set of events and N is the depth limit of the tree. This is trivially obtainable for subtrees with uncontrollable events (also shown in (Chung et al., 1992)). When the subtree has controllable events, the algorithm uses the A* search method which also has exponential worst-case time complexity.

The worst-case space complexity of the algorithm is also exponential, $O(|\Sigma|^N)$, since the explored portions of the lookahead tree are retained in memory for reuse at every iteration. The algorithm may be altered to "forget" explored nodes and the space complexity can be decreased to be linear $O(N)$ (since the exploration of the uncontrollable events is depth-first and for controllable events, IDA* can be used instead of A*). The choice depends on how demanding it is to compute the value function and how much run-time memory is available.

Although these results are discouraging, in practice the performance of the new algorithm may be reasonable especially since normalization for risk may reduce significantly the number of nodes that need to be explored. The results of a simulation which was carried out are presented next.

9. Simulation

In order to compare the performance of the near-optimal control algorithm with normalization to the performance of the simple optimal control algorithm, we modeled a DDES and ran simulations to gather statistical data.

9.1. Test system

Instead of choosing to model a system which clearly benefits from a greedy control attitude, we decided to use a system where long-term planning is advantageous, at least when the system is static. We wanted to see how the near-optimal algorithm would perform in such settings.

The system consists of a network of routes between a source and a destination city and packages which need to be delivered. There are five packages in total and each package may travel between the source (S) and the destination (T) as indicated in Fig. 12. Packages are scheduled for delivery

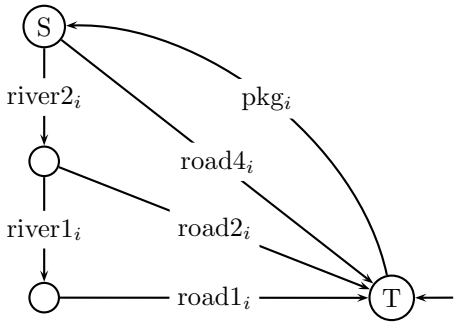
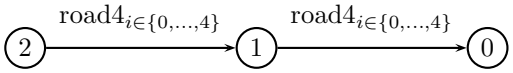


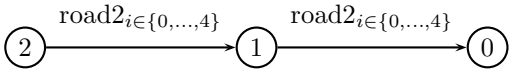
Figure 12. Model of each package for $i \in \{0, \ldots, 4\}$

after they enter the system via the uncontrollable event "pkg$_i$". A package can re-enter the system

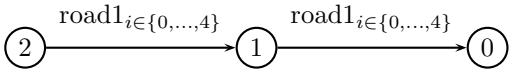
once it reaches the destination city, thus keeping the total number of packages constant. There are two types of connections between cities: roads and rivers, and the length of each connection is indicated with a number in the name of the event (e.g., the length of "road2_i" is two units). There are trucks that can transport packets via the roads (see Fig. 13) and ships that can transport packets via the rivers (see Fig. 14).



(a) Trucks on road 4

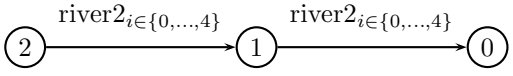


(b) Trucks on road 2

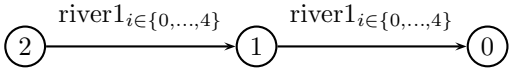


(c) Trucks on road 1

Figure 13. Models of trucks. The numbers inside the states indicate how many trucks are available.



(a) Ships on river 2



(b) Ships on river 1

Figure 14. Models of ships. The numbers inside the states indicate how many ships are available.

Transportation can be controlled and thus the transportation events are controllable. At each point in time, the DDES consists of the synchronous product of five packages and a certain number of trucks and ships. As time progresses, the number of available trucks and ships may change. For the sake of a more compact implementation, the different number of trucks and ships is not represented as the synchronous product of a number of separate truck and ship modules. Instead, the number is captured in the states of a single truck or ship module. Changing the number of trucks or ships involves the replacement of the truck or ship module with the same module only with a different current state. If all packets get "blocked", i.e., there are no trucks or boats available to transport them from their location, a special "wait" event (not modeled explicitly in the state machines) occurs.

Delivery via roads is more expensive than delivery via rivers. The price of use per unit road is 20, while the price per unit river is 8. The payoff of delivering a package is 100. Thus, the value function is defined as $v(s) = \sum_{\sigma \in s} c(\sigma)$, where the single-event values are defined as follows:

- $c(\text{road4}_i) = 20$ (i.e., $100 - 4 \times 20$)
- $c(\text{road2}_i) = 60$ (i.e., $100 - 2 \times 20$)
- $c(\text{road1}_i) = 80$ (i.e., $100 - 20$)
- $c(\text{river2}_i) = -16$ (i.e., 2×-8)
- $c(\text{river1}_i) = -8$

- $c(\text{pkg}_i) = 0$
- $c(\text{wait}) = 0$.

There are no illegal sequences (i.e., $K = L(G)$). This string evaluation scheme means that the longest delivery route brings the greatest payoff and vice versa. In a static system long-term planning should offer the best performance.

Two ways to vary the modules in the system during runtime were considered:

- *normal*: the system starts with two vehicles available for each connection. After each executed event, a random number between 0 and 1 is picked. If the number is greater than 0.66, the number of transport vehicles available is randomly chosen to be between 0 and 2 inclusive for each separate connection. This setup allows for some long-term planning and does not change the system radically at each step.
- *antagonistic*: the system starts with two vehicles available for each connection except for roads 1 and 2, where there is only one vehicle available. After each executed event, a random number between 0 and 1 is picked. If the number is greater than 0.66, the number of transport vehicles available is randomly chosen to be between 0 and 2 inclusive for each separate connection except for roads 1 and 2. If the random number is greater than 0.75, either 0 or 1 truck is randomly chosen to be available for road 2. If the random number is greater than 0.8, either 0 or 1 truck is randomly chosen to be available for road 1. This setup is designed to negate long-term planning by making packages get "blocked" and wait for a long time for transportation over roads 1 and 2.

The reliability function of the system is $R(s) = 0.66^{|s|}$. The normalization function used is

$$\begin{aligned} \mathcal{N}(x) &= 1 \quad \text{if } x > 0.25 \\ &= x \quad \text{if } x > 0.08 \\ &= 0 \quad \text{otherwise.} \end{aligned} \tag{6}$$

This specific function was chosen after a number of simulations comparing the performance of different variations.

9.2. Results

A number of simulation runs were administered in order to compare the performance of three different control algorithms:

- *simple*: the simple optimal control algorithm,
- *greedy*: the near-optimal control algorithm without a normalization function (i.e., $\forall x : \mathcal{N}(x) = 1$) and
- *normalized*: the near-optimal control algorithm with the normalization function described by (6).

Different depths for the lookahead tree were considered: from 2 to 9, both for the normal and antagonistic system setups. Each simulation run was of length 100 events and was repeated at least 500 times to collect more data for better averaging.

Two performance measures were considered: value of the generated strings and the average number of nodes inspected per step. The results of the simulations are shown in Table 1 and figures 15 and 16. When comparing the number of nodes inspected, it has to be pointed out that

Table 1. Results of the simulations

(a) Normal system setup

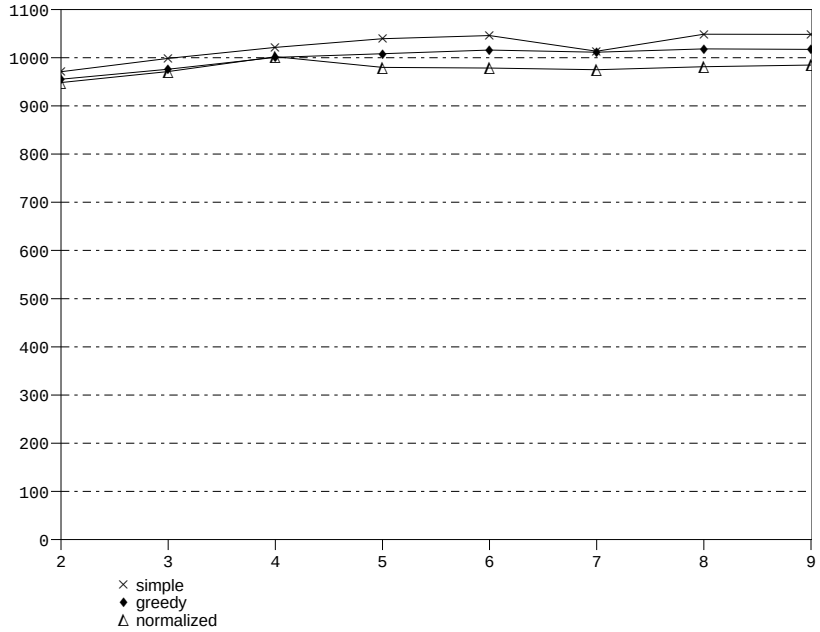
Tree depth	Value of generated string			Inspected nodes per step		
	simple	greedy	normalized	simple	greedy	normalized
2	970.9	955.3	948.4	16.6	17.3	17.1
3	998.8	976.0	971.2	49.9	45.5	45.1
4	1021.4	1000.6	1002.0	137.0	116.2	116.2
5	1039.6	1008.3	979.9	375.9	301.0	157.9
6	1046.1	1016.0	978.5	1015.4	784.7	215.9
7	1013.5	1011.3	975.1	2049.9	2026.1	397.4
8	1048.8	1018.2	981.2	6624.2	5147.8	510.3
9	1048.5	1017.5	984.7	16506.9	12381.4	495.1

(b) Antagonistic system setup

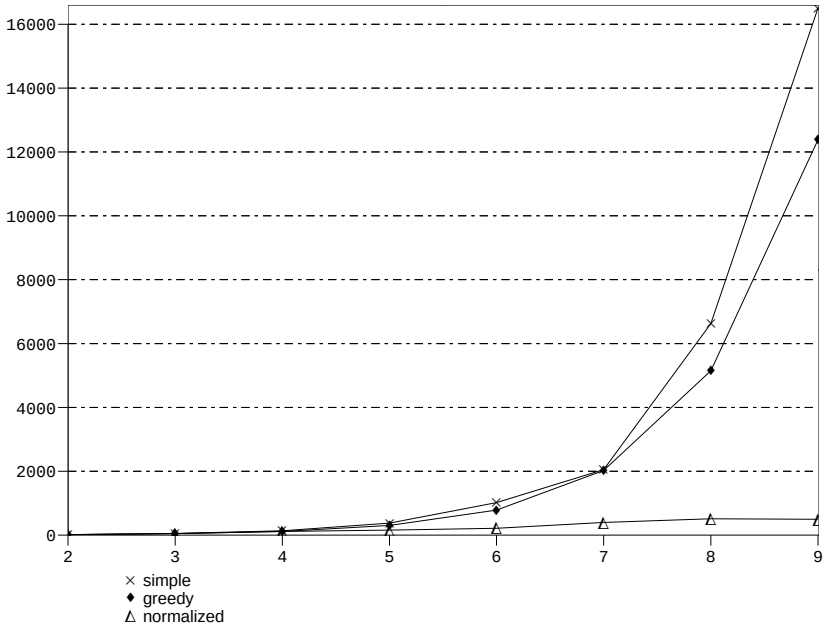
Tree depth	Value of generated string			Inspected nodes per step		
	simple	greedy	normalized	simple	greedy	normalized
2	904.5	882.4	883.4	14.9	15.5	15.4
3	885.6	888.5	890.1	39.0	40.1	40.2
4	878.0	891.6	889.4	95.9	99.2	97.6
5	877.5	899.1	893.0	233.8	248.4	133.8
6	865.2	903.0	886.0	556.2	620.5	188.0
7	862.4	895.7	887.6	1369.1	1480.6	367.7
8	858.0	904.0	895.4	3324.4	3539.8	470.4
9	847.6	893.9	890.6	8454.5	8157.3	465.2

the near-optimal control algorithms spend approximately 1.5 times more time on inspecting the same number of nodes as the simple control algorithm.

The simulations for the normal system setup show that, as expected, the simple control algorithm consistently produces strings with greater value than the other two algorithms. The greedy and normalized algorithms perform identically up to a depth of four: this is the depth after which the normalization function starts modifying node values. Generally, there is not a significant increase of the value of the generated strings with the increase of the depth of the lookahead tree after the initial over-restrictive depths. The normalized algorithm produces strings with value approximately 94% of the value of strings from the simple algorithm; this under-performance is due to the normalization applied to node values. The greedy algorithm produces strings with value approximately 97% of the value of the simple algorithm. While normalization is not applied to node values, the design of the system is such that greedy algorithms are less successful than algorithms with long-term planning (the longer is the route for the delivery of a package, the greater is the benefit). The second performance measure—the average number of nodes inspected per step—clearly distinguishes between the algorithms which do not use normalization and the algorithm with normalization. Both the simple algorithm and the greedy algorithm have exponential growth with the increase of the tree depth, with the simple algorithm growing faster. The number of nodes inspected by the normalized algorithm, on the other hand, grows much more slowly—in a way that makes the algorithm applicable to a real solution of the control of such a system.

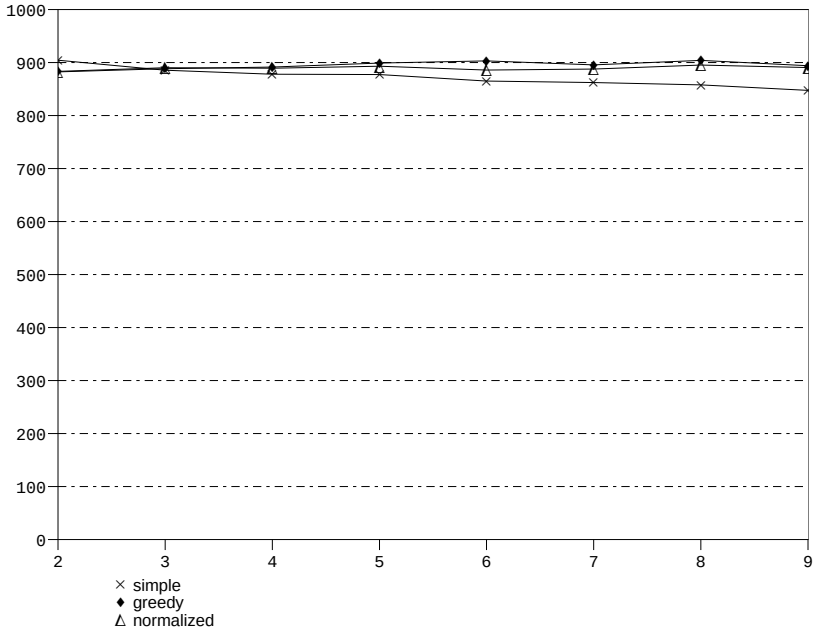


(a) Value of generated strings

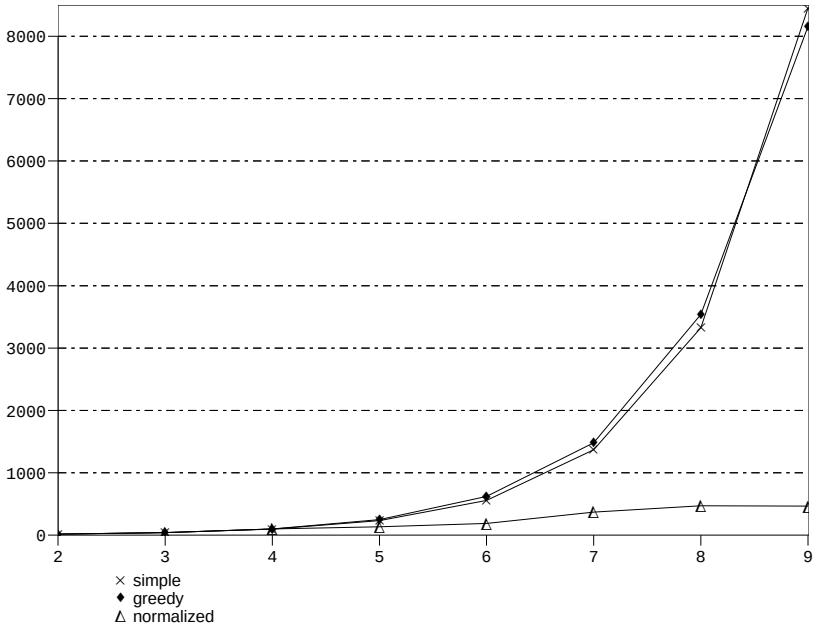


(b) Average number of nodes inspected per step

Figure 15. Graphical representation of the results of the simulations for the normal system setup.



(a) Value of generated strings



(b) Average number of nodes inspected per step

Figure 16. Graphical representation of the results of the simulations for the antagonistic system setup.

The second system setup, the antagonistic setup, was designed to induce a situation in which the simple control algorithm performs worse with the increase of the depth of the lookahead tree due to unrealistic assumptions about the future. This can be seen clearly in Fig. 16(a). The performance of the other two algorithms—greedy and normalized—is not affected in a similar way. These algorithms continue to generate strings of roughly the same value with the increase of the tree depth. This illustrates how important it is to consider system dynamics in the control algorithm. The graph in Fig. 16(b) is very similar to the graph in Fig. 15(b). The simple and greedy algorithms have exponential growth in the number of inspected nodes. In this case, however, the greedy algorithm inspects more nodes per step than the simple algorithm. This seems to contradict the design of the algorithm, i.e., to inspect at most as many nodes as the simple algorithm. The explanation is that in this system setup, the simple algorithm controls the system in a way that packages often get "blocked" and consequently there are fewer nodes in the tree to be inspected while the greedy algorithm does not get packages "blocked" as often and thus has to inspect more nodes in the tree. The number of inspected nodes by the normalized algorithm grows much more slowly with the increase of the tree depth. In this example the normalized algorithm outperforms significantly the simple control algorithm in both performance measures. Generally, it generates strings with values only a little smaller than the ones generated by the greedy algorithm, however, at an incomparably smaller computational cost.

The simulations we conducted are not exhaustive and do not cover a wide variety of DDESs. However, they allow us to demonstrate that, at least in some cases, the proposed near-optimal control algorithm with normalization is viable. Furthermore, it is experimentally shown that in dynamic DESs, deeper lookahead trees may lead to worse control decisions. Greedy algorithms, however, are able to offer a relatively stable performance.

10. Conclusions

In this paper we investigated issues and approaches to the optimal control of time-varying discrete-event systems. We defined a class of time-varying discrete-event systems, named dynamic discrete-event systems (DDESs), and a measure of strings using a value function which returns a real number describing the benefit of the strings for the user of the systems. A simple optimal control algorithm for such systems was proposed, followed by illustrative examples of different issues that can arise if it is used. This led to the specification of additional requirements for algorithms which attempt to optimize the control of DDESs. We then presented a new near-optimal online control algorithm which implements the requirements. It uses a greedy approach and a normalization function to adjusts string values according to the probability that they will occur. Thus, the time variability of the DDESs is accounted for and, additionally, the number of lookahead tree nodes which need to be examined is drastically reduced. The number of tree nodes explored is further reduced through the use of the A* algorithm to explore the subtrees of nodes with only controllable nodes leading out of them. The results of a set of simulations to compare the performance of the new algorithm with the simple optimal control algorithm were presented. It was shown that the performance of the near-optimal algorithm does not suffer when the system is dynamic and the algorithm, at a greatly reduced computational cost, generates strings with values close to the ones generated by the simple algorithm.

The control algorithm with normalization is a first step in the design of a solution for the optimal control of long-lived and time-varying discrete-event systems. In this paper we presented a framework in which many components may be substituted if more appropriate methods are needed: for example, a space-conservative search method may be used instead of the A* search or another method to determine normalization values may replace the measure of system reliability. If the algorithm presented here is to be used with long-lived systems, a method for the appropriate

reduction of the string history has to be developed, e.g., the use of checkpoints. We believe that, generally, using a heuristic approach for the control of complicated and uncertain systems is the key to real-world solutions. The incorporation of additional heuristic information to guide the lookahead tree exploration, such as decision trees, may improve the performance and reduce the complexity of a controller.

11. Acknowledgements

We would like to thank Stéphane Lafortune for his helpful comments during the initial stages of our research. This work was supported, in part, by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

Barr, D. R. and P. W. Zehna: 1983, *Probability: Modeling Uncertainty*. Boston, Massachusetts, USA: Addison-Wesley Publishing Company Inc.

Brandin, B. A., R. Malik, and P. Malik: 2004, 'Incremental Verification and Synthesis of Discrete-Event Systems Guided by Counter Examples'. *IEEE Transactions on Control Systems Technology* **12**(3), 387–401.

Brave, Y. and M. Heymann: 1993, 'On Optimal Attraction in Discrete-Event Processes'. *Information Sciences* **67**, 245–267.

Cassandras, C. G. and S. Lafortune: 1999, *Introduction to Discrete Event Systems*. Norwell, Massachusetts, USA: Kluwer Academic Publishers.

Chen, Y.-L., S. Lafortune, and F. Lin: 1997, 'How to Reuse Supervisors When Discrete Event System Models Evolve'. In: *Proceedings of the 36th Conference on Decision and Control*, Vol. 3. San Diego, California, USA, pp. 2964–2969.

Chen, Y.-L. and F. Lin: 2001, 'An Optimal Effective Controller for Discrete Event Systems'. In: *Proceedings of the 40th IEEE Conference on Decision and Control*, Vol. 5. pp. 4092–4097.

Chung, S.-L., S. Lafortune, and F. Lin: 1992, 'Limited Lookahead Policies in Supervisory Control of Discrete Event Systems'. *IEEE Transactions on Automatic Control* **37**(12), 1921–1935.

Chung, S.-L., S. Lafortune, and F. Lin: 1994, 'Supervisory Control Using Variable Lookahead Policies'. *Discrete Event Dynamic Systems: Theory and Applications* **4**, 237–268.

de Queiroz, M. H. and J. E. R. Cury: 2000, 'Modular Control of Composed Systems'. In: *Proceedings of the 2000 American Control Conference*, Vol. 6. pp. 4051–4055.

Gordon, D. and K. Kiriakidis: 2000, 'Adaptive Supervisory Control of Interconnected Discrete Event Systems'. In: *Proceedings of the 2000 IEEE International Conference on Control Applications*. Anchorage, Alaska, USA, pp. 935–940.

Grigorov, L. and K. Rudie: 2005, 'Issues in optimal control of dynamic discrete-event systems'. In: *Proceedings of the 16th IFAC World Congress*. Prague, Czech Republic. Paper 02005.

Grigorov, L. G.: 2004, 'Control of Dynamic Discrete-Event Systems'. Master's thesis, School of Computing, Queen's University, Kingston, Ontario, Canada.

Hadj-Alouane, N. B., S. Lafortune, and F. Lin: 1994, 'Variable Lookahead Supervisory Control with State Information'. *IEEE Transactions on Automatic Control* **39**(12), 2398–2410.

Hadj-Alouane, N. B., S. Lafortune, and F. Lin: 1996, 'Centralized and Distributed Algorithms for On-Line Synthesis of Maximal Control Policies under Partial Observation'. *Discrete Event Dynamic Systems: Theory and Applications* **6**, 379–427.

Hart, P. E., N. J. Nilsson, and B. Raphael: 1968, 'A formal basis for the heuristic determination of minimum cost paths'. *IEEE Transactions on Systems Science and Cybernetics* **SSC-4**(2), 100–107.

Heymann, M. and F. Lin: 1994, 'On-line control of partially observed discrete event systems'. *Discrete Event Dynamic Systems: Theory and Applications* **4**(3), 221–236.

Korf, R. E.: 1985, 'Depth-First Iterative-Deepening: An Optimal Admissible Tree Search'. *Artificial Intelligence* **27**, 97–109.

Kumar, R., H. M. Cheung, and S. I. Marcus: 1998, 'Extension based Limited Lookahed Supervision of Discrete Event Systems'. *Automatica* **34**(11), 1327–1344.

Kumar, R. and V. K. Garg: 1995, 'Optimal Supervisory Control of Discrete Event Dynamical Systems'. *SIAM Journal on Control and Optimization* **33**, 419–439.

Kumar, R. and V. K. Garg: 2001, 'Control of stochastic discrete event systems modeled by probabilistic languages'. *IEEE Transactions on Automatic Control* **46**(1), 593–606.

Lin, F.: 1993, 'Robust and Adaptive Supervisory Control of Discrete Event Systems'. *IEEE Transactions on Automatic Control* **38**(12), 1848–1852.

Minhas, R. and W. M. Wonham: 2003, 'Online Supervision of Discrete Event Systems'. In: *Proceedings of the 2003 American Control Conference*, Vol. 2. pp. 1685–1690.

Mohanty, S. R., V. Chandra, and R. Kumar: 1999, 'A computer implementable algorithm for the synthesis of an optimal controller for acyclic discrete event processes'. In: *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, Vol. 1. pp. 126–130.

Ramadge, P. J. and W. M. Wonham: 1987, 'Supervisory control of a class of discrete event processes'. *SIAM Journal on Control and Optimization* **25**(1), 206–230.

Russell, S. J. and P. Norvig: 2003, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, New Jersey, USA: Pearson Education, Inc., second edition.

Sengupta, R. and S. Lafortune: 1998, 'An Optimal Control Theory for Discrete Event Systems'. *SIAM Journal on Control and Optimization* **36**(2), 488–541.

Young, S. and V. K. Garg: 1995, 'Model Uncertainty in Discrete Event Systems'. *SIAM Journal on Control and Optimization* **33**(1).