On the Synthesis of Optimal Schedulers in Discrete Event Control Problems with Multiple Goals*

Hervé Marchand, Olivier Boivineau and Stéphane Lafortune
Dept. of Electrical Eng. & Computer Science, Univ. of Michigan
1301 Beal avenue, Ann Arbor, Michigan 48109-2122
e-mail: {marchand, oboivine, stephane}@eecs.umich.edu

July 1998

Report No. CGR-98-10
College of Engineering • Control Group Reports
University of Michigan
Ann Arbor

Abstract

This paper deals with a new type of optimal control for Discrete Event Systems. Our control problem extends the theory of [18], that is characterized by the presence of uncontrollable events, the notion of occurrence and control costs for events, and a worst-case objective function. A significant difference with the work in [18] is that our aim is to make the system evolve through a set of multiple goals, one by one, with no order necessarily pre-specified, whereas the previous theory only deals with a single goal. Our solution approach is divided into two steps. In the first step, we use the optimal control theory in [18] to synthesize individual controllers for each goal. In the second step, we develop the solution of another optimal control problem, namely, how to modify if necessary and piece together, or schedule, all of the controllers built in the first step in order to visit each of the goals with least total cost. We solve this problem by defining the notion of a scheduler and then by mapping the problem of finding an optimal scheduler to an instance of the well-known Traveling Salesman Problem (TSP) [7]. We finally suggest various strategies to reduce the complexity of the TSP resolution while still preserving global optimality.

Keywords
Discrete Event Systems, Optimal Control, Scheduler, Traveling Salesman Problem

*This research is supported in part by INRIA and by the Department of Defense Research & Engineering (DDR&E) Multidisciplinary University Research Initiative (MURI) on Low Energy Electronics Design for Mobile Platforms and managed by the Army Research Office (ARO) under grant DAAH04-96-1-0377.

1 Introduction and Motivation

We are interested in a new class of optimal control problems for Discrete Event Systems (DES). We adopt the formalism of supervisory control theory [13] and model the system as the regular language generated by a Finite State Machine (FSM). Our control problem follows the theory in [16, 17, 18] and is characterized by the presence of uncontrollable events, the notion of occurrence and control costs for events and a worst-case objective function. A significant difference with the work in [18] and with the other works dealing with optimal control of DES [5, 8, 11, 19] is that we wish to make the system evolve through a set of marked states (or multiple goals), one by one, with no order necessarily specified *a priori*; in contrast, the previous theories only deal with a single marked state.

Our problem formulation is motivated by several application domains such as test objective generation in verification and diagnostics, planning in environments with uncertain results of actions, and routing in communication networks.

- In test objective generation, a given system has been designed to meet some specific requirements. However, it may happen that some of these requirements have been overlooked or neglected. Failures can occur as a consequence of negligence. Test objective generation is a way of (ideally exhaustively) checking for inconsistencies in the behavior of the system [1, 2, 14]. The marked states (the states of interest) would be some particular states in which the behavior of the system to be tested is suspected to be flawed. The method that we develop generates a behavior for the system that allows it to reach all these states in an optimal way, with respect to the given occurrence and control cost functions for the events. Each time a state of interest is reached, a behavioral test can be performed on this particular state in order to check if it meets the requirements and conforms to the designed or expected behavior.

- In Artificial Intelligence (AI), the behavior of an agent is often sought to be optimized with respect to an optimality criterion [4]. Moreover, dealing with multiple goals is an active area of research in AI [10]. The model and the methods that we develop in this work can easily be applied to an agent evolving in an environment where the results of its actions are not always the ones expected. Under certain restrictions, there is a mapping between partial controllability in DES and the notion of a nondeterministic environment[1] in AI [15]. The notion of an optimal scheduler that we define and construct can be used to do planning with multiple goals.

- Broadcasting and multicasting in a communication network is an instance of a multi-agent system. Here, the marked states would represent the nodes of the network to which we would want the information to be sent. The uncontrollability of certain events would be interpreted as the uncertainty regarding the actual route that the information would take, since the entire route is not up to the decision of the single sending agent. The solution that we generate can be used to determine the number of duplicated messages that must be sent in parallel through the network, in order for all the desired recipients to receive the piece of information.

Our solution approach consists of two steps. The starting point is a FSM which represents the desired behavior of a given system. From this FSM, we can generate a controller that verifies any

[1] The notion of a nondeterministic environment in AI is different from the notion of a nondeterministic FSM in control of DES. In AI, a nondeterministic environment is one where the actions undertaken by the agent might not lead to the expected arrival state of the world; whereas in control of DES, a nondeterministic FSM is one in which there are identically labeled transitions that lead from one state to different states.

property that we would wish to associate to it, from the set of acceptable controllers. The desirable property is often taken to minimize a quantitative performance measure. In our case, we generate a controller which verifies a range of properties. This is what has been called the DP-Optimality property of a FSM [18]. DP-Optimality stands for Dynamic Programming Optimality. We use back-propagation from the goal state to generate the controller, based on event cost functions. The controller is represented as a FSM also. The theory of DP-Optimal controllers has been developed in the restricted case of one unique marked state [16, 17, 18]. We use the theory in [18] to synthesize a set of optimal controllers corresponding to the different marked states, each treated individually. This yields a set of FSMs that are generated independently from each other. These controllers are synthesized in a manner that gives them optimal sub-structure, consistent with the notion of DP-Optimality of [18]. The objective function has a worst-case form. The total worst-case computational complexity of the first step is cubic in the number of states in the system. At this point, the notion of a DP-Optimal controller is replaced by the notion of a Stepwise DP-Optimal scheduler. By scheduler, we mean a sequence of behaviors that are modeled by FSMs. We develop the solution of a "higher-level" optimal control problem where we use all the controllers built in the first step in order to visit each of the marked states with least total cost; we call this problem that of finding a "Stepwise DP-Optimal scheduler". We solve this problem by defining the notion of a scheduler and then by mapping the problem of finding a Stepwise DP-Optimal scheduler to an instance of the well-known Traveling Salesman Problem (TSP) [7]. We finally suggest strategies to reduce the computational complexity of this step while still preserving global optimality by taking advantage of particular properties of the structure of Stepwise DP-Optimal schedulers.

One of the differences between DP-Optimality and Stepwise DP-Optimality resides in the controller having a FSM structure, whereas the scheduler is a concatenation of FSMs. All the states appear only once in a controller, whereas states can appear several times in a scheduler, but under different circumstances, i.e., in different submachines. Also, another difference between DP-Optimal controllers and a Stepwise DP-Optimal scheduler for a FSM is the existence of a unique maximal DP-Optimal controller which contains all the other DP-Optimal controllers as submachines, whereas there is no notion of a unique maximal Stepwise DP-Optimal scheduler.

This paper is organized as follows. In §2, the necessary notations are introduced. In §3, we recall the basic definitions and properties of the optimal control theory of discrete event systems of [16, 17, 18]. More precisely, we review the notion of a DP-Optimal submachine of a FSM G. This definition is used as a springboard to §4, where we introduce the enlarged problem in the case of multiple marked states. In §5, we define the notion of an optimal scheduler; such a scheduler ensures that the system will visit at least once each state in a given set of states while minimizing a given cost function over the trajectories of the system. §6 suggests possible simplifications that can be made to reduce the overall complexity of the computation of a Stepwise DP-Optimal scheduler. §7 illustrates this new notion of optimality with an example. §8 presents possible applications of the theory that is developed throughout this paper. A conclusion and discussion on future works are presented in §9. Finally, two appendices complete the paper. The first appendix presents a new version of the DP-Optimal algorithm of [18], used in section §3 to construct the maximal DP-Optimal submachine of a given FSM G, in the case where control costs are 0 for controllable events, and ∞ for uncontrollable events. The second appendix outlines the Branch and Bound method, that is a general method for solving the TSP.

2 Preliminaries

In this section, the main concepts and notations are defined. (More definitions will be made when necessary in the following sections.)

The system to be controlled is a Finite State Machine (FSM) defined as follows:

Definition 2.1 *A Finite State Machine (FSM) is a 5-tuple* $G = \langle \Sigma, Q, q_0, Q_m, \delta \rangle$, *where* Σ *is the set of events,* Q *is the (finite) set of states,* q_0 *is the initial state,* Q_m *is the set of marked states, and* δ *is the partial transition function defined over* $\Sigma^* \times Q \to Q$.

The notation $\delta_G(\sigma, q)!$ means that $\delta_G(\sigma, q)$ is defined, i.e., there is a transition labeled by event σ out of state q in machine G. Likewise, $\delta_G(s, q)$ denotes the state reached by taking the sequence of events defined by trace s from state q in machine G. The behavior of the system is described by the prefix-closed language $\mathcal{L}(G)$ [13], generated by G. $\mathcal{L}(G)$ is a subset of Σ^*, where Σ^* denotes the Kleene closure of the set Σ [3]. Similarly, the language $\mathcal{L}_m(G)$ corresponds to the marked behavior of the FSM G, i.e., the set of trajectories of the system ending in one of the marked states of G.

Some of the events in Σ are uncontrollable, i.e., their occurrence cannot be prevented by a controller, while the others are controllable. In this regard, Σ is partitioned as $\Sigma = \Sigma_c \cup \Sigma_{uc}$, where Σ_c represents the set of controllable events and Σ_{uc} the set of uncontrollable events.

In the sequel, we will only be interested in *trim* FSMs (i.e., FSMs whose states are all accessible from q_0 and coaccessible to Q_m). We also introduce the *Trim Operation* for particular states of a FSM. It is defined as follows. Let G be a FSM and X_1 and X_2 be two states of this FSM. Then $G_T = Trim(G, X_1, X_2)$ is a submachine of G, with X_1 as initial state and X_2 as marked state, such that each state of G_T is coaccessible to X_2 and accessible from X_1. For explicit mathematical definitions, the reader may refer to [12].

We also define for any FSM G and $q \in Q$:

$$\begin{aligned} \mathcal{T}(G) &= \{(\sigma, q, q') : \sigma \in \Sigma, q \in Q, \delta(\sigma, q) = q'\}, \\ \mathcal{T}(G, q) &= \{(\sigma, q, q') : \sigma \in \Sigma, \delta(\sigma, q) = q'\}. \end{aligned}$$

These two functions represent the transitions in the machine G and the transitions defined at each state of G, respectively. Similarly, we introduce $\mathcal{T}_c(G)$ (resp. $\mathcal{T}_{uc}(G)$) and $\mathcal{T}_c(G, q)$ (resp. $\mathcal{T}_{uc}(G, q)$) as the set of controllable (resp. uncontrollable) transitions in the machine G and the set of controllable (resp. uncontrollable) transitions defined at each state q of G. $\Sigma(q)$ will denote the active event set at state q of machine G. The projection functions π_1, π_2, π_3 are used to represent the first, second and third components of the 3-tuple $(\sigma, q, q') \in \mathcal{T}(G)$, respectively.

Definition 2.2 *A FSM* $A = \langle \Sigma, Q_A, q_{0A}, Q_{mA}, \delta_A \rangle$ *is a submachine of* G *if*

$$\Sigma_A \subseteq \Sigma,\ Q_A \subseteq Q,\ Q_{m_A} \subseteq Q_m,\ \forall \sigma \in \Sigma_A, q \in Q_A\ \ \delta_A(\sigma, q)! \Rightarrow (\delta_A(\sigma, q) = \delta(\sigma, q)).$$

From definition (2.2), it is immediate that $\mathcal{T}(A) \subseteq \mathcal{T}(G)$. The statement '$A \subseteq G$' denotes that A is a submachine of G. We also say that A is a submachine of G at q whenever $q_{0A} = q \in Q$ and $A \subseteq G$. For any $q \in Q$, we will use $\mathcal{M}(G, q, Q_m) = \{A \subseteq G : A$ is trim with respect to Q_{m_A} and $q_{0_A} = q\}$ to represent the set of trim submachines of G at q with respect to Q_m. This set has a maximal element in the sense that this maximal element contains all other elements as submachines. It is denoted as $M(G, q, Q_m)$. For convenience, we write $\mathcal{M}(G, q)$ and $M(G, q)$ when there is only one marked state, i.e., when $Q_m = \{q_m\}$.

As stated in [18], to take into account the numerical aspect of the optimal control problem, costs are associated with each event of Σ. To this effect, we introduce an occurrence cost function[2] $c_e : \Sigma \to \mathbb{R}^+ \cup \{0\}$ and a control cost function $c_c : \Sigma \to \mathbb{R}^+ \cup \{0, \infty\}$. Occurrence cost functions are used to model the cost incurred in executing an event (energy, time, etc.). Control cost functions are used to represent the fact that disabling a transition possibly incurs a cost. The control cost function is infinity for events of Σ_{uc}. These cost functions are then used to introduce a cost on the trajectories of a submachine A of G.

To this effect, we define a projection p_j that, when applied to a trace of events $s = \sigma_1^s \sigma_2^s \dots \sigma_{\|s\|}^s$, returns the subtrace (prefix) of s of length j starting from σ_1^s; $\|s\|$ denotes the length of s. Formally:

$$\forall s = \sigma_1^s \sigma_2^s \dots \sigma_{\|s\|}^s \in \mathcal{L}(G),\ p_j(s) = \begin{cases} \sigma_1^s \sigma_2^s \dots \sigma_j^s \text{ if } j \leq \|s\|, \\ \text{undefined otherwise (i.e., when } j > \|s\|). \end{cases}$$

We also define $\Sigma_d^G(A, q)$ as the set of disabled events at state q for the system to remain in submachine A of G. $\Sigma_d^G(A, q)$ is then a subset of $\pi_1(\mathcal{T}(G, q))$.

Definition 2.3 *Let A be a submachine of G and $\mathcal{L}_m(A)$ be the marked language generated by A, then:*

- *For any state $q \in Q_A$ and string $s = \sigma_1^s \sigma_2^s \dots \sigma_{\|s\|}^s$ such that $\delta_A^*(s, q)$ exists, the cost of the string s is given by:*

$$c^g(q, A, s) = \sum_{j=1}^{\|s\|} c_e(\sigma_j^s) \ + \ \sum_{j=1}^{\|s\|} \sum_{\substack{\sigma \in \Sigma_d^G(A, q') \\ q' = \delta_A(p_j(s), q)}} c_c(\sigma) \tag{1}$$

- *The objective function denoted as $c_{sup}^g(.)$ is given by:*

$$c_{sup}^g(A) = \sup_{s \in \mathcal{L}_m(A)} c^g(q_{0A}, A, s). \tag{2}$$

Basically, the cost of a trajectory is the sum of the occurrence costs of the events belonging to this trajectory to which is added the cost of disabling events on the way to remain in A. If an uncontrollable event is disabled, this renders the cost of a trajectory infinite because the second term of (1) becomes infinity. The notation $c_{sup}^g(A)$ represents the worst-case behavior that is possible in submachine A.

3 Review of the DP-Optimal problem for one final state

In general, the purpose of optimal control is to study the behavioral properties of a system, to take advantage of a particular structure, and to generate a controller which constrains the system to a desired behavior according to quantitative and qualitative aspects. In the field of control of DES, the supervisory control theory initiated by Ramadge and Wonham [13] provides an appropriate theoretical environment for starting to reason about optimal control. In the basic setup of supervisory control theory, optimality is with respect to set inclusion and thus all legal behaviors are equally good (zero cost) and illegal behaviors are equally bad (infinite cost). The work in [18]

[2] This function is called "event cost function" in [18]; we retain the same c_e notation as in [18].

enriches this setup by the addition of quantitative measures in the form of occurrence and control cost functions, to capture the fact that some legal behaviors are better than others. The problem is then to synthesize a controller that is not only legal, but also "good" in the sense of given quantitative measures. Some other studies appear in [5, 8, 11, 19]. In this section, we present some results of [18] that are necessary for developing the solution procedure for optimal schedulers, the problem of interest in this paper. Our aim here is not to describe in detail all the theory, which can be found in [16, 17, 18], but to present the principal notations and results that we use in the sequel.

3.1 Principal Results

We first consider a FSM $G = \langle \Sigma, Q, q_0, q_m, \delta \rangle$ as defined in §2, but with a single marked state, i.e., $Q_m = \{q_m\}$. Before dealing with the optimal control problem, we recall the definition of controllability for a submachine.

Definition 3.1 *A submachine $A \subseteq G$ is said to be controllable if for all $q \in Q_A$ such that there exists $s \in \Sigma^*$ and $\delta(s, q_{o_A}) = q$, the following proposition is satisfied:*

$$\forall \sigma((\sigma \in \Sigma_{uc}) \wedge (\delta(\sigma, q)!)) \Rightarrow \delta_A(\sigma, q)! \tag{3}$$

In language terms [20], the previous definition can be rephrased as follows. A submachine $A \subseteq G$ is said controllable if

$$\overline{\mathcal{L}_m(A)}\ \Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{\mathcal{L}_m(A)}. \tag{4}$$

Relation (4) says that for a desired language to be controllable, the occurrence of an uncontrollable event must not generate a trace that is not in $\overline{\mathcal{L}_m(A)}$, or in other words, uncontrollable events cannot be prevented from occurring when restricting the behavior of G to A.

Definition 3.2 *For all $q \in Q$, $A_o \in \mathcal{M}(G, q)$ is an optimal submachine of G at q if*

$$c^g_{sup}(A_o) = \min_{A \in \mathcal{M}(G,q)} c^g_{sup}(A) < \infty. \tag{5}$$

In particular, an optimal solution A_o of G at q is in the set $\mathcal{M}(G, q_0)$ (i.e., the set of trim submachines of G). In the light of [16] and [18], the cost $c^g_{sup}(A_o)$ of A_o represents the minimum worst case cost incurred to reach q_m from q_0 when the behavior of G is restricted to a submachine of it. Under the constraint that some events are not controllable (have infinite cost), optimality is met when there is no other control policy with lower worst-case cost that allows to reach the marked state q_m certainly. The following lemma (lemma (2.15) in [16]) is stated to note that optimal solutions lie within the class of controllable submachines. In general, there will be several optimal submachines for a FSM.

Lemma 3.3 *Let $A \in \mathcal{M}(G, q, Q_m)$. If $c^g_{sup}(A) < \infty$ then A is controllable.* ■

From lemma 3.3, uncontrollable submachines are not candidates for optimality since the cost for restricting the system to those submachines is infinite (it is impossible to restrict the system to such a behavior).

Theorem (4.2) of [16] gives necessary and sufficient conditions for the existence of optimal submachines and is recalled without proof.

Theorem 3.4 *An optimal submachine of G exists if and only if there exists $A \subseteq G$ such that A is trim, controllable and for all $s \in \mathcal{L}(G)$ and $q \in Q$ such that $\delta(s,q) = q$ we have $c^g(q, A, s) = 0$.* ■

Intuitively, this theorem states that an optimal solution exists when there are controllable submachines of G in which all cycles have zero cost. The controllability assumption ensures that the positive cost cycles can be broken using controllable events alone.

We now introduce the notion of a DP-Optimal submachine, as it is stated in [16]. This kind of submachine is used extensively in the following sections.

Definition 3.5 *A submachine $A_{DO} \in \mathcal{M}(G,q)$ is DP-Optimal if it is optimal and for all $q' \in Q_{A_{DO}}$, $M(A_{DO}, q')$ is an optimal submachine in $\mathcal{M}(G,q')$.*

If a particular DP-Optimal FSM includes all other DP-Optimal FSMs as submachines of itself, then we call it the *maximal DP-Optimal submachine*. The maximal DP-Optimal submachine of a machine G at q with respect to q_m will be denoted as $M_D^o(G, q, q_m)$. Note that all DP-Optimal submachines are acyclic. The existence of a DP-Optimal submachine of G is given by the following theorem (theorem 4.3 of [16]).

Theorem 3.6 *If an optimal submachine of G exists, then the unique maximal DP-Optimal submachine $G_{des}^m = M_D^o(G, q_0, q_m)$ of G w.r.t. the final state q_m also exists.* ■

This section recalled the notions of controllability of a FSM G as well as of optimality of a FSM in the case of a single marked state. This reflects the fact that the marked state is always reachable with a finite and minimum worst-case cost. Finally, we have recalled the important definition of DP-Optimality, which ensures a minimum worst-case cost from any state to the marked state. Proofs and details can be found in [16, 17, 18].

3.2 The cyclic DP-Optimal algorithm

In order to take full advantage of the DP-Optimal property of a submachine of G, we need to generate such a DP-Optimal submachine.

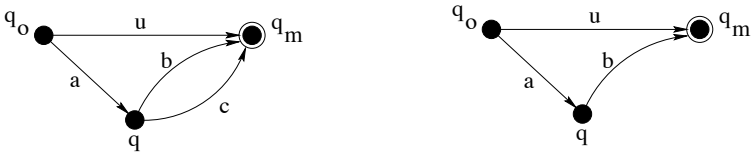
Consider a FSM $G = \langle \Sigma, Q, q_0, q_m, \delta \rangle$ with a unique initial state q_0, and a unique final state q_m. Assume that all occurrence costs are strictly positive; then there exists an algorithm with worst case complexity $\mathcal{O}(|Q|^2|\Sigma|log(|\Sigma|) + |Q|^3|\Sigma|)$ (theorem 6.10 of [18]), that constructs the maximal (in the sense of the partial order relation of inclusion) DP-Optimal submachine of the FSM G. An outline of this algorithm can be found in §6 of [18]. However we present, in Appendix A, a new simplified version of it, named **DP-Opt**, in the particular case where the control cost function is equal to zero for all controllable events and infinity for uncontrollable events. When a DP-Optimal solution exists, the result of the **DP-Opt** program is an acyclic submachine G_{des}^m. The algorithm also returns the worst inevitable cost $c_{sup}^g(G_{des}^m)$. Moreover, during the algorithm computation, we can recover the submachines $M_D^o(G, q, q_m)$ associated with $c_{sup}^g(M_D^o(G, q, q_m))$, for all the states visited during the computation. Of course we will only keep the submachines and the associated costs that are relevant to our optimal control problem with multiple marked states, which we formally present in §4.

3.3 Example of the DP-Optimal problem

We conclude this section by first giving insight in the difference between optimality and DP-Optimality, and then by illustrating the DP-Optimal problem through a more intricate example that is reused in §7.

3.3.1 Optimality versus DP-Optimality

In this part, we give a simple example (figure (1)) that allows to grasp the notion of DP-Optimality versus that of optimality. We have already seen that optimality actually exists when the worst-case cost from the initial state q_0 is to q_m is finite once minimized. DP-Optimality is obtained when the terminal path from any state of a submachine to the goal state q_m is optimal in the previous sense. This is illustrated by the following example.



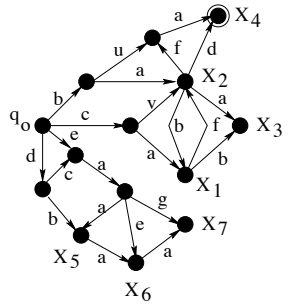
Event	c_e	Remarks
a,b	1	Controllable
c	3	Controllable
u	5	Uncontrollable

Figure 1: A simple example giving the difference between Optimality and DP-Optimality

For simplicity, we have assumed that the control cost function was reduced to 0 for controllable events and to ∞ for uncontrollable events. We can observe that, in both submachines, the cost for going from q_0 to q_m is optimal since there exist no other controllable path with a lower cost. However, in the first submachine, the worst-case cost to go from state q to state q_m is 3 (through event c); whereas the worst-case cost in the second submachine is optimized to be reduced to 1. Consequently, we can say that the second submachine is DP-Optimal (thereby optimal), but the first submachine is not DP-Optimal, but is optimal (since the optimality is obtained only regarding the paths between the initial and final states and never the postfix paths between any state of the corresponding FSM and the final state).

3.3.2 A more intricate example

Let G be a FSM and $\Sigma = \{a, b, c, d, e, f, g, u, v\}$ such that $a, b, c, d, e, f,$ and g are controllable; u and v are uncontrollable. G and the event cost function defined on Σ are as in figure (2). Once again, we assume $c_c \in \{0, \infty\}$. We assume that the initial state is q_0 and the final state is X_4.



Event	c_e	Remarks
a,f	1	Controllable
b,c,d,e	2	Controllable
g	4	Controllable
u	3	Uncontrollable
v	2	Uncontrollable

Figure 2: The initial system G and the event cost function

Using the **DP-Opt** program, we obtain the maximal DP-Optimal submachine of G, denoted G^4_{des} (figure (3)), for which the worst inevitable cost is equal to $c^g_{sup}(G^4_{des}) = 6$.

We can observe all the properties of the generated submachine. First, it is controllable, since from any state, there exists a path that leads surely to the goal X_4. Also, it is optimal, since all the paths leading to X_4 have a finite and minimized worst-case cost (notably, no uncontrollable event at state X_4 needs to be disabled). Finally, the DP-Optimality property can be observed. From

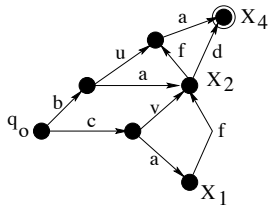


Figure 3: The maximal DP-Optimal submachine G^4_{des}

every state q of G^4_{des}, the path from q to X_4 which has the highest cost contains an uncontrollable event u, that cannot be disabled.

We have reviewed the optimal control problem and the notion of DP-Optimal submachines when only one marked state is present in the system. We now turn attention to the case of multiple marked states and present our results for this new problem. This will require the introduction of a new, more comprehensive, optimality criterion.

4 The Optimal Control problem with multiple marked states

In the previous section, we were interested in finding a DP-Optimal submachine of G that makes the system evolve from an initial state q_0 to a final state q_m by minimizing a cost function along the various trajectories of the system. Here, our goal is different. We consider a FSM G with a set of multiple marked (or final) states, $\mathcal{X} = (X_i)_{i\in[1,\ldots,n]}$. Our aim is now to have the system reach each and every one of the states of $\mathcal{X}$. To account for the fact that it may not be possible to find such a path, we assume in the following the possibility of *resetting* the system to its initial state q_0, when the system has evolved in one of the states of $\mathcal{X}$. The *Reset* event that is added in this section is much more than an artifact for developing the theory. Indeed, many interpretations can be associated with it. First, there are physical systems that can actually be reset to their initial state (like a World Wide Web browser, for example). Second, the *Reset* event can be seen as an event whose occurrence signals the impossibility of visiting all the states of $\mathcal{X}$ without visiting the initial state q_0 more than once. This apparent impossibility can be alleviated by having multiple systems perform in parallel. This possibility is explored in §7. Also, we will give a method for computing the minimum number of parallel systems needed to achieve the goal of visiting all the X_i in $\mathcal{X}$ without re-visiting the initial state q_0. For example, in the case of a communication network, a message that is sent cannot be brought back to the initial state. However, it can be regenerated, and then the number of *Reset* events can be regarded as an indicator of the number of copies of the message that must be generated and sent in parallel in a broadcast or a multicast.

4.1 Stepwise DP-Optimality Definition

Due to the *Reset* event, the system is now represented by the following FSM $G = \langle \Sigma \cup \{Reset\}, Q, q_0, \mathcal{X}, \delta\rangle$, with $\delta(Reset, X_i) = q_0$ for all $X_i \in \mathcal{X}$. As in the previous section, we introduce cost functions that take into account the particular *Reset* event: the occurrence cost function $c_e : \Sigma \cup \{Reset\} \to \mathbb{R}^+ \cup \{0\}$ such that $\forall\sigma \in \Sigma, c_e(\sigma) \geq 0$ and $c_e(Reset) = 0$, and the control cost function $c_c : \Sigma \cup \{Reset\} \to \mathbb{R}^+ \cup \{0, \infty\}$ such that $\forall\sigma \in \Sigma, c_c(\sigma) \geq 0$ and $c_c(Reset) = 0$. We discuss these assumptions in §5.3.

Definition 4.1 *Let $s \in \mathcal{L}_m(G)$. The trajectory s is said to be* **valid** *if there exists at least n prefixes of s, $(s_i)_{i\in[1,\dots,n]}$, such that $\delta(q_0, s_i) = X_i \in \mathcal{X}$.*

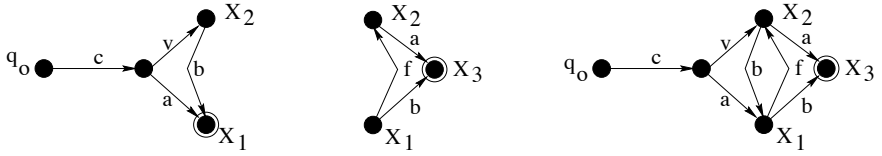
In other words, a trajectory is valid if it makes the system evolve into each of the marked states in $\mathcal{X}$. Note that the definition does not require that the trajectory visit each marked state exactly once. Besides, due to the *Reset* event, the system has the possibility of coming back in its initial state along the trajectory. The set of valid trajectories of the FSM G will be denoted as $\mathcal{S}$.

Given that our primary interest is in the states of $\mathcal{X}$, we introduce the notion of a *valid state trajectory.*

Definition 4.2 *Let s be a valid trajectory in $\mathcal{S}$, such that $s = t_1^s \dots t_l^s$, with $l > n$ and $\delta(q_0, t_1^s \dots t_k^s) = X_k^s \in \mathcal{X} \cup \{q_0\}$. We define the function D from $\mathcal{S}$ into $\{q_0\}(\mathcal{X}^*\{q_0\})^*$, such that $D(s) = (X_k^s)_{k\in[1,\dots,l]}$[3]. Such a trajectory is called a* **valid state trajectory** *w.r.t. $\mathcal{X}$. We denote as $\mathcal{D}$ the set of valid state trajectories in G, w.r.t. the set of valid trajectories $\mathcal{S}$: $\mathcal{D} = D(\mathcal{S})$.*

A valid state trajectory $d \in \mathcal{D}$ corresponds to a trajectory in $\{q_0\}(\mathcal{X}^*\{q_0\})^*$ that contains all the states of $\mathcal{X}$ (with possible repetitions).

Since we must deal with a set of marked states rather than with a single marked state, we need to introduce a model that comprises all the states of the set $\mathcal{X}$ and that accounts for the global behavior of the system. It is not possible to use a classical merge operation ($\oplus$, definition (6.2) in [18]), because states might appear in different submachines in different contexts, i.e., with different partial transition functions associated with them. An example of this conflict is given below, in figure (4).



(a) G^1_{des} (b) $M^o_D(G, X_1, X_3)$ (c) $G^1_{des} \oplus M^o_D(G, X_1, X_3)$

Figure 4: Introduction of a cycle using the merge operation $\oplus$

Consider the FSM G represented in figure (2). We assume that the occurrence costs are the same as the ones given in figure (2). Let the final state be state X_1. The DP-Optimal submachine G^1_{des} w.r.t. this state is given in figure (4(a)). From this state, our goal is to make the system G evolve into the state X_3. This is done by using the DP-Optimal submachine $M^o_D(G, X_1, X_3)$ represented in figure (4(b)). Consider now $G^1_{des} \oplus M^o_D(G, X_1, X_3)$ (figure (4(c))). A cycle has appeared in the graph. Therefore, state X_3 may not be reached during the evolution of the system.

Therefore, instead of using a merge that would (basically) perform the union of all the submachines by overlapping them, we introduce the notion of a scheduler. A scheduler can be thought of as a concatenation of (DP-Optimal in our case) submachines. The role of the scheduler is then to make the system evolve according to one submachine at a time, and account for switching between them at appropriate instants.

[3] This function allows the "extraction" of the state trajectory in G from the valid trajectory s.

In the sequel, the symbol "$\circ$" will denote the concatenation of two submachines A and A' of G. It is defined is terms of languages. Let $\mathcal{L}_m(A)$ and $\mathcal{L}_m(A')$ be the marked languages of A and A'. Then $\mathcal{L}_m(A \circ A') = \{st \ : \ s \in \mathcal{L}_m(A), \ t \in \mathcal{L}_m(A')\}$. Note that $\mathcal{L}_m(A \circ A') \subseteq \mathcal{L}_m(G)$ if and only if $Q_{m_A} = \{q_{0_{A'}}\}$ and $Q_{m_{A'}} \subseteq Q_{m_G} = \mathcal{X}$. Also note that, due to possible cycles in the FSM G, $A \circ A'$ is in general no longer a submachine of G since some state q of G may be shared by the two submachines A and A' but without the same transitions (i.e., $\mathcal{T}(A, q) \neq \mathcal{T}(A', q)$).

Definition 4.3 *Let $d = (X^d_{k'})_{k' \in [0,\ldots,l]} \in \mathcal{D}$ be a valid state trajectory of $\mathcal{X} \cup \{q_0\}$ and let $(A_k)_{k \in [1,\ldots,l]}$ such that $l \geq n$ and $A_k \in \mathcal{M}(G, X^d_{k-1}, X^d_k)$ for all $k \in [1, \ldots, l]$, then the structure $A = A_1 \circ A_2 \circ \ldots \circ A_l$ is called* **a scheduler** *w.r.t. G and $\mathcal{X}$. The set of schedulers w.r.t. G and $\mathcal{X}$ is denoted as $\mathcal{M}^{sc}(G, \mathcal{X})$.*

In this particular case, for each submachine of the scheduler, there is only one initial state and one final state. Hence, for two consecutive submachines A_i and A_{i+1}, we have $q_{m_{A_i}} = q_{0_{A_{i+1}}}$. Note that for a scheduler $A = A_1 \circ A_2 \circ \ldots \circ A_l$, some A_k may be simply reduced to the simple transition $(X^d_k \xrightarrow{Reset} q_0)$. This transition is clearly a DP-Optimal submachine from X^d_k to q_0. Besides, in some cases, $\mathcal{M}^{sc}(G, \mathcal{X})$ can be reduced to $\emptyset$.

The cost associated with a scheduler $A = A_1 \circ A_2 \circ \ldots \circ A_l$, denoted as $C^{sc}_{sup}(A)$, is given by

$$C^{sc}_{sup}(A) = \sum_{i=1}^{l} c^g_{sup}(A_i). \tag{6}$$

The following definition extends the notion of DP-Optimality to the notion of *Stepwise DP-Optimality.*

Definition 4.4 *Let $A \in \mathcal{M}^{sc}(G, \mathcal{X})$ be a scheduler, such that A makes the system evolve through a valid state trajectory $d = (X^d_{k'})_{k' \in [0,\ldots,l]}$ of $\mathcal{D}$. $A = A_1 \circ A_2 \circ \ldots \circ A_l$ is said to be* **Stepwise DP-Optimal** *if each of the submachines $A_k \in \mathcal{M}(G, X^d_{k-1}, X^d_k)$ is DP-Optimal with respect to its initial state X^d_{k-1} and final state X^d_k, and if the following condition is satisfied:*

$$C^{sc}_{sup}(A_o) = \min_{A \in \mathcal{M}^{sc}(G,\mathcal{X})} C^{sc}_{sup}(A) < \infty.$$

We wish to draw attention to the following assumption.

Assumption 4.5 *From now on, we assume that the DP-Optimal submachines under consideration, with the exception of $(X^d_k \xrightarrow{Reset} q_0)$, are maximal. This is done for two main reasons. First, the algorithm* **DP-Opt** *referred to in section (3.2) outputs exactly the maximal DP-Optimal submachines. Second, taking the maximal DP-Optimal submachines allows the system greater freedom. Indeed, it contains all the other DP-Optimal submachines; therefore, it has more possible paths from the initial state to the final marked state. In most applications, it is desirable to lower the probability of taking the worst-case cost path, which is the intent of taking the maximal DP-Optimal submachine for $(G^i_{des})_{i \in [1,\ldots,n]}$. The more possible paths there are, the less likely it is for the system to take the worst-case cost path. Note that the Reset machine $(X^d_k \xrightarrow{Reset} q_0)$ need not be maximal (this can only happen if occurrence costs cannot be equal to zero); in this case however, given our earlier interpretation of the role of the Reset event, we will include the single transition $(X^d_k \xrightarrow{Reset} q_0)$ in the scheduler.* ■

Under this assumption, the following property is a direct consequence of definition (4.4):

Property 4.6 *Let G be a FSM and $\mathcal{X}$ be the set of marked states of G. Let A be a Stepwise DP-Optimal scheduler, such that $A = A_1 \circ A_2 \circ \ldots \circ A_l$. Let $d = (X_k^d)_{k\in[0,\ldots,l]}$ of $\mathcal{D}$ be the associated valid state trajectory. Then $\forall k \in [1,\ldots,l]$, $A_k = M_D^o(G, X_{k-1}^d, X_k^d)$. Furthermore, the global cost of the scheduler is*

$$C_{sup}^{sc}(A) = \sum_{k=1}^{l} c_{sup}^g(M_D^o(G, X_{k-1}^d, X_k^d)) < \infty. \tag{7}$$

■

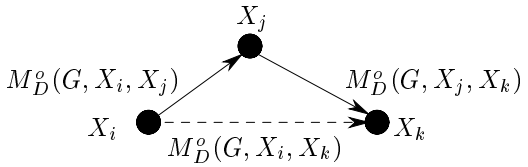
This property states that if a Stepwise DP-Optimal scheduler exists, then all the submachines constituting this scheduler are the respective $M_D^o(G, X_{k-1}, X_k)$. Moreover the cost of the scheduler is then simply equal to the sum of the costs of these DP-Optimal submachines. We will refer to this important result as the *additivity property* of the Stepwise DP-Optimal scheduler. In the sequel, the set of all schedulers A such that all the submachines of A are of the form $M_D^o(G, X_i, Xj)$, for $X_i, X_j \in \mathcal{X} \cup \{q_0\}$, is denoted $\mathcal{M}_D^{sc}(G, \mathcal{X})$.

Now that we have defined the notion of a Stepwise DP-Optimal scheduler and given some of its properties, we need to give necessary and sufficient conditions for its existence. The next subsection gives these conditions and also proves desirable properties of such a scheduler.

4.2 Existence of a Stepwise DP-Optimal scheduler

Theorem (4.9) gives necessary and sufficient conditions for the existence of a Stepwise DP-Optimal scheduler. Before we prove the following lemma.

Lemma 4.7 *If the DP-Optimal submachines $M_D^o(G, X_i, X_j)$ and $M_D^o(G, X_j, X_k)$ of G exist, then there exists a DP-Optimal submachine $M_D^o(G, X_i, X_k)$.*



Moreover, we have the following triangular inequality*:*

$$c_{sup}^g(M_D^o(G, X_i, X_k)) \leq c_{sup}^g(M_D^o(G, X_i, X_j)) + c_{sup}^g(M_D^o(G, X_j, X_k)). \tag{8}$$

Proof: Assume the existence of $M_D^o(G, X_i, X_j) = \langle \Sigma_{ij}, Q_{ij}, X_i, X_j, \delta_{ij} \rangle$ and of $M_D^o(G, X_j, X_k) = \langle \Sigma_{jk}, Q_{jk}, X_j, \{X_k\}, \delta_{jk} \rangle$. Consider the intersection of the states of these two submachines as being $Q_{ij} \cap Q_{jk} = \{X_j, q_1, \ldots, q_n\}$. Note that this intersection might be reduced to $\{X_j\}$. We construct a new submachine, $G_{ik} = \langle \Sigma_{ik}, Q_{ik}, q_{0_{ik}}, Q_{m_{ik}}, \delta_{ik} \rangle$, from these submachines:

$$G_{ik} = \begin{cases} \Sigma_{ik} & = & \Sigma_{ij} \cup \Sigma_{jk} \\ Q_{ik} & = & Q_{ij} \cup Q_{jk} \\ q_{0_{ik}} & = & X_i \\ Q_{m_{ik}} & = & \{X_k\} \\ \delta_{ik}(\sigma, q) & = & \begin{cases} \delta_{jk}(\sigma, q) \text{ if it exists and } q \in Q_{jk} \\ \delta_{ij}(\sigma, q) \text{ if it exists and } q \in Q_{ij} - \{X_j, q_1, \ldots, q_n\} \\ \text{undefined otherwise.} \end{cases} \end{cases}$$

This submachine G_{ik} is well defined. Any possible ambiguity has been eliminated by separately dealing with the states $\{X_j, q_1, \ldots, q_n\}$ in the definition of δ_{ik}. G_{ik} is obtained by always following the partial transition function of $M_D^o(G, X_j, X_k)$ as a default behavior, and following the partial transition function of $M_D^o(G, X_i, X_j)$ otherwise whenever possible. First, the machines $M_D^o(G, X_i, X_j)$ and $M_D^o(G, X_j, X_k)$ are trim. Second, G_{ik} is constructed by forward propagation; therefore, all the states of G_{ik} are accessible with respect to the initial state X_i and are coaccessible with respect to the marked state X_k. Therefore, G_{ik} is trim.

Moreover, G_{ik} is controllable. Indeed, the partial transition function δ_{ik} says that as long as the system has not reached a state of the set $\{X_j, q_1, \ldots, q_n\}$, it follows the partial transition function of δ_{ij}. Due to the DP-Optimality of $M_D^o(G, X_i, X_j)$, the system will always reach a state of the set $\{X_j, q_1, \ldots, q_n\}$ with a finite cost. Indeed, if the system never visits a state in $\{q_1, \ldots, q_n\}$, it will eventually reach X_j. Let us call q the first state of the set $\{X_j, q_1, \ldots, q_n\}$ that is visited by the system as it evolves. At this point, the default partial transition function becomes δ_{jk}; therefore, the system will eventually reach the marked state X_k with a finite cost since the submachine $M_D^o(G, X_i, X_j)$ is DP-Optimal. Since the cost of reaching X_k from q is finite, the overall cost of reaching X_k is necessarily finite. From lemma (3.3), G_{ik} is controllable.

Finally, G_{ik} has no positive cost cycles. $M_D^o(G, X_i, X_j)$ and $M_D^o(G, X_i, X_k)$ do not have positive cost cycles (by definition of DP-Optimality). As we have described previously, before the system reaches a state of $\{X_j, q_1, \ldots, q_n\}$ for the first time, it will not complete a positive cost cycle (from the DP-Optimal nature of $M_D^o(G, X_i, X_j)$). After the system reaches a state of $\{X_j, q_1, \ldots, q_n\}$ for the first time, it will not complete a positive cost cycle either (from the DP-Optimal nature of $M_D^o(G, X_j, X_k)$). Therefore, no new cycles have been introduced. The only cycles that may exist in G_{ik} are those of $M_D^o(G, X_i, X_j)$ and $M_D^o(G, X_j, X_k)$.

Given that G_{ik} is trim, controllable, and contains no cycles of positive cost in G, FSM G_{ik} satisfies the preconditions of theorem (3.4), and there exists an optimal submachine of G_{ik}. Following theorem (3.6), there also exists a DP-Optimal submachine $M_D^o(G, X_i, X_k)$ of G_{ik}.

The proof of the *triangular inequality* relies on what we have said previously. The cost of reaching a state of the set $\{X_j, q_1, \ldots, q_n\}$, from the initial state X_i, is less than $c_{sup}^g(M_D^o(G, X_i, X_j))$ (equality is possible but not necessary when X_j is reached). Once one of the states $\{X_j, q_1, \ldots, q_n\}$ has been reached, the cost for the system to reach the marked state X_k is less than $c_{sup}^g(M_D^o(G, X_j, X_k))$ (equality is possible but not necessary when the system visits X_j) because the corresponding machine is DP-Optimal. More formally, let us take a trace s of events that leads from the initial state X_i to the final state X_k, i.e., such that $\delta_{ik}(s, X_i) = X_k$. As seen earlier, s visits at least one state of the set $\{X_j, q_1, \ldots, q_n\}$. Let us call it q again. We can now subdivide s into s_1 and s_2 such that $s = s_1 s_2$ and $\delta_{ik}(s_1, X_i) = q$ and $\delta_{ik}(s_2, q) = X_k$. From the DP-Optimality of the two submachines $M_D^o(G, X_i, X_j)$ and $M_D^o(G, X_j, X_k)$, for all s such that $s = s_1 s_2,\ \delta_{ik}(s_1, X_i) = q,\ \delta_{ik}(s_2, q) = X_k$, we can compare:

$$\begin{cases} c^g(X_i, M_D^o(G, X_i, X_j), s_1) \leq c_{sup}^g(M_D^o(G, X_i, X_j)), \\ c^g(q, M_D^o(G, X_j, X_k), s_2) \leq c_{sup}^g(M_D^o(G, X_j, X_k)). \end{cases} \tag{9}$$

Since this is true for all traces leading from X_i to X_j, we can deduce the triangular inequality. ■

The following corollary uses the construction in the proof of the previous lemma (4.7) to introduce a necessary condition for the existence of a Stepwise DP-Optimal scheduler.

Corollary 4.8 *If G_{des}^k does not exist, then there exists no subscheduler that makes the system evolve from q_0 to X_k, should it be indirectly via states of $\mathcal{X}$.*

Proof: The proof is done by contradiction. Assume that G^k_{des} does not exist and that there exists a scheduler that makes the system evolve from q_0 to the marked state X_k, through states of $\mathcal{X}$. From the construction used in the proof of lemma (4.7), we can generate a machine G_k that makes the system evolve from the initial state q_0 to the marked state X_k. The existence (by construction) of this submachine and its trim, acyclic and controllable nature indicate (by theorem (3.4)) the existence of a DP-Optimal submachine G^k_{des}, which contradicts the first assumption. ■

As a consequence of these results, we can ensure that a state X_k is accessible in an optimal way if and only if G^k_{des} exists. We are now able to give the necessary and sufficient conditions of the existence of a Stepwise DP-Optimal scheduler. This is stated by theorem (4.9).

Theorem 4.9 *Let G be a FSM and $\mathcal{X}$ be the set of n marked states of G, then there exists a corresponding Stepwise DP-Optimal scheduler $A \in \mathcal{M}^{sc}_D(G, \mathcal{X})$ if and only if the n DP-Optimal submachines G^i_{des} of G exist for all $X_i \in \mathcal{X}, i \in [1, \ldots, n]$.*

Proof: The necessary condition is given by corollary (4.8), which states that if there is a state X_i of $\mathcal{X}$ such that there does not exist a DP-Optimal submachine G^i_{des}, then there is no way to reach this state with a finite cost (thus in a DP-Optimal way) and the goal cannot be achieved. All the states of $\mathcal{X}$ cannot be visited, since one of them cannot be visited.

The condition is sufficient since FSM A, such that

$$A = G^1_{des} \circ (X_1 \xrightarrow{Reset} q_0) \circ G^2_{des} \circ (X_2 \xrightarrow{Reset} q_0) \circ \ldots \circ G^{i-1}_{des} \circ (X_{i-1} \xrightarrow{Reset} q_0) \circ G^i_{des} \circ \ldots \circ G^n_{des} \circ (X_n \xrightarrow{Reset} q_0)$$

visits all the states of $\mathcal{X}$. A is then a possible scheduler allowing the achievement of the goal. ■

This theorem implies that the Stepwise DP-Optimal problem has a solution when there exists a DP-Optimal submachine for each of the X_i. Besides, if a Stepwise DP-Optimal solution exists, it need not be unique in general. There is no notion of a maximal Stepwise DP-Optimal scheduler, as in the DP-Optimal problem [18]. The problem of finding one of the optimal schedulers is explored in §5.

5 Determination of a Stepwise DP-Optimal scheduler

In this section, we need to assume that the occurrence costs are strictly positive: $\forall \sigma \in \Sigma \; c_e(\sigma) > 0$. This assumption is necessary when we use the **DP-Opt** algorithm in order to ensure polynomial complexity. We also assume that a DP-Optimal submachine exists for all the states $X_i \in \mathcal{X}$. From here on, G^i_{des} will denote the maximal DP-Optimal submachine of the particular FSM $G_i = \langle \Sigma, Q, q_0, X_i, \delta \rangle$ output by the **DP-Opt** algorithm. We take advantage of the DP-Optimal structure of each of the G^i_{des}. We explore the possibility of starting the system at q_0, reaching a state X_i, and instead of doing a *Reset*, continuing the graph to a state X_j. To do so, we convert the problem to a path-cost minimization problem on a graph equivalent to a Traveling Salesman Problem (TSP).

5.1 Modeling of the problem

In order to convert the Stepwise DP-Optimal problem into a path-cost minimization problem, we use the **DP-Opt** algorithm introduced in §3.2. This algorithm computes, for each $X_i \in \mathcal{X}$, the DP-Optimal submachine G^i_{des}. Moreover, during this computation, a state X_j belonging to $\mathcal{X}$ can be reached. Due to the DP-Optimality definition, the algorithm also gives the DP-Optimal submachine between X_j and X_i. The minimum worst inevitable cost between these two states can

be collected as well. The next algorithm computes the DP-Optimal submachines between each states of $\mathcal{X} \cup \{q_0\}$, as well as the optimal costs from each state to another.

Matrix generation program

(I) INPUT: $G = \langle \Sigma, Q, q_0, \mathcal{X}, \delta \rangle$, OUTPUT: $C \in \mathbb{R}^{n+1} \times \mathbb{R}^{n+1}$

(II) INITIALIZE $E = \mathcal{X}$

(III) INITIALIZE MATRIX C: $C[i,j] = \infty$, $j \neq 0$, AND $C[i,0] = 0$ (*Reset*)

(IV) IF $E = \emptyset$ THEN STOP ELSE PICK ANY $X_i \in E$ AND UPDATE $E \leftarrow E - \{X_i\}$

(V) CALL SUBPROGRAM **DP-Opt** WITH $G = \langle \Sigma, Q, q_0, X_i, \delta \rangle$.
IF $G^i_{des} = \langle \Sigma, Q_i, q_0, X_i, \delta_i \rangle$ EXISTS, UPDATE C: $C[0,i] = c^g_{sup}(G^i_{des})$
IF G^i_{des} DOES NOT EXIST, THE PROCESS TERMINATES (THEOREM 4.9)

(VI) $\forall X_j \in Q_i$, UPDATE C: $C[j,i] = c^g_{sup}(M^o_D(G, X_j, X_i))$

(VII) GOTO (IV)

This algorithm performs the program **DP-Opt** once for each state of interest in $\mathcal{X}$. Given that there are n states in $\mathcal{X}$, and that the algorithm **DP-Opt** is to the order of $\mathcal{O}(|Q|^2|\Sigma|log(|\Sigma|) + |Q|^3|\Sigma|)$, the algorithm that generates all the maximal DP-Optimal submachines $(G^i_{des})_{i\in[1,\ldots,n]}$ is to the order of $\mathcal{O}(n|Q|^2|\Sigma|log(|\Sigma|) + n|Q|^3|\Sigma|)$. Note that point (VI) of the previous algorithm can actually be performed during the computation of the algorithm **DP-Opt** with no significant increase in complexity (point V) in the **DP-Opt** program; refer to Appendix A.

From the previous algorithm, the matrix $C \in \mathbb{R}^{n+1} \times \mathbb{R}^{n+1}$ has the following form:

- $C[i,i] = \infty$,
- $C[i,0] = 0, i \neq 0$,
- $C[0,i] = c^g_{sup}(G^i_{des}), i \neq 0$,
- $C[k,i] = \begin{cases} c^g_{sup}(M^o_D(G, X_k, X_i)) \text{ if it exists,} \\ \infty \text{ otherwise.} \end{cases}$

From additivity property (4.6), the cost of a scheduler $A = A_1 \circ A_2 \circ \ldots \circ A_l$ of $\mathcal{M}^{sc}_D$ is equal to:

$$C^{sc}_{sup}(A) = \sum_{k=1}^{l} c^g_{sup}(A_k) = \sum_{k=1}^{l} c^g_{sup}(M^o_D(G, X_{d_{k-1}}, X_{d_k})) = \sum_{k=1}^{l} C[d_{k-1}, d_k] \tag{10}$$

Considering equation (10), the new optimization problem is now reduced to finding a path with a minimal cost in the directed graph associated with the matrix C. This closely resembles the TSP with the slight difference that multiple visits to states of $\mathcal{X}$ are possible. In this new problem, the "cities" are represented by the set of nodes $\mathcal{X}$, and the "streets" are represented by machines $(G^i_{des})_{i\in[1,\ldots,n]}$ and $M^o_D(G, X_i, X_j)$ when these are available. The costs of these paths are given by the maximum costs for each machine, i.e., the $(c^g_{sup}(G^i_{des}))_{i\in[1,\ldots,n]}$ and the $(c^g_{sup}(M^o_D(G, X_i, X_j)))_{i,j\in[1,\ldots,n]}$. Figure (5.1) illustrates this conversion from the graph of the FSM to the reachability graph.

Note that some elements of C might be equal to ∞ after all G^i_{des} have been computed, which does not mean that the corresponding DP-Optimal submachines do not exist. This means that they have not been computed in the algorithm **DP-Opt**. Indeed, let us suppose that some $M^o_D(G, X_i, X_j)$

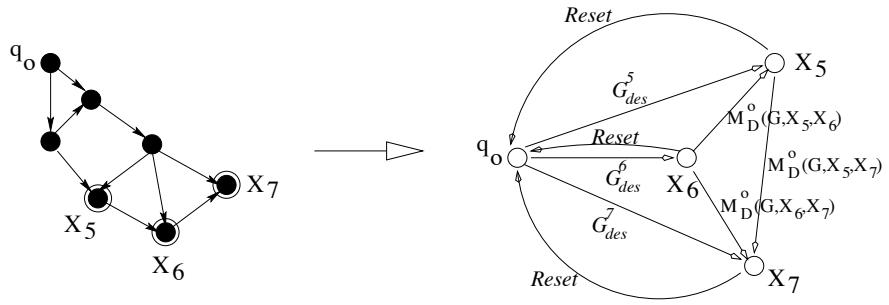


Figure 5: Conversion from the FSM to a reachability graph on the marked states

has not been computed (and that therefore $C[i,j] = \infty$). It means that it is less costly to perform a *Reset* from state X_i to state q_0, and to reach X_j through G^j_{des}. Another way of seeing this is to look at what the **DP-Opt** algorithm does. It backtracks from the marked state, say X_j. If it reaches X_i before q_0, this means that the cost from X_i to X_j is less than the cost from q_0 to X_j, in which case it is less costly to go directly from X_i to X_j than to reset the system. On the other hand, if state X_i is not reached when the algorithm reaches q_0 during its backtracking, it means that the cost to go from X_i to X_j is greater than the cost of reseting the system (0 in our case[4]) and taking the DP-Optimal submachine G^j_{des}. This explains why these particular paths are not taken into account as possible paths and are directly replaced in the matrix C by an infinite cost (the machine $M^o_D(G, X_i, X_j)$ never constitutes an optimal subscheduler allowing the visit of the two states X_i and X_j).

5.2 Generation of the Stepwise DP-Optimal scheduler

The problem of finding a Stepwise DP-Optimal scheduler A_0 has been brought down to solving an instance of the TSP, a classic combinatorial optimization problem. Many methods exist to solve the problem in an acceptable amount of time [7]. We specify once more the conditions in which we solve the TSP. The costs of the paths are all non-negative. The nodes of $\mathcal{X}$ must be visited *at least once*. One requirement of the TSP is that the salesman come back to the city he started from. This condition does not change anything to our problem since this maps to a *Reset*, which has null cost in our model. Finally, note that the cost matrix C, given in §5.1, is not necessarily symmetric.

Transformation: The first step is to transform our modified version of the TSP, where we can visit a node more than once (but at least once), into an ordinary TSP where we must visit each node exactly once. This is typically done by transforming the matrix C into a matrix C', called the all-pairs shortest-paths matrix [7]. Many techniques exist to perform such a computation. Among them is the Floyd-Warshall algorithm [7], which runs in a worst case of $\mathcal{O}(n^3)$ where n represents the number of vertices. Once the all-pairs shortest-paths matrix C' is obtained, we can feed it to a TSP solver.

C and C' have the same dimension, but represent different features of the graph. C contains, as non-infinite elements, the costs of the links that actually exist in the graph of the TSP. C' contains the minimum costs necessary to go from one marked state to another, along DP-Optimal submachines. C' is a reachability matrix; whereas C is a connectivity matrix. Notably, C' shows

[4]See section (5.3) for a discussion on when the *Reset* event has a strictly positive occurrence cost.

if states can be reached by using the *Reset* event. Concretely, to obtain C' from C, one only needs to replace any infinite value in C by the value in the same column in the first line (the cost of the G^i_{des} associated with column i).

Resolution of the TSP: The actual solving of the TSP from matrix C' can be done by using several methods. The most common method is the Branch & Bound method (chapters 9 and 10 of [9]). An outline of the method is given in Appendix B. The worst-case complexity for solving the TSP is $(n+1)!$. However the Branch & Bound method is expected to give a solution to the TSP in a tolerable amount of time[5].

The principle of the Branch & Bound method is quite natural. A branching strategy and a bounding strategy are used alternatively. The branching strategy consists of forcing a supplementary constraint on the system, usually done by forcing a set of subpaths in the graph. This allows to find a solution that is suboptimal in general but that provides a good heuristic to narrow down the search. The bounding strategy focuses on finding a lower bound to the cost of the optimal solution, by relaxing one of the constraints of the problem, usually done by relaxing the constraint that the solution must be a tour. The branching yields a search tree, and the bounding yields a way of quickly finding a suboptimal solution which is close to the optimal solution of the problem. The final solution is however optimal.

We also note that it is possible to obtain more than one solution to the TSP, by modifying the algorithm. We can thus obtain all the possible solutions of minimum cost. The interest behind this is to be able to have a choice in the nodes that we will be starting with. For example, in an FSM test problem, we may want to start by checking the nodes which are less likely to be flawed.

Restitution of the Stepwise DP-Optimal scheduler: From a solution of the TSP, we now build a corresponding Stepwise DP-Optimal scheduler.

The resolution of the TSP provides an optimal solution that gives the ordering in which the states should be visited so as to minimize the worst-case cost. A solution is under the form of a set of $n+1$ pairs (there are $n+1 = |\mathcal{X} \cup \{q_0\}|$ states), in which each state appears exactly once as an initial state and exactly once as a final state of a pair. For pairs (X_i, X_j) that represent a physically existing submachine $M^o_D(G, X_i, X_j)$, i.e., for which $C[i,j] < \infty$, it is sufficient to map these pairs to their associated submachine. As for the pairs (X_i, X_j) that do not map to an existing DP-Optimal submachine, i.e., those for which $C[i,j] = \infty$ and $C'[i,j] < \infty$, they are divided into two pairs, namely, (X_i, q_0) and (q_0, X_j). The first is mapped to a *Reset* to the initial state, and the second is mapped to the DP-Optimal submachine G^j_{des}.

Theorem 5.1 *Given a solution of the TSP, by adopting the previous mapping, the obtained scheduler is Stepwise DP-Optimal.*

Proof: The initial solution of the TSP with respect to the matrix C' is given by a tour of the form $\{(q_0, X_{i_1}); (X_{i_1}, X_{i_2}); \ldots (X_{i_j}, X_{i_{j+1}}); \ldots; (X_{i_n}, q_0)\}$ with a corresponding cost $TSP(C') = C'[0, i_1] + C'[i_1, i_2] + \ldots + C'[i_j, i_{j+1}] + \ldots + C'[i_n, 0]$. Consider now the transformation previously adopted. If the pair (X_i, X_j) originally exists, i.e., $C[i,j] < \infty$, then the path is admissible in

[5]The solving of the TSP does not imply too much overhead, especially because all the processing described in this paper is done off-line. To give a feel of the time complexity of this method, a 1,000 node fully-connected TSP can be solved in about 20 minutes on a standard workstation. We again emphasize that the number of nodes in the TSP, $n+1$, is the number of marked states in the DES G of interest, not the number of states in G, which is denoted as $|Q|$.

the original problem and we replace the pair by the submachine $M_D^o(G, X_i, X_j)$, where the corresponding cost $c_{sup}^g(M_D^o(G, X_i, X_j))$ is equal to $C[i, j]$. If the pair (X_i, X_j) does not map to an existing DP-Optimal submachine, i.e., $C[i, j] = \infty$, then we need to *Reset* the system before directly going to X_j through G_{des}^j. The *triangular inequality* of lemma (4.7) ensures that in this case, $C'[i, j] = c_{sup}^g(G_{des}^j)$. The pair is then replaced by the subscheduler $(X_i \xrightarrow{Reset} q_0) \circ G_{des}^j$, with the corresponding cost equal to $c_{sup}^g(G_{des}^j)$.

We then obtain a new sequence of pairs, with a cost equal to $TSP(C')$ but for which all the submachines actually exist in the original problem. The DP-Optimality of each submachine of the scheduler is given by construction, since we only consider submachines of the form $M_D^o(G, X_i, X_j)$ or G_{des}^j. The minimal cost of the scheduler is ensured by the optimality of the TSP solution and by the fact that the mapping does not add new costs. ■

An interesting property is given next. It states that all the submachines that constitute a Stepwise DP-Optimal scheduler are directly derived from all the DP-Optimal submachines built during the computation of the matrix C (section (5.1)).

Proposition 5.2 *A Stepwise DP-Optimal scheduler A_o obtained by the TSP solution is composed of exactly n different DP-Optimal submachines (not counting the possible Resets of the system). Moreover, all these submachines are obtained from the DP-Optimal submachines $(G_{des}^i)_{i\in[1,\ldots,n]}$ computed during the matrix generation step.*

Proof: The general solution of the TSP for the matrix C' is a tour of the form $\{(q_0, X_{i_1}); (X_{i_1}, X_{i_2}); \ldots; (X_{i_j}, X_{i_j}); \ldots; (X_{i_n}, q_0)\}$. Note that there are exactly $n + 1$ pairs in this tour (but the last pair is a trivial one, i.e., a *Reset*). If a pair (X_i, X_j) originally exists, i.e., if $C[i, j] < \infty$, then the path is in the original problem and it is replaced by the submachine $M_D^o(G, X_i, X_j)$. If not, i.e., if $C[i, j] = \infty$, then the system is reseted before directly going to X_j through G_{des}^j. The pair is then replaced by the subscheduler $(X_i \xrightarrow{Reset} q_0) \circ G_{des}^j$. The n pairs are then replaced by either the DP-Optimal submachine $M_D^o(G, X_i, X_j) = Trim(G_{des}^j, X_i, X_j)$, or by the subscheduler $(X_i \xrightarrow{Reset} q_0) \circ G_{des}^j$. The final solution of our problem has then exactly n non-trivial submachines that can be obtained from the n DP-Optimal submachine $(G_{des}^i)_{i=[1,\ldots,n]}$ of G, by a trim operation.

■

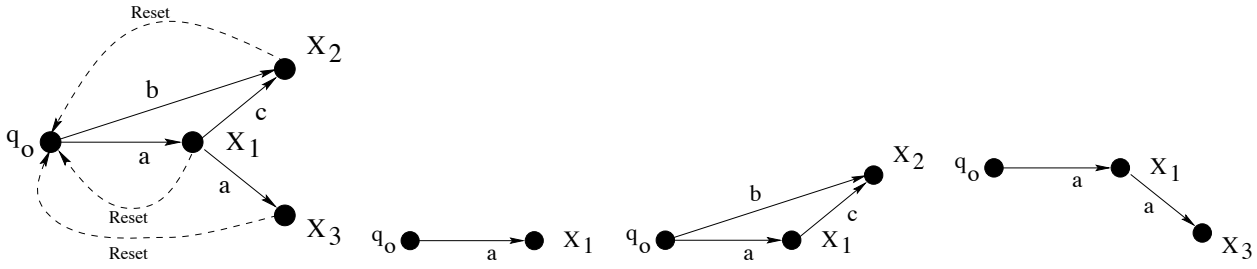
Corollary 5.3 *In a Stepwise DP-Optimal scheduler obtained by the TSP solution, the states of $\mathcal{X}$ are visited exactly once by the Stepwise DP-Optimal scheduler.*

Proof: From proposition (5.2), we know that there are n DP-Optimal submachines used in a Stepwise DP-Optimal scheduler when there are n marked states. Each DP-Optimal submachine has a unique marked state that is different for all the submachines. Also, given a solution to the TSP, we know that each marked state will appear exactly once in the right component of a pair and exactly once in the left component of a pair. The solution is a tour; therefore, the marked states are visited only once by the scheduler. ■

We wish to draw attention to the following fact. The Stepwise DP-Optimal scheduler visits each marked state exactly once when it is obtained from the TSP solution. However, the system itself, through its evolution described by the FSM G, may visit a marked state of G more than once. This comes from the fact that the scheduler is constructed on C', the all-pairs shortest-paths matrix,

whereas the behavior of the system modeled by the FSM G should be observed at a less abstract level, namely at the level of the FSM, G.

Let us illustrate this fact with an example. Consider the following FSM given in figure (6(a)).



(a) FSM G (b) G^1_{des} (c) G^2_{des} (d) G^3_{des}

Figure 6: Difference between the scheduler and the system

Assume the TSP solution is the set of pairs $\{(q_0, X_1); (X_1, X_2); (X_2, X_3); (X_3, q_0)\}$. The associated Stepwise DP-Optimal scheduler is $A_o = G^1_{des} \circ M^o_D(G, X_1, X_2) \circ (X_2 \stackrel{Reset}{\longrightarrow} q_0) \circ G^3_{des}$. The scheduler does visit a marked state exactly once. However, when the *system* evolves to visit all the marked states, it might perform the following ordered visit of states: $q_0, X_1, X_2, q_0, X_1, X_3$, in which case we can say that it visits X_1 twice. This distinguishes the scheduler from the system.

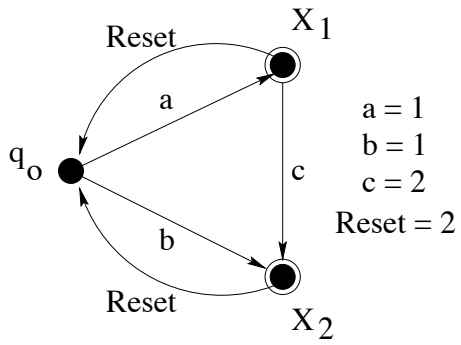
Moreover, there exists another Stepwise DP-Optimal scheduler that visits all the marked states: $A'_o = G^1_{des} \circ M^o_D(G, X_1, X_2) \circ (X_2 \stackrel{Reset}{\longrightarrow} q_0) \circ G^1_{des} \circ M^o_D(G, X_1, X_3)$. This scheduler does not visit each state exactly once, since it visits X_1 twice. This scheduler has not been derived from the TSP solution. This is why we specify that some of the properties hold only for schedulers derived from the TSP solution. On the other hand, some results have been derived in the general case where schedulers are not always derived from the TSP solution (see lemma (6.2) for example).

5.3 Case of a non-zero occurrence cost for the *Reset* event

In the framework of our theory, we have created and added a new event, namely the *Reset* event. We have also assumed that its (occurrence and disabling) costs were zero. The assumption that it is free to disable the *Reset* event is natural. On the other hand, its occurrence being cost-free may appear restrictive. For that purpose, we here show how the theory could be modified if we were to consider a non-zero occurrence cost for the *Reset* event.

The only place in our theory where a non-zero occurrence cost would intervene is when we recover the Stepwise DP-Optimal scheduler from the solution of the TSP. Indeed, we cannot map a pair (X_i, X_j) for which $C[i,j] = \infty$ but $C'[i,j] < \infty$ to the concatenation of submachines $(X_i \stackrel{Reset}{\longrightarrow} q_0) \circ G^j_{des}$, since it may be that we lose optimality when doing so. Let us give an example where $c_e(Reset) \neq 0$ (figure (7)).

The costs that are given in figure (7) are the occurrence costs. For simplicity, we assume that the control costs are all 0. If we run exactly the same computation as previously (notably, if we use the same algorithm and assume that $c_e(Reset) = 0$), we find that the Stepwise DP-Optimal scheduler is



Submachine	c^g_{sup}
G^1_{des}	1
G^2_{des}	1
$M^o_D(G, X_1, X_2)$	2
$X_1 \stackrel{Reset}{\longrightarrow} q_0$	2
$X_2 \stackrel{Reset}{\longrightarrow} q_0$	2

Figure 7: Simple example when $c_e(Reset) \neq 0$

$$A_o = G^1_{des} \circ (X_1 \stackrel{Reset}{\longrightarrow} q_0) \circ G^2_{des} \qquad c^{sc}_{sup}(A_o) = 4 \tag{11}$$

We do not show the final *Reset* which is optional, since we only want to reach all the marked states, which has been achieved once arrived at X_2. We observe that there is another scheduler which achieves the goal of visiting all the marked states with a lower global cost:

$$A_o = G^1_{des} \circ M^o_D(G, X_1, X_2) \qquad c^{sc}_{sup}(A_o) = 3 \tag{12}$$

Therefore, the methodology that was first followed is not correct. The change that should be made is to take into account the occurrence cost of the *Reset* event in the algorithm **DP-Opt**, so that the triangular inequality still holds. The backtracking that is done from the marked states (X_1 and X_2) should go further than the previous stopping condition of reaching the initial state q_0. The stopping condition should not be on the the initial state being reached, but on the cost function. The algorithm should stop backtracking once the residual cost, after having reached q_0 in the backtracking, is greater than the occurrence cost of the *Reset* event. Note that this occurrence cost can possibly be different for each of the marked states. This yields the Stepwise DP-Optimal scheduler in the general case of a non-zero occurrence cost for the *Reset* event.

5.4 Case of acyclic system model

Let us assume that the only cycles allowed in G are induced by the *Reset* event; in other words, the graph G without the *Reset* event is acyclic. The notations used are as defined in the prior sections. Some consequences of this assumption are:

1. For each of the states of $\mathcal{X}$, the maximal DP-Optimal submachine of G with respects to X_i can now be computed in $\mathcal{O}(|Q||\Sigma|log(|\Sigma|)$. The complete algorithm in the acyclic case can be found in §5.2 of [16].

2. As the FSM G is no longer cyclic, the graph G is unidirectionally oriented. It follows that if a DP-Optimal submachine $M^o_D(G, X_i, X_j)$ exists w.r.t. the states X_i and X_j, then the DP-Optimal submachine $M^o_D(G, X_j, X_i)$ does not exist. Consequently, the matrix C is at least half empty and the resolution of the corresponding TSP is easier, since fewer valid state trajectories need to be considered.

3. Finally, if we now consider a Stepwise DP-Optimal scheduler of G, A_o, such that

$$A_o = A_1^1 \circ \ldots \circ A_{k_1}^1 \circ Reset \circ \ldots \circ A_1^i \circ \ldots \circ A_{k_i}^i \circ Reset \circ \ldots \circ A_1^l \circ \ldots \circ A_{k_l}^l \circ Reset$$

then, $\forall i \in [1, \ldots, l], \forall j \in [i_1, \ldots, i_{k_i}]$, the subscheduler $A_j^i \circ A_{j+1}^i$ is a submachine of G, i.e., the $\circ$ operation is closed and well defined in the acyclic case. Consequently, the Stepwise DP-Optimal scheduler, A_o, can be rewritten $A_o = A^1 \circ Reset \circ \ldots \circ A^l \circ Reset$, where the $(A^i)_{i \in [1,\ldots,l]}$ are submachines of G.

These observations illustrate various simplifications that can be performed in the particular case where the system to be controlled is acyclic. Some further work should be performed in this direction in order to fully take advantage of the acyclicity of G.

6 Some simplifications of the TSP resolution

In order to solve the Stepwise DP-Optimal problem, we have to solve the corresponding TSP for the matrix C. The TSP is an NP-complete problem. It is then greatly advantageous to find some simplification methods, taking advantage of the special structure of a Stepwise DP-Optimal scheduler, in order to reduce the computational complexity of the corresponding TSP without loss of global optimality.

6.1 Divide and conquer

In some cases, it is possible to divide the matrix C into several smaller ones. In such cases, it suffices to solve the TSP on each of these sub-matrices. The following proposition states the necessary and sufficient conditions for this simplification.

Proposition 6.1 *Assume there exists a partition of* $\mathcal{X} = \cup_{k \in [1,\ldots,l]}(\mathcal{X}_k)$ *such that* $\forall k_1, k_2 \in [1, \ldots, l]$, *and* $\forall X_i \in \mathcal{X}_{k_1}$ *and* $\forall X_j \in \mathcal{X}_{k_2}$, *the submachine* $M_D^o(G, X_i, X_j)$ *is not defined.*

If A_o *is a Stepwise DP-Optimal scheduler with respect to* $\mathcal{X}$, *then it is possible to find a set of schedulers* $A_{\mathcal{X}_k}$, *where each* $A_{\mathcal{X}_k}$ *is Stepwise DP-Optimal with respect to* $\mathcal{X}_k$ *with an optimal cost* $c_{sup}^{sc}(A_{\mathcal{X}_k})$ *to visit of all the states of* $\mathcal{X}_k$, *and such that* $A_o = \circ_{k=1}^{l} A_{\mathcal{X}_k}$ *with*

$$c_{sup}^{sc}(A_o) = \sum_{k=1}^{l} c_{sup}^{sc}(A_{\mathcal{X}_k}).$$

Proof: From the assumptions of proposition (6.1), there does not exist a submachine of the form $M_D^o(G, X_i, X_j)$, when X_i and X_j belong to two different subsets of the partition of $\mathcal{X}$. Therefore, after reordering the submachines according to the different subsets $(\mathcal{X}_k)_{k \in [1,\ldots,l]}$, the Stepwise DP-Optimal scheduler A_o is composed by subschedulers $(A_{\mathcal{X}_k})_{k \in [1,\ldots,l]}$, each of them belonging to $\mathcal{M}^{sc}(G, \mathcal{X}_k)$. Moreover, each subscheduler $A_{\mathcal{X}_k}$ is actually a Stepwise DP-Optimal scheduler for the particular subset $\mathcal{X}_k$. Indeed, assume that there exists a subscheduler $A_{\mathcal{X}_k}$ that is not Stepwise DP-Optimal. Consider A_{o_k} the Stepwise DP-Optimal scheduler for this particular set $\mathcal{X}_k$ (such a scheduler exists from theorem (4.9)). The cost of this new scheduler is strictly lower than the one of $A_{\mathcal{X}_k}$. Therefore, by substituting $A_{\mathcal{X}_k}$ by the new Stepwise DP-Optimal scheduler A_{o_k} in A_o, we obtain a new scheduler, visiting all the marked states at least once, with a cost strictly lower. There is a contradiction with the fact that A_o is optimal and then each $A_{\mathcal{X}_k}$ is Stepwise DP-Optimal. ■

In view of proposition (6.1), the global problem can be solved on each sub-matrix $C_k,\ k \in [1,\ldots,l]$, corresponding to the particular set of states $\mathcal{X}_k \cup \{q_0\}$. This is performed by the following algorithm. $Conn(X)$ represents the one-step connectivity list associated with any state of the FSM G, regardless of directionality of the transition linking X to another state. Formally, $Conn$ is a state to set function defined over $\mathcal{X} \to 2^X$. CC is a state to integer function defined over $\mathcal{X} \to [1,\ldots,n]$, giving, for each state X the index of the connectivity list it belongs to.

Div-Conq program

(I) INPUT: $\mathcal{X}$, $Conn(X)_{X\in\mathcal{X}}$; OUTPUT: PARTITION $(\mathcal{X}_i)_{i\in[1,\ldots,k]}$
(II) INITIALIZE $E = \mathcal{X}$, $k = 0$
(III) WHILE $E \neq \emptyset$,
  PICK ANY $X \in E$, UPDATE $E \leftarrow E - \{X\}$
  $k = k + 1$, $CC(X) = k$
  CALL SUBPROGRAM **DFS-Visit**$(k,\ X,\ E)$
(IV) FOR $j = 1 \to k$, INITIALIZE $\mathcal{X}_j = \emptyset$
(V) FOR $i = 1 \to n$, $\mathcal{X}_{CC(X_i)} \leftarrow \mathcal{X}_{CC(X_i)} \cup \{X_i\}$

DFS-Visit program

(I) INPUT: $k,\ X,\ E$
(II) FOR ALL $X' \in Conn(X) \cap E$,
  $CC(X') = k$, UPDATE $E \leftarrow E - \{X\}$
  DFS-Visit$(k,\ X',\ E)$
(III) RETURN **Div-Conq** PROGRAM

Let us explain the **Div-Conq** algorithm. The inputs are the set of marked states and the connectivity lists associated with each marked state[6]. The output is a partition. Basically, we are looking for the connected components of the set $\mathcal{X}$, while cutting the node q_0 away. E is a global variable containing the states of $\mathcal{X}$ that have not yet been processed. k is the incremented number of subsets in the partition. Each time we pick a state in (III), it means that we create a new subset of the partition. **DFS-Visit** simply does a Depth-First Search to find states that are connected (irrespective of the directionality, since we are only interested in finding the connected components) to the processed states and marks these states with the current partition label $CC(X_i)$. Line (IV) creates the empty subsets of the partition; whereas line (V) fills the subsets with the states X_i that have the same partition label $CC(X_i)$.

The update of E ensures that each state is treated exactly once. Moreover, the fact that we only consider $Conn(X) \cap E$ as states actually connected to the current visited state ensures both the convergence of **DFS-Visit** and the single treatment of each edge of the graph of the FSM (regardless of transition directions). Therefore, the complexity of the **Div-Conq** algorithm is $\mathcal{O}(n+\mathcal{E})$, where n is the number of states of the FSM, and $\mathcal{E}$ is the number of one-step connections (or edges) among the states of $\mathcal{X}$.

6.2 Terminal path simplification

We address here a property of the scheduler, that can lead to a simplification on the matrix C. This property states that if there exists a kind of "dead-end" in the graph of the matrix, then it is

[6] The computation of these lists can be easily extracted from the main program **DP-Opt**, line (V).

always better to follow this path until the end than to perform a *Reset* and come back to visit the end of this path later.

Lemma 6.2 *Assume that there exists a subset* $\mathcal{X}_i = (X_{i_k})_{k\in[1,\ldots,m]}$ *of* $\mathcal{X}$*, with* $m < n$ *and such that*

1. $\forall k \in [1,\ldots,m-1]$, $M_D^o(G, X_{i_k}, X_{i_{k+j}})$ *exists for* $j \in [1,\ldots,m-k]$,
2. $\forall k \in [2,\ldots,m]$, $M_D^o(G, X_{i_k}, X_{i_{k-j}})$ *does not exist for* $j \in [1,\ldots,k-1]$,
3. $\forall k \in [1,\ldots,m]$ *and* $\forall X_l \in \mathcal{X} - \mathcal{X}_i$, $M_D^o(G, X_{i_k}, X_l)$ *is not defined.*

Let A_o *be a Stepwise DP-Optimal solution. Then all the states of* $\mathcal{X}_i$ *are only visited once by* A_o*.*

Proof: From the assumptions, we can deduce that for a particular $X_{i_j} \in \mathcal{X}_i$, the only accessible states from X_{i_j} (without the *Reset*) which belong to $\mathcal{X}$ are $X_{i_k} \in \mathcal{X}_i$, with $j < k \leq m$. Note that X_m is automatically followed by q_0 since there exists no state in $\mathcal{X}$ that can be reached from X_m without performing a *Reset*. From now on, $X_{j_i} \to X_{j_k}$ will denote the existence of a DP-Optimal submachine between these two states. In the sequel, we assume that the states of $\mathcal{X}_i$ (and of all the subsets of $\mathcal{X}_i$) are ordered according to assumptions (1) and (2) (i.e., $X_{i_k} \to X_{i_j}$ for $j > k$, and $X_{i_k} \nrightarrow X_{i_j}$ for $j < k$). This total order relation over the state of $\mathcal{X}_i$ is denoted $\xrightarrow{\sqsubset}$. Figure **(6.2)** sums up the assumptions and represents possible connections between the states of $\mathcal{X}$. Dashed lines constitute possible transitions from a state to another (depending of the behavior of the system), whereas solid lines represent the actual connections between states that exist in view of condition 1. in lemma (6.2).

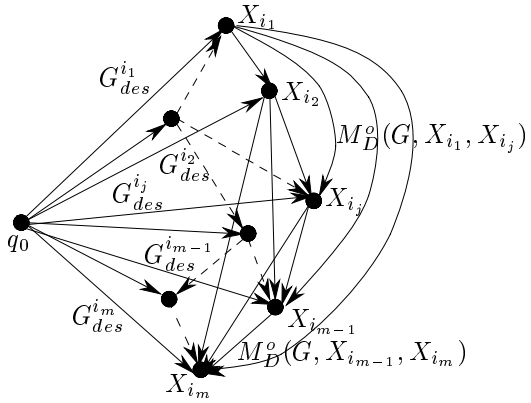


Figure 8: Example of terminal path in the graph associated with the matrix C

The proof proceeds by contradiction. Assume that there is at least one state X which is visited twice. Without loss of generality, we can assume that X is the last state of $\mathcal{X}_i$ visited twice by the Stepwise DP-Optimal scheduler A_o with respect to the ordering. The associated state trajectory d_0 is then given by:

$$d_0 = \overbrace{d_1' \to X_j}^{d_1} \to X \to X_{j_1} \to \ldots \to X_{j_l} \to \overbrace{q_0 \to d_2' \to X_j'}^{d_2} \to X \to X_{j_1'} \to \ldots \to X_{j_{l'}'} \to q_0 \to d_3,$$

where d_1', d_2' and d_3 represent particular sub-paths associated with the scheduler; $\{X_{j_1}, \ldots, X_{j_l}\}$ and $\{X_{j_1'}, \ldots, X_{j_{l'}'}\}$ are states contained in $\mathcal{X}_i$. Also, we have the following relations: $\forall j_i \in$

$[j_1,\ldots,j_l]X \xrightarrow{\sqsubset} X_{j_i}$ and $\forall j'_i \in [j'_1,\ldots,j'_l]X \xrightarrow{\sqsubset} X_{j'_i}$. By the additivity property given in 4.6, the cost associated with this trajectory is equal to:

$$\begin{aligned}\underline{c}(d_0) \;=\;& \underline{c}(d_1) + c^g_{sup}(M^o_D(X_j, X)) + \\ & c^g_{sup}(M^o_D(X, X_{j_1})) + c^g_{sup}(M^o_D(X_{j_1}, X_{j_2})) + \ldots + c^g_{sup}(M^o_D(X_{j_{l-1}}, X_{j_l})) + \\ & \underline{c}(d_2) + c^g_{sup}(M^o_D(X'_j, X)) + \\ & c^g_{sup}(M^o_D(X, X_{j'_1})) + c^g_{sup}(M^o_D(X_{j'_1}, X_{j'_2})) + \ldots + c^g_{sup}(M^o_D(X_{j'_{l'-1}}, X_{j'_{l'}})) + \\ & \underline{c}(d_3)\end{aligned}$$

Consider now the set $\{X_{j''_1}\ldots X_{j''_{l''}}\} = \{X_{j'_1}\ldots X_{j'_{l'}}\} \cup \{X_{j_1}\ldots X_{j_l}\}$ (note that these three sets are ordered according to the order relation over the states of $\mathcal{X}_i$: $\xrightarrow{\sqsubset}$). As X is the last element of $\mathcal{X}_i$ to be present twice, $\{X_{j_1}\ldots X_{j_{l'}}\}$ and $\{X_{j'_1}\ldots X_{j'_{l'}}\}$ form a partition of $\{X_{j''_1}\ldots X_{j''_{l''}}\}$, with $l'' = l + l'$. Consider now the path d, such that:

$$d = \overbrace{d'_1 \to X_j}^{d_1} \to X \to X_{j''_1} \to \ldots \to X_{j''_{l''}} \to \overbrace{q_0 \to d'_2 \to X'_j}^{d_2} \to q_0 \to d_3. \tag{13}$$

The scheduler corresponding to this path constitutes a valid scheduler since it allows the visit of all the states of $\mathcal{X}$ and all the submachines of this scheduler exist. Its associated cost is equal to:

$$\begin{aligned}\underline{c}(d) \;=\;& \underline{c}(d_1) + c^g_{sup}(M^o_D(X_j, X)) + c^g_{sup}(M^o_D(X, X_{j''_1})) + \\ & c^g_{sup}(M^o_D(X_{j''_1}, X_{j''_2})) + \ldots + c^g_{sup}(M^o_D(X_{j''_{l''-1}}, X_{j''_{l''}})) + \\ & \underline{c}(d_2) + \underline{c}(d_3)\end{aligned}$$

The difference between the cost associated with d_o and the one associated with d is:

$$\begin{aligned}\underline{c}(d_0) - \underline{c}(d) \;=\;& \left\{ \begin{array}{l} c^g_{sup}(M^o_D(X, X_{j_1})) + \\ c^g_{sup}(M^o_D(X_{j_1}, X_{j_2})) + \ldots + c^g_{sup}(M^o_D(X_{j_{l-1}}, X_{j_l})) + \\ c^g_{sup}(M^o_D(X'_j, X)) + \\ c^g_{sup}(M^o_D(X, X_{j'_1})) + c^g_{sup}(M^o_D(X_{j'_1}, X_{j'_2})) + \ldots + c^g_{sup}(M^o_D(X_{j'_{l'-1}}, X_{j'_{l'}})) \end{array} \right\} \quad (\alpha) \\ & - \quad \left\{ \; c^g_{sup}(M^o_D(X, X_{j''_1})) + c^g_{sup}(M^o_D(X_{j''_1}, X_{j''_2})) + \ldots + c^g_{sup}(M^o_D(X_{j''_{l-1}}, X_{j''_{l''}})) \; \right\} \quad (\beta)\end{aligned}$$

To obtain the contradiction, we want to prove that $\underline{c}(d_0) - \underline{c}(d) > 0$. To do that, we show that each term in (α) can be compared with a corresponding term in (β) and prove that the terms of (α) are either strictly greater, or greater than the terms of (β). As $l'' = l' + l$, (α) contains one term more than (β) (in fact it is $c^g_{sup}(M^o_D(X'_j, X))$). Moreover each state of $\{X_{j''_1}\ldots X_{j''_{l''}}\}$ appears only once in the right of the sub-expression in (β) as in (α). Consequently, for each term of the form $c^g_{sup}(M^o_D(X_{j_{i-1}}, X_{j_i}))$ in (α), there exists a term in (β), $c^g_{sup}(M^o_D(X_{j''_{k-1}}, X_{j''_k} = X_{j_i}))$. As $\{X_{j_1}\ldots X_{j_l}\} \subseteq \{X_{j''_1}\ldots X_{j''_{l''}}\}$, we can say that $X_{j_{i-1}} \to X_{j''_{k-1}} \to X_{j''_k} = X_{j_i}$ (note that $X_{j_{i-1}}$ and $X_{j''_{k-1}}$ can be equal), and consequently $c^g_{sup}(M^o_D(X_{j_{i-1}}, X_{j_i})) \geq c^g_{sup}(M^o_D(X_{j''_{k-1}}, X_{j''_k}))$. As $c^g_{sup}(M^o_D(X'_j, X)) > 0$, we can say that $\underline{c}(d_0) - \underline{c}(d) > 0$. The scheduler corresponding to the trajectory d has then a cost strictly lower that A_o, which contradicts the assumed optimality of A_o. As A_o has been constructed according to the fact that the state X was visited at least twice, we can conclude that X is present only once in any Stepwise DP-Optimal solution.

■

Following lemma (6.2), for the next property, we assume that each state of $\mathcal{X}_i$ is visited only once by the scheduler.

Proposition 6.3 *Under assumptions (1), (2) and (3) of lemma (6.2), the submachines* $(G_{des}^{i_k})_{k\in[2,\ldots,m]}$ *do not belong to the Stepwise DP-Optimal scheduler* A_o.

Proof: The proof proceeds by induction on the elements of $\mathcal{X}_i$.

- The base case is argued as follows. Assume that $G_{des}^{i_m}$ belongs to the scheduler A_o. Consider the particular state $X_{i_{m-1}}$. As the continuation of $X_{i_{m-1}}$ is either q_0 or X_{i_m} and that $G_{des}^{i_m}$ is in the scheduler, using the lemma (6.2), we can deduce that the corresponding state trajectory d_0 has the following form: $d_0 = d_1 \to X_{i_{m-1}} \to q_0 \to d_2' \to q_0 \to X_{i_m} \to q_0 \to d_3'$. Consider now the state trajectory $d = d_1 \to X_{i_{m-1}} \to X_{i_m} \to q_0 \to d_2' \to \ q_0 \to d_3'$. The difference between the cost associated with d_o and the one associated with d is :

  $$\underline{c}(d_0) - \underline{c}(d) = c_{sup}^g(G_{des}^{i_m}) - c_{sup}^g(M_D^o(X_{i_{m-1}}, X_{i_m})) > 0.$$

  The scheduler corresponding to the state trajectory constitutes a valid scheduler since it allows the visit of all the states of $\mathcal{X}$ and all the submachines of this scheduler exist. Moreover its global cost is lower than that of the A_o scheduler, which contradicts the assumed optimality of A_o. Then $G_{des}^{i_m}$ does not belong to the scheduler A_o.

- We now consider the general case:

  Assume (by induction) that for the $(m-k)^{th}$ last states, the $G_{des}^{i_{m-k'}}$ for $k' \leq k$ do not belong to a Stepwise DP-Optimal scheduler A_o. Assume that $G_{des}^{i_{k-1}}$ belongs to A_o and consider the state $X_{i_{k-2}}$. The state trajectory associated with the scheduler A_o can have various forms:

  • If $d_0 = d_1 \to X_{i_{k-2}} \to q_0 \to d_2 \to q_0 \to X_{i_{k-1}} \to d_3$, then it is obvious that the state trajectory d composed by $d_0 = d_1 \to X_{i_{k-2}} \to X_{i_{k-1}} \to q_0 \to d_2 \to q_0 \to d_3$ as a lower cost than d_0, which leads to a contradiction (same arguments as the ones used in the base case).

  • If $d_0 = d_1 \to X_{i_{k-2}} \to X_{j_1} \to \ldots \to X_{j_l} \to q_0 \to d_2 \to q_0 \to X_{i_{k-1}} \to X_{j_1'} \to \ldots \to X_{j_{l'}'} \to q_0 \to d_3$, with $\{X_{j_1} \ldots X_{j_l}\}$ and $\{X_{j_1'} \ldots X_{j_{l'}'}\}$ included in $\{X_{i_k} \ldots X_{i_m}\}$. As in lemma (6.2), we consider the set $\{X_{j_1''} \ldots X_{j_{l''}''}\} = \{X_{j_1'} \ldots X_{j_{l'}'}\} \cup \{X_{i_k} \ldots X_{i_m}\}$ (note that these three sets are ordered according to the order relation over the state of $\mathcal{X}_i$: $\stackrel{\sqsubset}{\longrightarrow}$). From lemma (6.2), $l'' = l + l'$.

  Consider now the state trajectory $d = d_1 \to X_{i_{k-2}} \to X_{i_{k-1}} \to X_{j_1''} \ldots X_{j_{l''}''} \to q_0 \to d_2 \to q_0 \to d_3$. Using lemma (4.7) and the same kind of arguments as the ones in lemma (6.2), we can easily prove that $\underline{c}(d_0) - \underline{c}(d) > 0$. Therefore, the scheduler corresponding to the trajectory d has then a cost strictly lower that A_o, which leads to a contradiction. Therefore $G_{des}^{i_{k-1}}$ does not belong A_o.

■

Proposition (6.3) deals with situations where a dead-end occurs. By dead-end, we mean a set of states $\{q_1, \ldots, q_n\}$ in which $\forall i \in [1, \ldots, n]$, $(q_j)_{j>i}$ are the only states coaccessible from q_i. If there exists a dead-end in the graph of marked states, the system will never enter that dead-end directly through one of the G_{des}^i, but will only enter indirectly from the initial state q_0. This means that no direct submachine of the type G_{des}^i will be used by a scheduler to enter a dead-end. Any visit to a state of the dead-end is done via a visit to a state that does not belong to the dead-end.

Due to the previous proposition (6.3), this simplification can be performed on the matrix C through the following algorithm

Term-Path program

(I) INPUT: $\mathcal{X}$, C, $(Pre(X))_{X\in\mathcal{X}}$, $(Post(X))_{X\in\mathcal{X}}$; OUTPUT: MATRIX C UPDATED
(II) INITIALIZE $CS = \emptyset$, $E = \mathcal{X}$, $A = \{X_i : \forall j\ C[i,j] = \infty\}$
(III) FOR ALL $X_i \in A$
  $D_i = \{X_i\}$
  CALL SUBPROGRAM **DFS-Path**$(X_i,\ D_i)$
(IV) $\forall X_i \in CS$, UPDATE MATRIX: $C[0,i] = \infty$

DFS-Path program

(I) INPUT: X, D
(II) INITIALIZE $Ind = \emptyset$
(III) FOR ALL $Y \in Pre(X) \cap E$
  IF $Post(Y) = D$ THEN **DFS-Path**$(Y,\ D + \{Y\})$
  ELSE $E \leftarrow E - \{Y\}$, $Ind \leftarrow Ind + \{Y\}$
(IV) IF $Ind = Pre(X)$, $CS = CS \cup \{D - D_{\|D\|}\}$

Let us describe the above **Term-Path** algorithm. The input is the set of marked states, the matrix C, the connectivity lists $Pre(X)$ and $Post(X)$ for all X defined as follows. $Pre(X_i) = \{X_j : C[j,i] \neq \infty\}$, and $Post(X_i) = \{X_j :\ C[i,j] \neq \infty\}$[7]. These can easily be obtained by extracting them from the main program **DP-Opt**. The variable A is initialized to be the set of states X for which $Post(X)$ is empty. These states of A are candidate states to be at the termination of terminal paths. D_i represents the possible terminal paths that are constructed by back-propagation from the elements of A. **DFS-Path** is a subprogram which performs a Depth First Search to find states that are connected to the current states according to assumptions (1), (2) and (3) of lemma (6.2) and the conditions of lines (III) and (III)(A) of the program. If any one is not satisfied, the node is erased from the global variable E. Otherwise, it is added to the current terminal path and **DFS-Path** is called recursively on that node.

With this simplification, the paths of the form $q_0 \rightarrow X_{i_k}$ do not constitute valid paths any longer and consequently, will not be taken into account as possible solutions in the corresponding TSP solution. This terminal path simplification can narrow down the search space when solving the TSP, since it adds "∞" in matrix C.

6.3 Pre-defined partial order for the visit of $\mathcal{X}$

Throughout §(5), we have assumed that we had no pre-specified order in which to visit the marked states in $\mathcal{X}$. This may be the case in several applications. However, in other applications, such as test-generation, we may be interested in the path taken by a system more than in the final state it reaches. The designer may want to enforce the system to follow a given path. The path would be characterized by the states it traverses, which would be marked. This would yield an ordering, not on the marked states, but on the subpaths themselves.

Let us take a simple example. Assume that we have the marked states $\{X_1, \ldots, X_{10}\}$ to visit in an optimal way. If we do not pre-specify the order in which they should be visited, the TSP will be solved on a 10×10 matrix. The designer may want to observe the behavior of the system when it visits states X_1 through X_3 in that order, X_4 through X_7 in that order, and X_7 through X_{10} in that order. This would reduce the TSP to a 3×3 matrix, abstracting away from the ten states to four macro-states: $\{q_0\}, \{X_1, X_2.X_3\}, \{X_4, X_5, X_6, X_7\} and \{X_8, X_9, X_10\}$. The solution

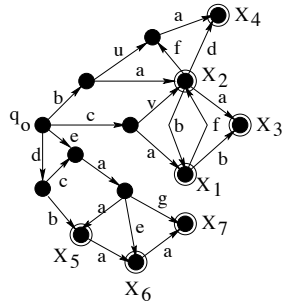
[7] Note that $Pre(X) \cup Post(X) = Conn(X)$.

thus obtained will not be Stepwise DP-Optimal per se. It will be optimal given the additional constraints imposed by the designer.

We suspect that more particular cases and simplifications could be found to take greater advantage of the structure of a scheduler.

7 Example

The following example is constructed to illustrate the essential stages of the optimal control problem for multiple marked states to visit. For the sake of simplicity, we have assumed that all control costs have zero cost for controllable events and infinite costs for uncontrollable events. This allows the use of the simplified version of the **DP-Opt** algorithm presented in Appendix A. We here consider a system modeled by the FSM G, which represents its legal behavior. In this example, there are 7 states denoted $(X_i)_{i\in[1,\dots,7]} = \mathcal{X}$ to visit in no particular order. Some costs are allocated to each of the events of the FSM G. The event costs and their status (controllable or not) are as depicted in figure (9).



Event	c_e	Remarks
a,f	1	Controllable
b,c,d,e	2	Controllable
g	4	Controllable
Reset	0	Controllable
u	3	Uncontrollable
v	2	Uncontrollable

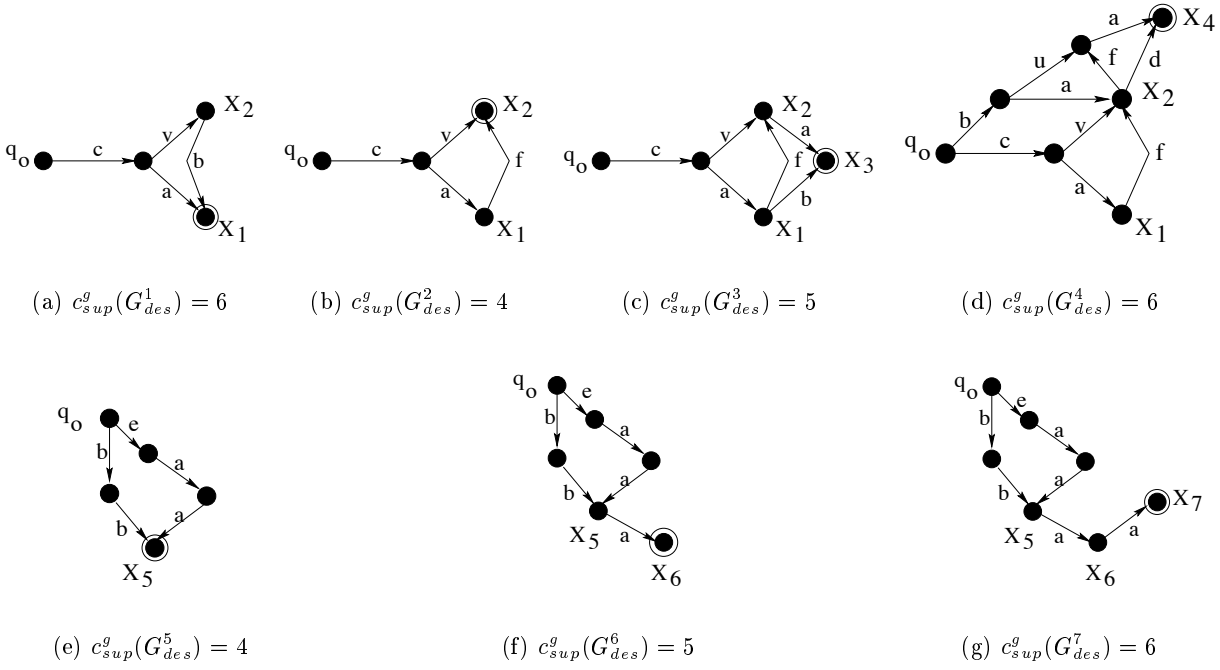
Figure 9: The initial system G and the event cost function

Note that in figure 9, the *Reset* events are not represented, but exist between each of the $(X_i)_{i\in[1,\dots,7]}$ and the initial state q_0. The first phase of the algorithm consists in computing the various DP-Optimal submachines $(G^i_{des})_{i\in[1,\dots,7]}$ for each of the final states of $\mathcal{X}$. This part is performed using the **DP-Opt** algorithm briefly described in §3.2 and Appendix A. The seven figures given next (figure (10)) correspond to the DP-Optimal submachines for each of the final states $(X_i)_{i\in[1,\dots,7]}$. We also give the worst inevitable cost for each submachine $(G^i_{des})_{i\in[1,\dots,7]}$.

By applying the algorithm described in §5.1, we obtain the matrix C, encoding the worst inevitable cost between two states X_i and X_j.

	q_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7
q_0	∞	**6**	**4**	**5**	**5**	**4**	**5**	**6**
X_2	0	**2**	∞	**1**	**2**	∞	∞	∞
X_3	0	∞	∞	∞	∞	∞	∞	∞
X_4	0	∞	∞	∞	∞	∞	∞	∞
X_5	0	∞	∞	∞	∞	∞	**1**	**2**
X_6	0	∞	∞	∞	∞	∞	∞	**1**
X_7	0	∞	∞	∞	∞	∞	∞	∞

Following proposition (6.1), we can see that $\mathcal{X}_1 = \{X_1, X_2, X_3, X_4\}$ and $\mathcal{X}_2 = \{X_5, X_6, X_7\}$ form a partition of the set of states $\mathcal{X} = (X_i)_{i\in[1,\dots,7]}$. Moreover, the states of $\mathcal{X}_2$ satisfy proposition



(a) $c^g_{sup}(G^1_{des}) = 6$ (b) $c^g_{sup}(G^2_{des}) = 4$ (c) $c^g_{sup}(G^3_{des}) = 5$ (d) $c^g_{sup}(G^4_{des}) = 6$

(e) $c^g_{sup}(G^5_{des}) = 4$ (f) $c^g_{sup}(G^6_{des}) = 5$ (g) $c^g_{sup}(G^7_{des}) = 6$

Figure 10: The DP-Optimal FSMs for the different state of $\mathcal{X}$

(6.3). Thus, in order to solve the TSP, we can now consider the two following matrices. Note that in the second one, we have replaced $C[0,6]$ and $C[0,7]$ by ∞ as stated by proposition (6.3). A full derivation of the first matrix is given in Appendix B. Appendix B shows the all-pairs shortest-paths matrix and traces the Branch & Bound method.

	q_0	X_1	X_2	X_3	X_4
q_0	∞	**6**	**4**	**5**	**5**
X_1	0	∞	**1**	**2**	**3**
X_2	0	**2**	∞	**1**	**2**
X_3	0	∞	∞	∞	∞
X_4	0	∞	∞	∞	∞

	q_0	X_5	X_6	X_7
q_0	∞	**4**	∞	∞
X_5	0	∞	**1**	**2**
X_6	0	∞	∞	**1**
X_7	0	∞	∞	∞

One solution (there are several) of the TSP for each sub-matrix is:

$$\{(q_0, X_4); (X_4, X_1); (X_1, X_2); (X_2, X_3); (X_3, q_0)\} \quad (14)$$

$$\{(q_0, X_5); (X_5, X_6); (X_6, X_7); (X_7, q_0)\} \quad (15)$$

This is the output of the TSP resolution method run on each of the subproblems. The optimal worst-case costs are 13 and 6, respectively. From theorem (5.1), there exist 2 Stepwise DP-Optimal schedulers A_{0_1} and A_{0_2}. We need to retrieve them from the output of the resolution of the TSP and build a global Stepwise DP-Optimal scheduler A_0.

We look at each one of the pairs and see if they correspond to a DP-Optimal submachine. In this case (q_0, X_4), (X_1, X_2), (X_2, X_3), and (X_3, q_0) correspond to G^4_{des}, $M^o_D(G, X_1, X_2)$, $M^o_D(G, X_2, X_3)$, and $X_3 \xrightarrow{Reset} q_0$, respectively. (X_4, X_1) does not have any associated DP-Optimal submachine. We decompose it: (X_4, X_1) becomes $(X_4 \xrightarrow{Reset} q_0)$ concatenated with G^1_{des}. An optimal DP-Optimal

scheduler A_{o_1} is the following:

$$A_{o_1} = G^4_{des} \circ (X_4 \stackrel{Reset}{\longrightarrow} q_0) \circ G^1_{des} \circ M^o_D(G, X_1, X_2) \circ M^o_D(G, X_2, X_3) \circ (X_3 \stackrel{Reset}{\longrightarrow} q_0) \tag{16}$$

For the second subproblem of the divide and conquer method, we also map the pairs to original DP-Optimal submachines, yielding scheduler A_{o_2}.

$$A_{o_2} = G^5_{des} \circ M^o_D(G, X_5, X_6) \circ M^o_D(G, X_6, X_7) \circ (X_7 \stackrel{Reset}{\longrightarrow} q_0) \tag{17}$$

Note that we use exactly 4 DP-Optimal submachines for the first subproblem and 3 for the second, as expected from Proposition (5.2)[8]. We finally generate the global Stepwise DP-Optimal scheduler A_o as the concatenation of the two subschedulers A_{o_1} and A_{o_2}, yielding the following scheduler, A_o.

$$\begin{aligned} A_o &= A_{o_1} \circ A_{o_2} \\ &= G^4_{des} \circ (X_4 \stackrel{Reset}{\longrightarrow} q_0) \circ G^1_{des} \circ M^o_D(G, X_1, X_2) \circ M^o_D(G, X_2, X_3) \circ (X_3 \stackrel{Reset}{\longrightarrow} q_0) \circ \\ &\quad G^5_{des} \circ M^o_D(G, X_5, X_6) \circ M^o_D(G, X_6, X_7) \circ (X_7 \stackrel{Reset}{\longrightarrow} q_0) \end{aligned} \tag{18}$$

In fact, this scheduler is actually composed of three different non-trivial subschedulers. The Stepwise DP-Optimal A_o can be rewritten as:

$$A_o = G^4_{des} \circ (X_4 \stackrel{Reset}{\longrightarrow} q_0) \circ A_1 \circ (X_3 \stackrel{Reset}{\longrightarrow} q_0) \circ A_2 \circ (X_7 \stackrel{Reset}{\longrightarrow} q_0)$$

$$where \begin{cases} A_1 = M^o_D(G, X_1, X_2) \circ M^o_D(G, X_2, X_3) \\ A_2 = G^5_{des} \circ M^o_D(G, X_5, X_6) \circ M^o_D(G, X_6, X_7) \end{cases}$$

This last expression shows the minimum number of *Reset*s that are necessary to visit all the X_i in an optimal way (3, in this case).

This form also highlights the manner in which the information would need to be sent through a communication network. Given that there are exactly 3 *Reset* events, the sender should generate exactly 3 messages and send them in parallel. For each message, the sender can specify (in each header for example) the desired route that each one of the messages should take, according to the sender's view of the network and calculations (this routing is actually given by the corresponding Stepwise DP-Optimal subscheduler). However, the uncontrollability is represented by the fact that other intermediate routing nodes may have a view of the network that is different from that of the sender, in which case, the former might decide on a new route.

8 Potential applications of the theory

Applications of the theory that we have elaborated cover various fields of engineering. One application that can be developed from the theory is test objective generation. In test objective generation, the goal is to check whether a particular system meets the expectations or the requirements that are associated with it. In this framework, the states of interest may be states in which the system is suspected to behave incoherently or incorrectly, or states in which misbehavior could be dramatic or dangerous. These would be the states that would be marked. The theory that we developed

[8] We have indeed stated that the scheduler requires exactly n different DP-Optimal submachines to be constructed (not counting the possible *Reset*s of the system).

allows to visit all these states and to test the behavior of the system in each one. Once the system has reached one of the marked states, all the known events can be disabled to check if the system stops or enters a forbidden state. A timeout can be set, for example. If the system has not behaved incoherently after that timeout, we can decide to pursue the visit of the marked states. Other more involved strategies can be applied to determine whether a state is faulty or not. For each state, either the behavior of the system is acceptable, or it is not. In the first case where the state is flawless, the next submachine of the scheduler is activated in order to make the system evolve in the next state of interest to be tested. In the case where a failure has been detected in the states, either we stop since the system is faulty and does not correspond to the awaited specifications, or we pursue to determine other possible faults. To do so, we *reset* the system to its initial state q_0, and go directly to the next state, say X_i, through its direct DP-Optimal submachine, namely G^i_{des}, and the process continues.

Another application area is planning in the case of multiple goals in Artificial Intelligence. Several search algorithms exist when one unique goal is sought (see part II of [15]). Planning in the case of multiple goals remains challenging and interesting. The framework in which we have developed the theory allows goals to be independent or related. Once again, the *Reset* event has an interesting interpretation in AI. It represents the impossibility to meet all the goals without returning to the initial state. It may represent the possibility of using several agents to achieve the goals, each running in parallel. The number of *Reset* events gives the necessary and sufficient number of agents that are needed to perform the goal of reaching all the subgoals in parallel, without any conflicts. These applications constitute interesting further work.

We give a last potential application example of our theory: routing in a communication network. In the same way that several agents can perform in parallel to achieve different tasks, in a communication network a message can be broadcast by generating multiple copies of it, and sending these copies in parallel, along the Stepwise DP-Optimal paths. These paths are actually the Stepwise DP-Optimal Schedulers seen as Stepwise DP-Optimal subnetworks. The marked states represent the agents to whom the messages are destined. The costs may be the energy consumed for each transmission between nodes. The uncontrollability of certain events may reflect the possibility of other agents changing the terminal path to certain nodes, based on their own view of the network.

9 Conclusion

In this paper, we have introduced a new type of optimal control for Discrete Event Systems. Previous work in optimal control deals with numerical performances in supervisory control theory when the goal to achieve is a unique state of interest. In contrast, our goal was to make the system evolve through a set of goals, one by one, with no order necessarily specified *a priori*. The order in which the states are visited was part of the optimization problem since it had an influence on the cost of visiting all the goal states.

The system to be controlled is represented by a FSM with a set of multiple marked states $\mathcal{X} = (X_i)_{i\in[1,\dots,n]}$ representing the states of interest. Our aim was to have the system reach each and every one of the $(X_i)_{i\in[1,\dots,n]}$. To do so, we have introduced the notion of a scheduler. A scheduler can be thought of as a concatenation of submachines. The role of the scheduler is to make the system evolve according to one submachine at a time, and account for switching between them at appropriate instants, i.e., when one of the states of interest has been reached. We have then introduced the notion of a Stepwise DP-Optimal scheduler of a FSM G w.r.t. the set $\mathcal{X}$. This particular type of scheduler is custom made given the system on which the optimization is to be run. It has the particularity of being composed of DP-Optimal submachines which allow optimality

from state of interest to state of interest (stepwise). Moreover, the ordering of these DP-Optimal submachines allows global optimality in the sense that the total worst-case cost of visiting all the states of $\mathcal{X}$ is minimized.

We gave a necessary and sufficient condition for the existence of a Stepwise DP-Optimal scheduler, namely, the existence of n DP-Optimal submachines between the initial state q_0 and each state of the n states of $\mathcal{X}$. This condition is not very restrictive, since if it does not hold, that means that one of the states is not reachable in a controllable manner, i.e., not surely reachable from the initial state q_0. In such a case, it is obvious that the state in question will never be reachable with a surely finite cost.

From a computational point of view, we showed that our optimal problem could be brought down to an instance of the TSP. The solution of this particular TSP gives a direct access to both the structure of a Stepwise DP-Optimal scheduler and the worst-case cost for visiting all the states of interest. Considering the high computational complexity of this step, we also gave ways of taking advantage of some particular properties of the structure of a Stepwise DP-Optimal scheduler, leading to the reduction of the computational complexity of the corresponding TSP without loss of global optimality. We have given a modified version of the algorithm returning DP-Optimal submachines, in the case where the control costs were 0 for controllable events and ∞ for uncontrollable events. This modeling of control costs can be useful in several applications where the emphasis is on the cost of events to occur, more than on the cost of disabling events.

Finally, besides the possible applications briefly presented in §8, future work will most probably extend the theory to the case of a system where the events are partially observable.

A The cyclic DP-Optimal algorithm

We recall in this section the algorithm for the construction of a submachine of the FSM G which solves the optimal control problem in the particular case where the control cost function is equal to zero for all controllable events and infinity for all uncontrollable events. To obtain a polynomial computability, two assumptions are required. They are

1. $\forall \sigma \in \Sigma,\ c_e(\sigma) > 0$
2. $|Q_m| = 1$

In other words, all event costs are strictly positive and there only exists one marked state. The following notations will be used in the various algorithms in the sequel. We first need to introduce an object called a one-step submachine, defined as follows:

Definition A.1 *A is a one-step submachine of G at $q \in Q$ if $A = \langle \Sigma, Q_A, q_{0A}, Q_{mA}, \delta_A \rangle$ satisfies:*

(i) $q_{0A} = q$
(ii) $\mathcal{T}(A) \subseteq \mathcal{T}(G, q)$
(iii) $\mathcal{T}(A) = \emptyset \Rightarrow q_{0A} \in Q_m$
(iv) $Q_A = \{q' \in Q : \exists \tau \in \mathcal{T}(A), (\pi_3(\tau) = q')\} \cup \{q\}$
(v) $Q_{mA} = Q_m \cap Q_A$.

The set of all one-step submachines of G at q is noted $\mathcal{M}_1(G, q)$. This set also has a maximal element denoted by $M_1(G, q)$. Moreover, for convenience, we write $M_D^o(G, q, Q_m) = M_D^o(G, q)$. The notations used in the algorithm are the followings:

(i) SL = Solved list of states (also sometimes called the closed list).

(ii) $\mathcal{T}_{opt}(q)$ = The set of transitions of $M_1(M_D^o(G, q), q)$, the maximal one-step submachine of $M_D^o(G, q)$ rooted at q.

(iii) $CL = \{(q, c_{sup}^q(M_D^o(G, q))) : q \in Q_G\}$. This is the cost list maintained by the algorithm for the recursive computation of the cost associated with a particular submachine. The variables CL^{temp} and $\mathcal{T}_{opt}^{temp}(.)$ are also used for temporary storage of the same quantities.

(iv) C = Set of states to be processed in the current iteration.

(v) $P_f(C) = \{q \in Q : \exists \tau \in \mathcal{T}(G), (\pi_2(\tau) = q) \wedge (\pi_3(\tau) \in C)\}$.

In other words $P_f(C)$ is the set of parent states of set C. A function denoted by $c_{\max}(E)$, where E is some set of transitions of a one-step submachine will also be used in the algorithm.

The following algorithm is a direct adaptation of the one described in [17], section 6. In this particular case, we assume that the control costs are equal to 0. This assumption leads to simplifications in the algorithm. As in [17], we choose to split the program into 3 sub-programs: The **DP-Opt program**, the **Optimize subprogram** and the **One-step optimize subprogram**. The DP-Opt program primarily orders the backward recursive search and updates C and SL based on data provided by the subprogram Optimize. The optimization at each state is described in the subprogram Optimize, which in turn calls the subprogram **One-step optimize**.

The DP-Opt program :

(i) Input: $\Sigma, Q, q_o, q_m, \delta, c_e$. Output G_{des}

(ii) Initialize: $C = \{q_m\}, SL = \emptyset, CL = \emptyset, CL^{temp} = \emptyset$. If there exists $\sigma \in \Sigma_{uc}$ and $q \in Q$ such that $(\sigma, q_m, q) \in \mathcal{T}(G)$ then STOP since no optimal solution exists. Otherwise set $\hat{E}_0(q_m) = E_0(q_m) = \emptyset$.

(iii) Optimize: Call subprogram Optimize with argument C.

(iv) Compute

$$A = \{q_d \in C : c_{\max}(\mathcal{T}_{opt}^{temp}(q_d)) = \min_{q \in C} c_{\max}(\mathcal{T}_{opt}^{temp}(q))\}.$$

(v) $\forall q \in A$:

$$\left| \begin{array}{lcl} \mathcal{T}_{opt}(q) & = & \mathcal{T}_{opt}^{temp}(q) \\ CL & \leftarrow & CL \cup \{(q, c_{\max}(\mathcal{T}_{opt}(q))\} \\ SL & \leftarrow & SL \cup A \\ CL^{temp} & = & \emptyset. \end{array} \right.$$

(vi) Termination Condition: Is $q_o \in SL$? If yes then STOP. Otherwise continue.

(vii) Computation of the states to be optimized in the next iteration: Compute the following $P_f(SL)$ and $A = P_f(SL) - SL$

$$\forall q \in A, \left| \begin{array}{rcl} \hat{E}_0(q) & = & \{\tau \in \mathcal{T}(G,q) : \pi_3(\tau) \notin SL\} \\ E_0(q) & = & \mathcal{T}(G,q) - \hat{E}_0(q) \\ B & = & \{q \in A : \exists \tau \in \hat{E}_0(q), \pi_1(\tau) \in \Sigma_{uc}\} \\ C & = & A - B \end{array} \right.$$

If $C = \emptyset$ then STOP since no optimal solution exists.

(viii) GOTO (iii).

The Optimize subprogram :

(i) Input: $C, \{E_0(q) : q \in C\}$.

(ii) $C_l = C$, $n = \|E_0(q)\|$

(iii) Pick any $q_d \in C_l$ and update C_l: $C_l = C_l - \{q_d\}$.

(iv) If $n > 1$ order $E_0(q_d)$ such that

$$i < j \Rightarrow c_e(\pi_1(\tau_i)) + c_{\sup}^g(M_D^o(G, \pi_3(\tau_i))) \leq c_e(\pi_1(\tau_j)) + c_{\sup}^g(M_D^o(G, \pi_3(\tau_j))$$

(v) If $n = 0$ then set $c_{\max}(E_0(q_d)) = 0$. Else set

$$c_{\max}(E_0(q_d)) = c_e(\pi_1(\tau_n)) + c_{\sup}^g(M_D^o(G, \pi_3(\tau_n))).$$

(vi) Call subprogram One-step optimize.

(vii) Termination condition: Is $C_l = \emptyset$? If yes then return to the main program. Otherwise GOTO (iii).

The One-step optimize subprogram :

(i) Input: $q_d, E_0(q_d)$ (ordered)

(ii) For $i = n \rightarrow 1$ in $E_0 = \{(\tau_i), i = 1 \ldots n\}$ do

if $\pi_1(\tau_i) \in \Sigma_{uc}$ then $E = \{(\tau_j), j = 1 \ldots i\}$, GOTO (iii)

(iii) if $i = 1$ then

$$E = \{\tau \ / \ c_e(\pi_1(\tau)) + c_{sup}^g(M_D^o(G, \pi_3(\tau)) = \min_{\tau_i \in E_0} [c_e(\pi_1(\tau_i)) + c_{sup}^g(M_D^o(G, \pi_3(\tau_i))]\}$$

(iv) set

$$\left|\begin{array}{lcl} \mathcal{T}_{opt}^{temp}(q_d) & \leftarrow & E, \\ CL^{temp} & \leftarrow & CL^{temp} \cup \{(q_d, c_{\max}(E))\} \end{array}\right.$$

where $c_{\max}(E) = c_e(\pi_1(\tau_i)) + c^g_{\sup}(M^o_D(G, \pi_3(\tau_i)))$ and return to Optimize.

It is important to note that the algorithm as stated will generally produce a submachine that is not trim. In particular it will have states not accessible from q_0. However, this particularity is taken into account to generate the submachines of the form $M^o_D(G, X_i, X_j)$. Indeed it could happen that the X_i does not belong to the final DP-Optimal submachine G^j_{des}, even if a DP-Optimal submachine exists between this two states. Note that X_j is added in the solved list SL if and only if $c^g_{sup}(M^o_D(G, X_i, X_j)) < c^g_{sup}(G^j_{des})$. In the other case, even if a submachine exists between these two states, this path does not have to be considered, because it is better to perform a *Reset* and to directly use the DP-Optimal submachine G^j_{des}. A consequence of the one-step optimize sub-program is given by:

Property A.2 $\forall q \in Q_i$,

1. *when* $\mathcal{T}_{uc}(G_{des}, q) \neq \emptyset$,

$$\max_{\tau_{uc} \in \mathcal{T}_{uc}(G_{des}, q)} \left\{ \begin{array}{c} c_e(\pi_1(\tau_{uc})) \\ + \\ c^g_{sup}(M^o_D(G, \pi_3(\tau_{uc}), q_m)) \end{array} \right\} \geq \max_{\tau_c \in \mathcal{T}_c(G_{des}, q)} \left\{ \begin{array}{c} c_e(\pi_1(\tau_c)) \\ + \\ c^g_{sup}(M^o_D(G, \pi_3(\tau_{uc}), q_m)) \end{array} \right\}$$

and

$$c^g_{sup}(M^o_D(G, q, q_m)) = \max_{\tau_{uc} \in \mathcal{T}_{uc}(G_{des}, q)} (c_e(\pi_1(\tau_{uc})) + c^g_{sup}(M^o_D(G, \pi_3(\tau_{uc}), q_m))).$$

2. *otherwise,* $\forall \tau \in \mathcal{T}(G_{des}, q)$,

$$c^g_{sup}(M^o_D(G, q, q_m)) = c_e(\pi_1(\tau)) + c^g_{sup}(M^o_D(G, \pi_3(\tau), q_m))$$

■

In other words, the worst case cost to go from any state $q \in Q_i$ to a state q_m is given by a trace all of whose prefixes contain an uncontrollable event (when it exists). Otherwise, the worst case cost is independent of the transition taken at q.

B Finding a minimum cost path visiting all the X_i

We have brought down the minimization of the worse inevitable cost of a path visiting all the $(X_i)_{i \in [1,\ldots,n]}$ in the original FSM G to a modified Traveling Salesman Problem. We can represent this problem by a graph (V, E) on which we wish to find a minimum cost path that visits all the nodes in V. The vertices in V are the $(X_i)_{i \in [1,\ldots,n]}$, and the costs of the edges in E are the $(c^g_{\sup}(G^i_{des}))_{i \in [1,\ldots,n]}$ and the $(c^g_{\sup}(M^o_D(G, X_i, X_j)))_{(i,j) \in [1,\ldots,n]}$ when the latter are defined. In this

section we show, in the particular context of this problem, how the TSP can be solved by the Branch and Bound method (B&B).

Before we explain the actual method, we must put the problem in canonical form. The start point is the cost matrix C, defined in §5.1. From this matrix, we generate the all-pairs shortest-paths matrix D. Several methods exist to achieve this transformation. Dijkstra on every node and Floyd-Warshall are the most common ones. Both run in a worst-case complexity of the order $\mathcal{O}(\|V\|^3)$ in the number $\|V\|$ of nodes. From this point, general methods can be used to solve the TSP.

B.1 Solving the TSP

The strategy used here to solve the TSP is similar to a divide and conquer approach. It is the **Branch and Bound method (B&B)**. We will not formally describe it. A thorough description of the underlying theory can be found in [7, 6, 9]. Instead, we will explain its principle, borrowed from [9], and directly apply the method to our problem.

B.1.1 Definitions and notations

We will call a *general solution* of the TSP any set of cardinality $\|V\|$, containing pairs of the type $(X_i, X_j)_{(i,j)\in[1,\ldots,n]}$ such that each X_k appears exactly once in the left component of a pair and exactly once in the right component of a pair. In other words, a general solution is a path that visits all the nodes $(X_i)_{i\in[1,\ldots,n]}$ exactly once, but is not necessarily a tour (i.e., a cycle). For that purpose, we will call a *tour solution* a general solution which is a tour. Finally, we will call an *optimal tour solution* a tour solution with minimum cost. We call z_0 the cost of any optimal tour solution. We will represent the fact that a given pair (X_i, X_j) appears in a general solution for a problem P by the equation $x_{ij}^P = 1$. Likewise, we will represent the fact that a given pair (X_i, X_j) does not appear in a general solution for a problem P by the equation $x_{ij}^P = 0$. We call the *incumbent* the best tour solution found so far.

B.1.2 The Branch and Bound Method

The B&B method allows to give both lower and upper bounds for the minimum cost z_0 of an optimal solution. The bounding method consists of relaxing a constraint of a problem and solving the *relaxed problem*, yielding a *relaxed optimum*, and thereby a lower bound for the minimum cost z_0 of an optimal solution. In our particular application, the condition that is relaxed is the condition of a solution being a tour (i.e., a cycle). The branching operation consists of imposing a supplementary constraint to a given problem. In our particular application this is done by imposing a pair of the type (X_i, X_j) to be in the solution that we wish to find. We impose $x_{ij}^P = 1$ in the problem we are dealing with. When a solution to the hardened problem (the problem with a supplementary constraint) is found, its cost will constitute an upper bound for the cost z_0 of an optimal solution to the original problem.

The bounding strategy We present an algorithm for finding a general solution for a TSP, that is, a solution to the relaxed TSP. This is called the **Hungarian Method**.

The Hungarian Method runs systematically on a matrix, independently of what it represents. The input to the algorithm is a matrix over which we wish to find a general solution. The output of the algorithm is a set of n pairs of the type $(X_i, X_j)_{(i,j)\in[1,\ldots,n]}$. Let us call this matrix D since it

is that matrix that will be processed at the first application of the Hungarian Method. We suppose that $D = [d_{ij}]_{(i,j)\in[1,\ldots,n]}$.

Step 1.1 - Initialization Define $\forall i \in [1,\ldots,n], u_i = min(d_{i1},\ldots,d_{iN})$. For all $i \in [1,\ldots,n]$, update D by subtracting u_i to every element in row i. Define $\forall j \in [1,\ldots,n], v_j = min(d_{1j},\ldots,d_{nj})$. For all $j \in [1,\ldots,n]$, update the resulting matrix by subtracting v_j to every element in column j. In other words, we subtract from each row its minimum, and on the resulting matrix, we subtract from each column its minimum. The resulting matrix is assured to have at least one null element in each row and in each column.

Step 1.2 - Generating an initial Assignment Look for a row or column which has a single null element. If there is one, **mark** it, and **cross out** all the cells d_{ij} in the same row and all the cells in the same column. We will say that a row or a column is marked if one of its cells is marked. Repeat the process until all non crossed out cells are non zero, or until a row or column has more than one null cell. If there is such a row or column, randomly pick one of the null cells and mark it, cross out all the cells in the same row and all the cells in the same column. Continue until all rows and columns are marked or until all null cells are crossed out.

If each row and each column has a marked cell, this means we have found a general solution to the underlying problem. The method terminates. Otherwise, go to Step 2.1.

Step 2.1 - Labeling the unmarked rows For each i such that row i is unmarked at this stage, label it $(s, +)$. This label simply says that this row needs to be marked. We say at this stage that the rows are now labeled but not scanned. Put them in a list of labeled but unscanned rows.

Step 2.2 - Scanning Unmarked Rows and Columns If the list of marked but unscanned (the definition of scanning has not yet been given on the first loop) rows and columns is empty, the labeling stage is over. This means that the present assignment of marked cells is maximal. In this case, we say that *non-breakthrough* has occurred, and go to Step 4. Otherwise, select a marked row or column that needs to be scanned, remove it from the list. If it is a row, apply the forward labeling, if it is a column apply the reverse labeling, at Step 2.3.

Step 2.3 - Scanning The following two labeling processes differentiate the scanning process when applied to rows and columns. Scanning a row or column is simply processing it, as defined next.

Forward labeling For each labeled row i, if a cell d_{ij} is 0, label column j with $(row_i, +)$ unless it is already labeled. This simply says that this cell can possibly be marked later in the process. During the process of forward labeling, if a column is labeled and not marked, we say that we have a situation of *breakthrough*. This means that there is a way of finding another marking of the matrix cells with one more marked cell. If breakthrough occurs, go to step 3.

Reverse labeling Any column j to be scanned will be marked. If the row containing the corresponding marked cell is unlabeled yet, label it with $(col_j, -)$.

If breakthrough has not occurred, include all the newly labeled rows and columns in the list of labeled but unscanned rows and columns, and go back to Step 2.2.

Step 3 - Changing the marking We process a labeled column j which is not marked. Suppose it is labeled with $(row_i, +)$. Mark cell d_{ij}. If the label of row i is $(col_{j_1}, -)$, remove the mark of cell d_{ij_1} (this removes the former mark of that row). Continue along the path, guided by the labels. When a row labeled $(s, +)$ is marked, stop. If at this stage all columns and all rows are marked, the process terminates. If not, erase all the labels and go back to step 2.1, with the new cost matrix, the new markings, and u and v as they are now.

Step 4 - Updating the cost matrix Compute δ which is the minimum value of the cells in labeled rows and unlabeled columns. The way the computation is organized, δ will be strictly positive. If $\delta = \infty$, this means that no solution to the problem exists, and the process terminates. If δ is finite, update $u_i \leftarrow u_i + \delta$ where i is a labeled row, and update $v_j \leftarrow v_j - \delta$ where j is a labeled column. Update the matrix by the following operations:

- d_{ij} stays unchanged when row i and column j are both labeled or both unlabeled,
- $d_{ij} \leftarrow d_{ij} - \delta$ when row i is labeled and column j is unlabeled,
- $d_{ij} \leftarrow d_{ij} + \delta$ when row i is unlabeled and column j is labeled.

All the new values of the matrix are still positive at this point, but new 0-valued cells have been created thereby allowing different marking configurations. At this point, retain all the labels on the rows and columns, but include all the labeled rows in the list of unscanned rows. Go back to the labeling process where it was left, i.e., in step 2.2.

The cost of the general solution obtained can be read by summing the values in the vector u and adding this sum to the sum of the values of the vector v.

We have presented a method for solving a general problem by relaxing the condition that the solution must be a tour. Let us go through the first steps of the process. We have a matrix D representing the original problem P_0. First, we apply the lower bounding strategy by relaxing the constraint that the optimal tour solution must be a tour. To do so, we use the **Hungarian Method** described above. We thereby obtain a relaxed solution which is a general solution to the original problem P_0. If this general solution happens to be a tour solution, it becomes the incumbent, it is an optimal tour solution, and the process terminates. Otherwise, we retain the cost L_0 of that solution as a lower bound for the cost of the optimal solution. We then apply the branching strategy on P_0.

The branching strategy The branching strategy simply relies on the fact that for a given pair (X_k, X_l) of the set $(X_m, X_n)_{x_{mn}^{P_0}=1}$ of the general solution,

1. Either it does not appears in an optimal tour solution,
2. Or it does appear in an optimal tour solution of the original problem P_0 and its inverse (X_l, X_k) does not.

Using the notations defined above, this means that

1. Either (1) $x_{kl}^{P_0} = 0$,
2. Or $x_{kl}^{P_0} = 1$ and $x_{lk}^{P_0} = 0$.

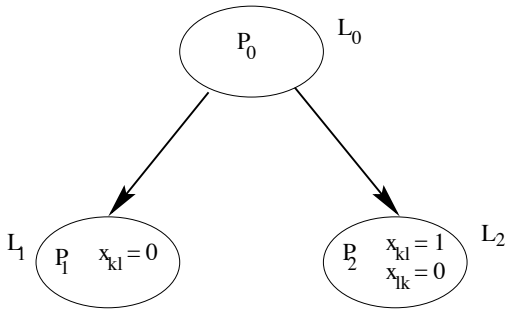


Figure 11: Branching the initial problem into two subproblems

This last supplementary conditions avoids an otherwise possible tour of length 2 (figure (11)). Before applying the branching strategy, we wish to determine which of the pairs $(X_i, X_j)_{x_{ij}^{P_0}=1}$ of the general solution should be branched.

The following computation gives a heuristic to choose such a pair. On the new cost matrix that results from the Hungarian Method, compute the following quantity for each pair $(X_i, X_j)_{x_{ij}^{P_0}=1}$ of the relaxed optimum:

$$\beta_{ij} = \min_{m \neq j}(d_{im}) + \min_{n \neq i}(d_{nj}) \tag{19}$$

Note that although β is doubly indexed, it represents only N values, one for each pair of the general solution. Pick the pair (X_i, X_j) for which β_{ij} is the greatest.

If a tie occurs, pick a pair at random. This is the pair on which branching should be applied because it is the one for which the next lower bound obtained will be closest to z_0, the cost of an optimal tour solution.

Now that we have the "best" pair to base our branching upon, we branch the original problem into subproblems P_1 and P_2 (see figure (11)). We apply the lower bounding strategy on both children of the parent problem, yielding L_1 and L_2. Several cases appear:

- $L_1 \leq L_2$. If the general solution $(X_m, X_n)_{x_{mn}^{P_1}=1}$ found for P_1 is a tour, then it verifies P_0 with the additional constraint that (X_k, X_l) occurs in an optimal solution. Therefore, L_1 is a current upper bound of z_0. In this case, we can *prune* the branch corresponding to P_2, because it will never yield a better upper bound, since its lower bound is higher than the upper bound L_1. The process terminates. The solution for the original problem P_0 is the relaxed optimum for P_1.

- $L_1 < L_2$. If the general solution $(X_m, X_n)_{x_{mn}^{P_2}=1}$ found for P_2 is a tour, then it verifies P_0 with the additional constraint that $x_{kl} = 0$, i.e., that (X_k, X_l) does not occur in an optimal solution (not meaning that (X_l, X_k) does). If the general solution $(X_m, X_n)_{x_{mn}^{P_1}=1}$ found for the relaxed problem of P_1 is not a tour, then we cannot prune P_1 because it may yield a better solution, i.e., with a cost lower than the current upper bound given by L_2. P_1 is entered in a queue, and will be processed next.

- $L_1 < L_2$. If neither the relaxed optimum for P_1 nor the relaxed optimum for P_2 are tours, then both problems may later give a tour solution which has cost less than L_2 (if so, it will constitute an optimal tour solution); therefore, both problems are put in the queue. Due

to its looser lower bound, P_1 will be branched before P_2. P_2 will be branched after only if necessary.

When a problem is branched, it is removed from the list, and will never enter it again. The children of the branched problem (P_{11} and P_{12} if P_1 is branched) are generated with an additional branching constraint and join the queue of subproblems to be treated.

At any stage during the algorithm, the branching constraints for a problem Q will be given by a set of equations of the type:

$$\begin{cases} x_{i_1 j_1} = x_{i_2 j_2} = \ldots = x_{i_r j_r} = 1 \\ x_{p_1 q_1} = x_{p_2 q_2} = \ldots = x_{p_u q_u} = 0 \end{cases} \tag{20}$$

These equations result from the successive branchings. Since our goal is to find a tour solution, we can remove all rows i_k such that $x_{i_k j_k} = 1$, and also remove all columns j_l such that $x_{i_l j_l} = 1$. Overall, for a subproblem with the above supplementary constraints, the relaxed assignment problem is of size $(N - r) \times (N - r)$. Then all the $d_{p_1 q_1}, d_{p_2 q_2}, \ldots, d_{p_u q_u}$ are set to ∞, i.e., we force the corresponding pairs not to appear in a general solution to the relaxed subproblem.

The process repeats, i.e., the search tree is expanded. Note that any subproblem inherits all the constraints of its parent; therefore, the lower bound found for a subproblem is higher than that of its parent. The queuing strategy is to branch the subproblem with lowest lower bound. When solving the relaxed versions of the problems, we always keep the solution which (1) satisfies the constraints of its parent and (2) which has the highest cost. This tour solution is the **incumbent**. Any problem in the queue that has a lower bound higher than the current cost of the incumbent is pruned and removed from the queue at each update of the incumbent. The stopping condition is when the queue is empty. At termination, if an incumbent is available, it is an optimal tour solution for the original problem P_0. Otherwise, if there is no available incumbent, the initial problem P_0 has no optimal tour solution, therefore no tour solution.

B.2 Example

Now that the method to solve the TSP has been given, in order to find the minimum cost path visiting all the $(X_i)_{i \in I}$, we can apply it to the example that is followed throughout the paper. The initial cost matrix C is given below.By transforming it into the all-pairs shortest-path cost matrix D (by applying the Floyd-Warshall algorithm for example), we obtain the second matrix:

	q_0	X_1	X_2	X_3	X_4
q_0	∞	**6**	**4**	**5**	**5**
X_1	**0**	∞	**1**	**2**	**3**
X_2	**0**	**2**	∞	**1**	**2**
X_3	**0**	∞	∞	∞	∞
X_4	**0**	∞	∞	∞	∞

	q_0	X_1	X_2	X_3	X_4
q_0	∞	**6**	**4**	**5**	**5**
X_1	**0**	∞	**1**	**2**	**3**
X_2	**0**	**2**	∞	**1**	**2**
X_3	**0**	**6**	**4**	∞	**5**
X_4	**0**	**6**	**4**	**5**	∞

The first step is to apply the Hungarian method on this matrix. We go through step 1.1, compute the initial u_i for this matrix D and update it by subtracting u_i to each row i. We compute likewise the v_j for the newly obtained matrix, and update it once more by subtracting v_j to each column j.

	q_0	X_1	X_2	X_3	X_4	u_i
q_0	∞	**6**	**4**	**5**	**5**	4
X_1	**0**	∞	**1**	**2**	**3**	0
X_2	**0**	**2**	∞	**1**	**2**	0
X_3	**0**	**6**	**4**	∞	**5**	0
X_4	**0**	**6**	**4**	**5**	∞	0

	q_0	X_1	X_2	X_3	X_4	u_i
q_0	∞	**0**	**0**	**0**	**0**	4
X_1	**0**	∞	**1**	**1**	**2**	0
X_2	**0**	**0**	∞	**0**	**1**	0
X_3	**0**	**4**	**4**	∞	**4**	0
X_4	**0**	**4**	**4**	**4**	∞	0
v_j	0	2	0	1	1	

At this point, we mark the null cells in each row, starting from the ones that are alone in their row, yielding the following marking. We now label the rows that are not yet marked with a label s^+ (short for $(s,+)$). We then propagate the labels to column 1 which shares a null cell will row 2. Labels of the type $(row_i,+)$ will be abbreviated r_i^+ in the tables. Likewise, the labels of the type $(col_j,-)$ will be abbreviated c_j^-.

	q_0	X_1	X_2	X_3	X_4	u_i
q_0	∞	**0**	**0**$\clubsuit$	**0**	**0**	4
X_1	**0**$\clubsuit$	∞	**1**	**1**	**2**	0
X_2	**0**	**0**$\clubsuit$	∞	**0**	**1**	0
X_3	**0**	**4**	**4**	∞	**4**	0
X_4	**0**	**4**	**4**	**4**	∞	0
v_j	0	2	0	1	1	

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**$\clubsuit$	**0**	**0**	4	
X_1	**0**$\clubsuit$	∞	**1**	**1**	**2**	0	c_1^-
X_2	**0**	**0**$\clubsuit$	∞	**0**	**1**	0	
X_3	**0**	**4**	**4**	∞	**4**	0	s^+
X_4	**0**	**4**	**4**	**4**	∞	0	s^+
v_j	0 r_4^+	2	0	1	1		

After another run through the scanning step of the Hungarian method, the list of labeled but unscanned rows becomes empty; therefore, we have non-breakthrough. We go to step 4, and compute δ by taking the minimum of the values of the cells that are in a marked row but in an unmarked column, yielding $\delta = 1$. We update the matrix D according to the cases described in step 4 of the algorithm. The matrix becomes the following, and the u_i and the v_j are also updated accordingly. This yields the first matrix below. We go back to the labeling process where it was left and update the labels on the columns as follows:

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**$\clubsuit$	**0**	**0**	4	
X_1	**0**$\clubsuit$	∞	**0**	**0**	**1**	1	c_1^-
X_2	**1**	**0**$\clubsuit$	∞	**0**	**1**	0	
X_3	**0**	**3**	**3**	∞	**3**	1	s^+
X_4	**0**	**3**	**3**	**3**	∞	1	s^+
v_j	-1 r_4^+	2	0	1	1		

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**$\clubsuit$	**0**	**0**	4	
X_1	**0**$\clubsuit$	∞	**0**	**0**	**1**	1	c_1^-
X_2	**1**	**0**$\clubsuit$	∞	**0**	**1**	0	
X_3	**0**	**3**	**3**	∞	**3**	1	s^+
X_4	**0**	**3**	**3**	**3**	∞	1	s^+
v_j	-1 r_4^+	2	0 r_2^+	1 r_2^+	1		

We here arrive at a breakthrough situation, because column 4 is labeled but unmarked. We go to step 3 to obtain a new arrangement of the markings. Starting from column 4 which is labeled but unmarked, we follow the labels to obtain the new marking of the matrix (shown below, left). We now restart the labeling process from the beginning by erasing the old labels. Relabeling the matrix yields the matrix shown below, right.

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**♣	**0**	**0**	4	
X_1	**0**	∞	**0**	**0**♣	**1**	1	c_1^-
X_2	**1**	**0**♣	∞	**0**	**1**	0	
X_3	**0**♣	**3**	**3**	∞	**3**	1	s^+
X_4	**0**	**3**	**3**	**3**	∞	1	s^+
v_j	-1 r_4^+	2	0 r_2^+	1 r_2^+	1		

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**♣	**0**	**0**	4	
X_1	**0**	∞	**0**	**0**♣	**1**	1	
X_2	**1**	**0**♣	∞	**0**	**1**	0	
X_3	**0**♣	**3**	**3**	∞	**3**	1	$c1^-$
X_4	**0**	**3**	**3**	**3**	∞	1	s^+
v_j	-1 r_5^+	2	0	1	1		

Again, we arrive at a non-breakthrough situation. We go to step 4 and compute δ, the minimum of the values of the cells in a marked row and unmarked column. We obtain $\delta = 3$. We then update the matrix and update the labels according to the usual procedure (we have put two steps into one to yield the matrix below, right). We continue the labeling process where it was left (matrix to the left):

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**♣	**0**	**0**	4	
X_1	**0**	∞	**0**	**0**♣	**1**	1	
X_2	**1**	**0**♣	∞	**0**	**1**	0	
X_3	**0**♣	**0**	**0**	∞	**0**	4	c_1^-
X_4	**0**	**0**	**0**	**0**	∞	4	s^+
v_j	-4 r_5^+	2	0	1	1		

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**♣	**0**	**0**	4	
X_1	**0**	∞	**0**	**0**♣	**1**	1	
X_2	**1**	**0**♣	∞	**0**	**1**	0	
X_3	**0**♣	**0**	**0**	∞	**0**	4	c_1^-
X_4	**0**	**0**	**0**	**0**	∞	4	s^+
v_j	-4 r_5^+	2 r_5^+	0 r_5^+	1 r_5^+	1 r_4^+		

At this stage, there is a column that is labeled but unmarked (column 5). Therefore, we have a breakthrough and we go to step 3. At this step, we follow the labels and assign a new marking to the cells, yielding:

	q_0	X_1	X_2	X_3	X_4	u_i	
q_0	∞	**0**	**0**♣	**0**	**0**	4	
X_1	**0**	∞	**0**	**0**♣	**1**	1	
X_2	**1**	**0**♣	∞	**0**	**1**	0	
X_3	**0**	**0**	**0**	∞	**0**♣	4	c_1^-
X_4	**0**♣	**0**	**0**	**0**	∞	4	s^+
v_j	-4 r_5^+	2 r_5^+	0 r_5^+	1 r_5^+	1 r_4^+		

We can see at this point that all rows and columns have a marked cell, which means that the process terminates. We have found a general solution to the general problem represented by the original matrix D. This solution can be read off the marked cells of the matrix directly:

$$(q_0, X_2); (X_2, X_1); (X_1, X_3); (X_3, X_4); (X_4, q_0) \tag{21}$$

We obtain a general solution that is a tour. Therefore, we can say that this solution is an optimal tour solution to the general problem. We need to interpret the output of the TSP solution. To do so, we take each pair and see if it exists in reality in our original problem (the one described by the matrix C). If a pair (X_i, X_j) originally exists, i.e., if $c_{ij} < \infty$, then the path is admissible in the original problem. If not, i.e., if $c_{ij} = \infty$, then we need to reset the system before directly going

to X_j through G^i_{des}. In our example, the paths $(q_0, X_4), (X_2, X_1), (X_1, X_3)$ and (X_3, q_o) exist. As for the path (X_4, X_2), it is decomposed into the set of pairs (X_4, q_0) and (q_0, X_2) that both exist.

The fact that we did not need to branch is due to the small dimension of our example. On the other hand, taking a larger example would lead to a very tedious description and tracing of the algorithm.

Nevertheless, we give a description of several particular cases one might encounter in the B & B algorithm.

- In the case where a solution is found that is not a tour, we must branch the problem as described earlier. The heuristic described in (19) is used to determine which is the best *a priori* pair to branch. The algorithm repeats the branching process as long as the list of subproblems to be treated is not empty. The list becomes empty when all the lower bounds of the relaxed optimums that are not tours are higher than the current best upper bound (i.e., the cost of the current incumbent).

- It may occur that $\beta_{ij} = 0$ for all the pairs $(X_i, X_j)_{x^Q_{ij}=1}$ for a problem Q. In this case, we are sure to have hit the cost z_0 of an optimal solution. It now suffices for the algorithm to get the pairs in an order that gives a tour. This may take several steps; however, these will be of very little overhead. Indeed, this situation means that the reduced (by the Hungarian Method) matrix has at least two 0's in each row and in each column.

This concludes the appendix, which described the Branch and Bound method and applied it to the example that we have been working with throughout the paper.

References

[1] E. Brinskma. A theory for the derivation of tests. *Protocol Specification, Testing and verification*, 7:63–74, 1988.

[2] J.C. Fernandez, C. Jard, T. Jéron, L. Nedelka, and C. Viho. An experiment in automatic generation of test suites for protocols with verification methodology. Technical Report 1035, IRISA, June 1996.

[3] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, Languages, and computation.* Addison-wesley, Reading, MA., 1979.

[4] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. Technical Report CS-96-08, Department of Computer Science, Brown University, February 1996.

[5] R. Kumar and V. K. Garg. Optimal control of discrete event dynamical systems using network flow techniques. In *Proc. of 29th Allerton Conf. on Communication, Control and Computing*, Champaign, IL, USA, October 1991.

[6] Vipin Kumar and Laveen N. Kanal. A general branch and bound formulation for and/or graph and game tree search. In *search-ai*, New York, NY, 1988. Springer-Verlag.

[7] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[8] H. Marchand and M. Le Borgne. On the optimal control of polynomial dynamical systems over z/pz. In *4th International Workshop on Discrete Event Systems*, pages 385–390, Cagliari, Italy, August 1998.

[9] K. Murty. *Operations research : deterministic optimization models.* Upper Saddle River, N.J. : Prentice Hall, 1995.

[10] David John Musliner. Circa: The cooperative intelligent real-time control architecture. Technical Report CS-TR-3157, University of Maryland, College Park, October 1993.

[11] K. M. Passino and P. J. Antsaklis. On the optimal control of discrete event systems. In *Proc. of 28th Conf. Decision and Control*, pages 2713–2718, Tampa, Floride, December 1989.

[12] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1):206–230, January 1987.

[13] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE; Special issue on Dynamics of Discrete Event Systems*, 77(1):81–98, 1989.

[14] A. Rouger and M. Phalippou. Test cases generation from formal specifications. *Proc. ISS'92, Oct. 25-30, 1992*, 2:C10.2, 1992.

[15] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[16] R. Sengupta and S. Lafortune. A deterministic optimal control theory for discrete event systems: Computational results. Technical Report n° CGR-93-16, Control Group, College of Engineering, University of Michigan, USA, December 1993.

[17] R. Sengupta and S. Lafortune. A deterministic optimal control theory for discrete event systems: Formulation and existence theory. Technical Report n° CGR-93-7, Control Group, College of Engineering, University of Michigan, USA, March 1993. Revised Dec. 1993.

[18] R. Sengupta and S. Lafortune. An optimal control theory for discrete event systems. *SIAM Journal on Control and Optimization*, 36(2), March 1998.

[19] E. Tronci. Optimal state supervisory control. In *Proc. of* 35^th *IEEE conf. on Decision and Control*, Kobe, Japon, December 1996.

[20] W. M. Wonham and P. J. Ramadge. Supervision of discrete event systems. In *1983 Proc. of the International Conference on Systems, Man and Cybernetics*, page 58, Bombay and New Delhi, India, January 1983. New York, NY, USA: IEEE.

Contents