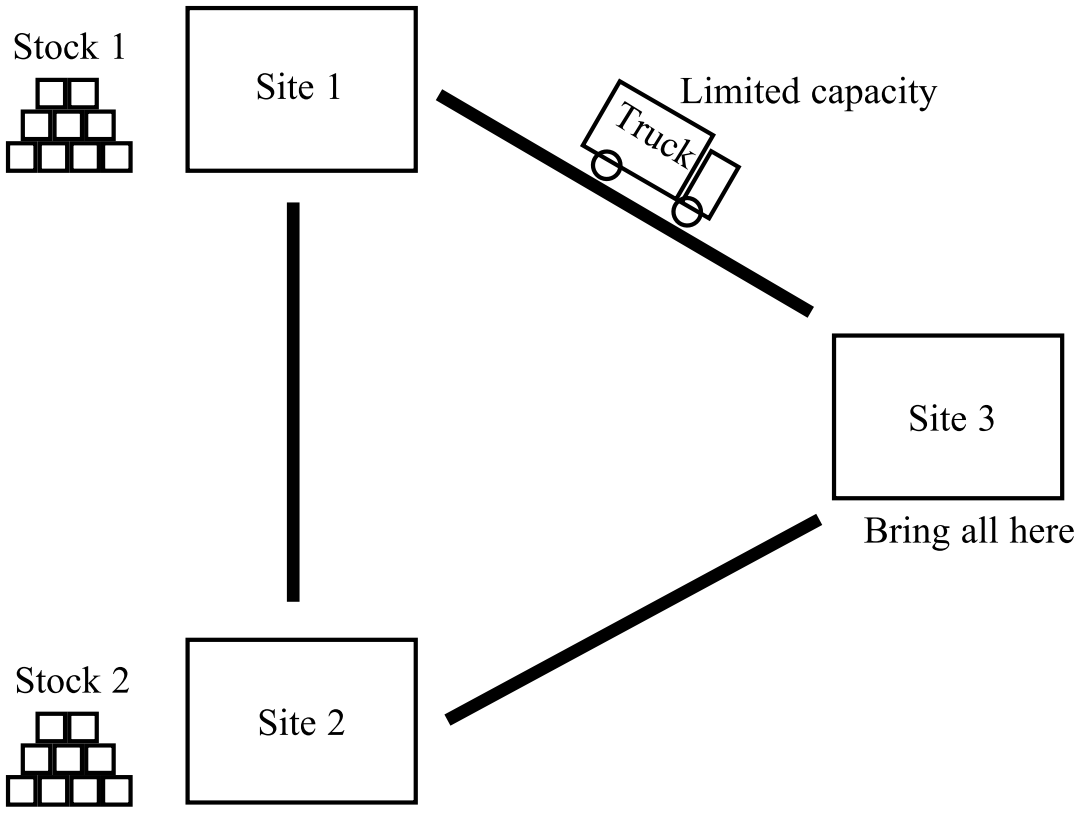
Distributed Cost-Optimal Planning

Loïg Jezequel

ENS Cachan Bretagne

November 13, 2012

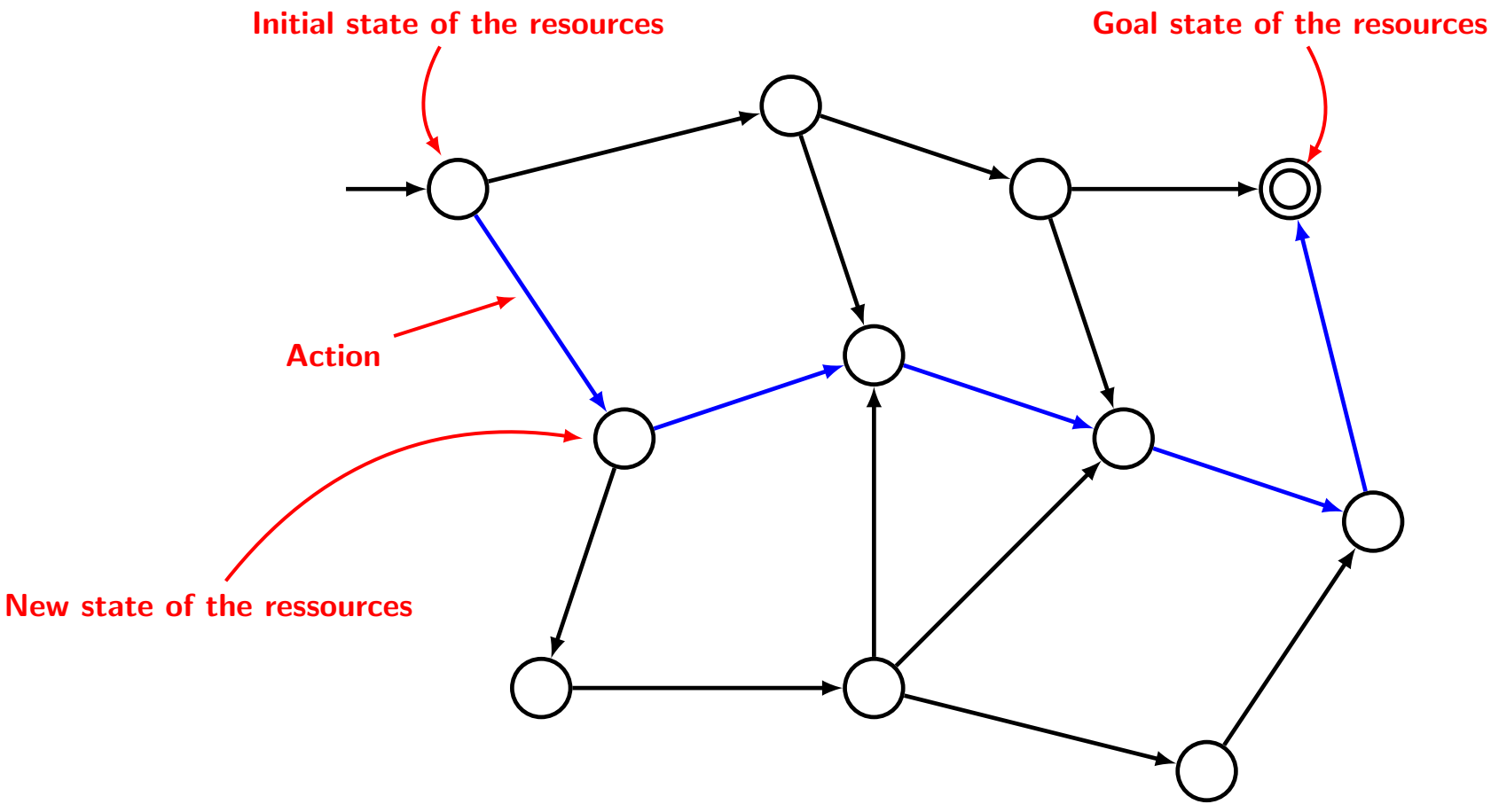
Planning problems: overview



Goal

Find a *plan*: a sequence of actions (with minimal cost) moving the system from its initial state to one of its goal states

Planning problems: search in graphs



Planning problems: resolution

Heuristic search

- A*-like algorithms: Hart et al. *1968*
- Various heuristics: Bonet and Geffner 2001, Helmert et al. 2007, Karpas and Domshlak 2009, ...

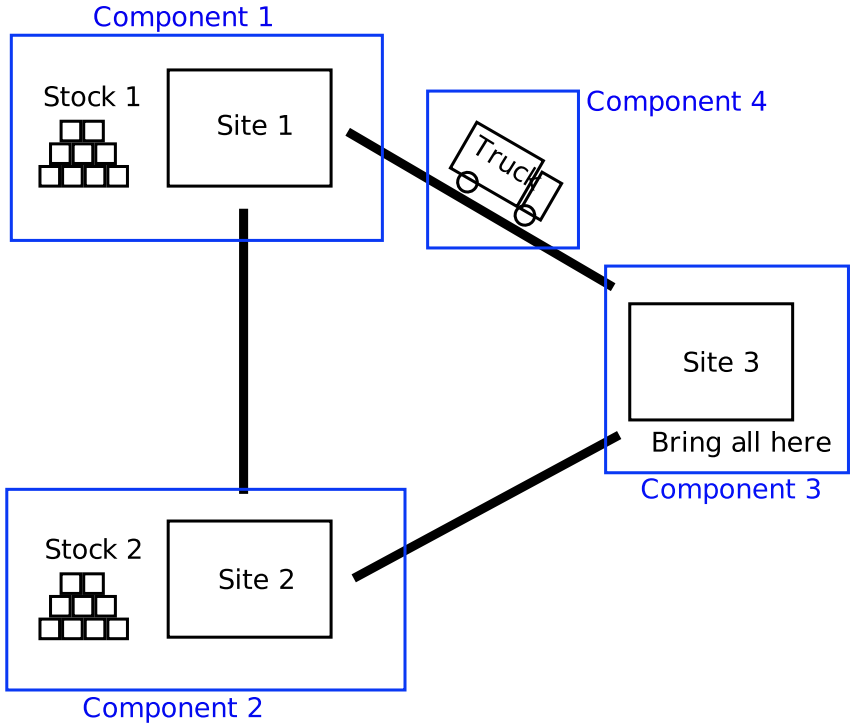
Parallelism of actions (concurrency)

- GRAPHPLAN: Blum and Furst *1995*
- Petri net unfolding: Hickmott et al. 2007, Bonet et al. 2008

Split problems into subproblems (Factored planning)

Amir and Engelhardt *2003*, Brafman and Domshlak 2006, Brafman and Domshlak 2008

Factored planning: principles

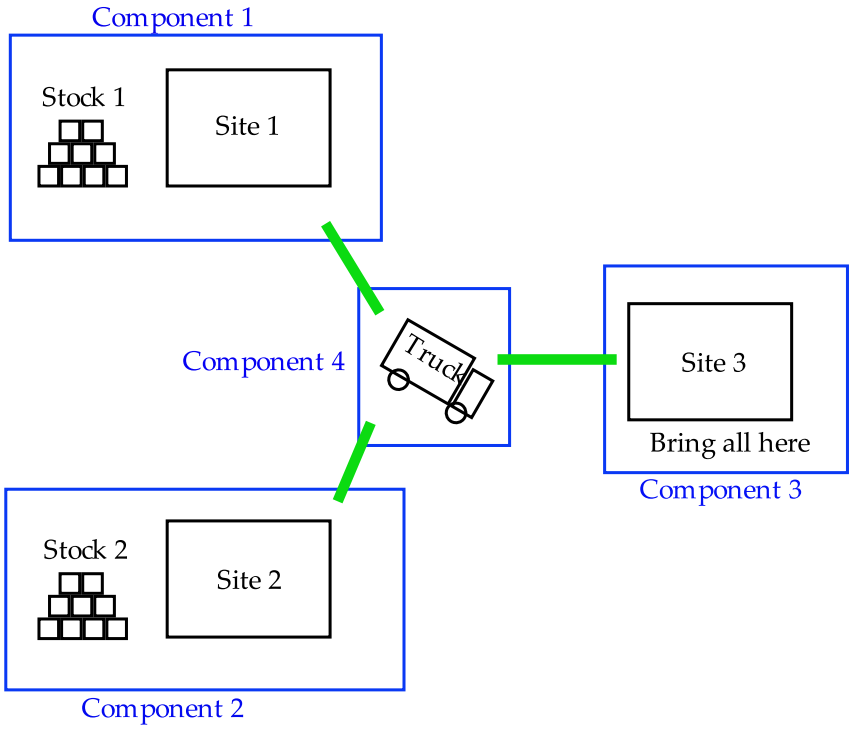


Each component is a planning problem with its own resources and actions

Goal

Find a set of *compatible* local plans: they can be *interleaved* into a global plan

Factored planning: principles



Each component is a planning problem with its own resources and actions

The components interact by resources and/or actions

Goal

Find a set of *compatible* local plans: they can be *interleaved* into a global plan

Factored planning: our contribution

Prior to this thesis, reasoning on the number of synchronizations:

- Absence of solution *can not be detected*
- Cost-optimality of plans *can not be achieved*

Our contribution

Two new approaches to *factored planning*, allowing to find *cost-optimal* plans with *distributed* algorithms

Top-down approach

Successive restrictions of the sets of local plans

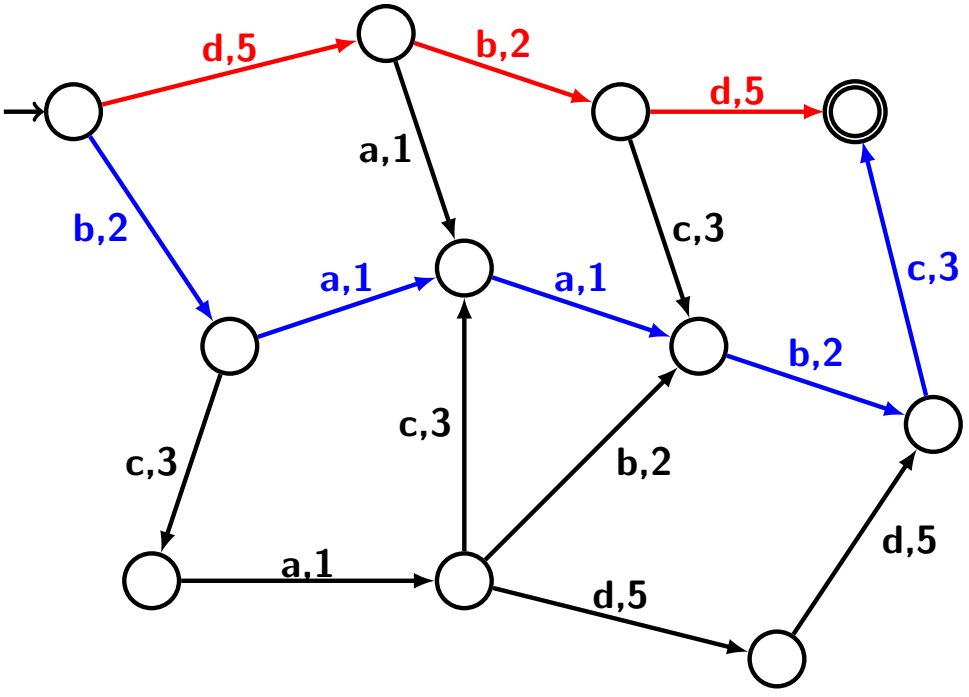
Bottom-up approach

Progressive construction of a local plan per component

Top-down approach

Factored cost-optimal planning using message passing algorithms

Centralized planning problem = weighted automaton



Set of actions Σ

The words are the plans

The words with minimal cost are the cost-optimal plans

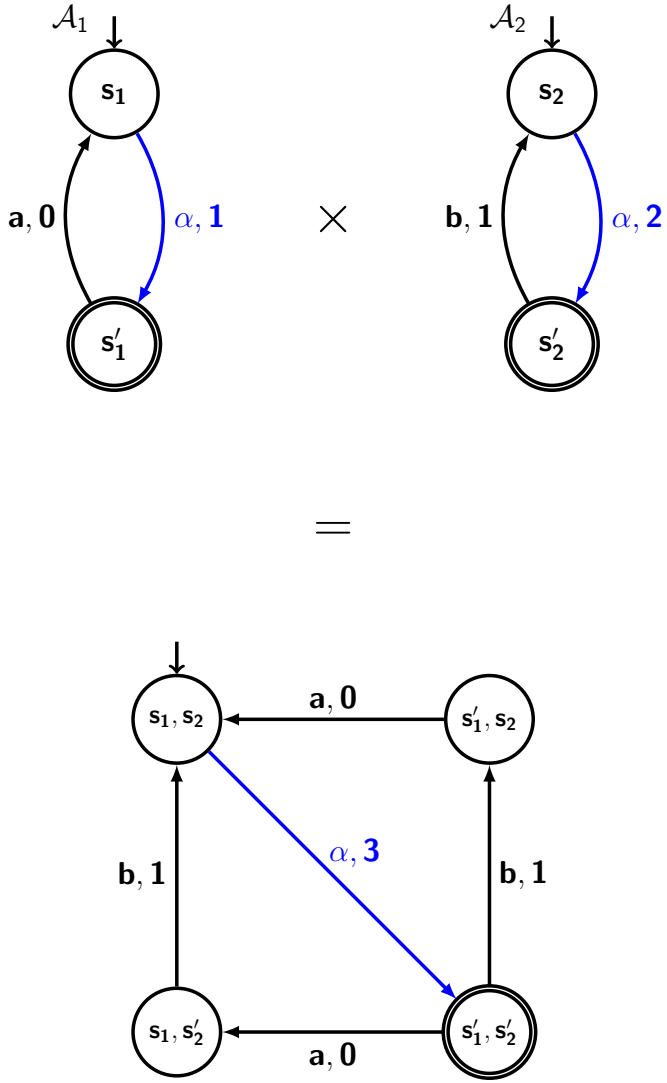
Goal

Find a minimal cost word in a weighted automaton

Factored planning problem

Components are weighted automata

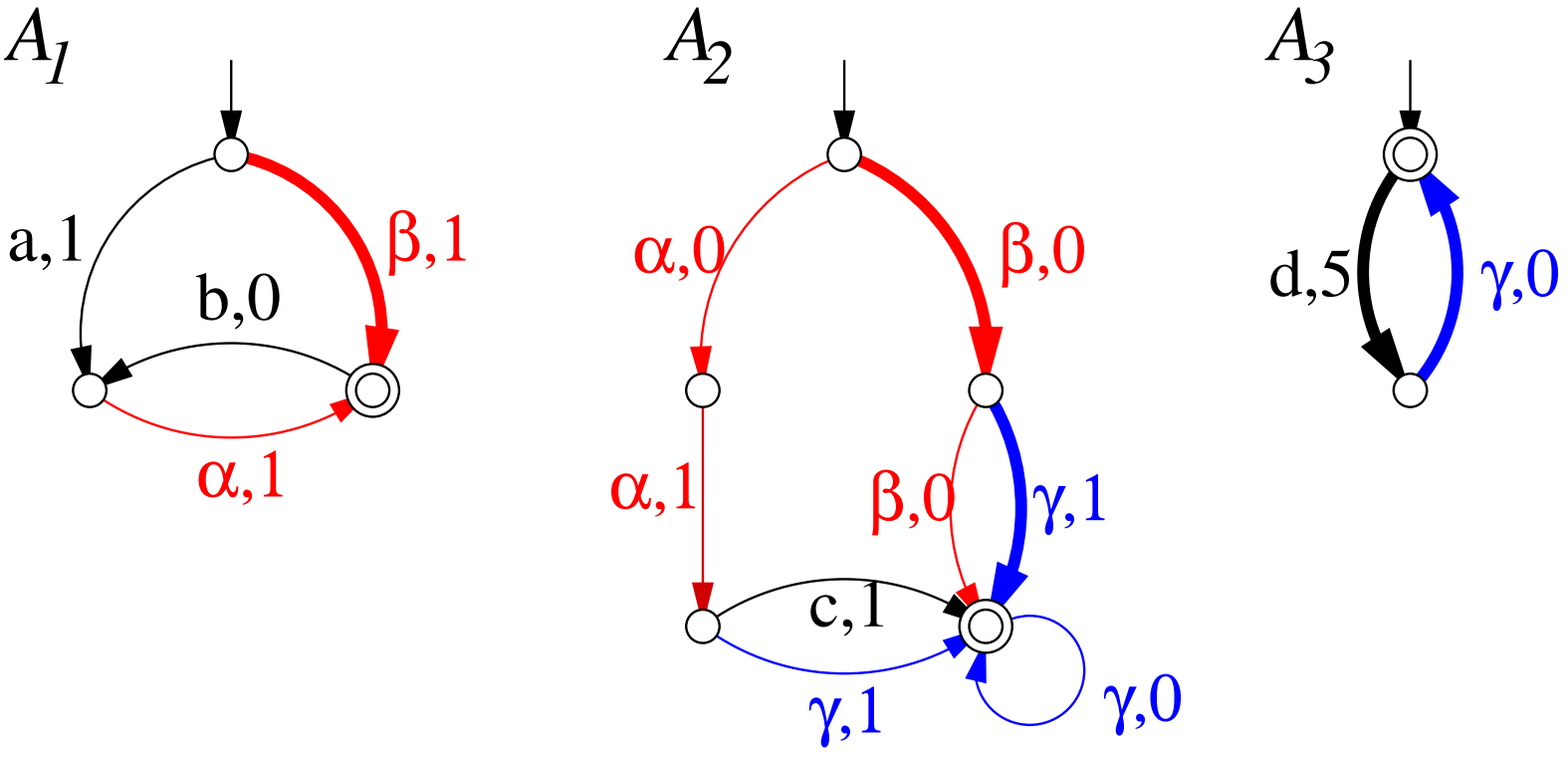
They interact by their shared actions: formalization using the notion of *synchronous product*



Goal

In $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$, find a tuple $(w_1, \ldots, w_n)$ of words which are all *compatible* and *minimize* the sum of their cost, *without computing* $\mathcal{A}$

Factored planning problem: example



Centralized plans: $\beta d \gamma$ and $d \beta \gamma$

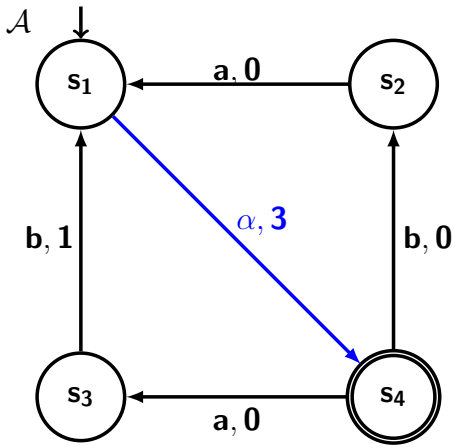
Factored/distributed/concurrent plan: $(\beta, \beta\gamma, d\gamma)$

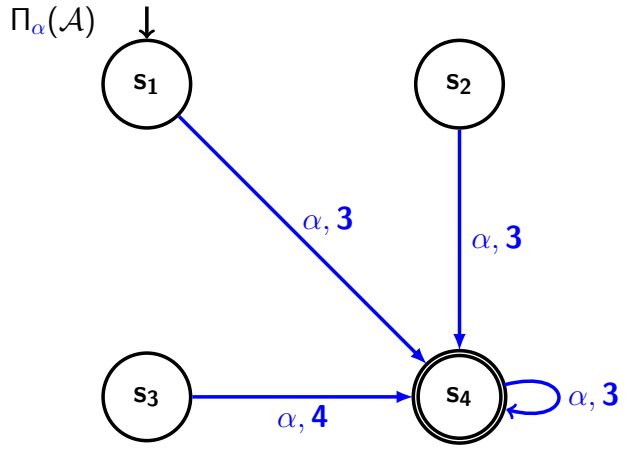
Projection: from global plans to local plans

Projection reduces a *global plan* to the actions of *a particular component*

$\Pi_{\Sigma'}$ corresponds to:

1. Replace each action not in Σ' by ε
2. Perform ε-reduction (to the left)
3. (Minimize)





MPA: computing $\Pi_{\Sigma_i}(\mathcal{A})$ without computing $\mathcal{A}$

Central properties of the projection of $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$

1. any *cost-optimal* word w of $\mathcal{A}$ can be projected into a *cost-optimal* word w_i of $\Pi_{\Sigma_i}(\mathcal{A})$, moreover $c(w) = c_i(w_i)$
2. any *cost-optimal* word w_i of $\Pi_{\Sigma_i}(\mathcal{A})$ is the projection of a *cost-optimal* word w of $\mathcal{A}$, moreover $c_i(w_i) = c(w)$

Consequence

Taking the minimal cost word in each $\Pi_{\Sigma_i}(\mathcal{A})$ gives a cost-optimal global plan (hypothesis: it is unique)

Building the $\Pi_{\Sigma_i}(\mathcal{A})$ by local computations

Successive refinements of the $\mathcal{A}_i$ from the constraints imposed by their neighbours

How to get the $\Pi_{\Sigma_i}(\mathcal{A})$: the message passing algorithms

Fundamental property (conditional independence)

$$\Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1 \times \mathcal{A}_2) \equiv_{\mathcal{L}} \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2)$$

Application:

$$\mathcal{A}_1 \overset{\Sigma_1 \cap \Sigma_2}{\text{———}} \mathcal{A}_2 \overset{\Sigma_2 \cap \Sigma_3}{\text{———}} \mathcal{A}_3$$

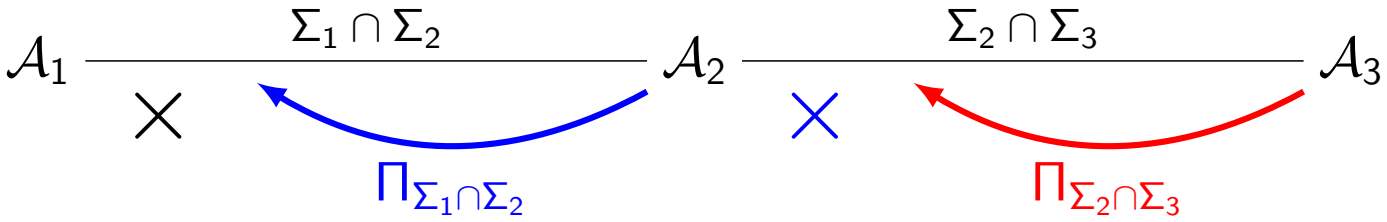
$$\begin{aligned}
\Pi_{\Sigma_1}(\mathcal{A}) &= \Pi_{\Sigma_1}(\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \Pi_{\Sigma_1}(\mathcal{A}_1) \times \Pi_{\Sigma_1}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(\mathcal{A}_3))
\end{aligned}$$

How to get the $\Pi_{\Sigma_i}(\mathcal{A})$: the message passing algorithms

Fundamental property (conditional independence)

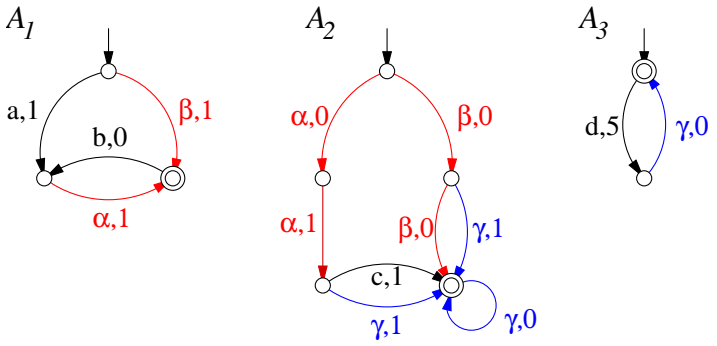
$$\Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1 \times \mathcal{A}_2) \equiv_{\mathcal{L}} \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_1) \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2)$$

Application:

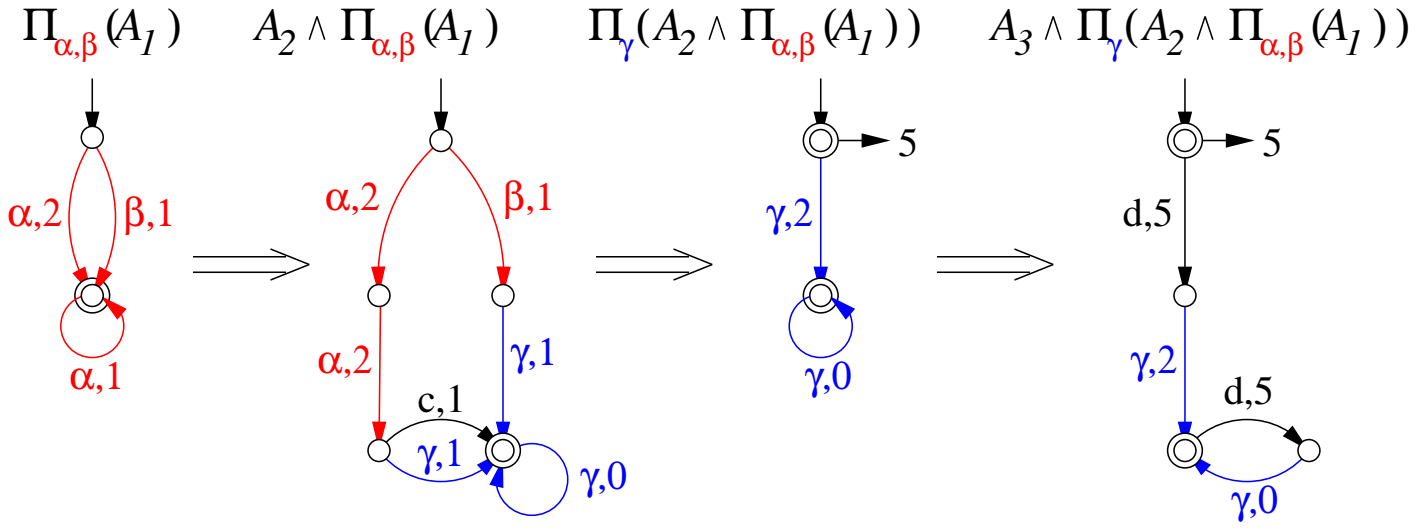


$$\begin{aligned}
\Pi_{\Sigma_1}(\mathcal{A}) &= \Pi_{\Sigma_1}(\mathcal{A}_1 \times \mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \Pi_{\Sigma_1}(\mathcal{A}_1) \times \Pi_{\Sigma_1}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \mathcal{A}_3) \\
&\equiv_{\mathcal{L}} \mathcal{A}_1 \times \Pi_{\Sigma_1 \cap \Sigma_2}(\mathcal{A}_2 \times \Pi_{\Sigma_2 \cap \Sigma_3}(\mathcal{A}_3))
\end{aligned}$$

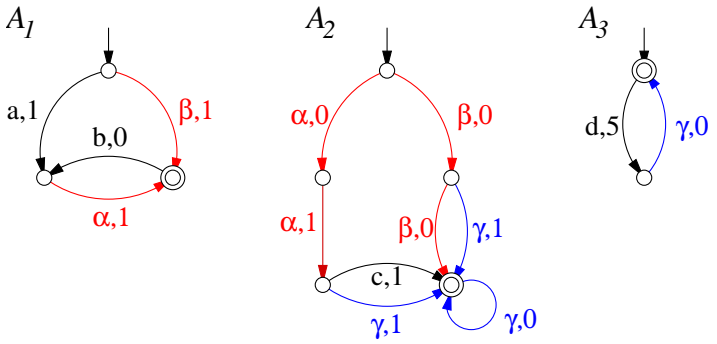
Example



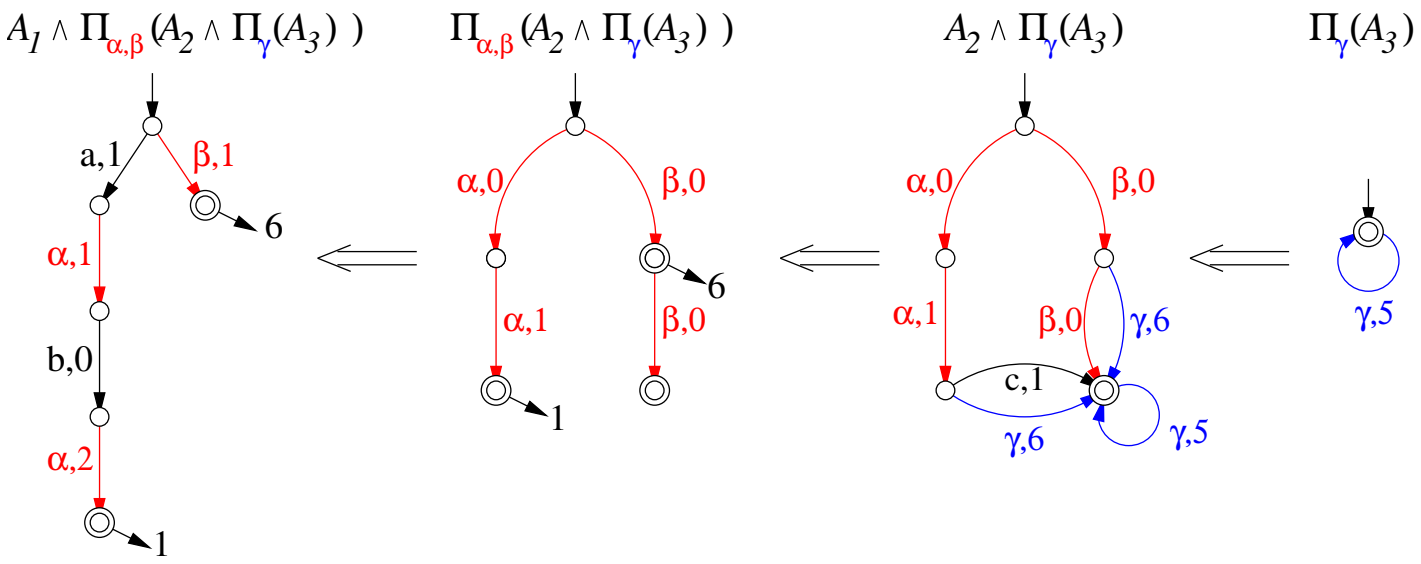
Messages from $\mathcal{A}_1$ to $\mathcal{A}_3$:



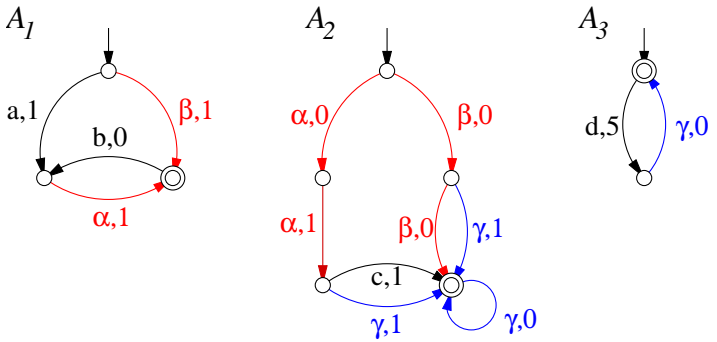
Example



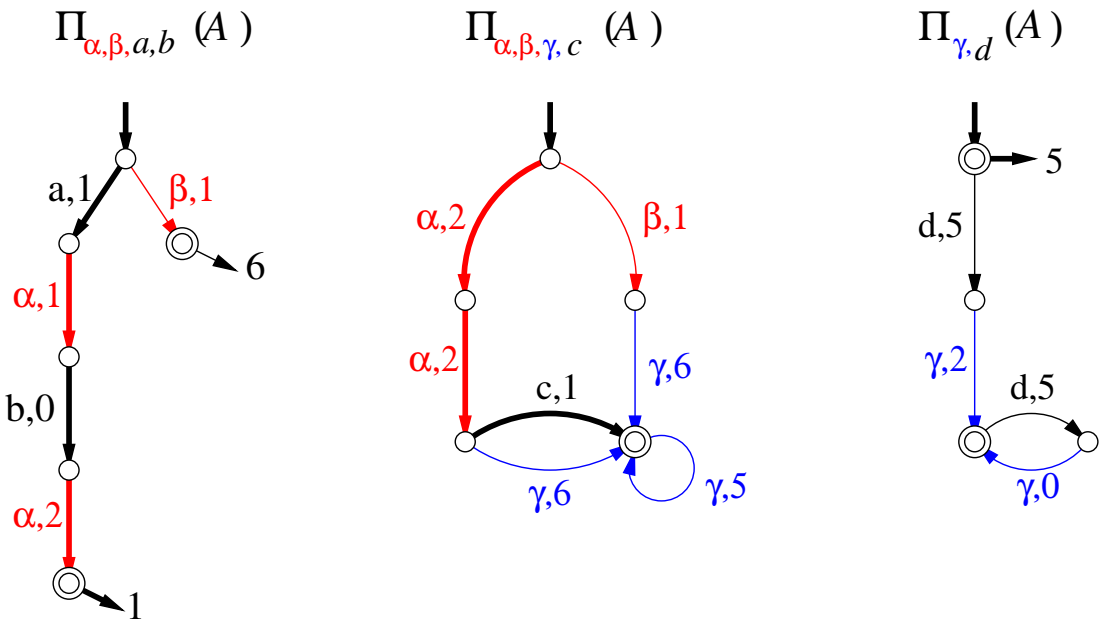
Messages from $\mathcal{A}_3$ to $\mathcal{A}_1$:



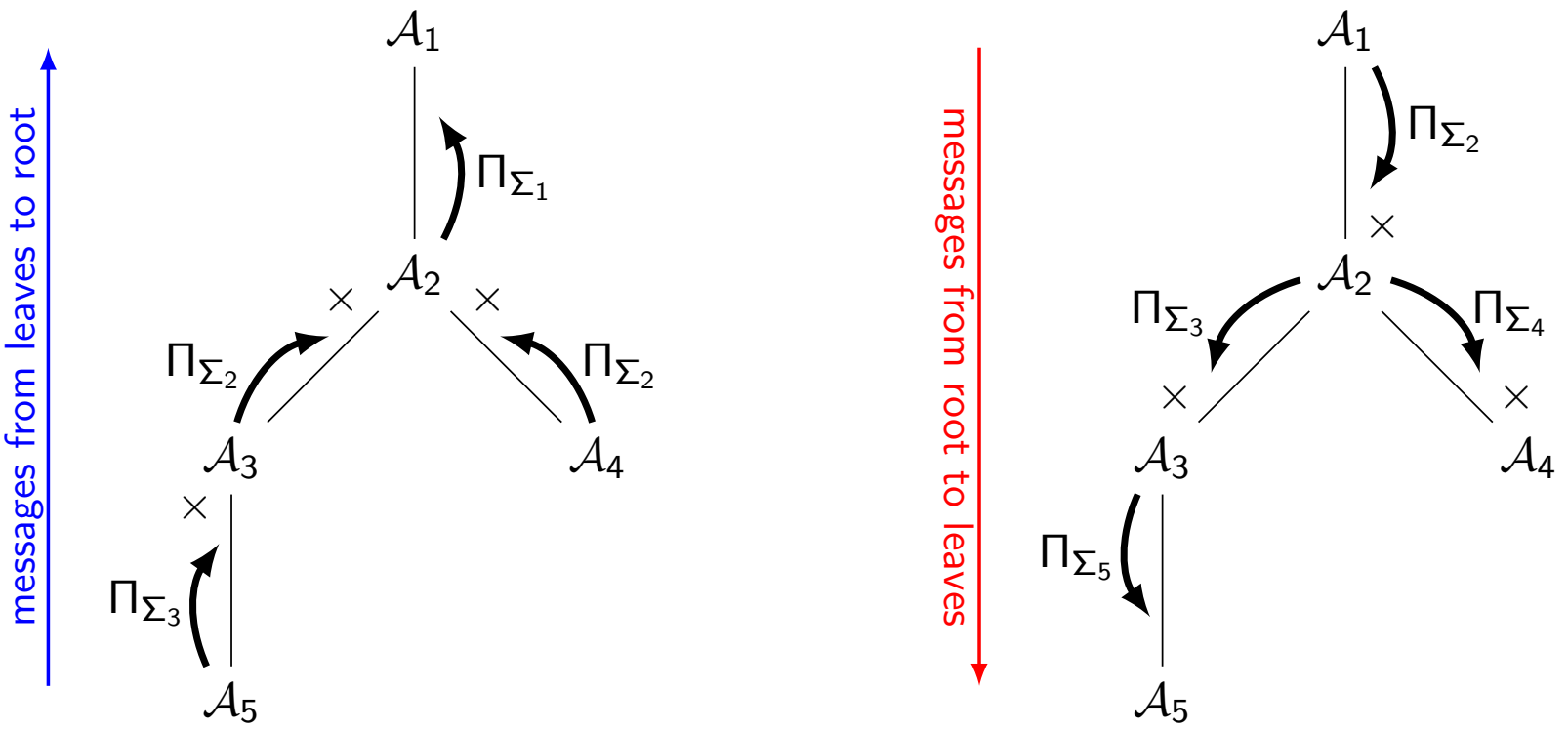
Example



Updated components:



Message passing algorithms: main results generalized[1]



Theorem

If $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ has a *tree shaped interaction* graph, the message passing algorithm *converges* and returns $\mathcal{A}'_i \equiv_{\mathcal{L}} \Pi_{\Sigma_i}(\mathcal{A})$ for each $\mathcal{A}_i$

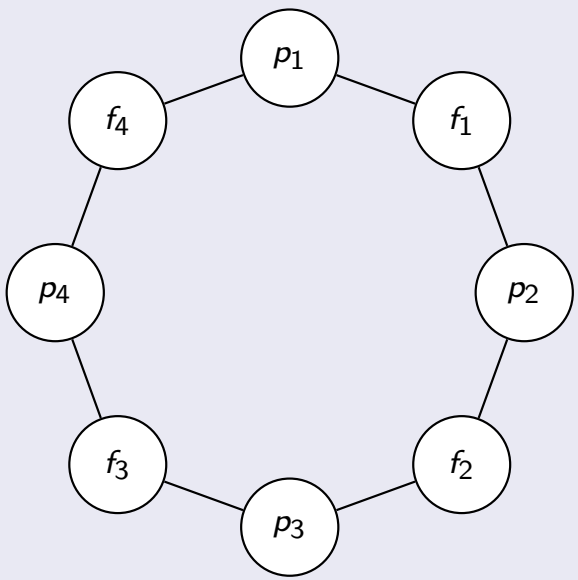
[1] Eric Fabre and Loïg Jezequel, *Distributed Optimal Planning: An Approach by Weighted Automata Calculus*, CDC 2009

Distoplan: presentation

Distoplan

C++ implementation of the message passing on weighted automata, on top of *openFST*[2] and the *HSP*'s parser*[3]

A benchmark: philosophers from IPC4



[2] http://www.openfst.org/

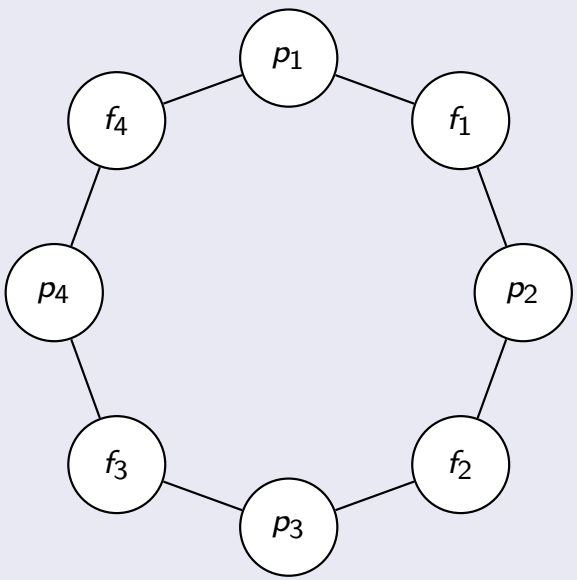
[3] Patrik Haslum, 4^{th} IPC Booklet,2004

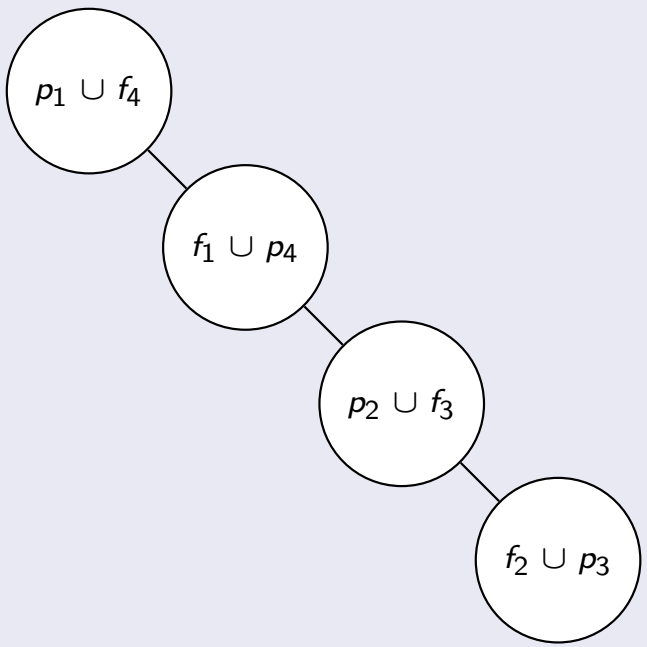
Distoplan: presentation

Distoplan

C++ implementation of the message passing on weighted automata, on top of *openFST*[2] and the *HSP*'s parser*[3]

A benchmark: philosophers from IPC4

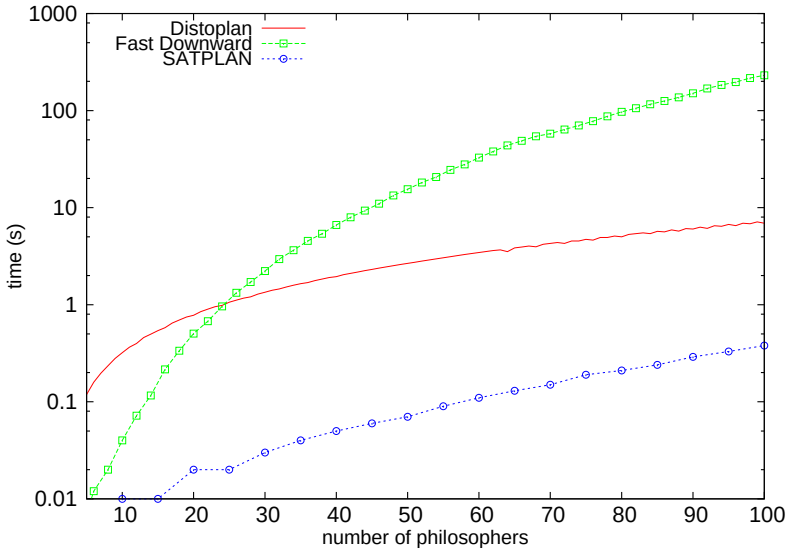


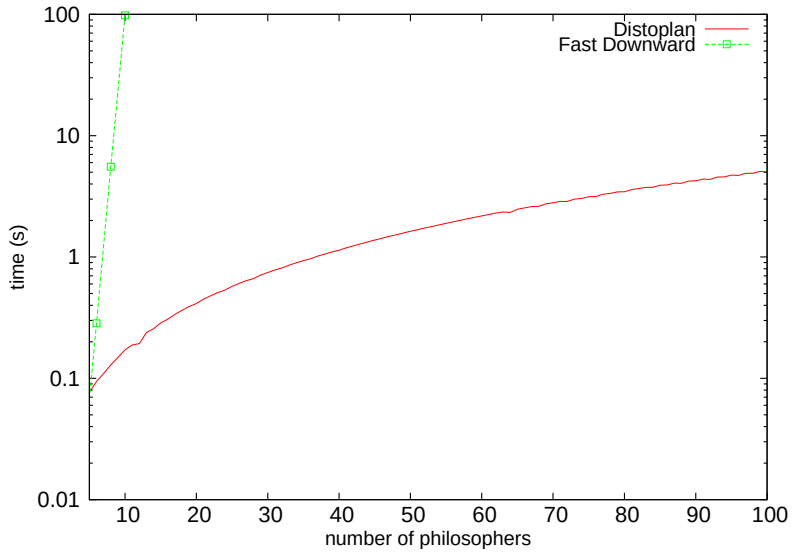


[2] http://www.openfst.org/

[3] Patrik Haslum, 4^{th} IPC Booklet,2004

Distoplan[4]



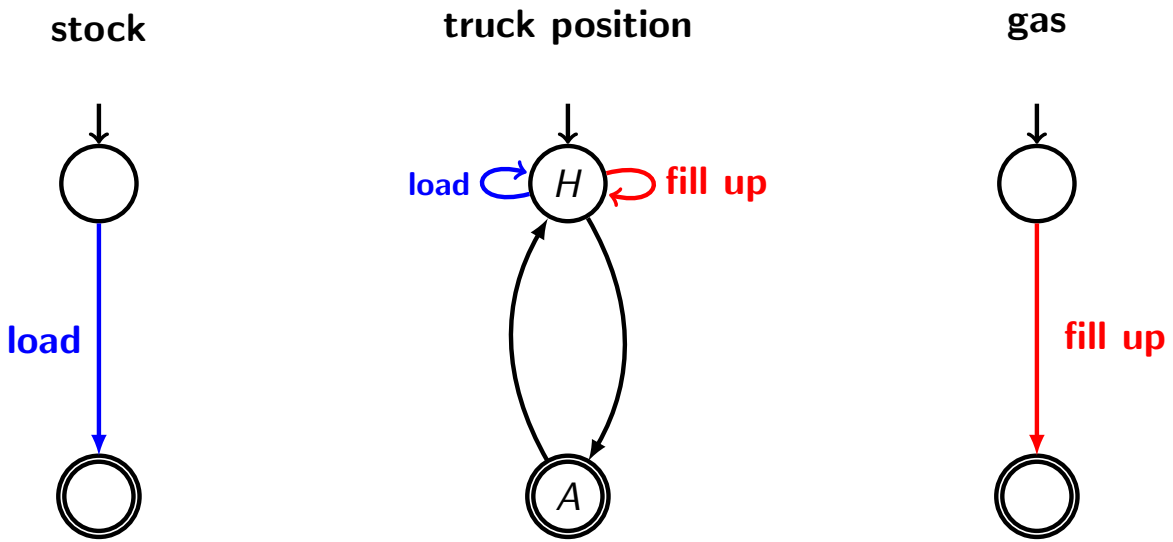


Conclusions

- Practical solving of planning problems
- Can be more efficient than centralized search
- Difficulty: find decompositions

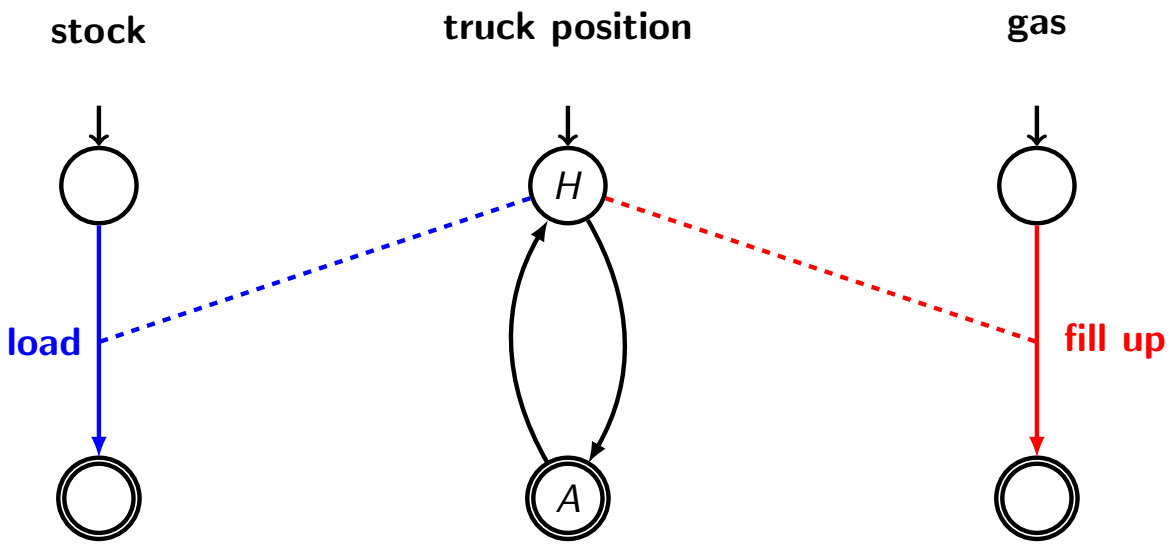
[4] Eric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux, *Cost-Optimal Factored Planning: Promises and Pitfalls*, ICAPS 2010

Extension 1: read arcs in networks of automata[5]



[5] Loïg Jezequel and Eric Fabre, *Networks of Automata with Read Arcs: A Tool for Distributed Planning*, IFAC World Congress 2011

Extension 1: read arcs in networks of automata[5]



Read arcs

Automata $\rightarrow$ automata with readings/writings on transitions

Theorem

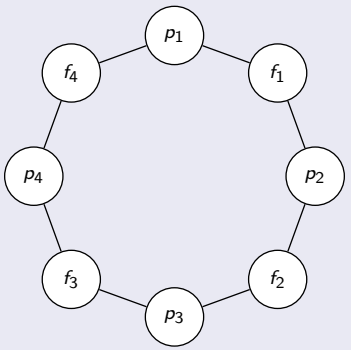
The message passing algorithms extend to this setting with minor modifications

[5] Loïg Jezequel and Eric Fabre, *Networks of Automata with Read Arcs: A Tool for Distributed Planning*, IFAC World Congress 2011

Extension 2: turbo planning[6]

Starting point

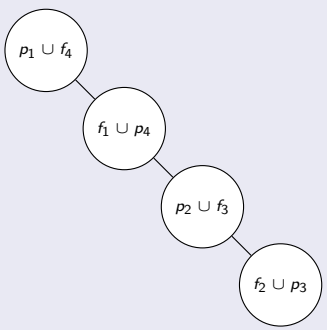
When interaction graphs contain cycles:



Existing solution

Tree decomposition of graphs:

- Not all parameters taken into account
- Tree-width can be huge

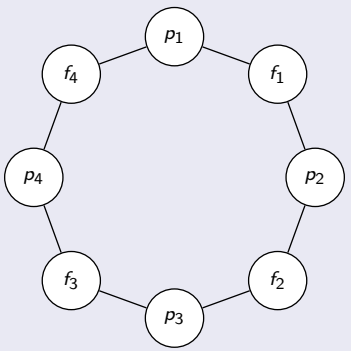


[6] Loïg Jezequel and Eric Fabre, *Turbo Planning*, WODES 2012

Extension 2: turbo planning[6]

Starting point

When interaction graphs contain cycles:



Turbo planning

Ignore cycles and perform *approximate planning*

Result: $\mathcal{A}'_i$ such that $\mathcal{L}(\Pi_{\Sigma_i}(\mathcal{A})) \subseteq \mathcal{L}(\mathcal{A}'_i) \subseteq \mathcal{L}(\mathcal{A}_i)$

[6] Loïg Jezequel and Eric Fabre, *Turbo Planning*, WODES 2012

Extension 2: convergence issues of turbo planning

As a constraint solving problem

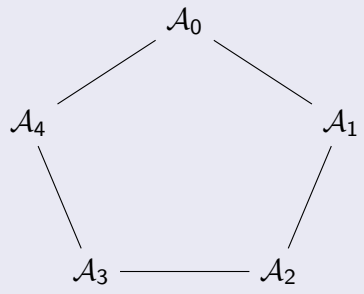
- Convergence in (possibly) infinite time
- Convergence in finite time for words of small length

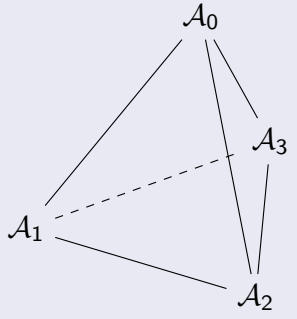
As an optimization problem

- Costs diverge in general
- Normalization:
  - Costs of optimal paths stabilize
  - Costs of other paths still diverge

Extension 2: experiments on turbo planning

Problem shapes



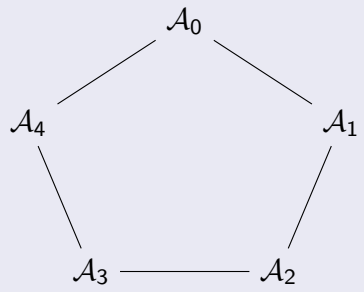


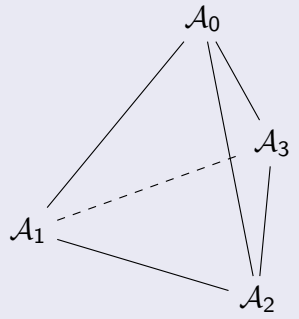
Results

- Fast convergence (always less than 5 iterations)
- Promising quality of the solutions found (70% of solutions cost less than 10% more than the optimal)

Extension 2: experiments on turbo planning

Problem shapes





Results

- Fast convergence (always less than 5 iterations)
- Promising quality of the solutions found (70% of solutions cost less than 10% more than the optimal)

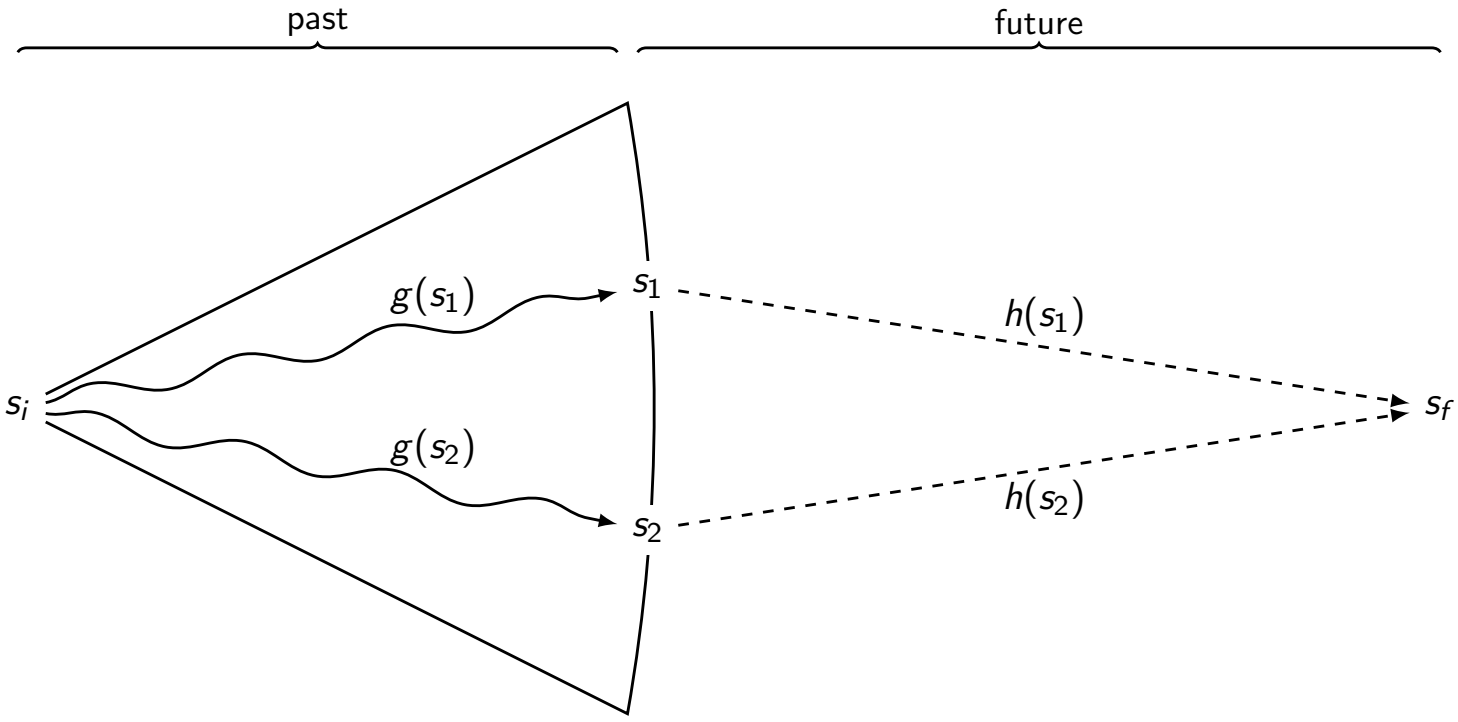
Open question

Theoretical explanation of this efficiency

Bottom-up approach

Cost-optimal planning using a distributed version of A*

A*: a best-first search algorithm

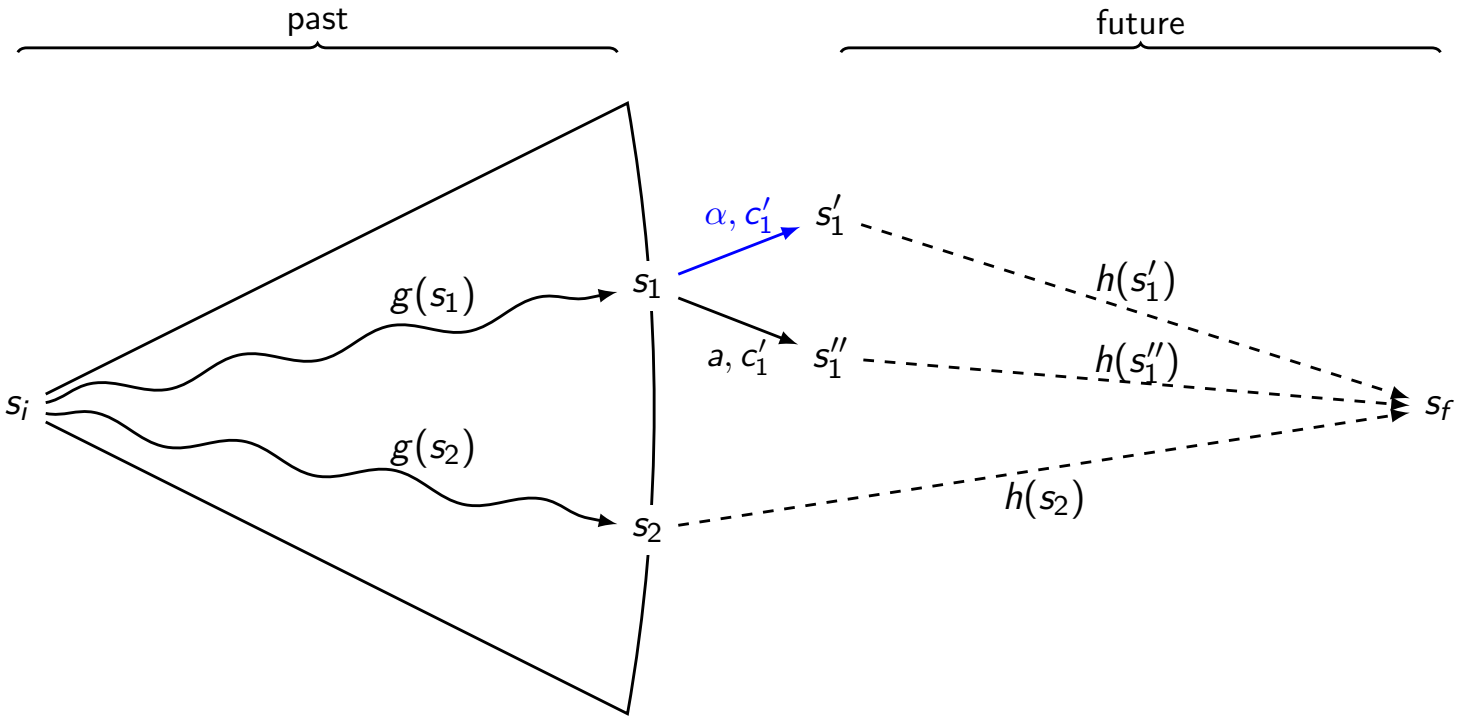


Rank of a node

Most promising node: $s^* = argmin_s(g(s) + h(s))$

Always expand from s^* first

A*: a best-first search algorithm

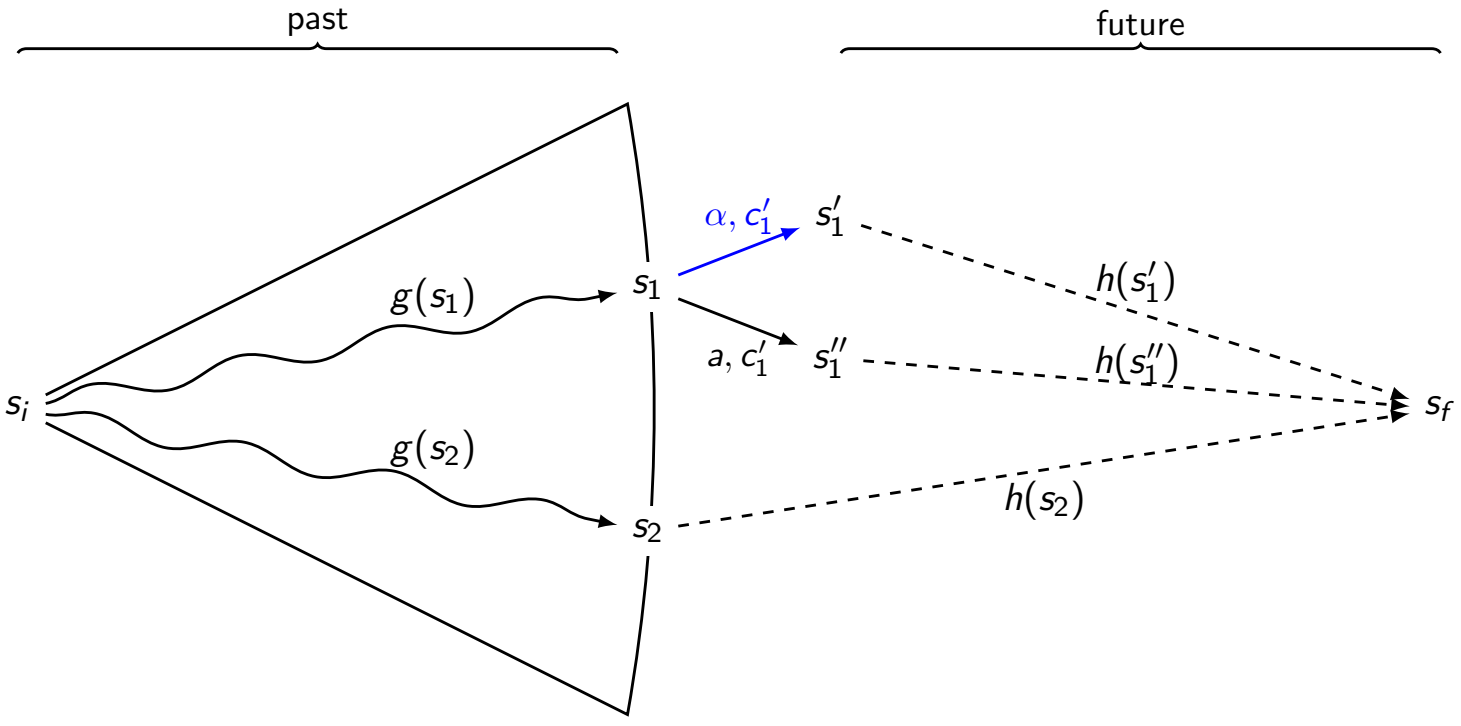


Rank of a node

Most promising node: $s^* = argmin_s(g(s) + h(s))$

Always expand from s^* first

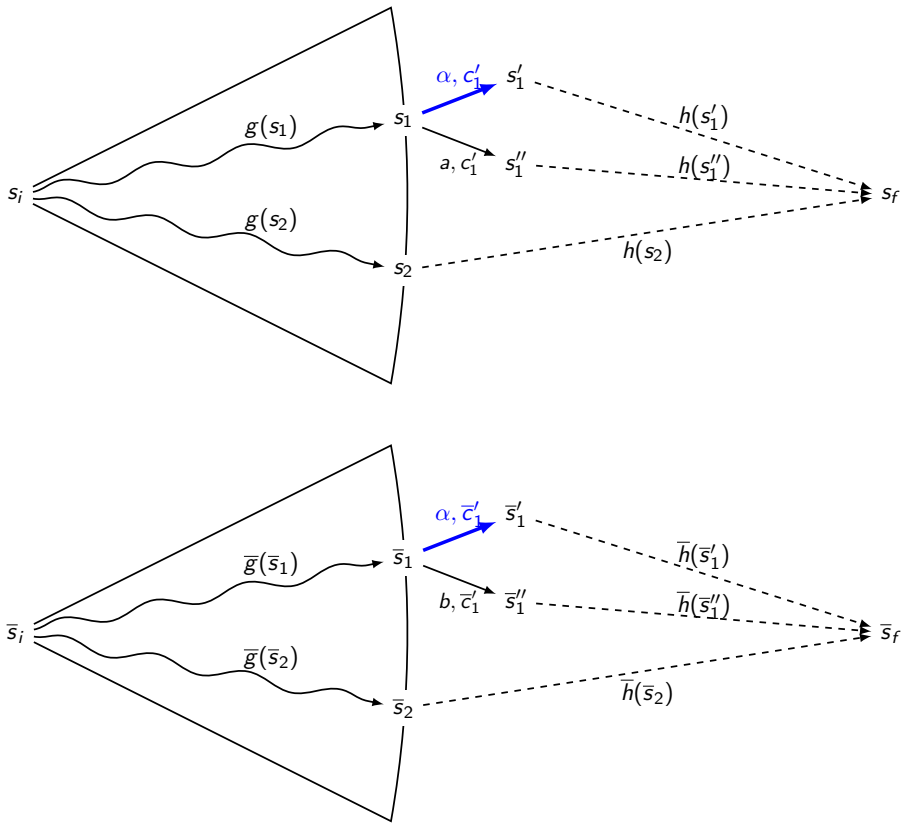
A*: a best-first search algorithm



Termination

When s_f is the most promising node

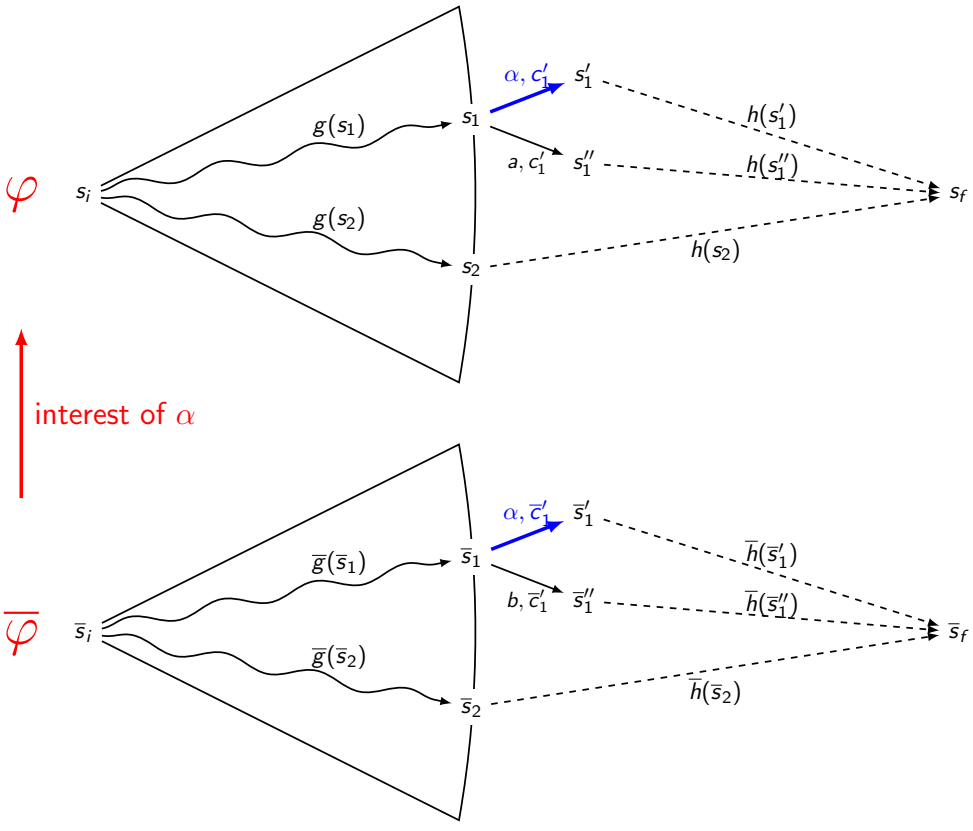
A#: intuition



Goal

A pair of compatible paths

A#: intuition



Goal

A pair of compatible paths

Idea

Parallel constrained searches

Agent φ

A* with information from $\overline{\varphi}$

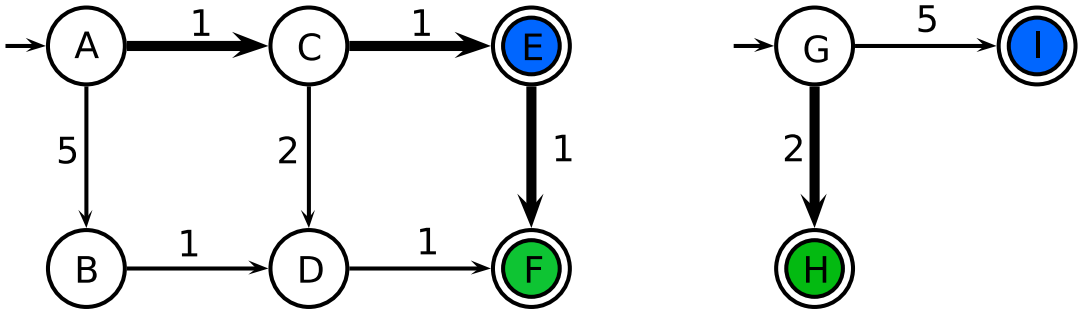
Compatible final states

The problem

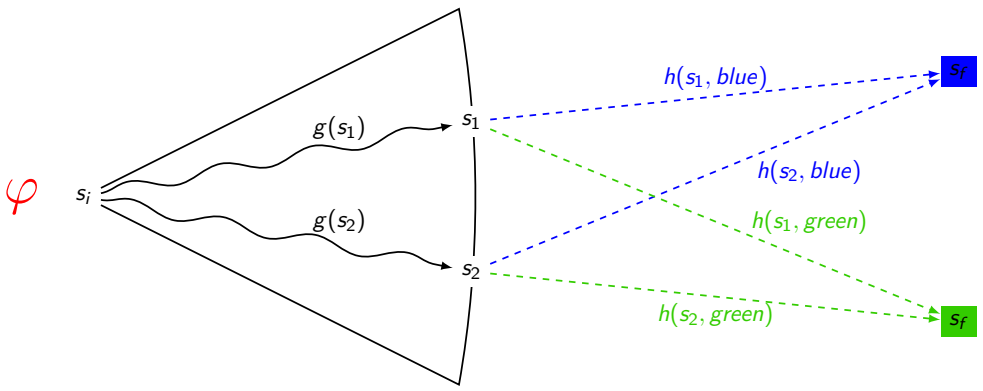
- Two automata (not sharing actions)
- A colouring function on final states

Goal: find a path in each automaton such that:

- They both reach final states of the same colour
- The sum of their costs is minimal among such paths



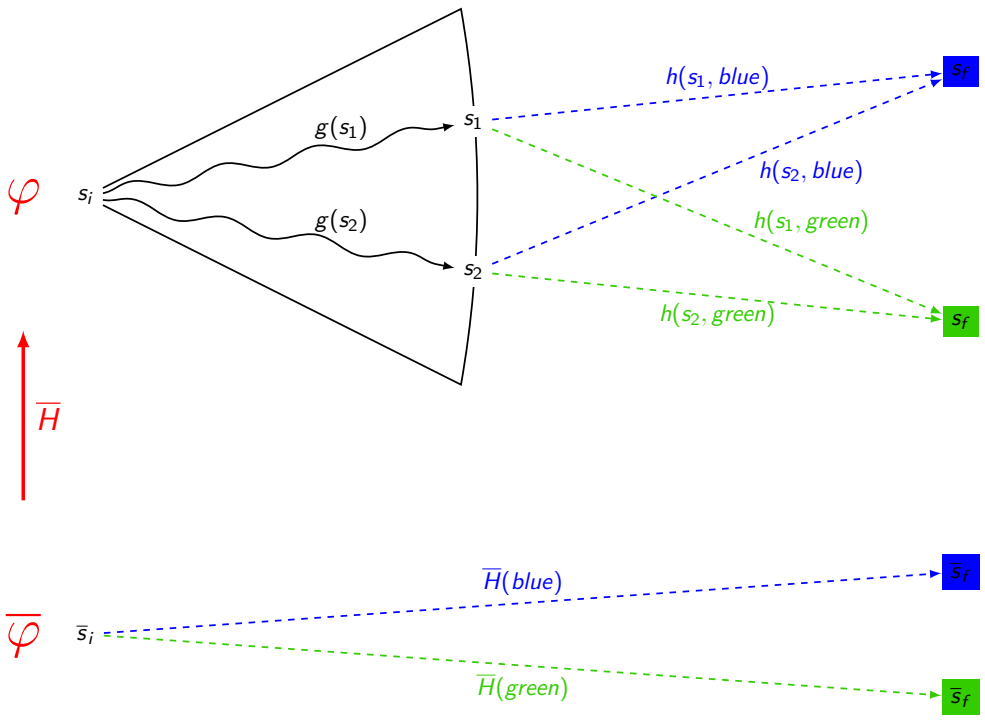
Compatible final states: ranking



Locally

One heuristic h per color

Compatible final states: ranking



Locally

One heuristic h per color

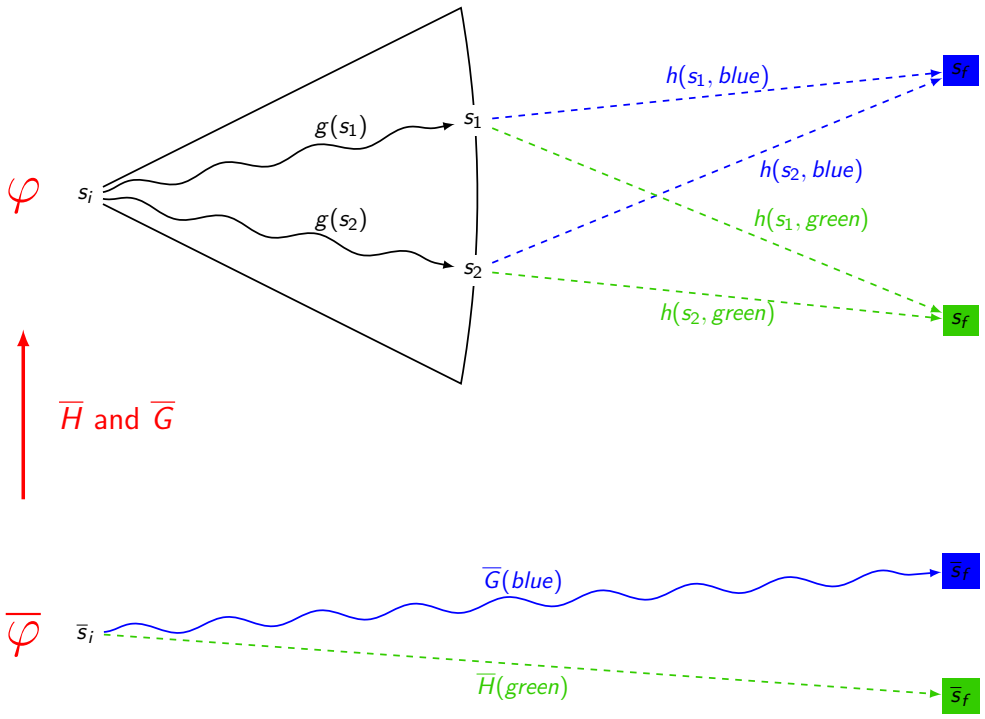
Externally

information $\overline{H}$ from $\overline{\varphi}$

Rank(s)

$g(s) + \min_c(h(s, c) + \overline{H}(c))$

Compatible final states: termination



$\overline{G}$

Achieved best costs for colors

Termination

s a goal with color c_s such that:

$g(s) + \overline{G}(c_s)$ lowest rank

Compatible final states: results[7]

Theorem

When executed by φ on any CFS problem:

- A# terminates,
- A# is sound,
- A# is complete,

assuming that φ has access to $\overline{G}$ and $\overline{H}$

[7] Loïg Jezequel and Eric Fabre, *A#: A Distributed Version of A* for Factored Planning*, CDC 2012

Compatible final states: results[7]

Theorem

When executed by φ on any CFS problem:

- A# terminates,
- A# is sound,
- A# is complete,

assuming that φ has access to $\overline{G}$ and $\overline{H}$

Theorem

$\overline{G}$ and $\overline{H}$ can be computed by $\overline{\varphi}$ along its own execution of A#

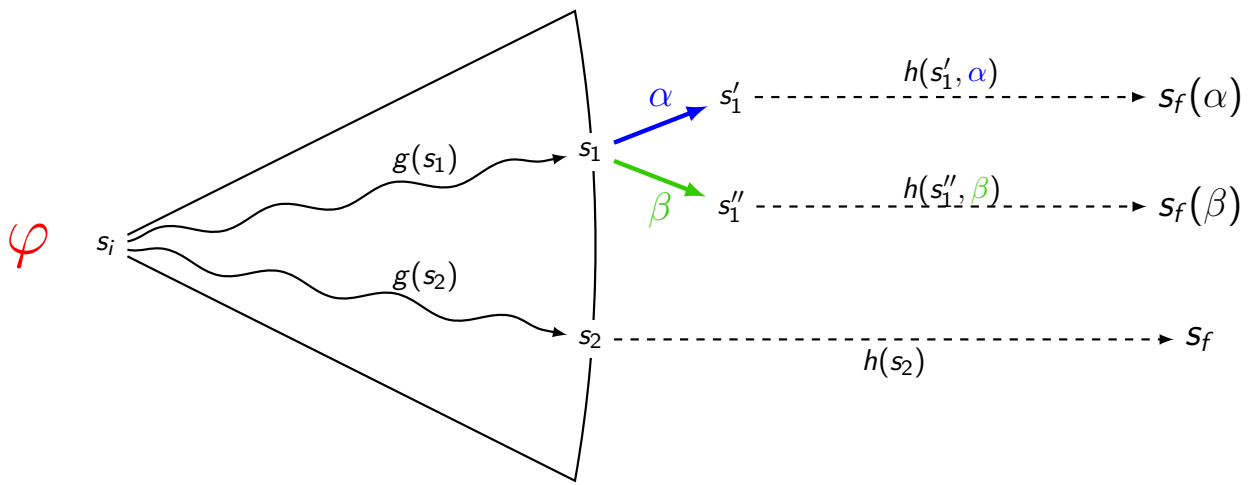
[7] Loïg Jezequel and Eric Fabre, *A#: A Distributed Version of A* for Factored Planning*, CDC 2012

A#: from CFS to factored planning

CCP and factored planning with two components

- Colour = sequence of (shared) actions
- Number of colours cannot be bounded locally

Consequence: computation of h, H, and G more difficult

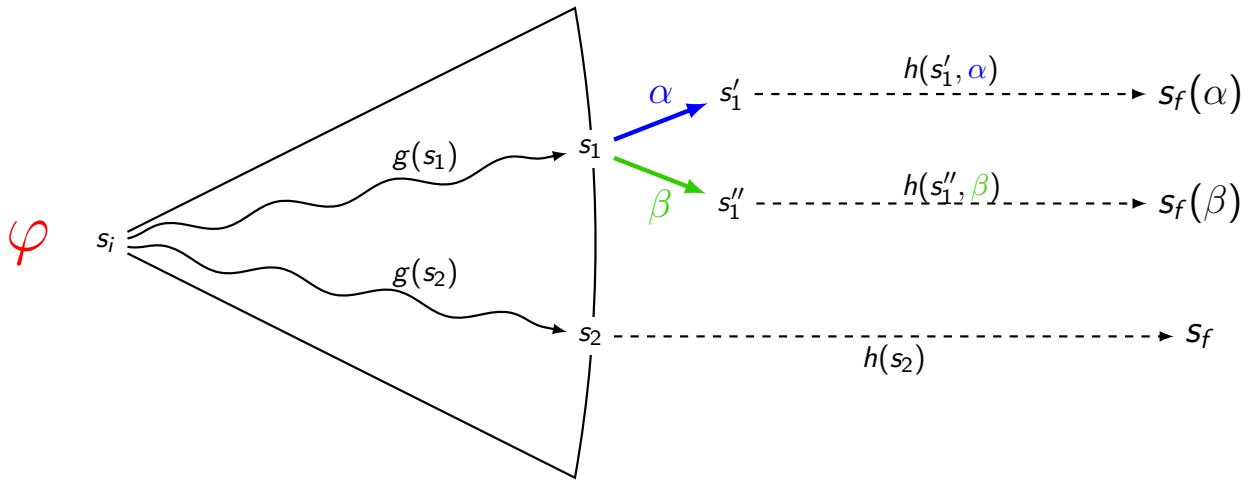


A#: from CFS to factored planning

CCP and factored planning with two components

- Colour = sequence of (shared) actions
- Number of colours cannot be bounded locally

Consequence: computation of h, H, and G more difficult



Theorem: termination

As soon as the considered factored planning problem has a solution, A# terminates

A#: extension to larger interaction graphs

From the point of view of agent φ_i, any factored planning problem has two components:

$$\mathcal{A}_i \text{ ——————— } \Pi_{k \neq i} \mathcal{A}_k$$

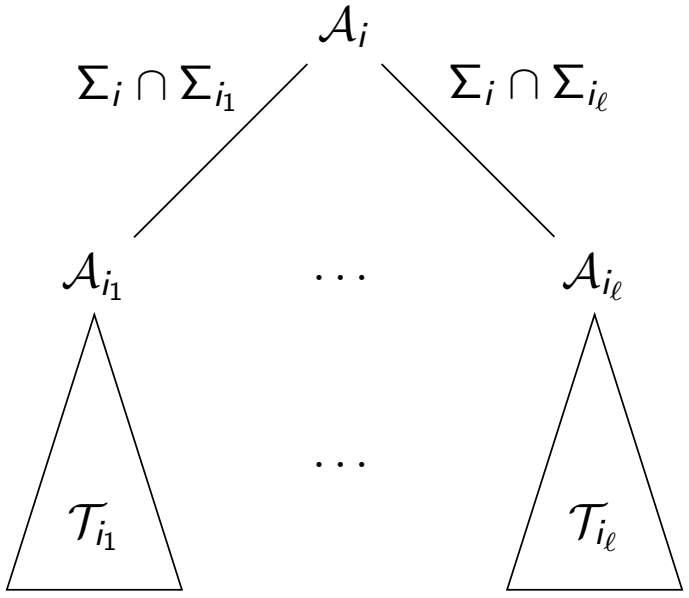
A#: extension to larger interaction graphs

From the point of view of agent φ_i, any factored planning problem has two components:

$$\mathcal{A}_i \quad \text{——————} \quad \Pi_{k \neq i} \mathcal{A}_k$$

Theorem

If the interaction graph is a tree $\overline{H}$ and $\overline{G}$ can be constructed using only information (messages) from the neighbours of φ_i



Conclusion and perspectives

Conclusion

Main contribution

Two planning algorithms allowing: *distributed planning* and *cost-optimal planning*

First approach

- Message passing algorithms + weighted automata calculus
- Implementation in Distoplan
- Reading variables
- Turbo planning (approximate methods for factored planning)

Second approach

- Distributed version of A*
- Proof of validity and implementability

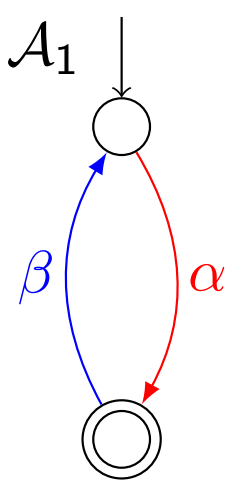
Perspectives

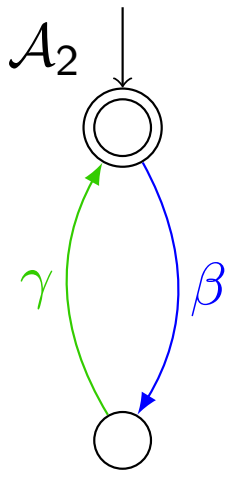
One needs *benchmarks* for factored planning

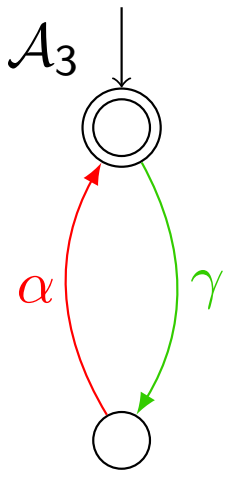
How to *automatically decompose* planning problems?

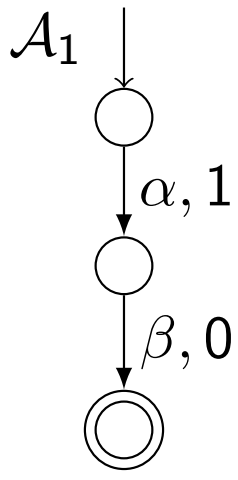
Is it possible to benefit from *local concurrency*?

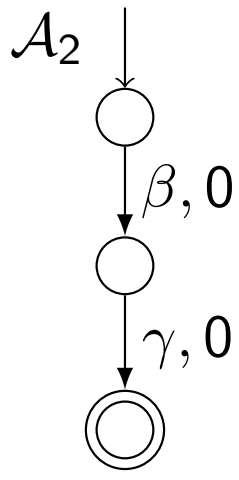
When/why turbo planning works?

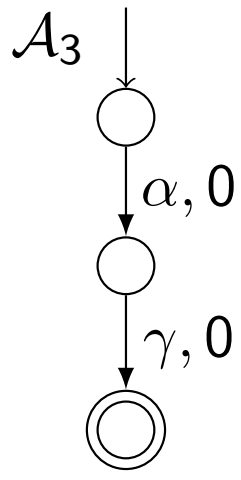

$\mathcal{A}_1$
β
α


$\mathcal{A}_2$
γ
β


$\mathcal{A}_3$
α
γ


$\mathcal{A}_1$
$\alpha, 1$
$\beta, 0$


$\mathcal{A}_2$
$\beta, 0$
$\gamma, 0$


$\mathcal{A}_3$
$\alpha, 0$
$\gamma, 0$