A#: A distributed version of A* for factored planning

Loïg Jezequel, Eric Fabre

A#: a distributed version of A* for factored planning

Loïg Jezequel[1], Eric Fabre[2]

***Abstract*— Factored planning consists in driving a modular or distributed system to a target state, in an optimal manner, assuming all actions are controllable. Such problems take the form of path search in a product of graphs. The state space of each component is a graph, in which one must find a path to the local goal of this component. But when all components are considered jointly, the problem amounts to finding a path in each of these state graphs, while ensuring their compatibility on the actions that must be performed jointly by some components of the system. This paper proposes a solution under the form of a multi-agent version of A*, assembling several A*, each one performing a biased depth-first search in the graph of each component.**

I. Introduction

Planning consists in organizing optimally a limited set of actions in order to reach some goal. Actions generally consume and produce resources, have a cost, and the goal is expressed as a desired value for some of these resources. From a control perspective, planning can also be regarded as driving an automaton to a target state, in an optimal manner, when all transitions are controllable. Each state then represents a tuple of values, one per resource, and transitions derive from the possible actions. In these terms, the problem amounts to finding a shortest path in a possibly huge weighted oriented graph, from an initial vertex to a set of possible final ones. Efficient algorithms have been proposed, as variants of the celebrated A* [1]. The latter is a search, guided to the goal by some heuristic function, i.e. a lower-bound on the distance to the goal, available at each node. In practice, this approach performs much better than the worst case bound, that requires exploring the whole graph, provided heuristics are smartly designed [2], [3], [4].

Distributed planning addresses a similar problem, for a network of automata. This setting appears when resources are partitioned into N subsets, each one associated to the actions that modify them. Each such reduced planning problem can be represented as a smaller automaton (or component), or as a smaller graph, making it more tractable. But some actions simultaneously modify the resources of several of these N subsets, in other words some components may have to agree on some shared actions. This amounts to finding a path in a product graph. It was soon recognized that distributed or factored planning could render manageable some large scale planning problems that can not be addressed in a centralized manner.

[1]L. Jezequel is with ENS Cachan Bretagne, Rennes, France loig.jezequel@irisa.fr

[2]E. Fabre is with INRIA Rennes Bretagne Atlantique, Rennes, France eric.fabre@irisa.fr

Different approaches to distributed planning have been proposed [5], [6], [7], with the simpler objective to find a possible plan, not an optimal one. [8] proposed an approach based on a message passing strategy and handling weighted automata to perfom computations. This solution actually provides all possible (distributed) plans, and identifies the best one(s). The present paper adopts another strategy and aims at a true distributed version of A*. The idea is to run modified versions of A* in parallel, one per component, and to bias the search of each one in order to favor the exploration of local paths/plans that are likely to be compatible with the ones explored in neighbouring components. Each such local A* thus has to inform its neighbors of the shared actions that are likely to lead to a solution, from its perspective. Such communicating parallel versions of A* suggested the name A#.

The paper is organized as follows. The optimal distributed planning is first formalized (Section II). Then a simplified version of the problem is examined, in order to clarify the mechanism of the proposed approach in the simple case of two components (Section III). These principles are then generalized (Section IV).

Due to page limitations, proofs are omited in this version of the paper. The reader can find the missing material in reference [9].

II. Planning for Distributed Systems

There exist several manners to set up a planning problem. For simplicity, this paper presents planning as an optimal path search problem in a graph.

A. Planning as path search in a graph

Let $\mathcal{G} = (V, E)$ be a (finite) directed graph, with V as set of vertices and $E \subseteq V \times V$ as set of edges. A *path* in $\mathcal{G}$ is a sequence of edges $p = e_1 \dots e_n$ such that, for any $1 \leq i < n$, one has $e_i = (v_i, v_{i+1})$. p is said to be a path from $v_1 = p^-$ to $v_{n+1} = p^+$. A labelling of $\mathcal{G}$ is a function $\lambda : E \to \Lambda$, where Λ is a finite set of labels, also called *actions* in the sequel. The labelling extends to paths $p = e_1 \dots e_n$ by $\lambda(p) = \lambda(e_1) \dots \lambda(e_n) \in \Lambda^*$. This labeling is *deterministic* iff for every pair of edges (v, v') and (v, v''), $\lambda(v, v') = a = \lambda(v, v'')$ entails $v' = v''$. In other words, the effect of action a at vertex v has a unique outcome. For simplicity, and to match standard planning problems, this paper considers deterministic labelings. By abuse of notation, we sometimes do not distinguish p and $\lambda(p)$. Similarly to labels, a cost function on $\mathcal{G}$ is defined as $c : E \to \mathbb{R}_+$, and associates costs to edges. It also extends to paths by $c(p) = \sum_{i=1}^{n} c(e_i)$.

A *planning problem* is now defined as a decorated graph $\mathcal{P} = (V, E, \Lambda, \lambda, c, i, F)$ where $i \in V$ is an initial vertex, and $F \subseteq V$ is a set of possible final (or goal) vertices. The objective is to find a path p such that $p^- = i,\ p^+ \in F$ and $c(p)$ is minimal among such paths. The word $\lambda(p)$ is called the (action) plan.

B. The A* solution

A* is a search strategy in $\mathcal{P}$ [1]. Therefore it is based on a set of active or open vertices $O \subseteq V$, initialized to $O = \{i\}$, that forms the (inner) boundary of the explored region $S \subseteq V$, progressively expanded to reach F. To each vertex v in O one associates two values: $g(v)$, which is the cost of the shortest path from i to v within the subgraph $\mathcal{P}_{|S}$ ($\mathcal{P}$ restricted to S), and $h(v)$, which is a *heuristic function*, that is a lower bound on the cost to reach F from v in $\mathcal{P}$. So $h(v) = 0$ for $v \in F$. Function h is often required to be consistent, i.e. to satisfy $h(v) \leq c(v, v') + h(v')$ for any edge $(v, v') \in E$. The heuristic h is used to guide the search towards the target F. Its selection influences greatly the performance of A* [2], [3], [4].

The algorithm proceeds as follows: it recursively 'expands' the most promising vertex in O, i.e. the vertex $v \in O$ with ranking $r(v) \triangleq g(v) + h(v) \leq r(v')$ for any other $v' \in O$. Expansion means that every successor v' of v in $\mathcal{P}$ is examined. Its g value is updated according to $g^{new}(v') = min(g^{old}(v'), g(v) + c(v, v'))$ (with $g^{old}(v') = +\infty$ when $v' \notin S$, i.e. when v' has not been visited),. If $g^{new}(v') < g^{old}(v')$, then v' is (re)activated, i.e. (re)placed into O. The expanded vertex v is then removed from O, but it remains in the set of visited nodes S. The algorithm stops when the vertex v chosen for expansion already belongs to F. The best path p from i to v yields an optimal action plan.

C. Distributed planning

Let $\mathcal{P}_1, \mathcal{P}_2$ be two planning problems, with $\mathcal{P}_k = (V_k, E_k, \Lambda_k, \lambda_k, c_k, i_k, F_k)$ for $k = 1, 2$, and such that $\Lambda_1 \cap \Lambda_2 \neq \emptyset$. We define a distributed (or factored) optimal planning problem as a pair $(\mathcal{P}_1, \mathcal{P}_2)$ of such interacting planning problems. The actions in $\Lambda_1 \cap \Lambda_2$ are said to be common or synchronized actions between the two problems, while those in $\Lambda_1 \setminus \Lambda_2$ (resp. $\Lambda_2 \setminus \Lambda_1$) are private to $\mathcal{P}_1$ (resp. $\mathcal{P}_2$). This setting is extremely natural in practical planning problems: $\mathcal{P}_1$ and $\mathcal{P}_2$ can represent the state of disjoints subsets of resources, that can be modified jointly by some actions.

A distributed planning problem $(\mathcal{P}_1, \mathcal{P}_2)$ can be recast into a standard planning problem $\mathcal{P}$ by means of a product operation: $\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2$ (similar to the synchronous product of labeled automata). This operation is defined as $V = V_1 \times V_2$, $i = (i_1, i_2)$, $F = F_1 \times F_2$, $\Lambda = \Lambda_1 \cup \Lambda_2$. For the edges, one has $E = E_s \uplus E_{p,1} \uplus E_{p,2}$ where E_s denotes synchronized transitions and $E_{p,k}$ the private transitions of $\mathcal{P}_k$. They are given by $E_s = \{((v_1, v_2), (v_1', v_2')) : (v_k, v_k') \in E_k,\ \lambda_1(v_1, v_1') = \lambda_2(v_2, v_2')\}$, while private moves of $\mathcal{P}_1$ assume $\mathcal{P}_2$ remains idle $E_{p,1} = \{(v_1, v_2), (v_1', v_2)) : (v_1, v_1') \in E_1,\ v_2 \in V_2,\ \lambda_1(v_1, v_1') \notin \Lambda_2\}$ and symmetrically for $E_{p,2}$. Labels follow accordingly: $\lambda((v_1, v_2), (v_1', v_2')) = \lambda_1(v_1, v_1') = \lambda_2(v_2, v_2')$ for E_s, and $\lambda((v_1, v_2), (v_1', v_2)) = \lambda_1(v_1, v_1')$ for $E_{p,1}$ (sym. for $E_{p,2}$). In the same way, costs are additive: $c((v_1, v_2), (v_1', v_2')) = c_1(v_1, v_1') + c_2(v_2, v_2')$ for E_s, and $c((v_1, v_2), (v_1', v_2)) = c_1(v_1, v_1')$ for $E_{p,1}$ (sym. for $E_{p,2}$).

The resulting product problem $\mathcal{P} = \mathcal{P}_1 \times \mathcal{P}_2$ may however be very large compared to the $\mathcal{P}_k$ alone (see Figure 1). This is the usual state explosion problem, which is doubled by the emergence of so-called concurrency diamonds on edges, i.e. the interleaving of private actions of $\mathcal{P}_1$ and $\mathcal{P}_2$.

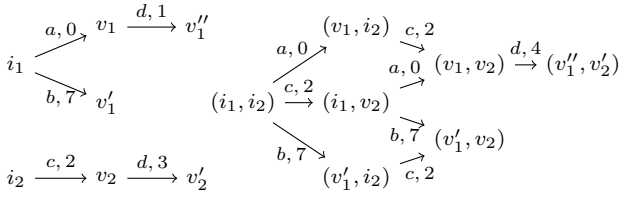

Fig. 1. Two planning problems (left) and their product (right).

Distributed planning aims at avoiding this double explosion of problem $\mathcal{P}$. The idea is to look for a *pair* of paths (p_1, p_2), such that

1) p_k defines a valid plan in $\mathcal{P}_k$, not necessarily optimal, for $k = 1, 2$
2) the two paths p_1 and p_2 are *compatible*, i.e. they coincinde on shared actions, which we translate by $\pi_{\Lambda_2}(\lambda_1(p_1)) = \pi_{\Lambda_1}(\lambda_2(p_2))$ (see below),
3) the pair (p_1, p_2) is jointly optimal, i.e. $c(p_1) + c_2(p_2)$ is minimal.

The natural projection $\pi_{\Lambda'}(w)$ of a word $w \in \Lambda^*$ on the subalphabet $\Lambda' \subseteq \Lambda$ is defined by $\pi_{\Lambda'}(\epsilon) = \epsilon$ for the empty word, and by $\pi_{\Lambda'}(aw) = a\pi_{\Lambda'}(w)$ if $a \in \Lambda'$, and $\pi_{\Lambda'}(aw) = \pi_{\Lambda'}(w)$ otherwise. The compatibility of p_1 and p_2 thus induces that there exists a global path p for problem $\mathcal{P}$ such that $\pi_{\Lambda_k}(\lambda(p)) = \lambda_k(p_k)$, $k = 1, 2$. Such a path p corresponds to an interleaving of the actions in p_1 and p_2. It is generally not unique: for example in Fig. 1 the distributed plan (ad, cd) corresponds to two global plans acd and cad This reveals that distributed planning actually aims at building *partially ordered plans*, which is a way to avoid the explosion due to concurrency: one saves the exploration of meaningless interleavings.

D. Coordinated parallel path searches

The approach proposed in this paper consists in associating an agent φ_k to each problem $\mathcal{P}_k$. Each agent performs an A*-like search in its local graph, and takes into account the constraints and costs of the other agent through an appropriate communication mechanism. Communications are asynchronous and can take place at any time. Nevertheless, it is proved that the algorithm converges to a distributed optimal plan (p_1, p_2).

For a matter of clarity, the next section first addresses a simpler problem called *compatible final states (CFS)*. $\mathcal{P}_1$ and

$\mathcal{P}_2$ have no common actions, so $\Lambda_1 \cap \Lambda_2 = \emptyset$. However, their final states are 'colored' by functions $\gamma_k : F_k \to \Gamma$ where Γ is a finite color set. The CFS problem amounts to finding an optimal distributed plan (p_1, p_2) where the compatibility condition (2) above is replaced by a compatibility of their final states: $\gamma_1(p_1^+) = \gamma_2(p_2^+)$. We shall assume that there is a unique optimal (common) final color, if ever the CFS problem has a solution. Indeed, selecting one optimal final color among several is an independent agreement problem.

The standard distributed planning problem can be reduced to the CFS as follows. First $\Lambda_1 \cap \Lambda_2 \neq \emptyset$ is ignored. Then, instead of assuming that the colors of final states are given beforehand, one *computes* them as functions of the current explored paths to reach any vertex of graph $\mathcal{P}_k$. Specifically, a path p_k in $\mathcal{P}_k$ with $p_k^+ \in F_k$ will have $\pi_{\Lambda_1 \cap \Lambda_2}(\lambda_k(p_k))$ as final 'color.' Compatibility of final colors thus entails the compatibility of p_1, p_2 in terms of shared actions. The main difference with CFS is thus that the color set becomes infinite: $\Gamma = (\Lambda_1 \cap \Lambda_2)^*$.

III. Compatible final states

A. Intuition on the approach

Let $\{k, \bar{k}\} = \{1, 2\}$. The agent φ_k attached to problem $\mathcal{P}_k$ relies on four functions. Two relate to the standard shape of a local A*: $g_k : V_k \to \mathbb{R}$ yields the (current) best known cost to reach any $v \in V_k$ and $h_k : V_k \times \Gamma \to \mathbb{R}$ is a set of heuristic functions towards F_k, one per terminal color. Equivalently, one has a heuristic function towards any $F_k \cap \gamma^{-1}(c)$ for $c \in \Gamma$. Besides, two other functions inform φ_k on the state of the search in $\mathcal{P}_{\bar{k}}$, where $\{k, \bar{k}\} = \{1, 2\}$. Namely, one has $H_{\bar{k}} : \Gamma \to \mathbb{R}$, and $G_{\bar{k}} : \Gamma \to \mathbb{R}$. $H_{\bar{k}}$ is a (generally) time varying heuristic that measures how much color $c \in \Gamma$ is promising at the current point of resolution of problem $\mathcal{P}_{\bar{k}}$. Similarly, $G_{\bar{k}}(c)$ eventually gives the best cost for reaching color c in $\mathcal{P}_{\bar{k}}$.

We now formalize these features and explain how these four functions are updated by each agent, how termination is detected, and how an optimal distributed plan is extracted.

B. Proposed algorithm

Let us consider first a non-varying distant heuristic $H_{\bar{k}}$: $H_{\bar{k}}(c) = h_{\bar{k}}(i_{\bar{k}}, c)$ for all $c \in \Gamma$. To the distant cost function on final colors $G_{\bar{k}}$ one associates an oracle $\Theta_{\bar{k}} : \Gamma \to \{null, optimal, useless\}$, with the following meaning. $\Theta_{\bar{k}}(c) = optimal$ means that a best plan towards final vertices of color c is known in $\mathcal{P}_{\bar{k}}$, and in that case $G_{\bar{k}}(c)$ represents the optimal cost to reach color c in $\mathcal{P}_{\bar{k}}$. $\Theta_{\bar{k}}(c) = useless$ means that $\varphi_{\bar{k}}$ can guarantee that for sure no optimal distributed plan $(p_k, p_{\bar{k}})$ exists which terminates in color c, and $null$ is the remaining default (and initial) value of $\Theta_{\bar{k}}$. This oracle satisfies the following property: for every color $c \in \Gamma$, there exists a finite time at which $\Theta_{\bar{k}}(c)$ jumps from $null$ to either $optimal$ or $useless$, and keeps this value forever.

Each agent φ_k executes Algorithm 1. Vertices can be marked as open, closed, or candidate. A candidate vertex v belongs to F_k, and thus represents a local plan in $\mathcal{P}_k$ that can be proposed to $\varphi_{\bar{k}}$ as a possible local component of a distributed plan. Initially all vertices v in $V_k \setminus \{i_k\}$ are closed and satisfy $g_k(v) = +\infty$. To progressively open them and explore graph $\mathcal{P}_k$, one relies on the ranking function R_k defined as follows. If $v \in V_k$ is not candidate

$$R_k(v) \quad = \quad g_k(v) + \min_{c \in \Gamma} \left(h_k(v, c) + H_{\bar{k}}(c) \right)$$

which integrates the cost of color c for agent $\varphi_{\bar{k}}$, and then optimizes on the possible final color. For a candidate vertex v, one takes

$$\begin{aligned} R_k(v) \quad &= \quad g_k(v) + G_{\bar{k}}(\gamma_k(v)) \text{ if } \Theta_{\bar{k}}(\gamma_k(v)) = optimal \\ &= \quad g_k(v) + H_{\bar{k}}(\gamma_k(v)) \text{ otherwise} \end{aligned}$$

which associates to the possible final vertex v the cost of its color $\gamma_k(v)$ for agent $\varphi_{\bar{k}}$.

The recursive (local) search then proceeds as follows. At each iteration φ_k selects the most promising non-closed (i.e. open or candidate) vertex v, i.e. the one that minimizes the ranking function R_k. According to the nature of v, agent φ_k either a) progresses in the exploration of $\mathcal{P}_k$ using an expansion function (Algorithm 2), this is the case in particular when v is open, or b) checks whether it can draw some conclusion using the information provided by the other agent $\varphi_{\bar{k}}$. These conclusions can be (1) that v is the goal vertex reached by a path part of a globally optimal plan (line 8), (2) that v will never be the goal vertex reached by a path part of a globally optimal plan (line 13), or (3) nothing for the moment (line 15). The reader familiar with A* may thus immediately identify its shape within Algorithm 1. The main difference lies in the stopping condition, due to the necessity to take into account constraints transmitted by the other agent.

Algorithm 1 executed by φ_k

```
1:  mark i_k open; g_k(i_k) ← 0; calculate R_k(i_k)
2:  while there exists non-closed vertices do
3:     let v be the non-closed vertex with minimal R_k(v)
4:     if v is open then
5:        expand(v)
6:     else
7:        case: Θ_k̄(γ_k(v)) = optimal
8:           if R_k(v) = g_k(v) + G_k̄(γ_k(v)) then
9:              return v and terminate
10:          else
11:             calculate R_k(v)
12:          end if
13:       case: Θ_k̄(γ_k(v)) = useless
14:          mark v closed
15:       case: Θ_k̄(γ_k(v)) = null
16:          if there exists open vertices then
17:             let v' be the open vertex with minimal R_k(v')
18:             expand(v')
19:          end if
20:    end if
21: end while
```

Algorithm 2 expand function

```
1: if v ∈ F_k then
2:    mark v candidate
3:    calculate R_k(v)
4: else
5:    mark v closed
6: end if
7: for all v' such that (v, v') ∈ E_k do
8:    g_k(v') ← min(g_k(v'), g_k(v) + c_k((v, v')))
9:    if g_k(v') strictly decreased then
10:      mark v' open
11:      pred(v') ← v
12:   end if
13:   calculate R_k(v')
14: end for
```

Notice that the call to the $expand$ function at line 18 of Algorithm 1 is not required for termination nor validity, however it will allow agent $\varphi_{\bar{k}}$ to maintain $G_{\bar{k}}$ and $\Theta_{\bar{k}}$ using its own instance of Algorithm 1. Otherwise $\varphi_{\bar{k}}$ should run a standard A^* algorithm in parallel with Algorithm 1.

Theorem 1: In this context, any execution of Algorithm 1 by φ_k on $\mathcal{P}_k$ terminates. Moreover, if the CFS problem $(\mathcal{P}_1, \mathcal{P}_2)$ has a solution, the output of Algorithm 1 for agent φ_k is a goal vertex $v_k \in F_k$, reached by a local plan p_k. The assembling of p_1 and p_2 provided by agents φ_1 and φ_2 resp. yields an optimal distributed plan (p_1, p_2) solving $(\mathcal{P}_1, \mathcal{P}_2)$.

C. Implementation of $G_{\bar{k}}$

The remaining of this section gives a feasible construction of the distant (color) cost function $G_{\bar{k}}$ and of the oracle $\Theta_{\bar{k}}$, showing that Algorithm 1 is usable in practice. These two functions have to be computed by agent $\varphi_{\bar{k}}$ independently of problem $\mathcal{P}_k$, and in particular, independently of G_k and Θ_k. The $expand$ function is considered atomic: no update of $\Theta_{\bar{k}}$ or $G_{\bar{k}}$ will occur during the execution of this function by φ_k.

A possible implementation follows, where $\Theta_{\bar{k}}$ and $G_{\bar{k}}$ are computed within Algorithm 1 by $\varphi_{\bar{k}}$:

- initialization: $\forall c \in \Gamma$, $G_{\bar{k}}(c) = +\infty$, and if $F_{\bar{k}} \cap \gamma_{\bar{k}}^{-1}(c) = \emptyset$ then $\Theta_{\bar{k}}(c) = useless$ otherwise $\Theta_{\bar{k}}(c) = null$,
- update: as soon as some final vertex $v \in F_{\bar{k}}$ is open or candidate, if no other open vertex $v' \in V_{\bar{k}}$ satisfies $g_{\bar{k}}(v') + h_{\bar{k}}(v', \gamma_{\bar{k}}(v)) < g_{\bar{k}}(v)$, then color $\gamma_{\bar{k}}(v)$ can not be reached with a lower cost in $\mathcal{P}_{\bar{k}}$, so $\Theta_{\bar{k}}(\gamma_{\bar{k}}(v))$ is set to $optimal$ and
$$G_{\bar{k}}(\gamma_{\bar{k}}(v)) = \min_{v' \in F_{\bar{k}}, \gamma_{\bar{k}}(v') = \gamma_{\bar{k}}(v)} g_{\bar{k}}(v')$$
- final update: when Algorithm 1 stops, for all $c \in \Gamma$ such that $\Theta_{\bar{k}}(c) = null$, set $\Theta_{\bar{k}}(c) = useless$, and $G_{\bar{k}}(c) = +\infty$.

D. Running example

Consider the graph of Figure 2. Heuristics h_1 should have the following properties: $h_1(i_1, r) \leq 1$, $h_1(v_1, r) \leq 0$, and $h_1(v, b) \leq +\infty$ for any v. In the same way the values of H_2 (provided to φ_1 by φ_2) should always be such that: $H_2(r) \leq 2$, and $H_2(b) \leq 2 + 0 = 2$.

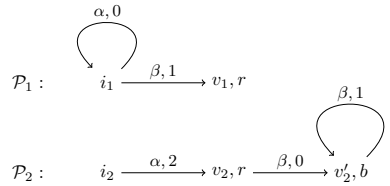

Fig. 2. A CFS problem. Goal vertices are represented with their color (ex. v_1 is a goal with color r). Costs and labels are written above edges.

Assume φ_1 is running Algorithm 1 on $\mathcal{P}_1$. Initially, i_1 is open, $g_1(i_1) = 0$, and $R_1(i_1) = g_1(i_1) + \min_{c \in \{r,b\}}(h_1(i_1, c) + H_2(c))$. All other vertices are closed and such that g_1 is infinite. Moreover, $\Theta_1(b) = useless$, as no goal vertex with color b exists in $\mathcal{P}_1$. The first execution of the while loop will directly call the expand function, as i_1 is not candidate. It will be marked as closed (as i_1 is not a goal vertex). As $g_1(i_1) = 0 \leq 0 = g_1(i_1) + 0$, i_1 will not be re-opened. As $g_1(v_1) = +\infty > 1 = g_1(i_1) + 1$, v_1 will be opened, with $g_1(v_1) = 1$, and $R_1(v_1) = g_1(v_1) + \min_{c \in \{r,b\}}(h_1(v_1, c) + H_2(c))$, and $pred(v_1) = i_1$. After that, the expand function terminates. Immediately, $\Theta_1(r) = optimal$ as v_1 is a goal state with color r and no other open vertex exists, $G_1(r) = g_1(v_1) = 1$. As there is open vertices, a second execution of the while loop starts. The open or candidate vertex with minimal value of R_k is v_1. As v_1 is not candidate, a call to expand occurs immediately. As $v_1 \in F_1$, it is now candidate, and $R_1(v_1) = g_1(v_1) + G_2(r)$ if $\Theta_2(r) = optimal$ or $R_1(v_1) = g_1(v_1) + H_2(r)$ else. As v_1 has no neighbors, no new vertices are opened. From that, a new execution of the while loop occurs. As v_1 is candidate it is checked if it allows to conclude. No more calls to expand function occur as there no longer exists open vertices. As soon as $\Theta_2(r) = optimal$, with $G_2(r) = 2$ it is possible to conclude. The only possible local solution is to go from i_1 to v_1 in one step. Its cost is 1 locally, but $1 + 2 = 3$ globally, as the part of the solution in $\mathcal{P}_2$ is to go from i_2 to v_2 in one step.

IV. Distributed planning with two components

This section extends the algorithm proposed to solve CFS problems to the more general framework of distributed planning (DP) problems, still in the limited case of two components. Compared to CFS, DP problems introduce two difficulties. First, colors are assigned dynamically to vertices: the color of vertex v is not given in advance by some coloring function γ, but is set as a function of the path p leading to this vertex $v = p^+$. Secondly, rather than a finite set Γ of colors, one potentially has an infinite set, since the idea is the 'color' of vertex $v = p^+$ is the sequence of shared actions met along path p leading to v. And there is generally no bound (efficiently computable in a distributed way) on the

number of shared actions in a globally optimal distributed plan

Let us recast a DP problem $(\mathcal{P}_1, \mathcal{P}_2)$ as a CFS problem $(\mathcal{P}'_1, \mathcal{P}'_2)$ with color set $\Gamma = (\Lambda_1 \cap \Lambda_2)^*$, the set of sequences of shared actions. One has $\mathcal{P}'_k = (V'_k, E'_k, \Lambda'_k, \lambda'_k, c'_k, i'_k, F'_k)$ with $V'_k = V_k \times \Gamma$, $E'_k = \{((v,w),(v',w')) : (v,v') \in E_k \wedge w' = w\,\pi_{\Lambda_1\cap\Lambda_2}(\gamma_k((v,v')))\}$, $i'_k = (i_k, \varepsilon)$, $F'_k = F_k \times \Gamma$, $\gamma'_k : F'_k \to \Gamma$ is such that $\gamma'_k((v,w)) = w$, and c'_k is such that $c'_k(((v,w),(v',w'))) = c_k((v,v'))$. $(\mathcal{P}'_1, \mathcal{P}'_2)$ has however a major difference with CFS problems considered in Section III: V'_1, E'_1, V'_2, and E'_2 may be infinite.

The remaining of this section is dedicated to extending the results of Section III to the case of the particular infinite graphs considered here. It will allow to use Algorithm 1 along with *expand* function given in Algorithm 3 for solving DP problems. This new *expand* function is in fact responsible for computing parts of P'_k from P_k (only when needed). Three points have to be addressed: (1) computation of $R_k((v,w))$ sometimes implies to take a minimum over an infinite number of elements, (2) termination of the algorithm relies on finiteness of the graphs, (3) computation of $G_{\bar{k}}$ and $\Theta_{\bar{k}}$ are not directly possible on infinite graphs as non-accessible colors can not be determined at initialization.

Algorithm 3 expand function

{called with an argument v of the form (v', w)}
if $v' \in F_k$ **then**
  mark $v = (v', w)$ candidate
  calculate $R_k(v)$
else
  mark $v = (v', w)$ closed
end if
for all v'' such that $(v', v'') \in E_k$ **do**
  $w' \leftarrow w\pi_\Gamma(\gamma_k((v', v'')))$
  $g_k((v'', w')) \leftarrow \min(g_k((v'', w')), g_k((v', w)) + c_k((v', v'')))$
  if $g_k((v'', w'))$ strictly decreased **then**
    mark (v'', w') open
    $pred((v'', w')) \leftarrow (v', w)$
  end if
  calculate $R_k((v'', w'))$
end for

A. Computation of R_k and $H_{\bar{k}}$

For any color w (which may correspond to an optimal distributed plan), $H_{\bar{k}}(w)$ should give a lower bound on the cost of reaching this color in $\mathcal{P}_{\bar{k}}$. Clearly, taking $H_{\bar{k}}(w) = 0$ for any w gives such a lower bound. However, it is usually better to get a tight bound in order to avoid as much exploration of the graphs as possible. For practical use of our algorithm, using a more accurate $H_{\bar{k}}$ would be recommended. An example of such an $H_{\bar{k}}$ is the following, where $w' < w$ is notation for w' is a prefix of w (recall that $H_{\bar{k}}$ is computed by $\varphi_{\bar{k}}$):

$$H_{\bar{k}}(w) = \min(H^o_{\bar{k}}(w), H^c_{\bar{k}}(w))$$

with:

$$H^o_{\bar{k}}(w) = \min_{\substack{(v_{\bar{k}}, w') \text{ open} \\ w' < w}} (g_{\bar{k}}((v_{\bar{k}}, w')), h_{\bar{k}}((v_{\bar{k}}, w'))),$$

$$H^c_{\bar{k}}(w) = \min_{\substack{(v_{\bar{k}}, w') \text{ candidate} \\ w' < w}} (g_{\bar{k}}((v_{\bar{k}}, w'))).$$

Notice that for any w it is possible to compute $H_{\bar{k}}(w)$, as the set of open and candidate (v, w) is always finite. Notice also that all $H_{\bar{k}}(w)$ can be computed by φ_k from a finite number of them given by $\varphi_{\bar{k}}$: the one which are such that $(v_{\bar{k}}, w)$ is open or candidate. We denote them by $\hat{H}_{\bar{k}}$. One then has: $H_{\bar{k}}(w) = \min_{w'<w} \hat{H}_{\bar{k}}(w')$.

When (v, w) is candidate, the computation of R_k is not an issue, it can be done exactly as in the simpler cases. Notice that, when (v, w) is candidate, v is necessarily in F_k. Hence, when $\Theta_{\bar{k}}(w) = optimal$ one has:

$$R_k((v,w)) = g_k((v,w)) + G_{\bar{k}}(w),$$

and in other cases:

$$R_k((v,w)) = g_k((v,w)) + H_{\bar{k}}(w).$$

However, when (v, w) is open it is not possible to directly use the previous definition of $R_k((v,w))$ as it may involve the computation of a minimum over an infinite number of elements. First of all, computing R_k as before would require the computation of $h_k((v,w), w')$ for any color w'. We consider instead $h_k((v,w)) = \min_{w'} h_k((v,w), w')$, which is computable as a lower bound on the cost of a path in $\mathcal{P}_k$ from v to a goal vertex.

From that, when (v, w) is open, we suggest to compute $R_k((v,w))$ as follows:

$$R_k((v,w)) = g_k((v,w)) + h_k((v,w)) + \min_{w'>w} H_{\bar{k}}(w').$$

The second difficulty is that there may be an infinite number of colors w to consider when computing $\min_{w'} H_{\bar{k}}(w') = \min_{w'>w} H_{\bar{k}}(w')$. This suggest to add a constraint on $H_{\bar{k}}$: it should be such that $\min_{w'>w} H_{\bar{k}}(w')$ is computable for any w. Fortunately, using the implementation of $H_{\bar{k}}$ proposed above it is possible. One just has to remark that:

$$\min_{w'>w} H_{\bar{k}}(w') = \min(H_{\bar{k}}(w), \min_{w'>w} \hat{H}_{\bar{k}}(w')),$$

as the number of w such that $\hat{H}_{\bar{k}}(w)$ is defined is always finite, this minimum can be computed.

B. Termination of the algorithm

The main difference here with the case of CFS problems is that the termination of the algorithm is not ensured when there is no solution. This is due to the fact that the graph to explore is in general infinite. In fact, it is possible to ensure termination, as there is a bound on the length of the color corresponding to a possible solution. This bound can be computed by considering the number of vertices in the product of $\mathcal{P}_1$ and $\mathcal{P}_2$: if a solution exists, one is such

that it passes at most one time in each vertex of this graph. However, it is not straightforward to tightly and modularly compute this bound. We focus on the case with a solution.

Theorem 2: In this context, if the considered planning problem $(\mathcal{P}_1, \mathcal{P}_2)$ has a solution, then: any execution of Algorithm 1 by φ_k on $\mathcal{P}_k$ terminates. Moreover, the output of Algorithm 1 for agent φ_k is a goal vertex $v_k \in F_k$, reached by a local plan p_k. The assembling of p_1 and p_2 provided by agents φ_1 and φ_2 resp. yields an optimal distributed plan (p_1, p_2) solving $(\mathcal{P}_1, \mathcal{P}_2)$.

C. Computation of $G_{\bar{k}}$ and $\Theta_{\bar{k}}$

As before, these two functions have to be computed by agent $\varphi_{\bar{k}}$ independently of $\mathcal{P}_k$, and in particular, independently of G_k and Θ_k. A possible implementation, where $\Theta_{\bar{k}}$ and $G_{\bar{k}}$ are computed along execution of Algorithm 1 by $\varphi_{\bar{k}}$, is the following:

- initialization: $\forall w \in \Gamma$, $\Theta_{\bar{k}}(w)$ is considered as $null$ and $G_{\bar{k}}(w) = +\infty$ (but only the $\Theta_{\bar{k}}(w) \neq null$ and the corresponding values of $G_{\bar{k}}$ are stored).
- update (1): as soon as there exists $v \in F_{\bar{k}}$ such that (v, w) is open or candidate, and there is no open couple (v', w') such that $g_{\bar{k}}((v', w')) + h_{\bar{k}}((v', w')) < g_{\bar{k}}((v, w))$ and $w' < w$, $\Theta_{\bar{k}}(\gamma_{\bar{k}}(v)) = optimal$ and

$$G_{\bar{k}}(w) = \min_{v' \in F_{\bar{k}}} g_{\bar{k}}((v', w)).$$

- update (2): as soon as for a given w there exists no $w' < w$ and v such that (v, w') is open or (v, w) is candidate, if $\Theta_{\bar{k}}(w) = null$, then $\Theta_{\bar{k}}(w)$ is set to $useless$.
- final update: when Algorithm 1 stops, for all $w \in \Gamma$ such that $\Theta_{\bar{k}}(w) = null$, set $\Theta_{\bar{k}}(w) = useless$, and $G_{\bar{k}}(w) = +\infty$.

D. Running example

Consider the graphs of Figure 2 as a DP problem (for that ignore final colors and focus on the labels on the edges).

An execution of Algorithm 1 by φ_1 on $\mathcal{P}_1$ starts with (i_1, ε) open. Then a call to expand function closes (i_1, ε) and opens (i_1, α) and (v_1, β). After that depending on the values of the different heuristics, a call to expand function will occur on either (i_1, α) or (v_1, β). Assume it is called on (i_1, α). Then (i_1, α) is closed and $(i_1, \alpha\alpha)$ and $(v_1, \alpha\beta)$ are opened. After that expand will be called on either (v_1, β), $(i_1, \alpha\alpha)$ or $(v_1, \alpha\beta)$. Which will either mark (v_1, β) or $(v_1, \alpha\beta)$ candidate, or close $(i_1, \alpha\alpha)$ and open $(i_1, \alpha\alpha\alpha)$ and $(v_1, \alpha\alpha\beta)$. After each time an element $(v_1, w\beta)$ is opened with $w \in \{\alpha\}^*$, $\Theta_1(w\beta) = optimal$ and $G_1(w\beta) = |w|.0+1$. As all costs of edges are positive, any open element of the form $(v_1, w\beta)$ with $w \in \{\alpha\}^*$ becomes candidate after a finite time. After some time $\Theta_2(\beta) = useless$ (it is not possible to reach a goal state in G_2 using only one edge with color β), and $\Theta_2(\alpha\beta) = optimal$ with $G_2(\alpha\beta) < \min(H_2(w\beta), G_2(w\beta))$ for all $w \in \{\alpha\}^*$ such that $\Theta_2(w\beta) \neq optimal$ and $G_2(\alpha\beta) < G_2(w\beta)$ for all $w \in \{\alpha\}^*$ such that $\Theta_2(w\beta) = optimal$. It allows φ_1 to conclude that its part of the optimal solution (which has a global cost of 3) reaches v_1 with color $\alpha\beta$. Moreover, the values of $pred$ allow to conclude that the path in $\mathcal{P}_1$ should be to loop on i_1 one time and then go to v_1.

V. Conclusion

A* is a celebrated search algorithm to find a shortest path in oriented graphs. Its variants are extremely used to solve optimal planning problems, which are weak versions of optimal control problems. This paper has presented A#, a multi-agent version of A*, dedicated to quickly find an optimal strategy to drive a distributed system to a target state. Its convergence and its consistency have been proved. Compared to other approaches to distributed planning, this is the first distributed algorithm that provides a *globally optimal* plan. Moreover, it is not hierarchical: all components run the same algorithm simultaneously, and bias each other's searches by their own guesses. It is more a 'consensus' approach than a master-slave approach.

The practical interest of A# will be soon tested. In terms of theoretical complexity, however, the worst case bounds of planning problems remain unchanged, and one may have to explore the whole global graph. However, compared to running A* on the equivalent product problem, one may be faster when components are loosely coupled. This is the standard key advantage of distributed planning, which explores possible plans as partial orders of actions rather than sequences, and thus saves the exploration of different interleavings of concurrent actions.

While the paper is limited to two components, A# extends to graphs of interacting components in a usual manner (these graphs are obtained by placing an edge between components that share actions). In particular, A# may become very efficient when the interaction graph between components is a tree.

References

[1] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[2] Stefan Edelkamp. Planning with pattern databases. In *Proceedings of the 12th International Conference on Automated Planning and Scheduling*, pages 13–24, 2001.

[3] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Fexible abstraction heuristics for optimal sequential planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling*, pages 176–183, 2007.

[4] Erez Karpas and Carmel Domshlak. Cost-optimal planning with landmarks. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1728–1733, 2009.

[5] Ronen Brafman and Carmel Domshlak. Factored planning: How, when, and when not. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence*, pages 809–814, 2006.

[6] Ronen Brafman and Carmel Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling*, pages 28–35, 2008.

[7] Eyal Amir and Barbara Engelhardt. Factored planning. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 929–935, 2003.

[8] Eric Fabre and Loïg Jezequel. Distributed optimal planning: an approach by weighted automata calculus. In *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 211–216, 2009.

[9] Loïg Jezequel and Eric Fabre. A-sharp: A distributed a-star for factored planning. Technical Report RR-7927, INRIA, 2012.