1998 IEEE Int. Conf. On Systems, Man, and Cybernetics, San Diego, USA Oct. 11-14

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

Abstract

This paper deals with a new type of optimal control for Discrete Event Systems that extends the theory of [7]. Our aim is to make a system optimally evolve through a set of multiple goals, one by one, with no order necessarily pre-specified. Our method is divided into two steps. We first use the results in [7] to synthesize individual optimal controllers for each goal. We then develop the solution of another optimal control problem, namely, how to adapt, if necessary, and schedule all of the controllers built in the first step in order to visit all 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 Traveling Salesman Problem [1].

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

1 Introduction

We are interested in a new class of optimal control problems for Discrete Event Systems. We adopt the formalism of supervisory control theory [4] and model the system as the regular language generated by a Finite State Machine (FSM). Our control problem follows the theory in [7] and is characterized by the presence of uncontrollable events, the notion of occurrence costs for events, and a worst-case objective function. A significant difference with the work in [7] 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 theory in [7] only deals with a single marked state. Our problem formulation is motivated by several application domains such as test objective generation in verification and diagnostic, and planning in environments with uncertain results of actions.

Our solution approach consists of two steps. In the first step, we specialize the theory in [7] to synthesize a set of optimal controllers corresponding to the different marked states (or goals), each treated individually. These controllers are synthesized in a manner that gives them optimal substructure, consistent with the notion of DP-optimality of [7]. In the second step, we develop the solution of another optimal control problem, namely, how to adapt and piece together, or schedule, the controllers built in the first step in order to visit all of the marked states with least total cost. We solve this problem by defining the notion of a scheduler and by mapping the problem of finding an optimal scheduler to an instance of the well-known Traveling Salesman Problem (TSP) [1]. We suggest different strategies to reduce the computational complexity of this step while preserving optimality. Finally, we show how to recover the solution to the original problem from the solution of the TSP. Proofs and details, omitted here due to space limitations, can be found in report [2].

*This work is partially supported by INRIA and by ARO grant DAAH04-96-1-0377.

2 Notations

The system to be controlled is modeled as a FSM defined by 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 on $\Sigma^* \times Q$. The behavior of the system is described by the prefix-closed language $\mathcal{L}(G)$ [4], generated by G. 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 for which all states of Q are accessible from q_0 and coaccessible to Q_m [4]. We say that FSM $A = \langle \Sigma, Q_A, q_{0A}, Q_{mA}, \delta_A \rangle$ is a submachine of G, denoted $A \sqsubseteq G$, if $\Sigma_A \subseteq \Sigma$, $Q_A \subseteq Q$, $Q_{m_A} \subseteq Q_m$, $\forall \sigma \in \Sigma_A, q \in Q_A \;\; \delta_A(\sigma, q)! \Rightarrow (\delta_A(\sigma, q) = \delta(\sigma, q))$ (The notation $\delta_A(\sigma, q)!$ means that $\delta_A(\sigma, q)$ is defined, i.e., there is a transition labeled by event σ out of state q in machine A). We say that A is a submachine of G at q whenever $q_{0A} = q \in Q$ and $A \sqsubseteq G$. For any $q \in Q$, we will use $\mathcal{M}(G, q, Q_m) = \{A : A$ is a trim submachine of G 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 the maximal element contains all other elements as submachines. It is denoted by $M(G, q, Q_m)$; for convenience, we will write $\mathcal{M}(G, q)$ and $M(G, q)$ when there is only one marked state, i.e., when $Q_m = \{q_m\}$.

3 Review of DP-Optimality for one final state

In this section, we recall some results for the optimal control of DES that can be found in [7]. In this section, it is assumed that $Q_m = \{q_m\}$. Before dealing with the optimal control problem, we recall the definition of controllability for a submachine:

Definition 3.1 A submachine A of 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 is satisfied: $\forall\sigma((\sigma \in \Sigma_{uc}) \wedge (\delta(\sigma, q)!)) \Rightarrow \delta_A(\sigma, q)!$.

In language terms, the previous definition can be rephrased as follows: a submachine A of G is controllable if $\overline{\mathcal{L}_m(A)}\Sigma_{uc} \cap \mathcal{L}(G) \subseteq \overline{\mathcal{L}_m(A)}$.

To take into account the numerical aspect of the optimal control problem, two cost values are associated to each event of Σ. To this effect, we introduce an occurrence cost function[1] $c_e : \Sigma \to \boldsymbol{R}^+ \cup \{0\}$ and a control cost function $c_c : \Sigma \to \boldsymbol{R}^+ \cup \{0, \infty\}$. Control costs are used to represent the fact that disabling a transition possibly incurs a cost. The control cost function is infinity for events in Σ_{uc}. The cost functions are then used to introduce a cost on the trajectories of a submachine A of G. We also define a projection p_j that, when applied to a trace of events $s = \sigma_1^s\sigma_2^s \ldots \sigma_{\|s\|}^s$, gives the subtrace of s of length j starting from σ_1^s ($p_j(s) = \sigma_1^s\sigma_2^s \ldots \sigma_j^s$ if $j \le \|s\|$, and is undefined otherwise). We also introduce $\Sigma_d^G(A, q)$ as the set of disabled events at state q for the system to remain in submachine A of G.

Definition 3.2 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 $s = \sigma_1^s\sigma_2^s \ldots \sigma_{\|s\|}^s$, such that $\delta_A(s, q_*)$ exists, the cost of s is given by:

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

• The objective function denoted by $c^g_{\sup}(.)$ is given by:

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

Basically, the cost of a trajectory is the sum of the occurrence costs of the events composing it, to which is added the cost of controlling events on the way to remain in machine A. If an uncontrollable event is disabled, the cost of a trajectory becomes infinite because the second term of (1) becomes infinity. Finally, $c^g_{\sup}(A)$ represents the worst case behavior that is possible in submachine A. We now define the optimization problem.

[1]This function is labeled “event cost function” in [7].

Definition 3.3 For all $q \in Q, A_o \in \mathcal{M}(G, q)$ is an optimal submachine if

$$c^g_{\sup}(A_o) = \min_{A \in \mathcal{M}(G,q)} c^g_{\sup}(A) < \infty. \qquad (3)$$

In particular, an optimal solution in the set $\mathcal{M}(G, q_0)$ is an optimal submachine of the plant G and represents a solution of the optimal control problem, which in general has more than one solution. For such a submachine A_o, $c^g_{\sup}(A_o)$ represents the optimal cost (in fact, the worst inevitable cost) necessary to reach q_m from q_0. It means that a submachine with a lower cost could not ensure the accessibility of q_m from q_0. The following lemma (lemma (2.15) in [6]) is stated to note that optimal solutions lie within the class of controllable submachines.

Lemma 3.4 *Let $A \in \mathcal{M}(G, q, Q_m)$. If $c^g_{sup}(A) < \infty$ then A is controllable.* ■

Theorem (4.2) of [6] gives necessary and sufficient conditions for the existence of optimal submachines:

Theorem 3.5 *An optimal submachine of G exists if and only if there exists a submachine A of 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 a zero cost. The controllability assumption ensures that the positive cost cycles can be broken using controllable events alone.

We now introduce the notion of DP-Optimal submachines. This kind of submachine will be used intensively in the next sections.

Definition 3.6 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 the final marked state q_m will be denoted by $M_D^o(G, q, q_m)$. 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 [6]).

Theorem 3.7 *If an optimal submachine of G exists, then the unique maximal DP-Optimal submachine $G^m_{des} = M_D^o(G, q_0, q_m)$ of G w.r.t. the final state q_m also exists.* ■

The cyclic DP-Optimal algorithm: Consider a FSM $G = \langle\Sigma, Q, q_0, q_m, \delta\rangle$ with a unique initial state q_0, and a unique marked state q_m. Assume that all occurrence costs are strictly positive; then there exists an algorithm [7], named **DP-Opt**, with a worst-case complexity $\mathcal{O}(|Q|^2|\Sigma|log(|\Sigma|) + |Q|^3|\Sigma|)$ (theorem 6.10 of [7]), that constructs the desired maximal DP-Optimal submachine of the FSM G w.r.t. q_0 and q_m, that we denote as G^m_{des}. The algorithm also returns the worst inevitable cost $c^g_{sup}(G^m_{des})$. Moreover, during the computation of the algorithm, we can recover the submachines $M^o_D(G,q,q_m)$ associated with $c^g_{sup}(M^o_D(G,q,q_m))$, for each state visited during the computation. A simplified version of this algorithm can be found in [2] (when the control cost function is reduced to the null function for controllable events).

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 into a final state q_m. Here, our goal is different. We consider a FSM G with a set of multiple marked states $\mathcal{X} = (X_i)_{i\in[1,\ldots,n]}$. Our goal is to have the system reach each and every one of the $(X_i)_{i\in[1,\ldots,n]}$. 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 $\mathcal{X}$ (we discuss this assumption in section 7).

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 $X_i \in \mathcal{X}$. Besides, we assume that the event *Reset* is controllable and that $c_e(Reset) = c_c(Reset) = 0$.

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: $(s_i)_{i\in[1,\ldots,n]}$ 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. Note that the definition does not require that the trajectory visit each marked state only once. The valid trajectory set of the FSM G will be denoted by $\mathcal{S}$.

Definition 4.2 Let s be a valid trajectory in $\mathcal{S}$, such that $s = t^s_1 \ldots t^s_l$, with $l \geq n$ and $\delta(q_0, t^s_1 \ldots t^s_k) = X^s_k \in \mathcal{X} \cup \{q_0\}$. We define the function D from $\mathcal{S}$ into $q_0(\mathcal{X}^*\{q_0\})^*$, such that $D(s) = (X^s_k)_{k\in[1,\ldots,l]}$. 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, with respect to the set of valid trajectories $\mathcal{S}$: $\mathcal{D} = D(\mathcal{S})$. A valid state trajectory $d \in \mathcal{D}$ corresponds to a trajectory of $q_0(\mathcal{X}^*\{q_0\})^*$, that contains all the states of $\mathcal{X}$ (with possible repetitions).

We now need a structure, named a **scheduler**, that makes the system evolve through all the states of $\mathcal{X}$. Such a structure can be thought of as a concatenation of (DP-Optimal) submachines. The role of the scheduler is to make the system evolve according to one submachine at a time, and account for switching between submachines at appropriate instants. To build a scheduler, we introduce the "$\circ$" operator that denotes the concatenation of two submachines A and A' of G. It is defined in terms of languages. Let $\mathcal{L}_m(A)$ and $\mathcal{L}_m(A')$ be the marked languages of A and A', respectively. Define $\mathcal{L}_m(A \circ A') = \{st,\ s \in \mathcal{L}_m(A),\ t \in \mathcal{L}_m(A')\}$. Note that $\mathcal{L}_m(A \circ A') \subseteq \mathcal{L}_m(G)$ if and only if $Q_{m_A} = \{q_{0_{A'}}\}$ and $Q_{m_{A'}} \subseteq Q_{m_G} = \mathcal{X}$. Due to possible cycles in the FSM G, $A \circ A'$ is in general no longer a submachine of G.

Definition 4.3 Let $d = (X^d_k)_{k\in[0,\ldots,l]} \in \mathcal{D}$ be a valid state trajectory visiting $\mathcal{X} \cup \{q_0\}$ and let $(A_k)_{k\in[1,\ldots,l]}$ be such that $l > n$ and $A_k \in \mathcal{M}(G, X^d_{k-1}, X^d_k)$ for all $k \in [1,\ldots,l]$, then $A = A_1 \circ A_2 \circ \ldots \circ A_l$ is called **a scheduler** w.r.t. G and $\mathcal{X}$. The set of schedulers w.r.t. G and $\mathcal{X}$ is denoted $\mathcal{M}^{sc}(G,\mathcal{X})$.

Note that, due to the *Reset* event, 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 $(X^d_k \xrightarrow{Reset} q_0)$. Besides, in some cases, $\mathcal{M}^{sc}(G,\mathcal{X})$ can be reduced to $\emptyset$.

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

$$C^{sc}_{sup}(A) = \sum_{i=1}^{l} c^g_{sup}(A_i) \tag{4}$$

The following definition extends the notion of DP-optimality of [7] to a scheduler.

Definition 4.4 Let $A_o \in \mathcal{M}^{sc}(G,\mathcal{X})$ be a scheduler that makes the system evolve through $d = (X^d_k)_{k\in[0,\ldots,l]}$ of $\mathcal{D}$. A_o is **Stepwise DP-Optimal** (SDP-Opt) if all the submachines A_k are DP-Optimal w.r.t. the initial state X^d_{k-1} and the marked state X^d_k, and if $C^{sc}_{sup}(A_o)$ is equal to:

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

We here wish to draw attention to the fact that we will only use maximal DP-Optimal submachines. This is done for two main reasons. First, the algorithm referred to in §3 outputs exactly maximal DP-Optimal submachines. Second, taking the maximal DP-Optimal submachines allows the system a greater freedom. Indeed, the maximal DP-Optimal submachine 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 using the maximal DP-Optimal submachines for $(G^i_{des})_{i\in[1,\ldots,n]}$.

A direct consequence of definition (4.4) is given by the following property:

Property 4.5 *Let A be a SDP-Opt scheduler w.r.t. G and $\mathcal{X}$, such that $A = A_1 \circ A_2 \circ \ldots \circ A_l$. Let $d = (X^d_k)_{k\in[0,\ldots,l]}$ of $\mathcal{D}$ be the associated valid state trajectory. Then $\forall k \in [1,\ldots,l]$, $A_k = M^o_D(G, X^d_{k-1}k, X^d_k)$. Furthermore, the global cost of the scheduler is:*

$$C^{sc}_{sup}(A) = \sum_{k=1}^{l} c^g_{sup}(M^o_D(G, X^d_{k-1}, X^d_k) < \infty$$

■

This property states that if a SDP-Opt scheduler exists, al-l the submachines constituting this scheduler are given by the $(M_D^o(G, X_k, X_{k+1}))_{k\in[1,\ldots,l]}$. The cost of the scheduler is simply the sum of the costs of the DP-Optimal submachines. The set of schedulers 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 $\mathcal{M}_D^{sc}(G, \mathcal{X})$.

4.2 Stepwise DP-Optimal scheduler Existence

Before we present the necessary and sufficient conditions for the existence of a SDP-Opt scheduler, we need the following lemma:

Lemma 4.6 *If 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*:*

$$c^g_{sup}(M_D^0(G, X_i, X_k)) \leq \begin{array}{l} c^g_{sup}(M_D^0(G, X_i, X_j)) + \\ c^g_{sup}(M_D^0(G, X_j, X_k)) \end{array} \qquad \blacksquare$$

From lemma (4.6), we can ensure that a state X_i is accessible in an optimal way if and only if G^i_{des} exists. Theorem (4.7) gives the necessary and sufficient conditions of the existence of a SDP-Opt scheduler.

Theorem 4.7 *A SDP-Opt scheduler $A \in \mathcal{M}_D^{sc}(G, \mathcal{X})$ exists if and only if the n DP-Optimal submachines G^i_{des} of G exist for all $X_i \in \mathcal{X}$, $i \in [1, \ldots, n]$.* $\blacksquare$

This theorem implies that the SDP-Opt problem has a solution when there exists a DP-Optimal submachine for each of the X_i.

If a SDP-Opt solution exists, it need not be unique in general. There is no maximal SDP-Opt scheduler, in contrast to the DP-Optimal problem [7]. The problem of finding a SDP-Opt schedulers is explored in the next section.

5 Stepwise DP-Optimal scheduler Determination

In this section, we assume that a DP-Optimal submachine exists for all the states $X_i \in \mathcal{X}$. G^i_{des} will denote the DP-Optimal submachine of the particular FSM $G_i = \langle \Sigma, Q, q_0, X_i, \delta\rangle$. 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 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 SDP-Opt problem into a path-cost minimization problem, we use the **DP-Opt** algorithm. This algorithm computes for each $X_i \in \mathcal{X}$, the DP-Optimal submachine G^i_{des}. During this computation, a state X_j belonging to $\mathcal{X}$ may be reached. Due to the DP-Optimality definition, the algorithm also gives the DP-Optimal submachine between X_j and X_i. The worst inevitable case cost between these two states can be collected as well and placed in a matrix $C \in \boldsymbol{R}^{n+1} \times \boldsymbol{R}^{n+1}$ that has the following form:

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

Note that some $C[i, j]$ can be infinite after all G^i_{des} have been computed. This does not mean that the corresponding DP-Optimal submachines do not exist, but that they have not been computed. Indeed, let us suppose that some $M_D^o(G, X_i, X_j)$ has not been computed ($C[i, j] = \infty$). It means that it is less costly to perform a *Reset* from X_i to q_0, and to reach X_j through G^j_{des}.

The cost $C^{sc}_{sup}(A)$ of a scheduler $A = A_1 \circ A_2 \circ \ldots \circ A_l$ of $\mathcal{M}_D^{sc}$ is then equal to

$$\sum_{k=1}^{l} c^g_{sup}(M_d^o(G, X_{d_k}, X_{d_{k+1}})) = \sum_{k=1}^{l} C[d_k, d_{k+1}]$$

With this equation, the new optimization problem is now reduced to finding a minimum cost path in the oriented 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.

5.2 Resolution of the TSP

The problem of finding a SDP-Opt scheduler A_0 has been brought down to a TSP, a classic combinatorial optimization problem. The cities are represented by the set of n-odes $\mathcal{X} \cup \{q_0\}$, and the streets are represented by the machines $(G^i_{des})_{i\in[1\ldots n]}$ and $M_D^o(G, X_i, X_j)$ when they exist. The costs of these paths are given by the optimal cost-s for each machine, i.e., the $(c^g_{sup}(G^i_{des})_{i\in[1\ldots n]}$ and the $(c^g_{sup}(M_D^o(G, X_i, X_j)))_{i,j\in[1\ldots n]}$. 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 (i.e., q_0). This condition does not change anything to our problem since this maps to a *Reset*, which has null cost in our model.

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 C', called the all-pairs shortest-paths matrix [1]. This can be performed in $\mathcal{O}(n^3)$ using for example the Floyd-Warshall algorithm [1].

Once C' is obtained, we can feed it to a TSP solver. Solving of the TSP from C' can be done by using the Branch & Bound method (chapters 9 and 10, [3]). This method is expected to give a solution to the TSP in a tolerable amount of time (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).

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 to the system, usually by forcing a set of sub-paths in the graph. This allows to find a solution that is suboptimal in general but that is sometimes optimal. The bounding strategy focuses on finding a lower bound on the cost of the optimal solution, by relaxing one of the constraints of the problem, usually 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 optimal.

5.3 Stepwise DP-Optimal scheduler Restitution

From a solution of the TSP, we now need to build a corresponding SDP-Opt scheduler. The resolution of the TSP provides an optimal solution that gives the ordering in which the states have to be visited so as to minimize the worst-case cost. A solution of the TSP is under the form of a set of $n+1$ pairs, 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 $M_D^o(G, X_i, X_j)$ (i.e., $C[i,j] < \infty$), it suffices 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., $C[i,j] = \infty, C'[i,j] < \infty$), they are divided into two pairs, namely, (X_i, q_0) and (q_0, X_j). The first is a *Reset* to the initial state ($X_i \xrightarrow{Reset} q_0$), and the second can be mapped to the DP-Optimal submachine G_{des}^j.

Theorem 5.1 *Given a solution of the TSP, the previous mapping yields a SDP-Opt scheduler.* ■

An interesting property is given next. It states that all the submachines that constitute a SDP-Opt scheduler can be directly derived from the DP-Optimal submachines built during the computation of the matrix C.

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

This results shows that the computation of the G_{des}^i is necessary and sufficient to both test for the existence of a solution and to generate the solution when it exists.

6 Some simplifications of the T-SP resolution

In order to solve the SDP-Opt problem, we first have to solve the TSP for the matrix C. As the TSP is a NP-complete problem, it is desirable to find some simplifications, taking advantage of the special structure of a SDP-Opt scheduler leading to the reduction of the computational complexity of the corresponding TSP without loss of global optimality.

6.1 Divide and conquer

In some cases, it will be possible to divide the matrix C into several smaller ones. It will then be equivalent to solving the TSP on each of these sub-matrices. The following property 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 $\forall X_j \in \mathcal{X}_{k_2}$, $M_D^o(G, X_i, X_j)$ is not defined. If A_o is a SDP-Opt scheduler w.r.t. G and $\mathcal{X}$, then it is possible to find a set of schedulers $A_{\mathcal{X}_k}$, SDP-Opt w.r.t. G and $\mathcal{X}_k$ with $c_{sup}^{sc}(A_{\mathcal{X}_k})$ as the optimal cost to perform the visit of all states of $\mathcal{X}_k$, such that $A_o = \circ_{k=1}^l A_{\mathcal{X}_k}$, and*

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

■

Considering 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 $\mathcal{X}_k \cup \{q_0\}$. The necessary computation to find the connected components before applying proposition (6.1) can be performed in $\mathcal{O}(n+E)$, where E is the number of vertices of the directed graph associated to matrix C [2].

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 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.

Proposition 6.2 *Assume that there exists a subset $\mathcal{X}_i = (X_{i_k})_{k \in [1,\ldots,m]}$ of $\mathcal{X}$, with $m < n$, 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. $\forall k \in [2,\ldots,m]$, $M_D^o(G, X_{i_{k-j}}, X_{i_k})$ 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.

Under these assumptions, the submachines $(G_{des}^{i_k})$ for $k \in [2,\ldots,m]$ do not belong to a SDP-Opt scheduler A_o. ■

An algorithm that performs this simplification on the matrix C according to the three assumptions is presented in [2]. Its complexity is linear in the number of states of $\mathcal{X}$. With this simplification, the paths of the form $q_0 \to X_{i_k}$ 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.

7 Potential applications of the theory

Applications of this theory cover various fields of engineering. One application is test objective generation where the goal is to check whether a system meets the expectations or requirements that are associated to it. In this framework, the (marked) 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. The presented theory 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. If a failure is detected in the state, either we stop since the system is faulty, or we continue to determine other possible faults. We then *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, 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 [5]). Planning with multiple goals remains challenging and interesting. Our framework allows goals to be independent or related. 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 necessity of using several agents to achieve the goals in parallel. The number of *Reset* events gives the necessary and sufficient number of agents needed to perform the goal of reaching all the subgoals in parallel, optimally and without any conflicts.

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. The marked states represent the agents to whom the message is 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.

8 Conclusion

In this paper, we have introduced a new type of optimal control for Discrete Event Systems. Previous work in optimal control deal with numerical performances in supervisory control theory when the goal to achieve is a unique state of interest. In contrast, our aim 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. We introduced the notion of a SDP-Opt scheduler of a FSM G w.r.t. the set of states $\mathcal{X}$. This particular type of scheduler is custom made given the system on which the optimization will 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. Finally, besides the possible applications briefly presented in §7, future work will focus on the issue of extending the theory to the case of a system under partial observation.

References

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

[2] H Marchand, O. Boivineau, and Lafortune S. On the synthesis of optimal schedulers in discrete event control problems with multiple goals. Technical Report CGR-98-10, Control Group, College of Engineering, Univeristy of Michigan, USA, July 1998.

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

[4] 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.

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

[6] 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.

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