An Efficient Model for Dynamic and Constrained Resource Allocation Problems

Camille Besse & Brahim Chaib-draa
Dep. of Comp. Science, Laval University,
Quebec (Qc), Canada
{besse,chaib}@damas.ift.ilaval.ca

Abstract

Dynamic constraint satisfaction is a useful tool for representing and solving sequential decision problems with complete knowledge in dynamic world and particularly constrained resource allocation problems. However, when resources are unreliable, this framework becomes limited due to the stochastic outcomes of the assignments chosen. On the contrary, Markov Decision Processes (MDPs) handle stochastic outcomes of unreliable actions, but their complexity explodes when using state-defined constraints. We thus propose an extension of the MDP framework so as to represent constrained and stochastic actions in sequential decision making. The basis of this extension consists in modeling the evolution of a dynamic constraint network by a MDP. We first study the complexity of the problem of finding an optimal policy for this model and then we propose an algorithm for solving it. Comparison to standard MDP shows that this framework noticeably improves policy computation.

Introduction

An autonomous agent, dynamically allocating stochastic resources to incoming tasks, faces increasingly complex situations when formulating its control policy. These situations are often constrained by limited resources of the agent, time limits, physical constraints or other agents. All these hindrances explain why complexity and state space dimension increase exponentially in size of the considered problem. Unfortunately, models that already exist either consider the sequential aspect of the environment, or its stochastic one or its constrained one. To the best of our knowledge, frameworks that model two of these aspects are not numerous, and even less numerous are frameworks that consider all three.

For example, dynamic constraint satisfaction problems (DCSPs) have been introduced by Dechter & Dechter (1988) to address problems that involve dynamics in constrained problems. In DCSPs, there is typically no transition model, and thus no concept of sequence of controls. Their objective is to minimize the work needed to repair a solution when a change occurs or to find robust solutions which could face changes. However, they are not in general used to plan in a stochastic world were solution at one step influences solutions in further steps when such knowledge is available.

On the other hand, Fargier, Lang, & Schiex (1996) proposed mixed CSPs (MCSPs) and probabilistic CSPs (PCSPs) (Fargier *et al.*, 1997; Shazeer, Littman, & Keim, 1999), in which some uncontrollable variables model uncertainty. Solutions then depend on their possible values. While this model produces robust solutions, it does not deal with sequence of events or with unexpected events in which case a practical solver will have to be able to fall back to existing DCSP methods. Thus, this approach considers only the stochastic and the constrained aspects of the environment.

In their approach, Fowler & Brown (2003) included sequential decision making in their Branching Constraint Satisfaction Problem (BCSP) to model problems in which there is uncertainty in the number of variables. They construct a tree of the sequential assignment of variables depending on probability that changes income during resolution. This model includes stochastic, constrained and dynamic aspects, but does not deal with sequence of complete assignments and changes in variable set and/or constraint set as the DCSP framework does.

In the same context, Walsh (2002) proposed the Stochastic CSP (SCSP) framework that encompasses all models described above. Thus, his approach includes stochastic and constrained aspects as well as sequential decision making. However, this approach does not aim to choose the best sequence of assignments but only a sequence that is satisfied with a certain probability threshold. Indeed, there is no concept of valuation of an assignment as usually deal resource allocation problems.

Dolgov & Durfee (2005a,b) presented several approaches to represent constraints in Markov Decision Processes mainly using linear programming techniques and weakly-coupled MDPs. These approaches are very efficient but suffer of limitation to linear constraints of linear programming and restriction to weakly-coupled agents.

In this paper, we introduce a new model based on DCSPs and Markov decision processes to address con-

strained stochastic resource allocation (SRA) problems by using expressiveness and powerfulness of CSPs. We thus propose a framework which aims to model dynamic and stochastic environments for constrained resources allocation decisions. Then, we present some efficient algorithms based on a combination of last researches in both constraint satisfaction and Markov decision problems. A complexity study is also made before presenting comparative experiments. We finally conclude with some remarks on the reasons for our model's success and future work.

Background

A classical *constraint satisfaction problem* (CSP) is a triple $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, where $\mathcal{X}$ is the set of *variables*, each of them can take its possible values in a *domain* $\mathcal{D}$ (supposed here finite), and $\mathcal{C}$ is a set of *constraints* restricting the possible values of some variables. We will denote by $\mathbb{S}(P) \subset \mathcal{D}^{|\mathcal{X}|}$ the set of all assignments satisfying all constraints $\mathcal{C}$ of P. Solving a CSP is equivalent to finding one element $\sigma \in \mathbb{S}(P)$.

Dynamic Constraint Satisfaction Problem (DCSP)

According to Verfaillie & Jussien (2005) dynamic constraint satisfaction problem (DCSP) is a sequence of η CSPs $\Psi = \{P_1, ..., P_\eta\}$, each P_i depending only on changes in the definition of the previous one P_{i-1}. These changes may affect any component in the problem definition: the set $\mathcal{X}$ of variables (by adding new variables or removing a subset of $\mathcal{X}$), the set $\mathcal{C}$ of constraints (by adding, removing or modifying) and eventually domains $\mathcal{D}$ if they are not modifiable by changing variables or constraints. Formally:

Definition 1. (Dynamic CSP)
A dynamic CSP is a finite set $\Psi = \{P_i\}$ *with each* $P_i = \langle \mathcal{X}_i, \mathcal{D}_i, \mathcal{C}_i \rangle$ *where:*

- $\mathcal{X}_i$ *is the set of variables at step* i*;*
- $\mathcal{D}_i$ *is the set of domains of these variables at step* i*;*
- $\mathcal{C}_i$ *is the set of constraints restricting variables at step* i*.*

Solving a DCSP consists in finding a satisfying solution $\sigma_i \in \mathbb{S}(P_i)$ for each i, $1 \leqslant i \leqslant \eta$. Most past work on DCSPs has been devoted to finding a solution σ_{i+1} based on σ_i. For instance, solution reuse and reasoning reuse (Verfaillie & Jussien, 2005) are two approaches that benefit from previous solutions to produce future ones. However, none of these approaches attempt to formalize the evolution of changes so they are not able to predict and control them.

Hence, dynamic CSPs are inadequate to address theory of control problems. We thus present Markov decision processes which are a well known approach in this domain before proposing a new framework based on the composition of these two approaches.

Markov Decision Processes (MDP)

A Markov Decision Process (MDP) models a planning problem in which action outcomes are stochastic but the world state is fully observable. The agent is assumed to know a probability distribution for action outcomes. Formally:

Definition 2. (MDP)
A MDP is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, s_0 \rangle$ *where:*

- $\mathcal{S}$ *is a finite set of states;*
- $\mathcal{A}$ *is a finite set of actions;*
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ *is a probability distribution over* $\mathcal{S}$ *for any* $s \in \mathcal{S}$ *and* $a \in \mathcal{A}$*;*
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ *is a bounded reward function;*
- s_0 *is the initial state;*

Intuitively, $\mathcal{T}^a_{ss'} = Pr(s'|s, a)$ denotes the probability of moving to state s' when action a is performed at state s, while $\mathcal{R}^a_s$ denotes the immediate utility associated with the action a in state s.

Thus, solving an MDP aims to find an optimal policy $\pi^* : \mathcal{S} \mapsto \mathcal{A}$ that associates to each state $s \in \mathcal{S}$ an action $a \in \mathcal{A}$. π^* aims to maximize the expected reward accumulated:

$$\begin{aligned} \pi^* &= \sup_{\pi \in \Pi} V_\pi(s) \\ &= \sup_{\pi \in \Pi} \left[\sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{T}^a_{ss'} \left[\mathcal{R}^a_s + \gamma V_\pi(s') \right] \right] \end{aligned}$$

where Π is the set of all policies.

Papadimitriou & Tsisiklis (1987) have shown that finding an optimal policy as described above is one of the most difficult polynomial problems under some restrictions recalled in the section about complexity. Nevertheless, as entries can have an exponential size, this problem is one of the current major difficulties in the planning community. In fact, two main factors are responsible for this tractability problem: the exponential size of the state space and the branching factor between each state which make search barely feasible.

Many methods already exist that address the state space problem. For example, Dearden's aggregation (Dearden & Boutilier, 1997) aims to group similar states while Sutton, Precup, & Singh (1999) organize states into a hierarchy to facilitate learning and planification. Other techniques like real-time exploration like Bonet's Labeled real-time dynamic programming algorithm (Bonet & Geffner, 2003) efficiently compute the optimal policy only on reachable states from a given state s_0. Finally, Boutilier, Dearden, & Goldszmidt (2000) present factored MDPs that reduce exponentially the size of the state space while bringing closer MDPs to constraint networks. This is why this paper aims to cover the branching factor problem instead, by using environmental constraints on actions that will thus limit state space explosion. Nonetheless, as we will

see in the following sections, using constraints in factored MDPs in the context of resource allocation problems reduces also the state space. In fact, one can also imagine constraints that prune value of state variables that are not consistent with the considered problem.

Markovian Constraint Satisfaction Problem (MaCSP)

A Markovian CSP (MaCSP) is a Markov Decision Process which describes the stochastic evolution of a bounded dynamic constraint network. States represent possible configurations of the DCSP among its evolution, and actions represent assignments of each configuration of this DCSP. Thus, in a Markovian CSP, the Markov property is satisfied as a future configuration depends only on the previous configuration and the assignment chosen. Formally:

Definition 3. (MARKOVIAN CSP)
A Markovian CSP (MaCSP) is defined by a tuple $\Phi = \langle \mathcal{S}, \Psi, \{\mathcal{A}\}, \mathcal{T}, \mathcal{R}, s_0 \rangle$ *where:*

- $\mathcal{S} = \{s_i, P_{s_i}\}$ *where* s_i *is the state and each* $P_{s_i} = \langle \mathcal{X}_{s_i}, \mathcal{D}_{s_i}, \mathcal{C}_{s_i} \rangle$ *is the current state of the underlying DCSP* Ψ *in the state* s_i*;*
- $\mathcal{A}_i = \{\sigma_{s_i} = \{x_1, ..., x_\varsigma\} : x_k \in \mathcal{D}_{s_i}, \sigma_{s_i} \in \mathbb{S}(P_{s_i})\}$ *is the set of consistent assignments of variables in each state* $s_i, 1 \leqslant i \leqslant \eta$*;*
- $\mathcal{T}^{\sigma}_{s_i s_{i+1}} = Pr(s_{i+1}|s_i, \sigma)$ *is the transition probability that the assignment* σ *leads from a state* s_i *to a state* s_{i+1}*;*
- $\mathcal{R}^{\sigma}_{s}$ *is a reward function leading to a satisfying subset of* $\mathcal{S}$*;*
- s_0 *is the initial state;*

In other words, $\mathcal{S}$ models a factored state space where constraints $\mathcal{C}_{s_i}$ can exist between variables in each state s_i. Variables are partitioned in two types: decision variables that are controllable and state variables that are not. $\mathcal{A}$ is the set of all possible assignments of decision variables, and $\mathcal{A}_i$ the subset of $\mathcal{A}$ that is consistent with current constraints in $\mathcal{C}_{s_i}$. The transition function $\mathcal{T}^{\sigma}_{s_i s_{i+1}}$ represents the evolution of all variables over time (including state variables). Thus, if some constraints in $\mathcal{C}_{s_i}$ depend on state variables, then, these constraints may change over time (and in some cases appear or disappear). The reward function $\mathcal{R}^{\sigma}_{s}$ that assigns a value to each assignment of variables depending on the state s_i can be viewed in the resource allocation context as a prioritization of certain tasks over others or as the return given by achieving some tasks.

In fact, a DCSP is a particular case of MaCSP where the transition model $\mathcal{T}^{\sigma}_{s_i s_{i+1}}$ is not specified and only decision variable exists. A DCSP can then be naturally defined as a MaCSP in which the transition model is deterministic whatever assignment chosen and reward function is $\{satisfied, unsatisfied\}$.

Moreover, as MaCSPs can be seen as factored MDPs as described by Boutilier, Dearden, & Goldszmidt (2000) where some constraints exist between variables of the factored MDP, this other point of view leads us to reconsider the transition function definition. In fact, as defined above, the transition function is a huge table which describes the evolution of each variable and of each constraint from one state to another. The size of the table is obviously non polynomial in the number of variables. So, an alternative way to represent this function is to describe evolution and dependencies of variables between different states or inside a same state by a Dynamic Bayesian Network (DBN) (Dean & Kanazawa, 1990) as described by Guestrin, Koller, & Parr (2001). Boutilier, Dearden, & Goldszmidt (2000) also noticed that this representation is at worst as compact as the explicit table and exponentially better in memory space in most of cases.

Solving a MaCSP consists in finding a consistent assignment σ_{s_i} for each state s_i defining a policy $\xi^* : \mathcal{S} \mapsto \mathcal{A}$ that associates for each state $s_i \in \mathcal{S}$ an assignment $\sigma_{s_i} \in \mathbb{S}(P_{s_i})$ over the finite horizon η. ξ^* is the optimal policy over the set of all policies Ξ such as:

$$\begin{aligned} \xi^* &= \sup_{\xi \in \Xi} V_\xi(s) \\ &= \sup_{\xi \in \Xi} \left[\sum_{\sigma \in \mathcal{A}} \xi(s, \sigma) \sum_{s' \in \mathcal{S}} \mathcal{T}^{\sigma}_{ss'} \left[\mathcal{R}^{\sigma}_{s} + \gamma V_\xi(s') \right] \right] \end{aligned}$$

where $0 < \gamma < 1$ is a discount factor.

Complexity

The complexity class of satisfying a dynamic CSP is easy to show since a dynamic CSP is a linear combination of CSPs in terms of complexity: As stated by Haralick *et al.* (1978), the problem of satisfiability of a constraint network as defined in previous section is NP-complete. Thus,

Lemma 1. *The static unrestricted Constraint Satisfaction Problem is* NP*-complete.*

Theorem 1. *The Dynamic Constraint Satisfaction Problem is* NP*-complete.*

Proof. First we show that the dynamic constraint satisfaction problem is solvable in polynomial time by a nondeterministic Turing machine. This is immediate, since the Turing machine can non-deterministically choose an assignment at each step and then verify its global consistency by checking each allowed tuple of each relation. Since R, the factor size of constraints[1] is fixed, the number of tuple to be checked is $C^R_{|\mathcal{X}|} \leqslant |\mathcal{X}|^R$ which is polynomial for each step and so for the η steps.

Now we show that some NP-complete problem would be solved by the dynamic constraint satisfaction problem. The problem to be considered is the static constraint satisfaction problem whose completeness is given by lemma 1. A CSP can be represented as a DCSP which has the same variables set and whose constraints

[1] i.e. the maximum number of variables implied in each constraint, e.g. R = 2 for binary constraints.

are added sequentially to obtain the targeted CSP. Thus, solving this DCSP will solve the corresponding CSP. This transformation is clearly polynomial. □

Furthermore, Papadimitriou & Tsisiklis (1987) have shown the following:

Lemma 2. *(Papadimitriou & Tsisiklis, 1987) Given an MDP $\mathcal{M}$, a horizon T, and an integer K, the problem of computing a policy in $\mathcal{M}$ under horizon T that yields total reward at least K is P-complete.*

Thus, as stated for MDPs, it is necessary to place some restrictions in order to hold the upper bounds. First, $\eta \ll |\mathcal{S}|$ and $|\mathbb{S}(P_s)| \ll |\mathcal{S}|, \forall s \in \mathcal{S}$. Then, we also assume that tables for the transition and the reward function can be represented with a constant number of bits. Under these restrictions:

Theorem 2. *Given an MaCSP Φ, a horizon η, and an integer K, the problem of computing a policy in Φ under horizon η that yields total reward at least K is* NP*-complete.*

Proof. The same methodology is applied as for proposition 1. We first show that MaCSPs are solvable in polynomial time by a non-deterministic Turing machine. Since the Turing machine can non-deterministically choose an assignment in each state of the underlying MDP and then verify its consistency by checking each allowed tuple of each relation of the current CSP. Thus the number of tuples to be checked is still polynomial in $|\mathcal{X}| \times |\mathcal{S}|$.

To show that solving a MaCSP helps to solve another NP-complete problem, one can show that solving an MaCSP is equivalent to solve a DCSP where changes between two CSP are the same whatever assignment chosen and transition probability is one. □

In fact we assume that, as in Markov Decision Processes, state space, action space and transition function are given and then we will just have to verify if actions are consistant with constraints which is linear in size of $\mathcal{A}$. Unfortunately, the practical problem associated to it is not as simple as stated above and we can conjecture that solving this problem is harder than it seems to be. In fact, we have to find *all the solutions* of a CSP in each state of the MDP to get the action space and this is already a #P-complete problem. However, we intend that this framework aims to reduce branching factor by action pruning, but do not aim to be applied to huge instances of dynamic CSPs.

Algorithms Combination

As MaCSPs mixes a MDP and a CSP, two approaches for which different algorithms have been proposed, we have elaborated an algorithm (as depicted in Alg. 1) which uses efficient existing techniques developed in those contexts. Precisely, we chose a real-time dynamic programming (RTDP) approach for the Markovian aspect of MaCSPs since it is considered as the most effective both in time and in quality. In fact, RTDP-class algorithms are shown to converge to the optimal policy. We found in the literature several derivative versions of RTDP algorithms. We chose Focused RTDP (FRTDP)

Algorithm 1 Focused RTDP for MaCSPs

function initNode(s):
  //Implicitly called the first time each state s is touched
  $(s.L, s.U) \leftarrow (\mathcal{H}_L, \mathcal{H}_U)$; $s.prio \leftarrow \Delta(s)$
end function

function FRTDP(s_0, ε,$\mathcal{H}_L$,$\mathcal{H}_U$,D_0,k_D):
  $D \leftarrow D_0$
  while $s_0.U - s_0.L > \varepsilon$ **do**
    $(q_p, n_p, q_c, n_c) \leftarrow (0, 0, 0, 0)$
    trialRecurse(s_0,$W = 1$,$d = 0$)
    if $(q_c/n_c) \geqslant (q_p/n_p)$ **then** $D \leftarrow k_D D$
  end while
end function

function trialRecurse(s, W, d):
  $(a^*, s^*, \delta) \leftarrow$ backup(s)
  trackUpdateQuality($\delta W, d$)
  if $\Delta(s) \leqslant 0$ **or** $d \geqslant D$ **then return**
  trialRecurse(s^*,$\gamma T^{a^*}_{s,s^*} W$,$d + 1$)
  backup(s)
end function

function trialUpdateQuality(q, d):
  if $d > D/k_D$ **then**
  $(q_c, n_c) \leftarrow (q_c + q, n_c + 1)$
  else $(q_p, n_p) \leftarrow (q_p + q, n_p + 1)$
end function

function backup(s_i):
  $\mathcal{A}_i \leftarrow$ bucketElimination(s_i)
  $s_i.L \leftarrow \max_{a \in \mathcal{A}_i} \mathrm{QL}(s_i, a)$
  $u \leftarrow \max_{a \in \mathcal{A}_i} \mathrm{QU}(s_i, a)$
  $a^* \leftarrow \sup_{a \in \mathcal{A}_i} \mathrm{QU}(s_i, a)$
  $\delta \leftarrow |s_i.U - u|$
  $s_i.U \leftarrow u$
  $p \leftarrow \max_{s_{i+1} \in \mathcal{S}} \gamma T^{a^*}_{s_i, s_{i+1}} s_{i+1}.prio$
  $s^* \leftarrow \sup_{s_{i+1} \in \mathcal{S}} \gamma T^{a^*}_{s, s_{i+1}} s_{i+1}.prio$
  $s_i.prio \leftarrow \min(\Delta(s_i), p)$
  return (a^*, s^*, δ)
end function

function $\Delta(s)$:
  return $|s.U - s.L| - \varepsilon/2$
end function

function QL(s, a):
  return $R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \gamma T^a_{s,s'} s'.L$
end function

function QU(s, a):
  return $R(s, a) + \gamma \sum_{s' \in \mathcal{S}} \gamma T^a_{s,s'} s'.U$
end function

developed by Smith & Simmons (2006) which is currently, according to the authors, the most efficient, and which principally uses two bounds. The use of a lower bound offers guarantees in terms of value, and the upper bound guides efficiently the search. Moreover, CSP's objective functions easily model lower bounds in planning problems. Thus, those kinds of algorithms can be also improved by CSP methods.

Concerning the constrained aspect, we use a bucket elimination procedure (line 26) that allows reducing the branching factor by selecting only feasible actions in a given state. Bucket elimination is essentially a method that propagates constraints applied on a variable of a CSP to the domain of the other variables involved in these constraints, and then eliminates the variable. Once all but one variables have been eliminated, a backward propagation on eliminated variables gives the set of all solutions. For details on bucket elimination, refer to (Dechter, 2003, chap. 13.3.3).

As in all RTDP-class, FRTDP's execution consists in trials (line 17) that begin in a given initial state s_0 and then explore reachable states of the state space, selecting actions according to an upper bound. Once a final state is reached, it performs backups (line 18) on the way back to s_0. Backups consists essentially in Bellman updates which are applied on each bound (line 27, 28), on the priority of the state (line 32) and on the quality of the trial: the larger is the update on the upper bound, the better is the quality (line 30). This function returns the currently optimal action based on the upper bound, the next state to explore based on the priority and the quality of the backup.

As previously stated, FRTDP maintains a lower bound and uses a priority criterion (line 32) to select actions outcomes and to detect trial termination. The lower bound is used to establish the policy and it also contributes in the priority calculation of states to expand on the fringe of the search tree (line 34). Trial termination detection has been modified and improved by adding an adaptive maximum depth D (line 16) in the search tree in order to avoid over-committing to long trials early on. Indeed, the maximum depth D is lengthened (line 22) each time the trial is not useful enough. This usefulness is represented by δW where δ measures how much the update changed the upper bound value of s and W the expected amount of time the current policy spends in s, adding up all possible paths from s_0 to s. Refer to the pseudo-code of Alg. 1 and to Smith & Simmons' paper (Smith & Simmons, 2006) for details.

Experimental Results

A typical class of problems that can be solved by MaCSP are the dynamic, constrained and unreliable resource allocation problems. A number of different tasks must be achieved and actions consist in assigning various resources at different times to each of these tasks. Moreover, there are constraints on the resources, like time constraints, limited amount and interactions between those resources. Furthermore, these resources are unreliable in the sense that some of them are more effective when assigned to specific tasks and consequently probabilities are attached to their capacity to achieve tasks. In this case, the objective is to maximize over the temporal horizon the probability to achieve each task while respecting given constraints.

MaCSP is particularly suited for this problem since it involves stochasticity in actions that correspond to assignment of resources to tasks. Moreover, constraints on resources can easily be modeled in this framework using CSP. In a classical MDP, all constraints must be hard coded in states and hence will increase the state space while preventing constraints from evolving along time line. MaCSPs offer both the control on events via Markov processes and expressiveness via CSPs without exploding complexity (see theorem 2).

A typical example of this class of problems is the Dynamic Weapon-Target Allocation (DWTA) problem described by Hosein, Athans, & Walton (1988) which has been shown to be NP-complete (Lloyd & Witsenhausen, 1986) even in the static unconstrained case.

The DWTA problem on which we experiment our algorithm is described as follows: Consider a naval platform attacked by a set $\mathcal{M}$ of m threats. This platform owns a set $\mathcal{W}$ of w weapons with limited ammunitions to defend itself on a finite temporal horizon T. Thus its objective is to survive the attack by destroying threats with its weapons. Unfortunately, weapons are unreliable and weapon w_i have a probability p_{ij}^t to destroy threat m_j at time $t \in \{0, ..., T\}$. As a result, the problem can be formally described by:

$$Maximize: \quad \sum_{t=1}^{T} \sum_{i=1}^{m} \left[1 - \prod_{j=1}^{w} (1 - \alpha_{ij}^t p_{ij}^t) \right]$$

$$under \quad \alpha_{ij}^t \in \{0, 1\} \quad and \quad \sum_{i=1}^{m} \sum_{j=1}^{w} \alpha_{ij}^t = 1$$

$$t \in \{1, ..., T\}$$

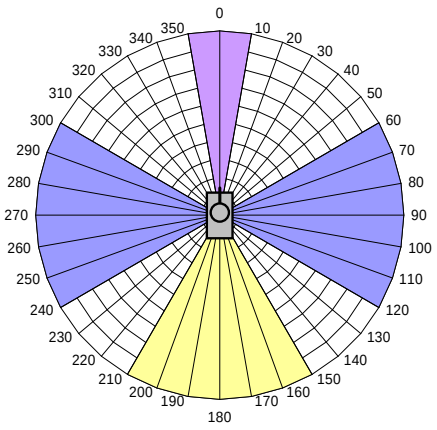
where $\alpha_{ij}^t = 1$ represents the assignation of weapon w_i on threat m_j. The solution thus consists in a sequence of assignments of α_{ij}^t for each $t \in \{1, ..., T\}$.

C_1:	A weapon must "see" the target (cf. table 2)
C_2:	A ML must be guided by a STIR from fire time to interception time,
	A Gun must use a STIR at fire time,
C_3:	Two STIRs cannot target the same threat.

Table 1: Examples of DWTA constraints

Moreover, the time for a weapon to intercept a threat depends on the range, the type and the speed of the threat and weapons also cannot freely fire at threats. Some constraints apply depending on their incoming azimut and their distance from the platform. In our example, we consider that the platform is equipped with two

Separate & Track Illumination Radar (STIR), two Missile Launchers (ML), a mid-ranged Gun and a Close-In Weapon System (CIWS). To ease example understanding all threats are assume to be the same type but with different starting range and speed[2]. The table 1 describes how weapons are constrained and table 3 what their probabilities of success are.



Base State	0 to 360°	1 STIR, 1 Gun, 1 CIWS, 2 MLs
Sector	Angles	Difference from base state
A ■	350 to 10°	No CIWS
B	10 to 60°	No difference – Base state
C ■	60 to 120°	An additional STIR
D	120 to 150°	No difference – Base state
E ■	150 to 210°	No Gun
F	210 to 240°	No difference – Base state
G ■	240 to 300°	An additional STIR
H	300 to 350°	No difference – Base state

Table 2: Examples of DWTA constraints: Blind zones

Weapon	Range	Probability of success
ML	From 2.2 to 20km	95%
Gun	From 1.5 to 5km	50%
CIWS	From 0.2 to 2km	30%

Table 3: Examples of DWTA outcomes of actions

In this application, we chose weapons as variables of the underlying DCSP and threats as values since constraints are mainly between weapons. The constraint network is trivial but is complex enough to prune the search and offers a gain versus a classical Markov Decision Process. Figure 1 shows the initial CSP and how it may evolve during an episode: There exists unary constraints that specify which threats are reachable regarding their distance from platform and the range of weapon, and binary constraints that bind firing weapons to STIRs depending on threats STIRs can "see".

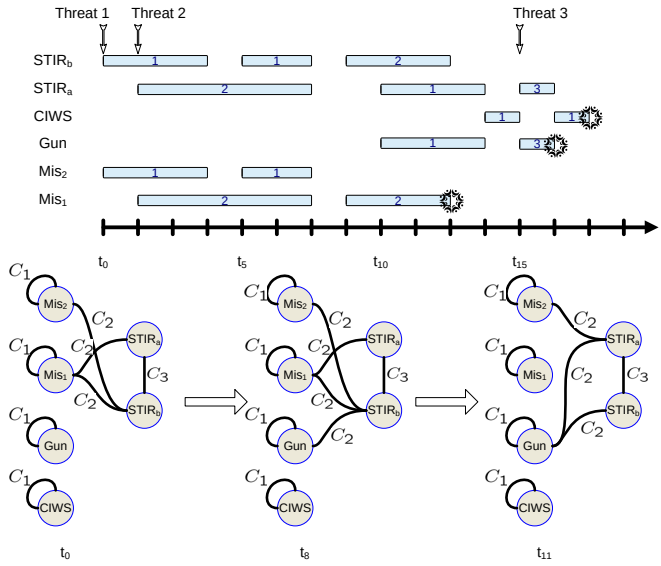


Figure 1: Example of a plan and the DCSP evolution during an episode

We then compare our framework to a classical MDP by specifying that actions that are not feasible have a null probability of success (e.g. a ML which fire to a too far target will never succeed): results in figure **??** show that MaCSP reduces the number of backups made and thus the policy computational time. Results were obtained with a FRTDP algorithm with the worst lower and upper bounds[3].

Tendency in figure 2 shows that MDP computation is exponential in number of actions and also in number of tasks as it could be forecast. As a consequence, MaCSP which prunes useless actions has an exponential gain over the classical MDP. In a same way, as the branching factor depends on the number of actions, the number of backups (Figure 3) that FRTDP needs to converge with bad heuristics is also exponentially reduced. Results for four tasks and over with Markov Decision Processes were not obtainable due to lack of memory space.

[2]Threats speed is assume to remain constant all over an episode.

[3]Null lower bound in every state except in goal states and the upper bound is computed as if actions were deterministic.

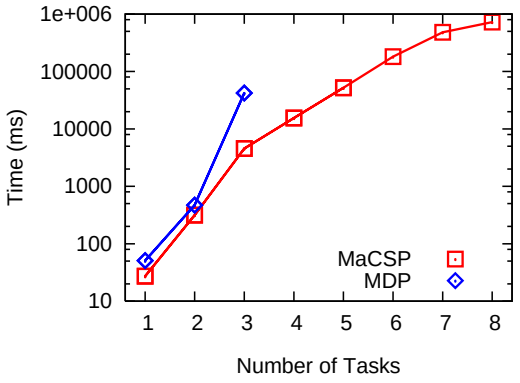


Figure 2: Time (ms) vs # Tasks

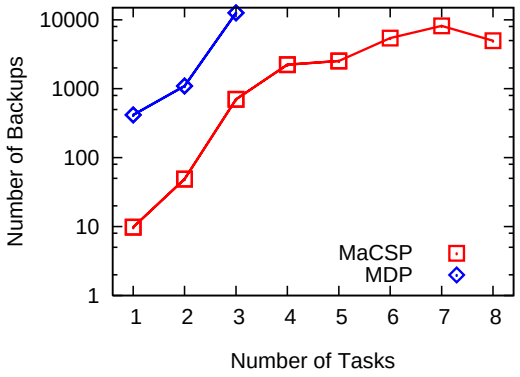


Figure 3: # Backups vs # Tasks

Discussion & Further Work

This paper proposed a useful framework which combines Markovian Decision Processes and Constraints Satisfaction Problems. We studied the complexity of this model, proposed a simple algorithm to solve it and gave an example of application. Objectives of this approach have been to model actions space reduction due to environmental constraints and then reduce the branching factor of the state space in order to facilitate the search of optimal policies with efficient algorithms like the Real-Time Dynamic Programming family (Bonet & Geffner, 2003; McMahan, Likhachev, & Gordon, 2005; Smith & Simmons, 2006).

Nonetheless, it has some drawbacks: the action space of a MaCSP with x constrained variables which have each a domain of size d is in $\mathcal{O}(x^d)$ which is obviously exponential in size of the domains. Thus, in the worst case, if there not enough constraints to prune the action space it will be as prejudicial to search all the solutions of this huge CSP, which is a #P-complete problem, as searching directly a solution with an efficient algorithm (assuming all actions are feasible but have a null probability to lead to another state). However, in the average, a gain may be done.

Regarding further, apart from studying the efficiency of this framework in some others problems, there are mainly two future research avenues. First, factored MDP of Boutilier, Dearden, & Goldszmidt (2000) is a very well suited framework to apply MaCSP, with constraints between state variables, allowing also to prune non consistent states if *a priori* knowledge about environment is available. Some work on framework generalization has already been made by Pralet, Verfaillie, & Schiex (2006) where an algebraic framework called PFU (standing for Plausibility, Feasibility, Utility) that encompasses MDP, CSP and also Bayesian Networks and Influence Diagrams has been proposed. However this framework does not encompass MaCSP since our framework models the evolution of constraints by allowing constraints between decision variables and state variables. Studying the consequences of an extension of PFU in this way could thus be interesting.

The second interesting research avenue is about Graphical Games Theory proposed by Kearns, Littman, & Singh (2001). Constraint satisfaction algorithms were recently applied to solve graphical games by representing interactions between agents by constraints. Graphical Games could lead by the mean of this framework to stochastic graphical games with few efforts.

Nevertheless, some other aspects still have to be studied in MaCSPs such as optimality criteria, bounds on error made by adding constraints and existing algorithm convergence. This framework also opens many research avenues since it combines two well-known techniques where much work has been done in recent years. Learning and Reinforcement Learning is one example from the Markovian community; solution reuse and reasoning reuse is another example from the dynamic CSP community.

Acknowledgements

We would like to thank Pascal Tesson for his helpful comments on complexity results, Gérard Verfaillie and Cédric Pralet for their comments on the model and reviewers for their useful overall suggestions.

References

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *ICAPS*, 12–31.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 2000. Stochastic dynamic programming with factored representations. *Artificial Intelligence* 121(1-2):49–107.

Dean, T., and Kanazawa, K. 1990. A model for reasoning about persistence and causation. *Comput. Intell.* 5(3):142–150.

Dearden, R., and Boutilier, C. 1997. Abstraction and Approximate Decision-Theoretic Planning. *Artificial Intelligence* 89(1–2):219–283.

Dechter, R., and Dechter, A. 1988. Belief Maintenance in Dynamic Constraint Networks. In *AAAI*, 37–42.

Dechter, R. 2003. *Constraint Processing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Dolgov, D. A., and Durfee, E. H. 2005a. Computationally-Efficient Combinatorial Auctions for Resource Allocation in Weakly-Coupled MDPs. In *AAMAS*.

Dolgov, D. A., and Durfee, E. H. 2005b. Stationary Deterministic Policies for Constrained MDPs with Multiple Rewards, Costs, and Discount Factors. In *IJCAI*.

Fargier, H.; Lang, J.; Martin-Clouaire, R.; and Schiex, T. 1997. Traitement de problèmes de décision sous incertitude par des CSPs. *Revue d'Intelligence Artificielle* 11(3):375–398.

Fargier, H.; Lang, J.; and Schiex, T. 1996. Mixed Constraint Satisfaction: A Framework for Decision Problems under Incomplete Knowledge. In *AAAI/IAAI, Vol. 1*, 175–180.

Fowler, D. W., and Brown, K. N. 2003. Branching Constraint Satisfaction and Markov Decision Problems compared. *Annals of Operation Research* 118:85–100.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Multiagent planning with factored mdps. In *NIPS*, 1523–1530.

Haralick, R. M.; Davis, L. S.; Rosenfeld, A.; and Milgram, D. L. 1978. Reduction Operations for Constraint Satisfaction. *Information Sciences* 14(3):199–219.

Hosein, P.; Athans, M.; and Walton, J. 1988. Dynamic Weapon-Target Assignment Problems with Vulnerable C2 nodes. In *Proceedings of the 1988 Command and Control Symposium*, 240–245.

Kearns, M. J.; Littman, M. L.; and Singh, S. P. 2001. Graphical models for game theory. In *UAI*, 253–260.

Lloyd, S. P., and Witsenhausen, H. S. 1986. Weapons allocation is np-complete. In *Proceedings of the 1986 Summer Computer Simulation Conference*.

McMahan, H. B.; Likhachev, M.; and Gordon, G. J. 2005. Bounded Real-Time Rynamic Programming: RTDP with Monotone Upper Bounds and Performance Guarantees. In *ICML*, 569–576.

Papadimitriou, C., and Tsisiklis, J. N. 1987. The Complexity of Markov Decision Processes. *Math. Oper. Res.* 12(3):441–450.

Pralet, C.; Verfaillie, G.; and Schiex, T. 2006. Decision with Uncertainties, Feasibilities, and Utilities: Towards a Unified Algebraic Framework. In *ECAI*, 427–431.

Shazeer, N. M.; Littman, M. L.; and Keim, G. A. 1999. Solving Crossword Puzzles as Probabilistic Constraint Satisfaction. In *AAAI/IAAI*, 156–162.

Smith, T., and Simmons, R. G. 2006. Focused Real-Time Dynamic Programming for MDPs: Squeezing More Out of a Heuristic. In *AAAI/IAAI*.

Sutton, R. S.; Precup, D.; and Singh, S. P. 1999. Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artif. Intell.* 112(1-2):181–211.

Verfaillie, G., and Jussien, N. 2005. Constraint Solving in Uncertain and Dynamic Environments: A Survey. *Constraints* 10(3):253–281.

Walsh, T. 2002. Stochastic Constraint Programming. In *Proceedings of the 15th ECAI.*, volume 8. IOS Press.