Distributed Optimal Planning: an Approach by Weighted Automata Calculus

Eric Fabre*, Loïg Jezequel**

***Abstract*— We consider a distributed system modeled as a possibly large network of automata. Planning in this system consists in selecting and organizing actions in order to reach a goal state in an optimal manner, assuming actions have a cost. To cope with the complexity of the system, we propose a distributed/modular planning approach. In each automaton or component, an agent explores local action plans that reach the local goal. The agents have to coordinate their search in order to select local plans that 1/ can be assembled into a valid global plan and 2/ ensure the optimality of this global plan. The proposed solution takes the form of a message passing algorithm, of peer-to-peer nature: no coordinator is needed. We show that local plan selections can be performed by combining operations on weighted languages, and then propose a more practical implementation in terms of weighted automata calculus.**

***Index Terms*— factored planning, distributed planning, optimal planning, discrete event system, distributed constraint solving, distributed optimization, weighted automaton, K-automaton, string to weight transducer, formal language theory**

I. Introduction

A planning problem [1] consists in optimally selecting and organizing a set of actions in order to reach a goal state from a given initial state. These "states" correspond to tuples $(v_i)_{i\in I}$ of values, one per variable $V_i, i \in I$, and the actions read and write on subsets of these variables. Expressed in these general terms, one easily guesses that a planning problem "simply" amounts to finding a path from an initial state to a (set of) goal state(s) in an automaton. In reality, the problem is more complex in several respects. First of all, the underlying automaton that encodes the problem is generally huge: the state space explodes, due to its vector nature, and actions operate on few components of the state vector, so a single action results in a huge number of transitions. Therefore, finding a path to the goal in such a huge automaton is not a trivial task and requires dedicated algorithms. Secondly, there exist planning problems of different difficulties. Some are more on the side of constraint solving: they admit few complex solutions, or even none, and one should dedicate his efforts to finding one solution, or to proving that there is no solution at all. Other problems are more accessible, in the sense that one can easily prove the existence of many solutions. The difficulty then amounts to finding the best one in an efficient manner, where "best" means that some criterion should be minimized, for example the number of actions in the plan, or the total cost of the plan, assuming each action involves some cost. The present paper addresses this second family of problems.

In order to address planning problems of growing size and complexity, several research directions have been recently explored. They essentially try to make use of the locality of actions, *i.e.* the fact that an action involves a small number of variables. One can for example take advantage of the concurrency of actions: when two actions are simultaneously firable and involve different sets of variables, they need not be ordered in a plan. This results in search strategies that handle plans as partial orders of actions rather than sequences, which reduces the search space [2], [3]. A stronger trend is known as "factored planning", and aims at solving planning problems by parts [5], [6], [7], [8]. Formally, one can imagine that the action set is partitioned into subsets, each subset representing an "agent." So each agent can only influence part of the resource set. The idea is then that one should build a plan for each agent, which corresponds to a smaller planning problem, and at the same time ensure that all such local plans are compatible, *i.e.* can be assembled to form a valid global plan. The difficulty is of course to obtain this compatibility of local plans: this is where the sparse interaction graph of agents is exploited, and where one may obtain a complexity gain.

The results presented here elaborate on this idea, but adopt a radically new perspective on the problem. Specifically, we assume that agents are sufficiently small to enable the handling of *all* local plans. We then focus on the *distributed* computations that 1/ will select local plans of each agent that can be extended into (or that are projection of) a global plan, and 2/ will at the same time select the tuple of local plans (one per agent) that corresponds to the best global plan. As a side-product, we also obtain global plans that are partially ordered sets of actions.

Our approach first encodes the planning problem as a reachability problem in a network of automata, one automaton per agent (section II). We then make use of classical tools in formal language theory, distributed constraint solving [10], [11], [9], distributed optimization [13] (section III), and weighted automata calculus [16], [17] (section IV) to solve the problem. Taken separately, none of these tools is original, but their assembling certainly is, and we believe this opens a promising research direction about planning problems.

* E. Fabre is with the DistribCom team, INRIA Centre Rennes - Bretagne Atlantique, Rennes, France, `eric.fabre@inria.fr`

** L. Jezequel is with ENS Cachan Bretagne, Rennes, France, `loig.jezequel@eleves.bretagne.ens-cachan.fr`

This work was partly supported by the FAST program (Franco-Australian program for Science and Technology), Grant 18616NL, and by the European Community's 7th Framework Programme under project DISC (DIstributed Supervisory Control of large plants), Grant Agreement INFSO-ICT-224498. This work was also supported by the joint research lab between INRIA and Alcatel-Lucent Bell Labs.

II. Planning in networks of automata

A. From planning to distributed planning

The definition of a planning problem assumes first a finite set of state variables $\{V_i\}_{i\in I} \triangleq V_I$, taking values in finite domains $\mathcal{D}_i$. The initial state is a specific tuple $(v_i)_{i\in I}$ and we assume here a set of goal states in product form $\times_i G_i$ with $G_i \subseteq \mathcal{D}_i$. The second ingredient is a finite collection of actions $\{a_k\}_{k\in K}$. An action a_k usually involves a small subset of variables $\mathcal{V}(a_k) \subseteq V_I$. To be firable, a_k must read specific values on (some of) the $\mathcal{V}(a_k)$, which form the preconditions of a_k. The firing of a_k writes specific values on (some of) the variables $\mathcal{V}(a_k)$, the so-called effect of a_k. In this paper, to avoid non-central technical complications, we assume that each a_k both reads and writes on all its variables $\mathcal{V}(a_k)$. Finding an optimal plan consists in selecting and organizing actions to go from the initial state $(v_i)_{i\in I}$ to one of the goal states of $G = \times_{i\in I} G_i$, and at the same time minimize a criterion like the number of actions for example. This is made more formal below. Planning problems are generally expressed in different formalisms: STRIPS or PDDL assume binary variables, while SAS+ [5] or the related notion of Domain Transition Graph [4] assume multi-valued variables. Here we are closer to this second family.

To make this setting distributed, we partition the variable set V_I into subsets V_{I_n}, with $\uplus_n I_n = I$, corresponding to the "agents" $\mathcal{A}_n$ (one could equivalently partition the action set). Agent $\mathcal{A}_n$ is provided with all the actions a_k restricted to its variables V_{I_n}, $a_{k|V_{I_n}}$, such that $\mathcal{V}(a_k) \cap V_{I_n} \neq \emptyset$. Agent $\mathcal{A}_n$ represents the restriction of the global planning problem to the subset of variables V_{I_n}. Since actions are now split into different agents, we introduce below a standard product formalism that synchronizes agents on these shared actions and allows us to recover the global planning problem from its restrictions. This way of splitting a planning problem into parts is standard and has been adopted by several "factored planning" approaches [5], [6], [7], [8], [12]. It is generally used to build global plans by parts, starting by some agent, looking for a local plan in this agent, and then trying to progressively extend it with a compatible local plan of another agent, and so on. Here, the compatibility of local plans corresponds to an agreement to jointly perform or reject some shared actions (this is formalized below).

In this paper, we adopt a different perspective. First of all, we look for a distributed planning approach and abandon the idea of a coordinator in charge of assembling the proposed local plans into a global one. We rather assume that the agents themselves are in charge of computations, relying on message exchanges, and that they only handle local information (typically sets of local plans), not global one. Secondly, rather than a search for *one* possible global plan (which assumes many backtrackings in the assembling of agent proposals), the method we propose is rather based on a filtering idea: it explores *all* local plans of an agent, and removes those that can not be the restriction of a valid global plan. Finally, beyond this filtering idea, the procedure we propose implements as well a distributed optimization function that will compute (all) the optimal global plan(s). To our knowledge, this is the first approach to optimal factored planning.

We proceed by formalizing the notion of agent as a weighted automaton, and the notion of plan as a word in the language of this automaton.

B. Weighted automata and their languages

Let $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$ denote the so-called tropical commutative semiring $(\mathbb{R}^+ \cup \{+\infty\}, \min, +, +\infty, 0)$. Following [16], a weighted automaton (WA), or equivalently a string to weight transducer, is a tuple $\mathcal{A} = (S, I, F, \Sigma, c_I, c_F, c)$ where S is a finite set of states, among which $I, F \subseteq S$ represent initial and final states respectively, Σ is a finite alphabet of actions, $c_I : I \to \mathbb{K}\setminus\{\bar{0}\}$ and $c_F : F \to \mathbb{K}\setminus\{\bar{0}\}$ are weight/cost functions on initial and terminal states. The last parameter $c : S\times\Sigma\times S \to \mathbb{K}$ is a weight or cost function over all possible transitions of $\mathcal{A}$, with the convention that only transitions in $T = c^{-1}(\mathbb{K}\setminus\{\bar{0}\})$ are possible in $\mathcal{A}$ (transitions of infinite cost are impossible). Given a transition $t \in T$, we denote by $(s^-(t), \sigma(t), s^+(t))$ its three components in $S \times \Sigma \times S$. A path $\pi = t_1...t_n$ is a sequence of transitions such that $s^+(t_i) = s^-(t_{i+1})$, $1 \leq i \leq n-1$. We define $s^-(\pi) = s^-(t_1)$, $s^+(\pi) = s^+(t_n)$, $\sigma(\pi) = \sigma(t_1)...\sigma(t_n)$ and for the cost of this path $c(\pi) = c(t_1) \otimes \ldots \otimes c(t_n)$, *i.e.* the sum of transition costs. The path π is accepted by $\mathcal{A}$, denoted $\pi \models \mathcal{A}$, iff $s^-(\pi) \in I$ and $s^+(\pi) \in F$. The language of $\mathcal{A}$ is defined as the formal power series

$$\mathcal{L}(\mathcal{A}) \quad = \quad \sum_{u\in\Sigma^*} \mathcal{L}(\mathcal{A}, u)\, u \tag{1}$$

where coefficients are given by

$$\mathcal{L}(\mathcal{A}, u) \quad = \quad \bigoplus_{\substack{\pi \models \mathcal{A} \\ u = \sigma(\pi)}} c_I[s^-(\pi)] \otimes c(\pi) \otimes c_F[s^+(\pi)] \tag{2}$$

$\mathcal{L}(\mathcal{A}, u)$ is the weight of the action sequence (or word) u, and it is thus obtained as the minimum weight over all accepted paths of $\mathcal{A}$ that produce u, with the convention that $\mathcal{L}(\mathcal{A}, u) = +\infty$ (*i.e.* $\bar{0}$) when no such path exists. The word u is said to belong to the language of $\mathcal{A}$ iff $\mathcal{L}(\mathcal{A}, u) \neq \bar{0}$.

One can associate a transition function $\delta : S \times \Sigma \to 2^S$ to $\mathcal{A}$ by $\delta(s, \sigma) = \{s', \exists(s, \sigma, c, s') \in T\}$, which extends naturally to state sets $S' \subseteq S$ by union and to words $u \in \Sigma^*$ by composition. We also denote $\delta(s) = \cup_{\sigma\in\Sigma}\delta(s, \sigma)$. $\mathcal{A}$ is said to be deterministic when $|I| = 1$ and δ is a partial function over $S \times \Sigma$, *i.e.* from any state s there is at most one outgoing transition carrying a given label σ.

A WA can be considered as an encoding of a planning problem with action costs. Optimal planning then consists in finding the word(s) u of minimal weight in the language $\mathcal{L}(\mathcal{A})$, or equivalently the optimal accepted path(s) in $\mathcal{A}$, which can be solved by traditional graph search. In the sequel, we examine the case where $\mathcal{A}$ is large, but obtained by combining smaller planning problems (called components), one per agent.

C. From distributed planning to (networks of) automata

We represent an agent $\mathcal{A}_n$ as a WA. Its state space encodes all possible values $(v_i)_{i\in I_n}$ on its variables, and its transitions define how actions modify these values. Transition costs represent how much the agent must spend for a given action. The goal of agent is defined by its subset F of final states.

The interaction of two agents is defined by sharing some actions, which formally takes the form of a product of WA. Let $\mathcal{A}_1, \mathcal{A}_2$ be two WA, $\mathcal{A}_i = (S_i, I_i, F_i, \Sigma_i, {c_I}^i, {c_F}^i, c_i)$ with T_i as associated transition sets, their product $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 = (S, I, F, \Sigma, c_I, c_F, c)$ is defined by $S = S_1 \times S_2, I = I_1 \times I_2, F = F_1 \times F_2, \Sigma = \Sigma_1 \cup \Sigma_2, c_I = {c_I}^1 \circ p_1 \otimes {c_I}^2 \circ p_2, c_F = {c_F}^1 \circ p_1 \otimes {c_F}^2 \circ p_2$ where the $p_i : S_1 \times S_2 \to S_i$ denote the canonical projections. For transition costs, one has

$$c((s_1, s_2), \sigma, (s'_1, s'_2)) = \begin{cases} c_1(s_1, \sigma, s'_1) \otimes c_2(s_2, \sigma, s'_2) \text{ if } \sigma \in \Sigma_1 \cap \Sigma_2 \\ c_1(s_1, \sigma, s'_1) \text{ if } \sigma \notin \Sigma_2 \text{ and } s_2 = s'_2 \\ c_2(s_2, \sigma, s'_2) \text{ if } \sigma \notin \Sigma_1 \text{ and } s_1 = s'_1 \\ \bar{0} \text{ otherwise} \end{cases} \quad (3)$$

The first line corresponds to synchronized actions: the two agents must agree to perform shared actions of $\Sigma_1 \cap \Sigma_2$, in which case action costs are added. By contrast, actions carrying a private label remain in the product as private actions, where only one agent changes state (next two lines).

We now model a distributed planning problem as a product $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_N$, which can be seen as a network of interacting agents. The objective is to find the/a path π from $I = I_1 \times ... \times I_n$ to the global objective $F = F_1 \times ... \times F_N$ that has minimal cost in $\mathcal{A}$. Equivalently, we look for a word u in the language of $\mathcal{A}$ that has minimal weight $\mathcal{L}(\mathcal{A}, u)$. We will actually look for an N-tuple of words $(u_1, ..., u_N)$, one word u_i per component $\mathcal{A}_i$, where each u_i corresponds to the canonical projection of u on (the action alphabet of) agent $\mathcal{A}_i$. Such local paths π_i are said to be *compatible*. The next section explains how to compute an optimal tuple of compatible local plans without computing optimal global plans.

III. Distributed optimal planning by (weighted) language calculus

A. Basic operations on weighted languages

Let us first define the product of weighted languages (WL). For a word $u \in \Sigma^*$ and $\Sigma' \subseteq \Sigma$, we denote by $u_{|\Sigma'}$ the natural projection of u on the sub-alphabet Σ'. Let $\mathcal{L}_1, \mathcal{L}_2$ be two WL defined as formal power series on Σ_1, Σ_2 respectively, their product is given by

$$(\mathcal{L}_1 \times_L \mathcal{L}_2)(u) = \mathcal{L}_1(u_{|\Sigma_1}) \otimes \mathcal{L}_2(u_{|\Sigma_2}) \quad (4)$$

Proposition 1: For $\mathcal{A} = \mathcal{A}_1 \times ... \times \mathcal{A}_N$ one has $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \times_L ... \times_L \mathcal{L}(\mathcal{A}_N)$.

Proof: The result is well known if weights are ignored [10], [14]. Regarding weights, let us consider the case of two components, without loss of generality. Let $u \in (\Sigma_1 \cup \Sigma_2)^*$ such that its projections $u_i = u_{|\Sigma_i}$ have non vanishing costs: $\mathcal{L}(\mathcal{A}_i, u_i) \neq \bar{0}$. Let π_i be an accepted path in $\mathcal{A}_i$ such that $\sigma_i(\pi_i) = u_i$. By definition of $\mathcal{A}_1 \times \mathcal{A}_2$, one can interleave π_1 and π_2 into a path $\pi \models \mathcal{A}$ such that $\sigma(\pi) = u$. Conversely, let $\pi \models \mathcal{A}$ such that $\sigma(\pi) = u$. Since transitions of $\mathcal{A}$ are pairs of transitions of $\mathcal{A}_1$ and $\mathcal{A}_2$, the canonical restriction of π to the $\mathcal{A}_i$ part yields a $\pi_i \models \mathcal{A}_i$ such that $\sigma_i(\pi_i) = u_i$. As a consequence, the $\oplus$ in (2) splits into a product of two sums, one for each component, which yields $\mathcal{L}(\mathcal{A}, u) = \mathcal{L}(\mathcal{A}_1, u_1) \otimes \mathcal{L}(\mathcal{A}_2, u_2)$. ■

The second operation we need is the projection of a WL $\mathcal{L}$ defined on alphabet Σ on a subset $\Sigma' \subseteq \Sigma$ of action labels. As for regular languages, this amounts to removing the non desired labels, but here we combine it with a cost optimization operation over the discarded labels:

$$\forall u' \in {\Sigma'}^*, \quad \Pi_{\Sigma'}(\mathcal{L})(u') = \bigoplus_{u \in \Sigma^*,\ u_{|\Sigma'} = u'} \mathcal{L}(u) \quad (5)$$

Proposition 2: Let u be an optimal word of $\mathcal{L}$, *i.e.* $\mathcal{L}(u) = \bigoplus_{v \in \Sigma^*} \mathcal{L}(v)$, then $u' = u_{|\Sigma'}$ is an optimal word of $\mathcal{L}' = \Pi_{\Sigma'}(\mathcal{L})$, *i.e.* $\mathcal{L}'(u') = \bigoplus_{v' \in {\Sigma'}^*} \mathcal{L}'(v')$. And conversely, an optimal word u' of $\mathcal{L}'$ is necessarily the projection on Σ' of an optimal word u of $\mathcal{L}$.

Proof: Direct consequence of (5). ■

Proposition 2 has an important meaning for distributed planning. Consider the set of global plans $\mathcal{L}(\mathcal{A})$ and its projections $\Pi_{\Sigma_i}(\mathcal{L}(\mathcal{A}))$ on the action sets of all components. Then an optimal local plan u_i in $\Pi_{\Sigma_i}(\mathcal{L}(\mathcal{A}))$ is necessarily the projection of an optimal global plan $u \in \mathcal{L}(\mathcal{A})$: $u_i = u_{|\Sigma_i}$. And the latter induces optimal local plans $u_j = u_{|\Sigma_j}$ in all the other projected languages $\Pi_{\Sigma_j}(\mathcal{L}(\mathcal{A}))$, $j \neq i$. Moreover, if the optimal local plan u_i is unique in every $\Pi_{\Sigma_i}(\mathcal{L}(\mathcal{A}))$, then these local plans are necessarily the projection of the same u, *i.e.* they are compatible (by definition). In summary, our objective is to compute the projections $\Pi_{\Sigma_i}(\mathcal{L}(\mathcal{A}))$ on the action alphabets of components $\mathcal{A}_i$, and then select the optimal words in these local languages. It turns out that these projected languages can be obtained *without computing* $\mathcal{L}(\mathcal{A})$, as we show below.

B. Distributed planning

Theorem 1: Let $\mathcal{L}_1, \mathcal{L}_2$ be weighted languages on Σ_1, Σ_2 respectively, and let $\Sigma_1 \cap \Sigma_2 \subseteq \Sigma'$, then

$$\Pi_{\Sigma'}(\mathcal{L}_1 \wedge \mathcal{L}_2) = \Pi_{\Sigma'}(\mathcal{L}_1) \wedge \Pi_{\Sigma'}(\mathcal{L}_2) \quad (6)$$

Proof: Again, the result is standard on languages when weights are ignored [9]. To take weights into account, assume for simplicity that $\Sigma' = \Sigma_1 \cap \Sigma_2$. The proof is then similar to the one of Prop. 1: for any two words $u_i \in \mathcal{L}_i, i = 1, 2$, such that $u_{1|\Sigma'} = u_{2|\Sigma'}$, one will have a joint word u in $\mathcal{L}_1 \wedge \mathcal{L}_2$, and *vice-versa*. It is then sufficient to notice that in (6) the sum $\oplus$ that removes the extra labels of $(\Sigma_1 \cup \Sigma_2) \backslash \Sigma'$ in the left-hand side projection can be split into a product of two independent sums, one removing labels of $\Sigma_1 \backslash \Sigma'$ in the u_1 terms, and another one removing labels of $\Sigma_2 \backslash \Sigma'$ in the u_2 terms. This gives the right-hand side of (6). ■

Theorem 1 is central to derive distributed constraint solving methods [12] (useful here to select compatibles local

plans), as well as distributed optimization methods [13] (useful here to derive the local views of optimal global plans). These approaches are actually two facets of a more general theory developed in [9]. We combine them here to design a distributed method for optimal planning. For a matter of simplicity, we illustrate the concepts on a simple example.

Consider a planning problem defined as $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3$ where $\mathcal{A}_i$ is defined on the action alphabet Σ_i and such that $\Sigma_1 \cap \Sigma_3 \subseteq \Sigma_2$. This assumption states that every interaction of $\mathcal{A}_1$ and $\mathcal{A}_3$ involves $\mathcal{A}_2$, or equivalently that $\mathcal{A}_1$ and $\mathcal{A}_3$ have conditionally independent behaviors given a behavior of $\mathcal{A}_2$. One can graphically represent this assumption by means of an interaction graph (Fig. 1). An interaction graph has components $\mathcal{A}_i$ as nodes; edges are obtained by recursively removing redundant edges, starting from the complete graph. The edge $(\mathcal{A}_i, \mathcal{A}_j)$ is declared redundant iff either $\Sigma_i \cap \Sigma_j = \emptyset$, or it is included in every Σ_k along an alternate path from $\mathcal{A}_i$ to $\mathcal{A}_j$ in the (remaining) graph.

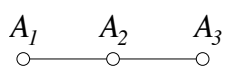

Fig. 1. *The interaction graph of* $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3$ *when* $\Sigma_1 \cap \Sigma_3 \subseteq \Sigma_2$.

Consider the derivation of $\Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A})]$. From Proposition 1, one has $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{A}_1) \times_L \mathcal{L}(\mathcal{A}_2) \times_L \mathcal{L}(\mathcal{A}_3)$. Then

$$\begin{aligned} & \Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A})] \\ = \; & \Pi_{\Sigma_1}[\, \mathcal{L}(\mathcal{A}_1) \times_L \mathcal{L}(\mathcal{A}_2) \times_L \mathcal{L}(\mathcal{A}_3)\,] \\ = \; & \mathcal{L}(\mathcal{A}_1) \times_L \Pi_{\Sigma_1}[\, \mathcal{L}(\mathcal{A}_2) \times_L \mathcal{L}(\mathcal{A}_3)\,] \\ = \; & \mathcal{L}(\mathcal{A}_1) \times_L \Pi_{\Sigma_1 \cap \Sigma_2}[\, \mathcal{L}(\mathcal{A}_2) \times_L \mathcal{L}(\mathcal{A}_3)\,] \\ = \; & \mathcal{L}(\mathcal{A}_1) \times_L \Pi_{\Sigma_1 \cap \Sigma_2}[\, \mathcal{L}(\mathcal{A}_2) \times_L \Pi_{\Sigma_2 \cap \Sigma_3}[\mathcal{L}(\mathcal{A}_3)]\,] \quad (7) \end{aligned}$$

The second equality uses Theorem 1 with $\Sigma' = \Sigma_1 \supseteq \Sigma_1 \cap (\Sigma_2 \cup \Sigma_3)$, and the fact that $\Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A}_1)] = \mathcal{L}(\mathcal{A}_1)$. For the third equality, observe that language $\mathcal{L} = \mathcal{L}(\mathcal{A}_2) \times_L \mathcal{L}(\mathcal{A}_3)$ is defined on the alphabet $\Sigma_2 \cup \Sigma_3$. So $\Pi_{\Sigma_1}(\mathcal{L}) = \Pi_{\Sigma_1}[\Pi_{\Sigma_2 \cup \Sigma_3}(\mathcal{L})] = \Pi_{\Sigma_1 \cap (\Sigma_2 \cup \Sigma_3)}(\mathcal{L})$. This is where our assumption comes into play to obtain $\Sigma_1 \cap (\Sigma_2 \cup \Sigma_3) = \Sigma_1 \cap \Sigma_2$. For the fourth equality, one replaces first $\Pi_{\Sigma_1 \cap \Sigma_2}$ by $\Pi_{\Sigma_1 \cap \Sigma_2} \circ \Pi_{\Sigma_2}$. The derivation of $\Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_2) \times_L \mathcal{L}(\mathcal{A}_3)] = \mathcal{L}(\mathcal{A}_2) \times_L \Pi_{\Sigma_2 \cap \Sigma_3}[\mathcal{L}(\mathcal{A}_3)]$ is again a direct application of Theorem 1, and reproduces the derivation of the third equality.

Equation (7) reveals that the desired projection $\Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A})]$ can be obtained by a message passing procedure, following the edges of the interaction graph. The message from $\mathcal{A}_3$ to $\mathcal{A}_2$ is $\Pi_{\Sigma_2 \cap \Sigma_3}[\mathcal{L}(\mathcal{A}_3)]$. It is combined with the knowledge of $\mathcal{A}_2$ and the result is projected on $\Sigma_1 \cap \Sigma_2$ to produce the message from $\mathcal{A}_2$ to $\mathcal{A}_1$. A symmetric message propagation rule would yield $\Pi_{\Sigma_3}[\mathcal{L}(\mathcal{A})]$, and one can also prove that

$$\begin{aligned} & \Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A})] \\ = \; & \Pi_{\Sigma_1 \cap \Sigma_2}[\mathcal{L}(\mathcal{A}_1)] \times_L \mathcal{L}(\mathcal{A}_2) \times_L \Pi_{\Sigma_2 \cap \Sigma_3}[\mathcal{L}(\mathcal{A}_3)] \quad (8) \end{aligned}$$

So the two incoming messages at $\mathcal{A}_2$ are sufficient to compute $\Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A})]$. The interest of this message passing strategy is triple: the procedure is fully distributed (no coordinator is needed), it only involves local information, and it has low complexity, in the sense that only two messages per edge are necessary (one in each direction). While the product generally increases the size of objects, one can expect the projection to reduce it, and thus save in complexity (this still has to be quantified more precisely, however).

A full theory allows one to extend this simple example to systems which interaction graph is a tree (and beyond, with more complications) [9].

C. Example

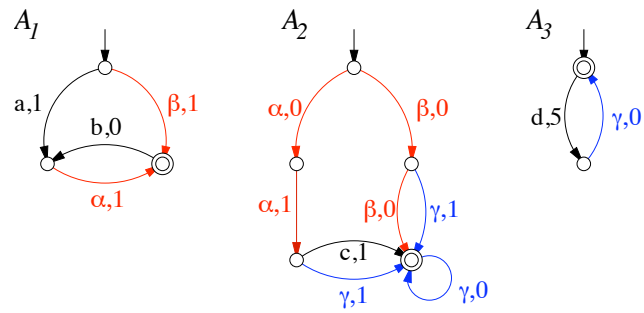

Fig. 2. *A network of 3 interacting weighted automata.*

Consider the distributed system $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3$ where the three components $\mathcal{A}_i$ are WA depicted in Fig. 2 (assuming $c_I = c_F = 0$). $\mathcal{A}_1$ and $\mathcal{A}_2$ share actions $\{\alpha, \beta\}$, and $\mathcal{A}_2, \mathcal{A}_3$ share action $\{\gamma\}$, which corresponds to the ineraction graph in Fig. 1. One has $\mathcal{L}(\mathcal{A}_1) = 1 \cdot \beta + 2 \cdot a\alpha + 2 \cdot \beta b\alpha + 3 \cdot a\alpha b\alpha + \ldots$, $\mathcal{L}(\mathcal{A}_2) = 0 \cdot \beta\beta\gamma^* + 1 \cdot \beta\gamma\gamma^* + \ldots$ and $\mathcal{L}(\mathcal{A}_3) = \sum_{n \geq 0} n \cdot (d\gamma)^n$. Observe that the minimal words in these language are β, $\beta\beta\gamma^*$ and ϵ, respecively, and that they are not compatible.

Let us follow (7) to compute $\Pi_{\Sigma_3}[\mathcal{L}(\mathcal{A})]$. The message sent by $\mathcal{A}_1$ to $\mathcal{A}_2$ is $\Pi_{\{\alpha,\beta\}}[\mathcal{L}(\mathcal{A}_1)] = 1 \cdot \beta + \sum_{n \geq 1}(1 + n) \cdot (\epsilon + \beta)\alpha^n$, which will kill all solutions with two β in $\mathcal{L}(\mathcal{A}_2)$: at most one β can be performed in $\mathcal{A}_1$. Specifically, composed with $\mathcal{L}(\mathcal{A}_2)$ this message yields $2 \cdot \beta\gamma\gamma^* + 5 \cdot \alpha\alpha(c + \gamma)\gamma^*$, which is also $\Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A}_1) \times_L \mathcal{L}(\mathcal{A}_2)]$, *i.e.* the vision from $\mathcal{A}_2$ of what $\mathcal{A}_1$ and $\mathcal{A}_2$ can perform together to reach their goals. Projected on γ, this yields $2 \cdot \gamma\gamma^* + 5 \cdot \epsilon$, the message from $\mathcal{A}_2$ to $\mathcal{A}_3$. Observe that the $5 \cdot \gamma\gamma^*$ part is discarded by the optimization step. Finally, composing this message with $\mathcal{L}(\mathcal{A}_3)$ yields the desired $\Pi_{\Sigma_3}[\mathcal{L}(\mathcal{A})] = 5 \cdot \epsilon + \sum_{n \geq 0}(7 + 5n) \cdot (d\gamma)^{n+1}$. This reveals (Proposition 2) that the best plans or words in $\mathcal{L}(\mathcal{A})$ have cost 5, and require $\mathcal{A}_3$ to do... nothing!

Following exactly (7) and (8) yields the other projections $\Pi_{\Sigma_1}[\mathcal{L}(\mathcal{A})] = 5 \cdot a\alpha b\alpha + 7 \cdot \beta$ and $\Pi_{\Sigma_2}[\mathcal{L}(\mathcal{A})] = \sum_{n \geq 0}[(5 + 5n) \cdot \alpha^2 c + (7 + 5n) \cdot \beta\gamma + (10 + 5n) \cdot \alpha^2\gamma]\gamma^n$. The minimal word in each $\Pi_i[\mathcal{L}(\mathcal{A})]$ is unique, and all of them have cost 5, which yields the triple $(a\alpha b\alpha, \alpha^2 c, \epsilon)$ as an optimal (factored) plan of cost 5. These three words are of course compatible. Notice that component $\mathcal{A}_1$ has to go twice through its local goal to help $\mathcal{A}_2$ and $\mathcal{A}_3$ reach their own goal.

IV. Implementation into weighted automata calculus

A. Recoding primitive operations

Languages of WA are generally infinite objects, so they can not be handled as such in practice. Fortunately, one starts computations with the regular languages $\mathcal{L}(\mathcal{A}_i)$, and the two primitive operations product $\times_L$ and projection $\Pi_.$ both preserve the regularity. Therefore one possibility to perform the distributed computations of section III is to replace every regular language by its finite representation as a WA. Specifically, one can choose to represent every regular language by its minimal deterministic weighted automaton (MDWA), provided it exists. The minimality is interesting to reduce the complexity of products and projections, and minimality is well defined for deterministic WA. Dealing with deterministic automata reduces as well the complexity of basic operations. But it has another important advantage for optimal planning applications: there is only one path representing a given word of the language, therefore all sub-optimal (and thus useless) paths for this word are removed in the determinization step.

Consider two minimal deterministic WA $\mathcal{A}$ and $\mathcal{A}'$. The product of their languages $\mathcal{L}(\mathcal{A})\times_L\mathcal{L}(\mathcal{A}')$ can be represented by $Min(\mathcal{A}\times\mathcal{A}')$. One already has $\mathcal{L}(\mathcal{A}\times\mathcal{A}') = \mathcal{L}(\mathcal{A})\times_L \mathcal{L}(\mathcal{A}')$ by Proposition 1, and $\mathcal{A}\times\mathcal{A}'$ is deterministic. Therefore only a minimization step (Min) is necessary, and there exist polynomial minimization algorithms for deterministic WA (not described here for a matter of space):One proceeds with a generic weight pushing procedure, followed by a standard minimization step [17].

Difficulties appear with the projection. Let $\mathcal{A}$ be a deterministic WA on alphabet Σ, its projection on $\Sigma' \subseteq \Sigma$ is obtained as for non-weighted automata, by first performing an epsilon-reduction, then determinizing the result. The epsilon-reduction collapses all transitions labeled by $\Sigma" = \Sigma\backslash\Sigma'$. Specifically, one obtains $\mathcal{A}' = (S, I', F', \Sigma', c'_I, c'_F, c')$ where the new transition function c' satisfies

$$c'(s,\sigma',s') \quad = \bigoplus_{\substack{\pi\,:\,s^-(\pi)=s,\,s^+(\pi)=s' \\ \sigma(\pi)\in\sigma'\Sigma"^*}} c(\pi) \qquad (9)$$

The initial and final cost functions are modified in a similar manner; again we refer the reader to [17] for details.

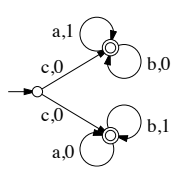

Fig. 3. *A weighted automaton $\mathcal{A}$ that can not be determinized.*

The true difficulty lies in the determinization step: *not all weighted automata can be determinized*. A counter-example is provided in Fig. 3. The point is that in a deterministic WA, a unique path (and therefore a unique and minimal weight) is associated to any accepted sequence of Σ^*. In $\mathcal{A}$, the accepted sequences are $c\{a,b\}^*$, and one either pays for the a or for the b, according to the path selected for the first c. The weight of an accepted sequence w is thus $\min(|w|_a, |w|_b)$. Intuitively, a deterministic automaton recognizing this language must count the a and the b in order to determine the weight of a word. And so it can not be finite.

A sufficient conditionfor determinizability is the so-called twin property:

Definition 1: In $\mathcal{A}$, two states $s, s' \in S$ are twins iff, either $\nexists u \in \Sigma^*$ such that $s, s' \in \delta(I,u)$, *i.e.* they can not be reached by the same label sequence from the initial states, or $\forall u \in \Sigma^*\ :\ s\in\delta(s,u),\ s'\in\delta(s',u)$, one has

$$\bigoplus_{\substack{\pi,\sigma(\pi)=u \\ s^-(\pi)=s^+(\pi)=s}} c(\pi) \quad = \bigoplus_{\substack{\pi',\sigma(\pi')=u \\ s^-(\pi')=s^+(\pi')=s'}} c(\pi')$$

$\mathcal{A}$ has the twin property iff all pairs of states are twins.

In other words, when states s, s' can be reached by the same label sequence, if it is possible to loop around s and around s' with the same label sequence u, then these loops must have identical weights.

The twin property can be tested in polynomial time [16]. It is clearly preserved by product, but unfortunately *not* by projection. See the counter-example above (Fig. 3) where one of the $(c,0)$ would be a $(d,0)$. Then $\mathcal{A}$ would have the twin property. But after projection on $\{a,b\}$ the property is obviously lost. Therefore, in order to perform computations on WA, *we have to assume that the twin property is preserved by all projections*. Otherwise there is no guarantee that the determinization procedure would terminate. Notice however that, strictly speaking, this is not an obstacle to computations since the latter can be performed with any compact representative of a given WL. In an extended version of this work, we show how to get rid of the twin property by a partial determinization.

The determinization procedure of a WA $\mathcal{A}$ elaborates on the classical subset construction for the determinization of standard automata, which may have an exponential complexity. For $u\in\Sigma^*$ and $s\in S$, let us define

$$C(u,s) = \bigoplus_{\substack{\pi\,:\,\sigma(\pi)=u, \\ s^-(\pi)\in I, s^+(\pi)=s}} c_I(s^-(\pi))\otimes c(\pi), \qquad (10)$$

and $C(u) = \bigoplus_{s\in S} C(u,s)$. So $C(u,s)$ is the minimal weight among paths that start in I, terminate in s and produce the label sequence u. States of $Det(\mathcal{A})$ take the form $q = (A,\lambda)$ where $A\subseteq S$ is a subset of states, and $\lambda : A \to \mathbb{R}^+\backslash\{\bar{0}\}$. The initial state of $Det(\mathcal{A})$ is $q_0 = (I,\lambda_0)$ with $\lambda_0(s) = C(\epsilon,s)\odot C(\epsilon) = C(\epsilon,s) - C(\epsilon)$. Given $u\in\Sigma^*$ accepted by $\mathcal{A}$, the unique state $q=(A,\lambda)$ reached by u in $Det(\mathcal{A})$ is such that: $A = \delta(I,u)$, as usual, and one has $\lambda(s) = C(u,s)\odot C(u) = C(u,s) - C(u)$. So $\lambda(s)$ is the (positive) residual over the best cost to produce u when one wants also to terminate in s. There is an obvious recursion determining the new state $q' = (A',\lambda')$ obtained by firing $\sigma\in\Sigma$ at state q. The reader is referred to [16], [17] for

the complete details of the algorithm, and for a termination proof when the twin property is satisfied.

B. Example

Let us reconsider the example in Fig. 2. Fig. 4 illustrates the propagation of messages from $\mathcal{A}_1$ to $\mathcal{A}_3$, that was described in terms of language computations in section III-C. Observe that the message from $\mathcal{A}_2$ to $\mathcal{A}_3$ (3rd automaton) now has a termination cost of 5 at the initial state. This corresponds to the path $\alpha^2 c$, that yields the empty string ϵ after projection on label γ. The rightmost automaton corresponds to $\Pi_{\Sigma_3}[\mathcal{L}(\mathcal{A})]$. Its optimal path to a terminal state is the empty string and has cost 5, the cost of an optimal global plan.

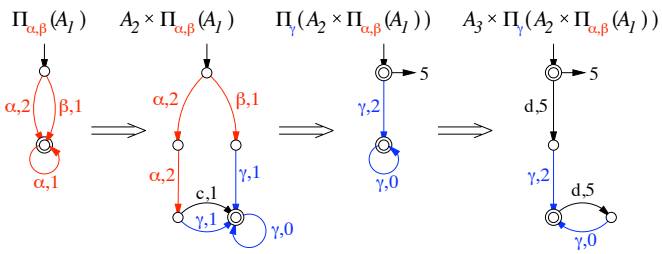

Fig. 4. *Propagation of messages from $\mathcal{A}_1$ to $\mathcal{A}_3$.*

Performing (7) and (8) in terms of WA computations completes the derivation of the MDWA representing the projected languages $\Pi_{\Sigma_i}[\mathcal{L}(\mathcal{A})]$ (Fig. 5). The optimal path in each of them appears in bold lines. These paths are unique, are associated to the same optimal cost of 5 (Proposition 2), and yield the triple $(a\alpha b\alpha, \alpha\alpha c, \epsilon)$ as best factored plan.

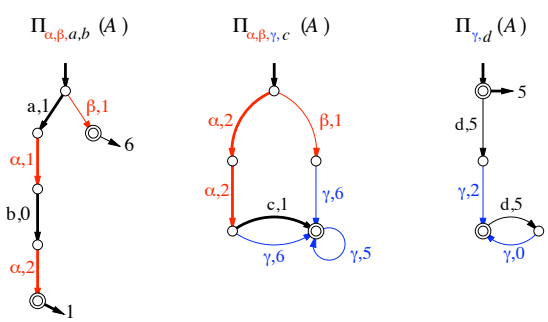

Fig. 5. *The 3 MDWA representing the projected languages $\Pi_{\Sigma_i}[\mathcal{L}(\mathcal{A})]$.*

V. Conclusion

We have described a distributed optimal planning procedure, based on a message passing strategy and on weighted automata calculus. To our knowledge, this is the first approach combining distributed planning to distributed optimization. The standpoint adopted here is unusual with respect to the planning literature, in the sense that one does not look for a single solution, but for all (optimal) solutions. This is made possible by several ingredients: working at the scale of small components makes computations tractable, looking for plans as tuples of local plans introduces a partial order semantics that implicitly reduces the trajectory space, and finally representing the trajectory space as a product of local trajectory spaces is generally more compact.

The limitations we have mentioned, namely the potential exponential complexity of determinization, and the possibility that determinization could not be possible at all, can easily be overcome. First of all because there is no necessity to perform computations with the minimal deterministic WA representing a weighted language: Any compact representative of this language can be used. Secondly, when determinization is not possible, one can perform a partial determinization (that will be described in an extended version of this work). Another controversial aspect may be that we aim at all solutions, which may be impractical in some cases. Again, classical approximations (handling subsets of the most promising plans for example) can be designed. We are currently working on these aspects, on a detailed complexity analysis and on the validation of this approach on classical benchmarks.

Acknowledgement: The authors would like to thank Philippe Darondeau for fruitful discussions about this work.

References

[1] Dana Nau, Malik Ghallab, Paolo Traverso, "Automated Planning: Theory & Practice," Morgan Kaufmann Publishers Inc.,San Francisco, CA, USA, 2004.
[2] Sarah Hickmott, Jussi Rintanen, Sylvie Thiebaux, Lang White, "Planning via Petri Net Unfolding," in Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI-07).
[3] B. Bonet, P. Haslum, S. Hickmott, S. Thiebaux, "Directed Unfolding of Petri Nets," in Proc. of the Workshop on Unfolding and Partial Ordered Techniques (UFO-07), associated to 28th Int. Conf. on Application and Theory of Petri Nets and Other Models of Concurrency, 2007.
[4] Yixin Chen, Ruoyun Huang, Weixiong Zhang, "Fast Planning by Search in Domain Transition Graphs," AAAI-08, 23rd Conference on Artificial Intelligence, 2008.
[5] Ronen I. Brafman, Carmel Domshlak, "Factored Planning: How, When, and When Not," AAAI-06, 21st Conference on Artificial Intelligence, Boston, MS, August 2006.
[6] Ronen I. Brafman, Carmel Domshlak, "From One to Many: Planning for Loosely Coupled Multi-Agent Systems," ICAPS-08, 18th International Conference on Automated Planning and Scheduling, Sydney, Australia, September 2008.
[7] Eyal Amir, B. Engelhardt, "Factored Planning," 18th Int. Joint Conference on Artificial Intelligence (IJCAI'03), 2003.
[8] Jaesik Choi, Eyal Amir, "Factored planning for controlling a robotic arm: theory," 5th Int. Cognitive Robotics workshop (CogRob 2006), 2006.
[9] E. Fabre, "Convergence of the turbo algorithm for systems defined by local constraints," INRIA research report no. PI 4860, May 2003.
[10] R. Su, W.M. Wonham, J. Kurien, X. Koutsoukos, "Distributed Diagnosis for Qualitative Systems," in Proc. 6th Int. Workshop on Discrete Event Systems, WODES02, pp. 169-174, 2002.
[11] Rong Su, "Distributed Diagnosis for Discrete-Event Systems," PhD Thesis, Dept. of Elec. and Comp. Eng., Univ. of Toronto, June 2004.
[12] Rina Dechter, "Constraint Processing," Morgan Kaufmann, 2003.
[13] Judea Pearl, "Fusion, Propagation, and Structuring in Belief Networks," Artificial Intelligence, vol. 29, pp. 241-288, 1986.
[14] C. Cassandras, S. Lafortune, "Introduction to Discrete Event Systems," Kluwer Academic Publishers, 1999.
[15] J. Berstel, "Transductions and Context-Free Languages," Electronic Edition, May 2007.
[16] Mehryar Mohri, "Finite-State Transducers in Language and Speech Processing," Computational Linguistics, 23:2, 1997.
[17] Mehryar Mohri, "Weighted automata algorithms," In Werner Kuich and Heiko Vogler, editors, Handbook of weighted automata. Springer (to appear), 2009.