On Optimal Control in the Presence of Adversaries with Application to Scheduling under Uncertainty

Oded Maler

CNRS-VERIMAG
Grenoble, France

September 2004

Two Parts

A unified framework for optimal and sub-optimal control in the presence of adversaries

A concrete example: scheduling under uncertainty (sketch)

Unified Framework for Optimal Control

How to evaluate/optimize open systems

Dynamic games

Bounded horizon

Dynamic programming

Best first forward search

How to Evaluate/Optimize Open Systems I

Two player games:

We, the good guys, the controller

They, the rest of the world, disturbances, the environment

The controller chooses actions $u \in U$ the environment picks $v \in V$ and this determines the outcome

The controller wants to optimize the outcome according to some criterion

The environment is indifferent

How to Evaluate/Optimize Open Systems II

Consider a one-shot game a-la von Neumann and Morgenstern

The outcome be defined as $c : U \times V \rightarrow \mathbb{R}$

c	v_1	v_2
u_1	c_{11}	c_{12}
u_2	c_{21}	c_{22}

Worst-case: $u = argmin\ \max\{c(u, v_1), c(u, v_2)\}$

Average case: $u = argmin\ p(v_1) \cdot c(u, v_1) + p(v_2) \cdot c(u, v_2)$

Typical case: $u = argmin\ c(u, v_1)$

Remark: when c is a continuous function the probabilistic approach is still standard function optimization

Dynamic Games I

Reactive systems, ongoing interaction between controller and environment

State-space X and a dynamic rule of the form $x' = f(x, u, v)$, which determines the next state as a function of the actions of the two players

In discrete time: $x_i = f(x_{i-1}, u_i, v_i)$

Differential games: $\dot{x} = f(x, u, v)$

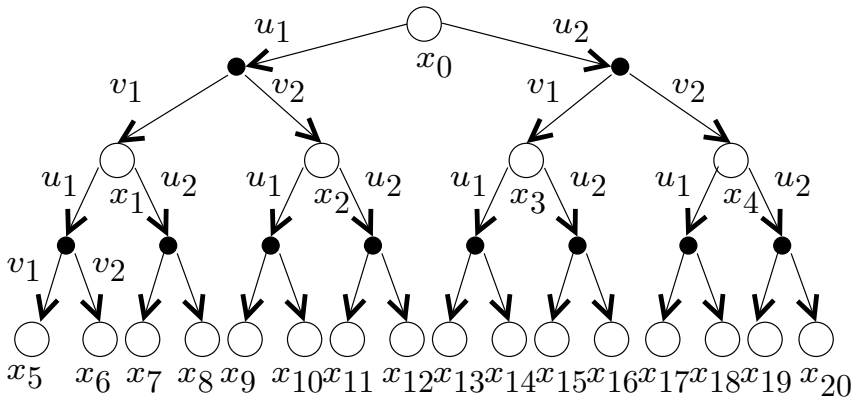
There are other more "asynchronous" games

Initial state x_0. Notation for sequences: $\bar{x} = x[0], x[1], \ldots, x[k]$, $\bar{u}$ and $\bar{v}$ for sequences of players actions.

Dynamic Games II

The predicate (constraint) $B(\bar{x}, \bar{u}, \bar{v})$ means that $\bar{x}$ is the behavior of the system when the two players apply the action sequences $\bar{u}$ and $\bar{v}$, respectively:

$$B(\bar{x}, \bar{u}, \bar{v}) \text{ iff } x[0] = x_0$$
$$\text{and } x[t] = f(x[t-1], u[t], v[t]) \; \forall t$$



$$x[0] \xrightarrow{u[1],v[1]} x[1] \cdots \xrightarrow{u[k],v[k]} x[k]$$

Evaluating Single Behaviors

Given a behavior $x[0] \xrightarrow{u[1],v[1]} x[1] \cdots \xrightarrow{u[k],v[k]} x[k]$ we can assign costs in various ways.

Any cost function $c : X \to \mathbb{R}$ on individual states can be extended to sequences:

Sum of costs:	$c(\bar{x}) = \sum_{t=1}^{k} c(x[t])$
Max of costs:	$c(\bar{x}) = \max\{c(x[t]) : t \in 1..k\}$
Time to reach a goal:	$c(\bar{x}) = \min\{t : x[t] \in F)\}$

One can add costs associated with actions of both players

Sometimes the cost function is chosen for historical reasons (quadratic norms)

Bounded Horizon Problems

Comparing strategies based on behaviors of fixed length

Motivations:

1) In many problems of “control to target” and “shortest path” all reasonable behaviors reach the same state after finitely many steps

2) Looking too far in the future is anyway unreliable (model-predictive control)

3) The problem can be reduced to standard finite dimensional optimization

Bounded Horizon Problems without Adversary

For $x' = f(x,u)$ we look for a sequence $\bar{u} = u[1], \ldots, u[k]$ which is the solution of the constrained optimization problem

$$\min_{\bar{u}} c(\bar{x}) \text{ subject to } B(\bar{x}, \bar{u})$$

The cost is based only on $\bar{x}$ while the fact that $\bar{x}$ is the result of following the dynamics f under control $\bar{u}$ is part of the constraints

For linear dynamics, $x' = Ax + Bu$, this reduces to linear programming

In discrete verification this reduces to Boolean satisfiability (bounded model checking).

Strategy without Adversary = Plan

In the absence of external disturbances $\bar{u}$ completely determines $\bar{x}$ and no feed-back from x is needed

The control "strategy" reduces to an open-loop "plan": at each time instant t apply the element $u[t]$ of $\bar{u}$

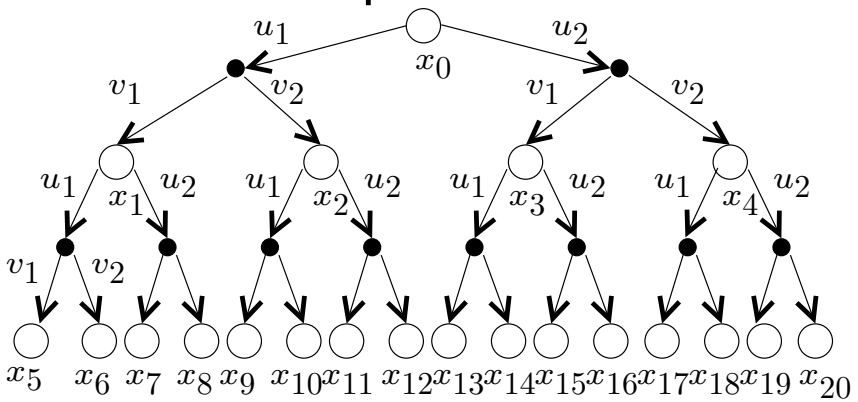
This could be rephrased it as a feed-back function (strategy) s defined over all $x[t]$ in $\bar{x}$ as $s(x[t]) = u[t+1]$ but that's an overkill

Reintroducing the Adversary

The same problem with adversary, applying the worst-case criterion, is:

$$\min_{\bar{u}} \max_{\bar{v}} c(\bar{x}) \text{ subject to } B(\bar{x}, \bar{u}, \bar{v})$$

We can enumerate all the possible control sequences and compute their cost:



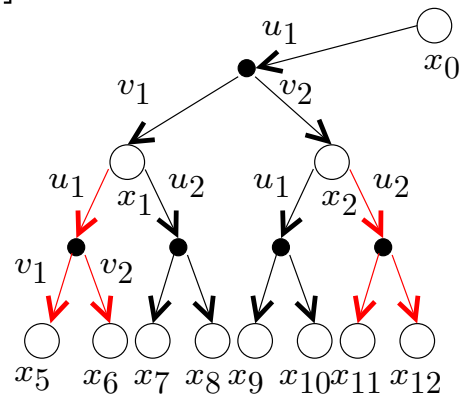
$$\begin{aligned}
u_1u_1: &\quad \max\{c(x_5), c(x_6), c(x_9), c(x_{10})\} \\
u_1u_2: &\quad \max\{c(x_7), c(x_8), c(x_{11}), c(x_{12})\} \\
u_2u_1: &\quad \max\{c(x_{13}), c(x_{14}), c(x_{17}), c(x_{18})\} \\
u_2u_2: &\quad \max\{c(x_{15}), c(x_{16}), c(x_{19}), c(x_{20})\}
\end{aligned}$$

Strategies based on Feedback

The resulting sequence is the optimal “open-loop”control achievable. It ignores information obtained during execution

If $\max\{c(x_5), c(x_6)\} < \max\{c(x_7), c(x_8)\}$
but $\max\{c(x_9), c(x_{10})\} > \max\{c(x_{11}), c(x_{12})\}$

we should apply u_1 when $x[1] = x_1$ and u_2 when $x[1] = x_2$



Control Strategies

A *control strategy* is a function $s : X \rightarrow U$ telling the controller what to do at any reachable state of the game

The following predicate indicates the fact that $\bar{x}$ is the behavior of the system in the presence of disturbance $\bar{v}$ when the controller employs strategy s:

$$B(\bar{x}, s, \bar{v}) \quad \textit{iff} \quad \begin{array}{l} x[0] = x_0 \\ u[t] = s(x[t-1]) \ \forall t \\ x[t] = f(x[t-1], u[t], v[t]) \ \forall t \end{array}$$

Finding the best strategy s is the following 2nd-order optimization problem:

$$\min_s \max_{\bar{v}} c(\bar{x}) \text{ subject to } B(\bar{x}, s, \bar{v})$$

Computing Strategies by Optimization

Finding an optimal strategy is harder than finding an optimal sequence.

In discrete finite-state systems there are $|U|^{|X|}$ potential strategies and each of them induces $|V|^k$ behaviors of length k.

In continuous domains (on continuous time) such a strategy is the solution of a partial differential equation known as the Hamilton-Jacobi-Bellman-Isaacs equation.

A strategy need not be defined all over X, only for elements reachable from x_0 when the controller employs that strategy.

Dynamic Programming

Backward value iteration, a technique for computing optimal strategies in an incremental way (Bellman)

For discrete systems the algorithm is polynomial in the size of the transition graph, which is better than the exponential enumeration of strategies

However, this polynomiality is practically irrelevant because the transition graph itself is typically *exponential* in the number of system variables

We illustrate it on a shortest path problem

Shortest Path

A subset F of X is designated as a target set, and a cost $c(x,u,v)$ is associated with each transition

The cost of a path

$$x[0] \xrightarrow{u[1],v[1]} x[1] \cdots \xrightarrow{u[k],v[k]} x[k]$$

from the initial state to a target state is

$$c(\bar{x},\bar{u},\bar{v}) = \sum_{t=1}^{k} c(x[t-1],u[t],v[t])$$

and our goal is to find the strategy that minimizes the worst-case

Value Function

An auxiliary function (value function, cost-to-go) $\vec{\mathcal{V}}$: $X \rightarrow \mathbb{R}$ is used

$\vec{\mathcal{V}}\ (x)$ is the performance of the optimal strategy for the sub-game starting from x

For "leveled" acyclic transition graphs (where all paths that reach a state x from x_0 have the same number of transitions) we have:

$$\vec{\mathcal{V}}\ (x) = 0 \qquad \text{when } x \in F$$

$$\vec{\mathcal{V}}\ (x) = \min_u \max_v (c(x,u,v) + \vec{\mathcal{V}}\ (f(x,u,v))).$$

Value Iteration

In the more general case $\vec{\mathcal{V}}$ is the fixed-point of the iteration:

$$\vec{\mathcal{V}}_0\,(x) = \begin{cases} 0 & \text{when } x \in F \\ \infty & \text{when } x \notin F \end{cases}$$

$$\vec{\mathcal{V}}_{i+1}\,(x) = \min \left\{ \begin{array}{l} \vec{\mathcal{V}}_i\,(x), \\ \min_u \max_v (c(x,u,v) + \vec{\mathcal{V}}\,(f(x,u,v))) \end{array} \right\}$$

If $\max$ is replaced by weighted sum this procedure gives the optimal average case strategy (Markov decision processes)

If summation is replaced by $\max$ we obtain the backward synthesis algorithm for automata. $\vec{\mathcal{V}}_i$ characterizes the states from which the controller cannot postpone reaching a forbidden state for more than i steps.

Forward Search

A dual shortest path algorithm (Dijkstra)

A "backward" value function $\overleftarrow{\mathcal{V}}(x)$, indicating the minimal cost for reaching x from x_0

We want to compute $\overleftarrow{\mathcal{V}}(x)$ for $x \in F$

$$\overleftarrow{\mathcal{V}}(x_0) = 0$$
$$\overleftarrow{\mathcal{V}}(x) = \min_u (c(x', u) + \overleftarrow{\mathcal{V}}(f(x', u)))$$

where x' ranges over all the immediate predecessors of x.

Same complexity as Bellman (in fact, you can reverse the graph)

A Generic Search Algorithm I

We view each incomplete path as a partial strategy defined only on states encountered along the path

We store triples $(s, x, \overleftarrow{\mathcal{V}})$ where s is a partial strategy, x is the last node in the path and $\overleftarrow{\mathcal{V}}$ is the cost for reaching x along the path.

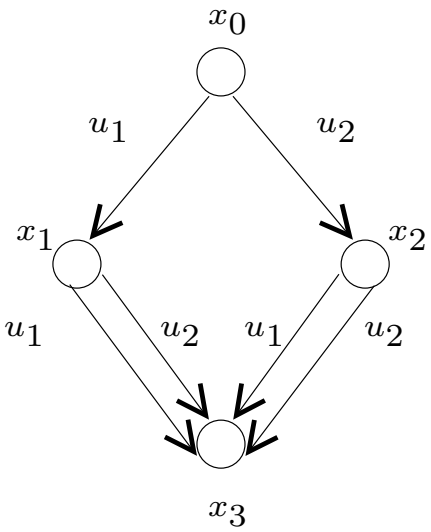
An exhaustive algorithm, a waiting list W containing partial paths that need to be explored further

When nodes are inserted at the end of W we obtain a breadth first search algorithm

A Generic Search Algorithm II

$W := \{(\emptyset, x_0, 0)\}$
repeat
  Pick a non-terminal node $(s, x, \overleftarrow{\mathcal{V}}) \in W$
  for every $u \in U$ **do**
    $(s', x', \overleftarrow{\mathcal{V}}') :=$
      $(s \cup \{x \mapsto u\}, f(x,u), \overleftarrow{\mathcal{V}} + c(x,u))$
    Insert $(s', x', \overleftarrow{\mathcal{V}}')$ into W
  end
  Remove $(s, x, \overleftarrow{\mathcal{V}})$ from W
until W contains only terminal nodes

Example



s	x	$\overleftarrow{\mathcal{V}}$
$\emptyset$	x_0	0
$\{x_0 \mapsto u_1\}$	x_1	$c(x_0, u_1)$
$\{x_0 \mapsto u_2\}$	x_2	$c(x_0, u_2)$
$\{x_0 \mapsto u_1, x_1 \mapsto u_1, \}$	x_3	$c(x_0, u_1) + c(x_1, u_1)$
$\{x_0 \mapsto u_1, x_1 \mapsto u_2, \}$	x_3	$c(x_0, u_1) + c(x_1, u_2)$
$\{x_0 \mapsto u_2, x_2 \mapsto u_1, \}$	x_3	$c(x_0, u_2) + c(x_2, u_1)$
$\{x_0 \mapsto u_2, x_2 \mapsto u_2, \}$	x_3	$c(x_0, u_2) + c(x_2, u_2)$

Guided Search: Estimation Function

An estimation function which gives an approximation of the cost of any extension of the path/strategy

This function is derived from domain-specific knowledge, macro reasoning

$$\mathcal{E}(s, x, \overleftarrow{\nu}) = \overleftarrow{\nu} + \underline{\overrightarrow{\nu}}(x)$$

$\overleftarrow{\nu}$ is the past component, something known, and $\underline{\overrightarrow{\nu}}$ is a future component, an approximation of the cost-to-go function $\overrightarrow{\nu}$

As x gets deeper, the past component becomes more dominant and the estimation more realistic.

Best First Search

The same search algorithm with W ordered according to $\mathcal{E}$ explores the most promising paths first

If $\underline{\vec{\mathcal{V}}}$ is an under approximation of $\vec{\mathcal{V}}$, then $\mathcal{E}$ is "optimistic"

In this case, we can stop the exploration when $\mathcal{E}(s, x, \overleftarrow{\mathcal{V}})$ for the first element in W is worse than any previously found solution, without missing the optimum

If we relax optimality, we can find reasonable solutions while exploring only a small fragment of the search space

Recall that all these problems are NP-hard and worse

Forward Search on Game Graphs (BFS) I

The value function becomes

$$\overleftarrow{\mathcal{V}}(x) = \min_u \max_v (c(x', u, v) + \overleftarrow{\mathcal{V}}(f(x', u, v)))$$

Because of the adversary, the result of applying a controller action u at state x is a set of states:

$$f(x, u) = \{f(x, u, v) : v \in V\}$$

The u-successor of (s, x) with s being a partial strategy is

$$\sigma((s, x), u)) = (s \cup \{x \mapsto u\}, f(x, u))$$

Forward Search on Game Graphs (BFS) II

Consider a node (s, L) where s is a partial strategy and L is the set of states reachable while following s.

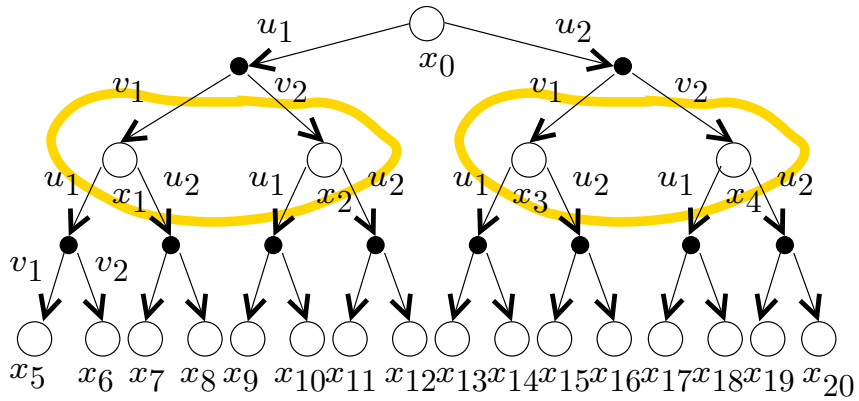
The successors of (s, L), i.e. the partial strategies that extend s and their respective sets of reachable states, are all combinations of all possible choices of u for every $x \in L$

$$\sigma(s, L) = \bigotimes_{x \in L} \{(\sigma(s, x), u) : u \in U)\}$$

where

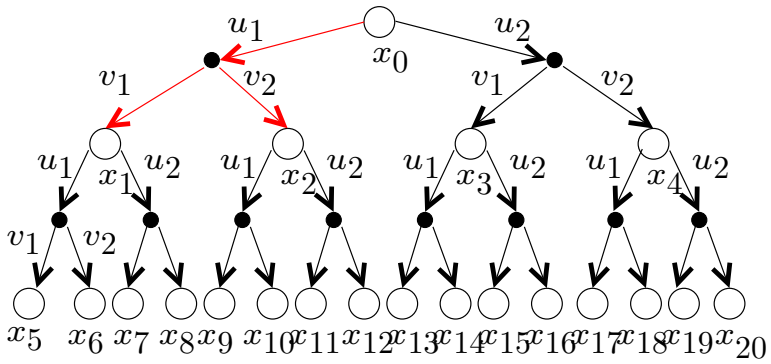
$$L_1 \otimes L_2 = \{(s_1 \cup s_2, m_1 \cup m_2) : (s_1, m_1) \in L_1, (s_2, m_2) \in L_2\}$$

Example



$$\sigma(\emptyset,\{x_0\}) = \{\sigma((\emptyset,x_0),u_1),\sigma((\emptyset,x_0),u_2)\} = \left\{ \begin{array}{l} (\{x_0 \mapsto u_1\},\{x_1,x_2\}) \\ (\{x_0 \mapsto u_2\},\{x_3,x_4\}) \end{array} \right\}$$

Example (cont.)



$$\sigma(\{x_0 \mapsto u_1\}, \{(x_1, x_2\})) =$$

$$\left\{ \begin{array}{l} (\{x_0 \mapsto u_1, x_1 \mapsto u_1\}, \{x_5, x_6\}), \\ (\{x_0 \mapsto u_1, x_1 \mapsto u_2\}, \{x_7, x_8\}) \end{array} \right\} \otimes \left\{ \begin{array}{l} (\{x_0 \mapsto u_1, x_2 \mapsto u_1\}, \{x_9, x_{10}\}), \\ (\{x_0 \mapsto u_1, x_2 \mapsto u_2\}, \{x_{11}, x_{12}\}) \end{array} \right\}$$

$=$

$$\left\{ \begin{array}{l} (\{x_0 \mapsto u_1, x_1 \mapsto u_1, x_2 \mapsto u_1\}, \{x_5, x_6, x_9, x_{10}\}), \\ (\{x_0 \mapsto u_1, x_1 \mapsto u_1, x_2 \mapsto u_2\}, \{x_5, x_6, x_{11}, x_{12}\}), \\ (\{x_0 \mapsto u_1, x_1 \mapsto u_2, x_2 \mapsto u_1\}, \{x_7, x_8, x_9, x_{10}\}), \\ (\{x_0 \mapsto u_1, x_1 \mapsto u_2, x_2 \mapsto u_2\}, \{x_7, x_8, x_{11}, x_{12}\}) \end{array} \right\}$$

Interim Summary

The BFS exhaustive forward algorithm on games is exponential in the size of the graph, so even worse than dynamic programming

A DFS forward algorithm with memorization has the same complexity as dynamic programming

Combination of DFS with estimation function can be used (and has been in game playing programs) for finding sub-optimal solutions with modest computational cost

The main advantage of forward search is that you restrict the strategy to be defined only on reachable states. If you go backwards there is no simple way to restrict the search

Application to Continuous and Hybrid Control

What do do when X, U and V are continuous?

One solution is to discretize U and V

Some toy examples:

Search-based verification (with J. Kapinski, B. Krogh and O. Stursberg)

Guiding a vehicle among obstacles (O. Ben Sik Ali)

Finding recovery sequences for power networks (A. Donze and S. Shapero)

Part II: Application to Scheduling

Principles:

State-space based approach

State: which tasks are waiting, enabled, executing (for how long), terminated

Controller actions: to choose which enabled tasks to start (or to wait)

Adversary actions: arrival of tasks, termination of tasks, evaluation of conditions, breaking of machines, change in criteria

Conceptual difficulty: not modeled naturally as synchronous games; more event-triggered than time triggered

Solution: modeling as timed automata = dense time + discrete transitions

Timed Systems

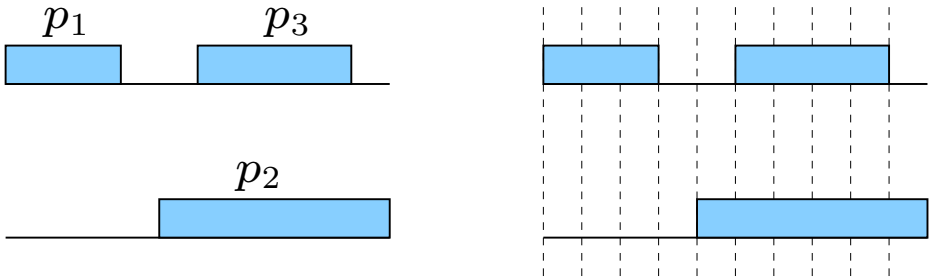
The model described so far assumes implicitly a uniform “synchronous” time scale, where something happens every time instance

Some application domains such as scheduling, digital circuit timing analysis, real-time systems, have a more “asynchronous” nature

A typical behavior consists of sparse events (starting, ending, rising, falling) separated by long periods where the only thing that happens is the passage of time

Timed automata are the natural dynamic model for such systems, on which controller synthesis can be done

Synchronous Modeling Style



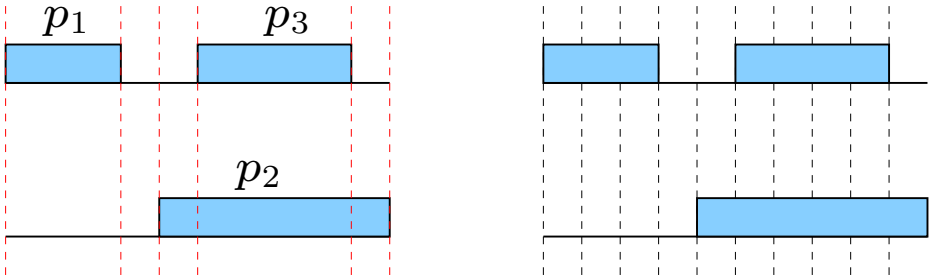
We can discretize time and have a similar type of a dynamical system where actions of the controller are $\perp$ (do nothing) and st_i (start executing p_i). The actions of the environment are $\perp$ and en_i (terminate p_i)

$$\xrightarrow{st_1} p_1 \xrightarrow{\perp} p_1 \xrightarrow{\perp} p_1 \xrightarrow{\perp,en_1} \emptyset \xrightarrow{\perp,} \emptyset \xrightarrow{\perp,st_2} p_2 \xrightarrow{\perp,st_3} \{p_2,p_3\}$$

$$\xrightarrow{\perp} \{p_2,p_3\} \xrightarrow{\perp} \{p_2,p_3\} \xrightarrow{\perp} \{p_2,p_3\} \xrightarrow{\perp,en_3} p_2 \xrightarrow{\perp,en_2} \emptyset$$

Asynchronous, Event-Triggered, Timed Style

The time index is not time but the events



$$\xrightarrow{st_1} (p_1,0) \xrightarrow{3} (p_1,3) \xrightarrow{en_1} \emptyset \xrightarrow{1} (p_2,0) \xrightarrow{1} (p_2,1) \xrightarrow{st_3} \{(p_2,1),(p_3,0)\}$$
$$\xrightarrow{4} \{(p_2,5),(p_3,4)\} \xrightarrow{en_2} (p_2,5) \xrightarrow{1} (p_2,6) \xrightarrow{en_2} \emptyset$$

Timed automata express processes that alternate between time passage (without a-priori commitment to a time step) and discrete transitions. Clocks measure elapsed time since transitions and are part of the state-space

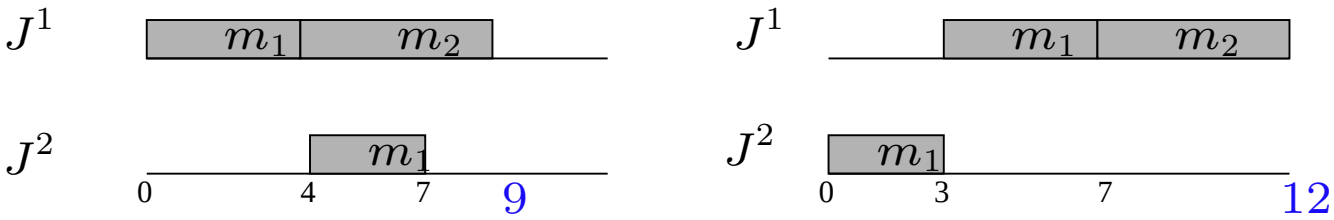
Example: Deterministic Job-Shop Scheduling

$$J^1: \quad (m_1, 4), (m_2, 5) \qquad\qquad J^2: \quad (m_1, 3)$$

Determine the execution times of the steps/tasks such that:

The termination time of the last step is minimal

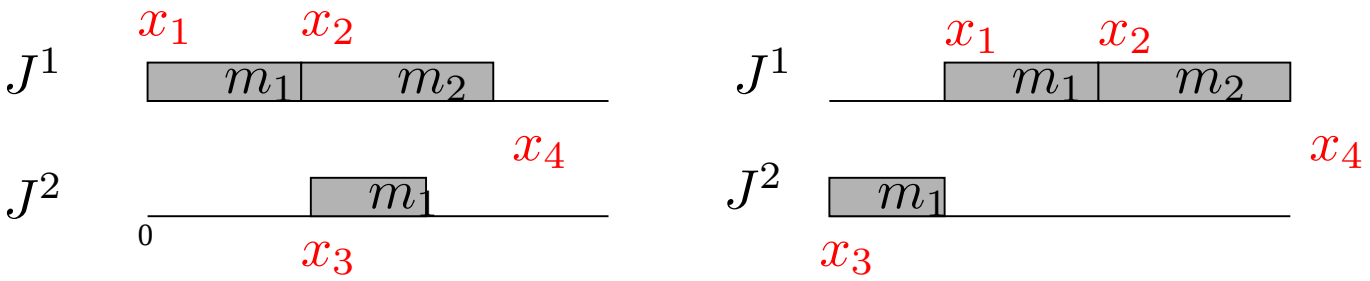
Precedence and resource constraints are satisfied



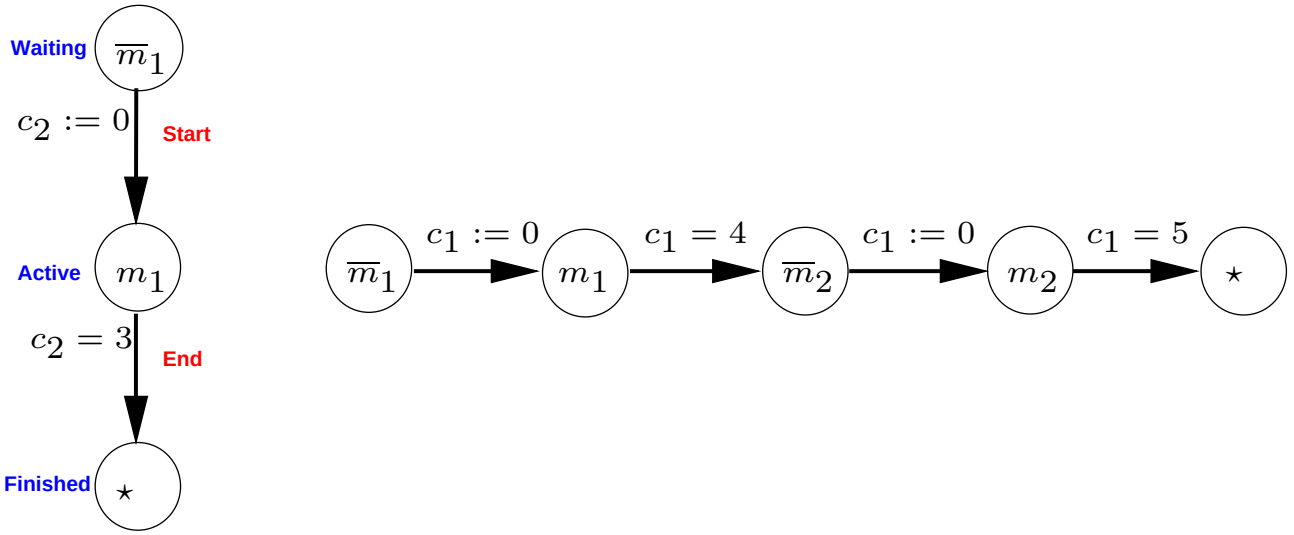
Sometimes it is better not to start a step although the machine is idle

Constrained Optimization (Bounded Horizon)

minimize x_4 subject to	(makespan)	minimize x_4 subject to
$x_2 \geq x_1 + 4$ $x_4 \geq x_2 + 5$ $x_4 \geq x_3 + 3$	(precedence)	$x_2 - x_1 \geq 4$ $x_4 - x_2 \geq 5$ $x_4 - x_3 \geq 3$
$[x_1, x_1 + 4] \cap$ $[x_3, x_3 + 3]$	(mutual exclusion)	$x_3 - x_1 \geq 4 \vee$ $x_1 - x_3 \geq 3$



Modeling with Timed Automata

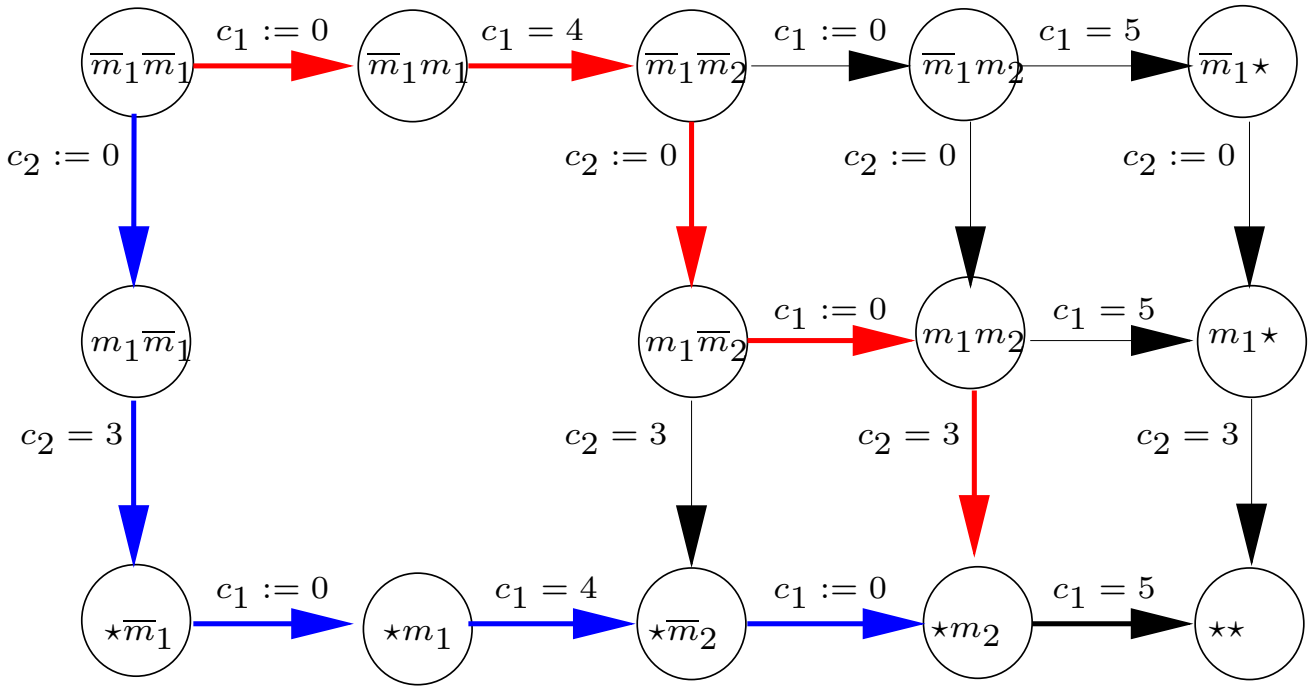


Each automaton represents the set of all possible behaviors of each task/job in isolation (respecting the precedence constraints)

The Start transitions are issued by the controller/scheduler and the End transitions by the environment

The Global Automaton

Resource constraints expressed via forbidden states in the product automaton



Optimal scheduling = shortest path problem timed automata

State-of-this-Art

Deterministic Job-Shop: search algorithms on automata (with heuristics) are not worse than other methods (with Y. Abdeddaïm, 2001)

Extension to deterministic task-graph problem. More general precedence constraints than in job-shop, uniform machines (Y. Abdeddaïm and A. Kerbaa 2003)

Extension to preemptive job-shop using stopwatch automata (Y. Abdeddaïm, 2002)

Strategy synthesis for job-shop with uncertainty in task durations. Steps of the form $(m_1, [2, 5])$. Strategy better than static worst-case (E. Asarin and Y. Abdeddaïm 2003)

Strategy synthesis for conditional precedence graph. Whether or not some tasks need to be executed will be known only after termination of other tasks (M. Bozga and A. Kerbaa)

Summary

Dynamic games are a natural model for many many problems in system design. The interesting questions about games are not necessarily those asked by “game theorists”

Clean semantic modeling precedes (but of course, does not replace) optimization algorithms

Scheduling could benefit from a general theory based on these principles