Decoupled Search: A New Form of State-Space Exploration

Álvaro Torralba

AALBORG UNIVERSITY

International Symposium on Combinatorial Search
SoCS 2023
Aquapalace Hotel Prague
Czech Republic
July 14-16, 2023

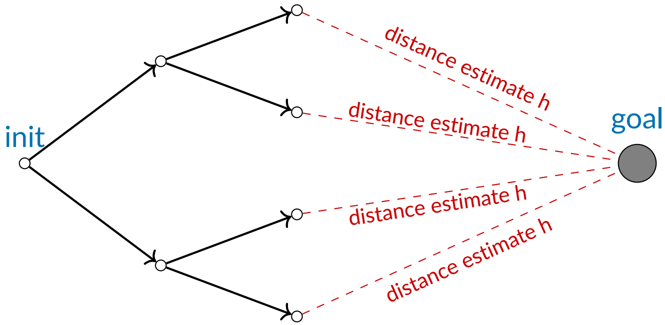
What's this About

- Decoupled Search:
  - New technique for state-space exploration in AI-planning and model-checking

Daniel Gnad (Gnad (2021))

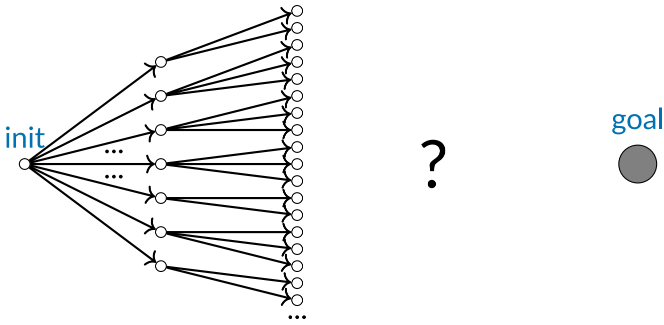
Joerg Hoffmann

A Successful Approach in General: Heuristic Search



$\rightarrow$ State space search with heuristic function h maps states s to an estimate $h(s)$ of goal distance.

Heuristic Search – Limitations



State explosion problem:
State space of a planning task is exponential in the number of variables.

Domain-independent planning versus domain-dependent solvers

Finding Optimal Solutions to Rubik's Cube Using Pattern Databases

Richard E. Korf
Computer Science Department
University of California, Los Angeles
Los Angeles, Ca. 90095
Korf@cs.ucla.edu

Abstract

We have found the first optimal solutions to random instances of Rubik's Cube. The median optimal solution length appears to be 18 moves. The algorithm used is iterative-deepening-A* (IDA*), with a lower-bound heuristic function based on large memory-based lookup tables, or "pattern databases" (Culberson and Schaeffer 1996). These tables store the exact number of moves required to solve various subgoals of the problem, in this case subsets of the individual movable cubies. We characterize the effectiveness of an admissible heuristic function by its expected value, and hypothesize that the overall performance of the program obeys a relation in which the product of the time and space used equals the size of the state space. Thus, the speed of the program increases linearly with the amount of memory available. As computer memories become larger and cheaper, we believe that this approach will become increasingly cost-effective.

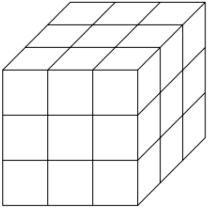
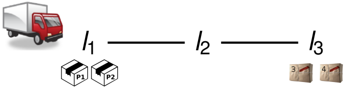


Figure 1: Rubik's Cube

The dream: reduce the gap to a point where domain-independent planners are as efficient than

Domain-independent vs domain-dependent

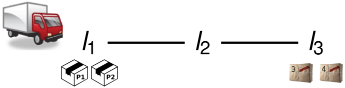
Running Example:



- $V = \{t, p_1, \dots p_N\}$ with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $I = \{(t, l_1), (p_1, l_1), (p_2, l_1), (p_3, l_3), (p_4, l_3)\}$
- A = $\{load(p_i, x), unload(p_i, x), drive(x, x')\}$, where:
  $pre_{load(p_i,x)} = \{(t, x), (p_i, x)\}$ and
  $eff_{load(i,x)} = \{(p_i, t)\}$
- $G = \{(p_1, l_3), (p_2, l_3), (p_3, l_1), (p_4, l_1)\}$

Domain-independent vs domain-dependent

Running Example:

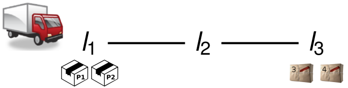


- $V = \{t, p_1, \ldots p_N\}$ with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $I = \{(t, l_1), (p_1, l_1), (p_2, l_1), (p_3, l_3), (p_4, l_3)\}$
- A = $\{load(p_i, x), unload(p_i, x), drive(x, x')\}$, where: $pre_{load(p_i,x)} = \{(t, x), (p_i, x)\}$ and $eff_{load(i,x)} = \{(p_i, t)\}$
- $G = \{(p_1, l_3), (p_2, l_3), (p_3, l_1), (p_4, l_1)\}$

- State `init` (s)
- set(A) `applicable` (s)
- State `apply` (s, a)
- bool `isGoal` (s)
- int `heuristic` (s)

Domain-independent vs domain-dependent

Running Example:



- $V = \{t, p_1, \dots p_N\}$ with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $I = \{(t, l_1), (p_1, l_1), (p_2, l_1), (p_3, l_3), (p_4, l_3)\}$
- A = $\{load(p_i, x), unload(p_i, x), drive(x, x')\}$, where: $pre_{load(p_i,x)} = \{(t, x), (p_i, x)\}$ and $eff_{load(i,x)} = \{(p_i, t)\}$
- $G = \{(p_1, l_3), (p_2, l_3), (p_3, l_1), (p_4, l_1)\}$

- State `init` (s) $\rightarrow$return I
- set(A) `applicable` (s) $\rightarrow$return $\{a \mid s \models pre(a)\}$
- State `apply` (s, a) $\rightarrow$return $s[a]$
- bool `isGoal` (s) $\rightarrow$return $s \models G$
- int `heuristic` (s) $\rightarrow$Any domain-independent planning heuristic

Exercise: Pick-up and Delivery

We have M trucks and N packages across L locations. Trucks drive around to pick and deliver the packages. We want to compute a (optimal) route. How do you design the search space? States? Actions?

Exercise: Pick-up and Delivery

We have M trucks and N packages across L locations. Trucks drive around to pick and deliver the packages. We want to compute a (optimal) route. How do you design the search space? States? Actions?

Option 1: Planning

- State: position of each package and truck
- Actions: drive-to(t, l), pick(p, t), deliver(p, t)

Exercise: Pick-up and Delivery

We have M trucks and N packages across L locations. Trucks drive around to pick and deliver the packages. We want to compute a (optimal) route. How do you design the search space? States? Actions?

Option 1: Planning

- State: position of each package and truck
- Actions: drive-to(t, l), pick(p, t), deliver(p, t)

Option 2: Package-centered

- State: position of each package and truck
- Actions: pick(p, t), deliver(p, t) (trucks move automatically)

Exercise: Pick-up and Delivery

We have M trucks and N packages across L locations. Trucks drive around to pick and deliver the packages. We want to compute a (optimal) route. How do you design the search space? States? Actions?

Option 1: Planning

- State: position of each package and truck
- Actions: drive-to(t, l), pick(p, t), deliver(p, t)

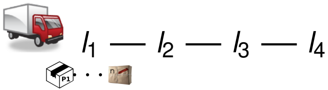
Option 2: Package-centered

- State: position of each package and truck
- Actions: pick(p, t), deliver(p, t) (trucks move automatically)

Option 3: Truck-centered

- State: truck routes, whether packages have been delivered
- Actions: drive-to(t, l)

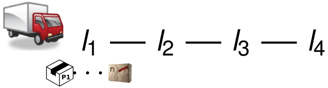
State Explosion Problem – Example



Running Example:

- $V = \{t, p_1, \ldots p_N\}$
  with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$

State Explosion Problem – Example

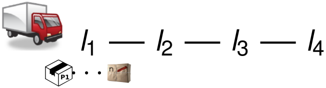


Running Example:

- $V = \{t, p_1, \dots p_N\}$
  with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$

Size of the state space (number of reachable states):

State Explosion Problem – Example

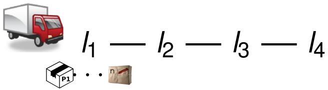


Running Example:

- $V = \{t, p_1, \ldots p_N\}$
  with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$

Size of the state space (number of reachable states): $4 \cdot 5^N$

State Explosion Problem – Example



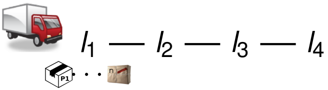
Running Example:

- $V = \{t, p_1, \ldots p_N\}$
  with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$

Size of the state space (number of reachable states): $4 \cdot 5^N$

How many different action permutations result from only loading all packages at l_1?

State Explosion Problem – Example



Running Example:

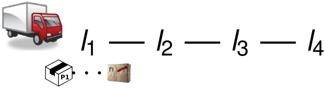
- $V = \{t, p_1, \dots p_N\}$
  with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$

Size of the state space (number of reachable states): $4 \cdot 5^N$

How many different action permutations result from only loading all packages at l_1?

$\rightarrow N!$ (2^N different states)

State Explosion Problem – Example



Running Example:

- $V = \{t, p_1, \dots p_N\}$
  with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$

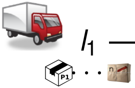
Size of the state space (number of reachable states): $4 \cdot 5^N$

How many different action permutations result from only loading all packages at l_1?

$\rightarrow N!$ (2^N different states)

Can this be avoided?

State Explosion Problem – Example



l_1 — l_2 — l_3 — l_4

Running Example:

- $V = \{t, p_1, \dots p_N\}$
  with $D_t = \{l_1, l_2, l_3, l_4\}$ and $D_{p_i} = \{t, l_1, l_2, l_3, l_4\}$.
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$

Size of the state space (number of reachable states): $4 \cdot 5^N$

How many different action permutations result from only loading all packages at l_1?

$\rightarrow N!$ (2^N different states)

Can this be avoided? What is the connection between the packages?

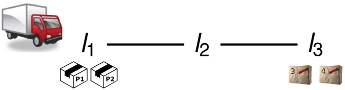
Exponential Reduction of the State Space

Domain	Reachable State Space. Right: Average over Instances Commonly Built Success Std	POR	Unfold.	Decoupled	Representation Size (in Thousands) Std	POR	Decoupled
Solvable Benchmarks: From the International Planning Competition (IPC)							
Depots	4	4	2	5	30,954.8	30,954.8	3,970.0
Driverlog	5	5	3	10	35,632.4	35,632.4	127.2
Elevators	21	17	3	41	22,652.1	22,651.1	186.7
Logistics	12	12	11	27	3,793.8	3,793.8	8.2
Miconic-STRIPS	50	45	30	145	52,728.9	52,673.1	2.4
Nomystery	11	11	7	40	29,459.3	25,581.5	10.0
Pathways	4	4	3	4	54,635.5	1,229.0	11,211.9
PSR	3	3	3	3	39.4	33.9	11.1
Rovers	5	6	4	5	98,051.6	6,534.4	4,032.9
Satellite	5	5	5	4	2,864.2	582.5	352.7
TPP	5	5	4	11	340,961.5	326,124.8	.8
Transport	28	23	11	34	4,958.6	4,958.5	173.3
Woodworking	11	20	22	16	438,638.5	226.8	9,688.9
Zenotravel	7	7	4	7	17,468.0	17,467.5	99.4
Unsolvable Benchmarks: Extended from Hoffmann and Nebel (2001)							
Nomystery	9	8	4	40	85,254.2	65,878.2	3.8
Rovers	4	4	0	4	697,778.9	302,608.9	20,924.4
$\sum$	186	181	116	398			

Agenda

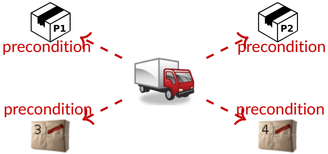
Decoupled Search – Intuition

Running Example:



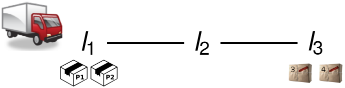
- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$, where:
  $pre_{load(p_i,x)} = \{(t, x), (p_i, x)\}$ and $eff_{load(i,x)} = \{(p_i, t)\}$,
  $pre_{unload(p_i,x)} = \{(t, x), (p_i, t)\}$ and $eff_{unload(i,x)} = \{(p_i, x)\}$.

Causal Graph: **Dependencies across (components of) state variables.**



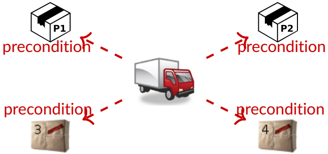
Decoupled Search – Intuition

Running Example:



- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$, where: $pre_{load(p_i,x)} = \{(t, x), (p_i, x)\}$ and $eff_{load(i,x)} = \{(p_i, t)\}$, $pre_{unload(p_i,x)} = \{(t, x), (p_i, t)\}$ and $eff_{unload(i,x)} = \{(p_i, x)\}$.

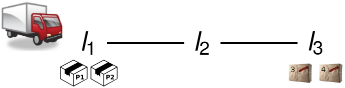
Causal Graph: **Dependencies across (components of) state variables.**



Decomposition: "Instantiate center to break the conditional dependencies".

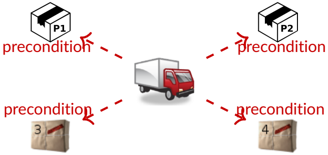
Decoupled Search – Intuition

Running Example:



- $A = \{load(p_i, x), unload(p_i, x), drive(x, x')\}$, where:
  $pre_{load(p_i,x)} = \{(t, x), (p_i, x)\}$ and $eff_{load(i,x)} = \{(p_i, t)\}$,
  $pre_{unload(p_i,x)} = \{(t, x), (p_i, t)\}$ and $eff_{unload(i,x)} = \{(p_i, x)\}$.

Causal Graph: **Dependencies across (components of) state variables.**



Decomposition: "Instantiate center to break the conditional dependencies".

Search over global actions; handle each leaf component separately.

How to Decompose a Planning Task?

Definition (Factoring). *Let Π be a planning task with variables V. A factoring $\mathcal{F}$ is a partitioning of V into non-empty subsets.*

How to Decompose a Planning Task?

Definition (Factoring). *Let Π be a planning task with variables V. A factoring $\mathcal{F}$ is a partitioning of V into non-empty subsets.*

$\rightarrow$ Each of the variable sub-sets if called a factor:

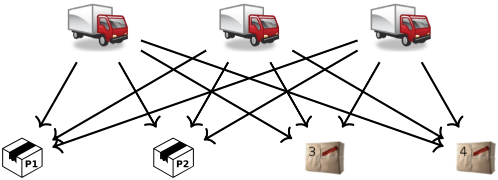
- One center factor (possibly empty)
- A set of leaf factors (typically two or more)

How to Decompose a Planning Task?

Definition (Factoring). *Let Π be a planning task with variables V. A factoring $\mathcal{F}$ is a partitioning of V into non-empty subsets.*

$\rightarrow$ Each of the variable sub-sets if called a factor:

- One center factor (possibly empty)
- A set of leaf factors (typically two or more)

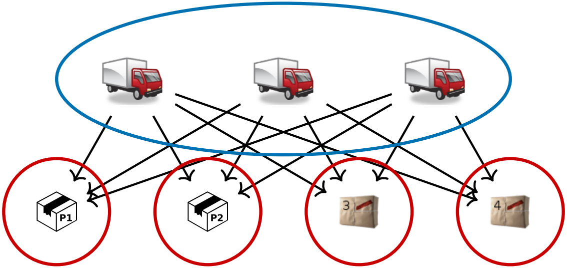


How to Decompose a Planning Task?

Definition (Factoring). *Let Π be a planning task with variables V. A factoring $\mathcal{F}$ is a partitioning of V into non-empty subsets.*

$\rightarrow$ Each of the variable sub-sets if called a factor:

- One center factor (possibly empty)
- A set of leaf factors (typically two or more)

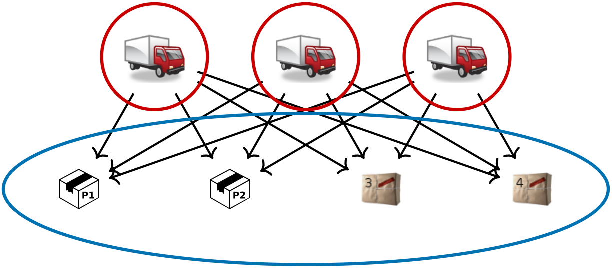


How to Decompose a Planning Task?

Definition **(Factoring).** *Let Π be a planning task with variables V. A factoring $\mathcal{F}$ is a partitioning of V into non-empty subsets.*

$\rightarrow$ Each of the variable sub-sets if called a factor:

- One center factor (possibly empty)
- A set of leaf factors (typically two or more)



Factorings – Special Cases

Definition (Interaction Graph) *The interaction graph of Π given $\mathcal{F}$ is the directed graph $\mathrm{IG}_{\Pi}(\mathcal{F})$, with vertices $\mathcal{F}$, and an arc $F \rightarrow F'$ if $F \neq F'$, and there exist $v \in F$ and $v' \in F'$, s.t. $v \rightarrow v'$ is an arc in $CG(\Pi)$.*

Factorings – Special Cases

Definition (Interaction Graph) *The interaction graph of* Π *given* $\mathcal{F}$ *is the directed graph* $\mathsf{IG}_{\Pi}(\mathcal{F})$*, with vertices* $\mathcal{F}$*, and an arc* $F \to F'$ *if* $F \neq F'$*, and there exist* $v \in F$ *and* $v' \in F'$*, s.t.* $v \to v'$ *is an arc in* $CG(\Pi)$*.*

The interaction graph is the quotient of $CG(\Pi)$ over $\mathcal{F}$.

Factorings – Special Cases

Definition (Interaction Graph) *The interaction graph of Π given $\mathcal{F}$ is the directed graph $\mathrm{IG}_\Pi(\mathcal{F})$, with vertices $\mathcal{F}$, and an arc $F \to F'$ if $F \neq F'$, and there exist $v \in F$ and $v' \in F'$, s.t. $v \to v'$ is an arc in $CG(\Pi)$.*

The interaction graph is the quotient of $CG(\Pi)$ over $\mathcal{F}$.

Definition *A factoring $\mathcal{F} = \{C\} \cup \mathcal{L}$ is a:*

- *fork factoring: all arcs in $\mathrm{IG}_\Pi(\mathcal{F})$ are of the form $C \to L$,*

Factorings – Special Cases

Definition (Interaction Graph) *The interaction graph of* Π *given* $\mathcal{F}$ *is the directed graph* $\mathrm{IG}_\Pi(\mathcal{F})$*, with vertices* $\mathcal{F}$*, and an arc* $F \rightarrow F'$ *if* $F \neq F'$*, and there exist* $v \in F$ *and* $v' \in F'$*, s.t.* $v \rightarrow v'$ *is an arc in* $CG(\Pi)$*.*

The interaction graph is the quotient of $CG(\Pi)$ over $\mathcal{F}$.

Definition *A factoring* $\mathcal{F} = \{C\} \cup \mathcal{L}$ *is a:*

- *fork factoring: all arcs in* $\mathrm{IG}_\Pi(\mathcal{F})$ *are of the form* $C \rightarrow L$*,*
- *inverted-fork factoring: all arcs in* $\mathrm{IG}_\Pi(\mathcal{F})$ *are of the form* $L \rightarrow C$*,*

Factorings – Special Cases

Definition (Interaction Graph) *The interaction graph of Π given $\mathcal{F}$ is the directed graph $\mathrm{IG}_\Pi(\mathcal{F})$, with vertices $\mathcal{F}$, and an arc $F \rightarrow F'$ if $F \neq F'$, and there exist $v \in F$ and $v' \in F'$, s.t. $v \rightarrow v'$ is an arc in $CG(\Pi)$.*

The interaction graph is the quotient of $CG(\Pi)$ over $\mathcal{F}$.

Definition *A factoring $\mathcal{F} = \{C\} \cup \mathcal{L}$ is a:*

- *fork factoring: all arcs in $\mathrm{IG}_\Pi(\mathcal{F})$ are of the form $C \rightarrow L$,*
- *inverted-fork factoring: all arcs in $\mathrm{IG}_\Pi(\mathcal{F})$ are of the form $L \rightarrow C$,*
- *strict-star factoring: all arcs in $\mathrm{IG}_\Pi(\mathcal{F})$ are incident to C.*

Factorings – Special Cases

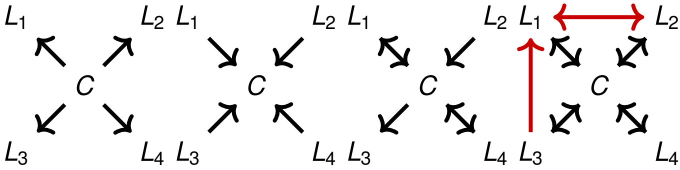
Definition (Interaction Graph) *The interaction graph of Π given $\mathcal{F}$ is the directed graph $\mathrm{IG}_\Pi(\mathcal{F})$, with vertices $\mathcal{F}$, and an arc $F \rightarrow F'$ if $F \neq F'$, and there exist $v \in F$ and $v' \in F'$, s.t. $v \rightarrow v'$ is an arc in $CG(\Pi)$.*

The interaction graph is the quotient of $CG(\Pi)$ over $\mathcal{F}$.

Definition *A factoring $\mathcal{F} = \{C\} \cup \mathcal{L}$ is a:*

- *fork factoring: all arcs in $\mathrm{IG}_\Pi(\mathcal{F})$ are of the form $C \rightarrow L$,*
- *inverted-fork factoring: all arcs in $\mathrm{IG}_\Pi(\mathcal{F})$ are of the form $L \rightarrow C$,*
- *strict-star factoring: all arcs in $\mathrm{IG}_\Pi(\mathcal{F})$ are incident to C.*

Examples:

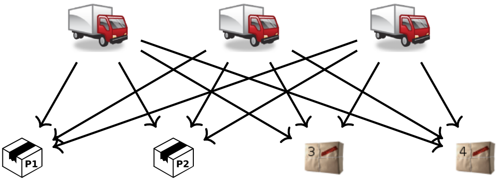


Fork Inverted-Fork Strict-Star Star

Dividing the Actions

Given a Factoring $\mathcal{F} = \{C, L_1, \ldots, L_n\}$, the set of actions is divided into $n+1$ *subsets*:

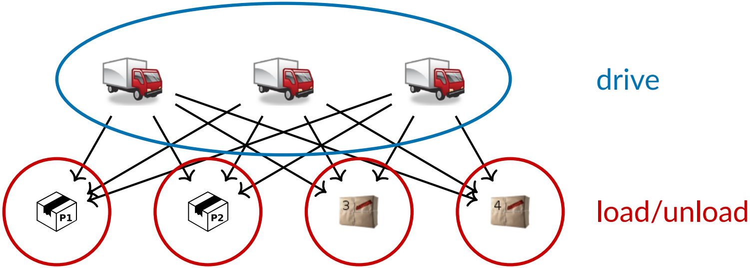
- Internal (leaf-only) Actions A^L: affect only one leaf $L \in \mathcal{L}$, $a \in A^L \Leftrightarrow V[\mathit{eff}_a] \subseteq L \wedge V[\mathit{pre}_a] \cup V[\mathit{eff}_a] \subseteq C \cup L$.
- Global Actions A^C: those that are not leaf actions, e.g.:
  - have an effect on a center variable
  - have effects and/or preconditions on two leaves



Dividing the Actions

Given a Factoring $\mathcal{F} = \{C, L_1, \ldots, L_n\}$, the set of actions is divided into $n+1$ *subsets*:

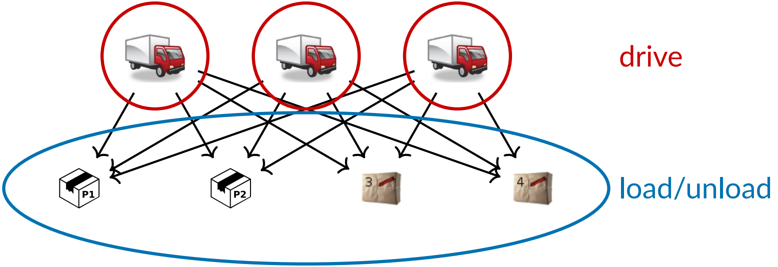
- Internal (leaf-only) Actions A^L: affect only one leaf $L \in \mathcal{L}$, $a \in A^L \Leftrightarrow V[eff_a] \subseteq L \wedge V[pre_a] \cup V[eff_a] \subseteq C \cup L$.
- Global Actions A^C: those that are not leaf actions, e.g.:
  - have an effect on a center variable
  - have effects and/or preconditions on two leaves



Dividing the Actions

Given a Factoring $\mathcal{F} = \{C, L_1, \dots, L_n\}$, the set of actions is divided into $n + 1$ *subsets*:

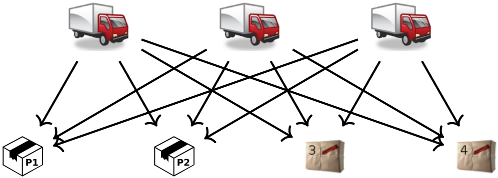
- Internal (leaf-only) Actions A^L: affect only one leaf $L \in \mathcal{L}$, $a \in A^L \Leftrightarrow V[\mathit{eff}_a] \subseteq L \wedge V[\mathit{pre}_a] \cup V[\mathit{eff}_a] \subseteq C \cup L$.
- Global Actions A^C: those that are not leaf actions, e.g.:
  - have an effect on a center variable
  - have effects and/or preconditions on two leaves



Applying a Factoring to a Planning Task

Given a Factoring $\mathcal{F} = \{C, L_1, \ldots, L_n\}$, we define

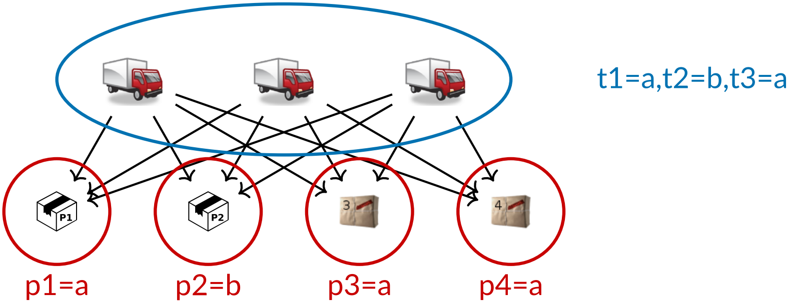
- Center States $s^C \in S^C$: complete assignment to C
- Leaf States $s^L \in S^L$: complete assignment to an $L \in \mathcal{L}$



Applying a Factoring to a Planning Task

Given a Factoring $\mathcal{F} = \{C, L_1, \dots, L_n\}$, we define

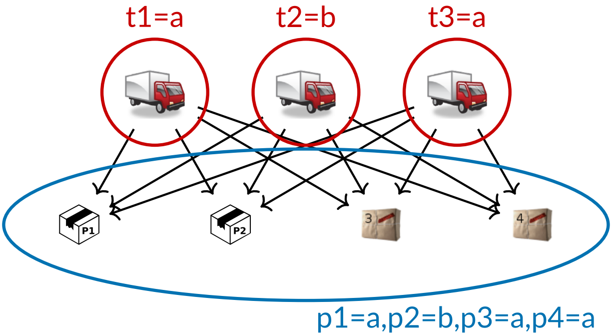
- Center States $s^C \in S^C$: complete assignment to C
- Leaf States $s^L \in S^L$: complete assignment to an $L \in \mathcal{L}$



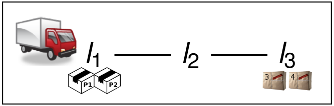
Applying a Factoring to a Planning Task

Given a Factoring $\mathcal{F} = \{C, L_1, \dots, L_n\}$, we define

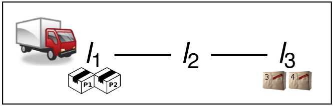
- Center States $s^C \in S^C$: complete assignment to C
- Leaf States $s^L \in S^L$: complete assignment to an $L \in \mathcal{L}$



Decoupled Search – Intuition II



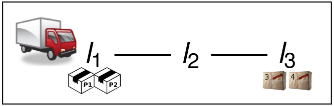
Decoupled Search – Intuition II



Center path:

$$l_1 \xrightarrow{\text{drive}(l_1, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_3)} l_3 \xrightarrow{\text{drive}(l_3, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_1)} l_1$$

Decoupled Search – Intuition II



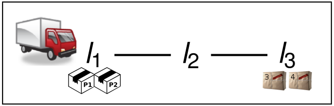
Center path:

$$l_1 \xrightarrow{\text{drive}(l_1, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_3)} l_3 \xrightarrow{\text{drive}(l_3, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_1)} l_1$$

:

(a) $l_1 \xrightarrow{\text{load}(p_1, l_1)} t \xrightarrow{\text{unload}(p_1, l_2)} l_2$

Decoupled Search – Intuition II



Center path:

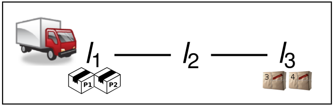
$$l_1 \xrightarrow{\text{drive}(l_1, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_3)} l_3 \xrightarrow{\text{drive}(l_3, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_1)} l_1$$

:

(a) $l_1 \xrightarrow{\text{load}(p_1, l_1)} t \xrightarrow{\text{unload}(p_1, l_2)} l_2$

(b) $l_1 \xrightarrow{\text{load}(p_1, l_1)} t \quad \xrightarrow{\text{unload}(p_1, l_3)} l_3$

Decoupled Search – Intuition II



Center path:

$l_1 \xrightarrow{drive(l_1, l_2)} l_2 \xrightarrow{drive(l_2, l_3)} l_3 \xrightarrow{drive(l_3, l_2)} l_2 \xrightarrow{drive(l_2, l_1)} l_1$

:

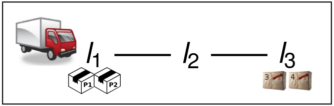
(a) $l_1 \xrightarrow{load(p_1, l_1)} t \xrightarrow{unload(p_1, l_2)} l_2$

(b) $l_1 \xrightarrow{load(p_1, l_1)} t \qquad \xrightarrow{unload(p_1, l_3)} l_3$

:

(a) $l_3 \xrightarrow{load(p_3, l_3)} t \xrightarrow{unload(p_3, l_2)} l_2$

Decoupled Search – Intuition II



Center path:

$$l_1 \xrightarrow{\text{drive}(l_1, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_3)} l_3 \xrightarrow{\text{drive}(l_3, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_1)} l_1$$

:

(a) $l_1 \xrightarrow{\text{load}(p_1, l_1)} t \xrightarrow{\text{unload}(p_1, l_2)} l_2$

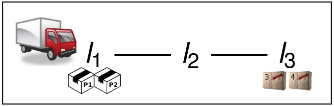
(b) $l_1 \xrightarrow{\text{load}(p_1, l_1)} t \qquad \xrightarrow{\text{unload}(p_1, l_3)} l_3$

:

(a) $l_3 \xrightarrow{\text{load}(p_3, l_3)} t \xrightarrow{\text{unload}(p_3, l_2)} l_2$

(b) $l_3 \xrightarrow{\text{load}(p_3, l_3)} t \qquad \xrightarrow{\text{unload}(p_3, l_1)} l$

Decoupled Search – Intuition II



Center path:

$$l_1 \xrightarrow{\text{drive}(l_1, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_3)} l_3 \xrightarrow{\text{drive}(l_3, l_2)} l_2 \xrightarrow{\text{drive}(l_2, l_1)} l_1$$

:

(a) $l_1 \xrightarrow{\text{load}(p_1, l_1)} t \xrightarrow{\text{unload}(p_1, l_2)} l_2$

(b) $l_1 \xrightarrow{\text{load}(p_1, l_1)} t \qquad \xrightarrow{\text{unload}(p_1, l_3)} l_3$

:

(a) $l_3 \xrightarrow{\text{load}(p_3, l_3)} t \xrightarrow{\text{unload}(p_3, l_2)} l_2$

(b) $l_3 \xrightarrow{\text{load}(p_3, l_3)} t \qquad \xrightarrow{\text{unload}(p_3, l_1)} l$

We can choose (a) or (b) for each of p_1 and p_3 independently

$\Longrightarrow$ **Maintain the compliant paths for each leaf separately.**

Decoupled Search – Intuition III

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

Decoupled State $s^{\mathcal{F}}$

Decoupled Search – Intuition III

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

Decoupled State $s^{\mathcal{F}}$

$tl_3, p_1 t, p_2 l_3, p_3 l_3, p_4 l_3$ 7
$tl_3, p_1 l_3, p_2 l_3, p_3 t, p_4 t$ 6
$tl_3, p_1 t, p_2 t, p_3 t, p_4 l_3$ 5
$tl_3, p_1 l_3, p_2 t, p_3 t, p_4 l_3$ 6
$tl_3, p_1 t, p_2 l_3, p_3 t, p_4 t$ 5
...

Hypercube $[s^{\mathcal{F}}]$ (144 member states!

Decoupled Search – Intuition III

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

Decoupled State $s^{\mathcal{F}}$

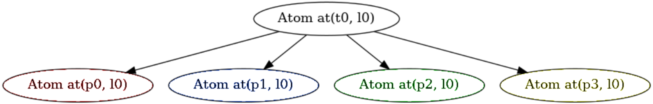
$$tl_3, p_1t, p_2l_3, p_3l_3, p_4l_3\ 7$$
$$tl_3, p_1l_3, p_2l_3, p_3t, p_4t\ 6$$
$$tl_3, p_1t, p_2t, p_3t, p_4l_3\ 5$$
$$tl_3, p_1l_3, p_2t, p_3t, p_4l_3\ 6$$
$$tl_3, p_1t, p_2l_3, p_3t, p_4t\ 5$$
$$\dots$$

Hypercube $[s^{\mathcal{F}}]$ (144 member states!

Every member state annotated with its price in $s^{\mathcal{F}}$.

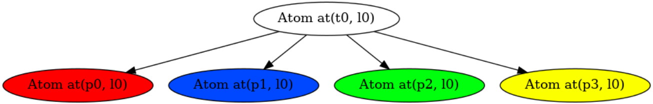
Hypercube dimensions = Leaves; Axis values = Leaf States.

Examples



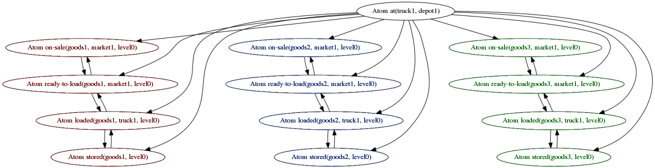
Logistics

Examples



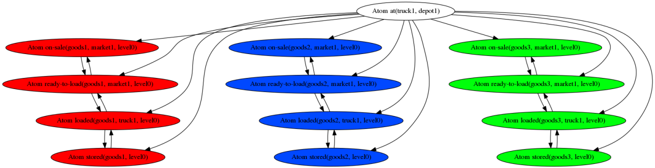
Logistics

Examples



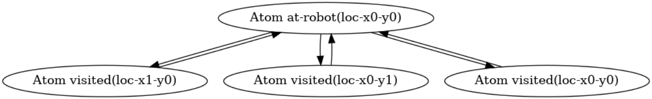
TPP

Examples



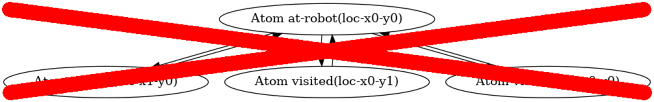
TPP

Examples



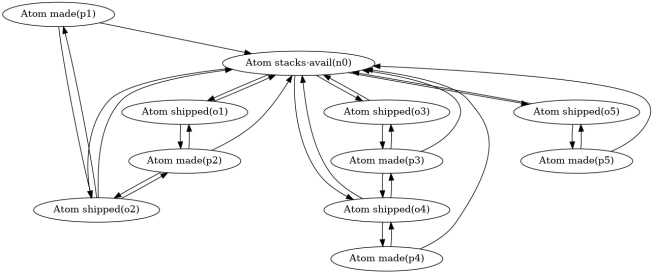
Visit-All

Examples



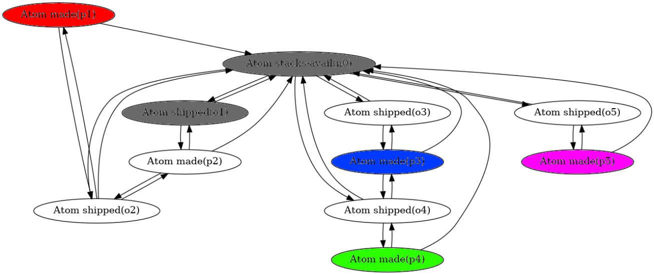
Visit-All

Examples



Openstacks

Examples

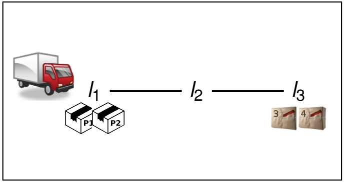


Openstacks

Initial Decoupled State

- Center variables get their value from the explicit state

Example:



Explicit initial state

	l_1	t	l_2	l_3		l_1	t	l_2	l_3
p_1	∞	∞	∞	∞	p_2	∞	∞	∞	∞

$t = l_1$

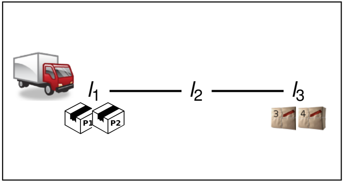
	l_1	t	l_2	l_3		l_1	t	l_2	l_3
p_3	∞	∞	∞	∞	p_4	∞	∞	∞	∞

Decoupled initial state

Initial Decoupled State

- Center variables get their value from the explicit state
- Set price of 0 for the leaf state that holds in the initial state

Example:



Explicit initial state

	l_1	t	l_2	l_3
p_1	0	∞	∞	∞

	l_1	t	l_2	l_3
p_2	0	∞	∞	∞

$t = l_1$

	l_1	t	l_2	l_3
p_3	∞	∞	∞	0

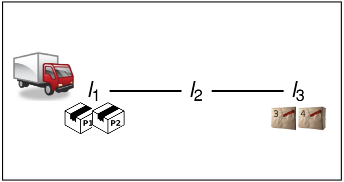
	l_1	t	l_2	l_3
p_4	∞	∞	∞	0

Decoupled initial state

Initial Decoupled State

- Center variables get their value from the explicit state
- Set price of 0 for the leaf state that holds in the initial state
- Saturate the leaves: reachability analysis (Dijkstra) on each leaf using leaf-only actions whose center preconditions hold

Example:



Explicit initial state

	l_1	t	l_2	l_3
p_1	0	∞	∞	∞

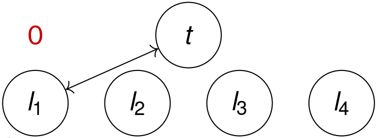
	l_1	t	l_2	l_3
p_2	0	∞	∞	∞

$t = l_1$

	l_1	t	l_2	l_3
p_3	∞	∞	∞	0

	l_1	t	l_2	l_3
p_4	∞	∞	∞	0

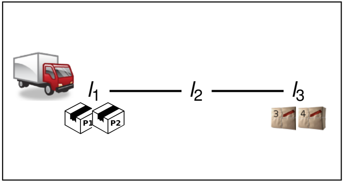
Decoupled initial state



Initial Decoupled State

- Center variables get their value from the explicit state
- Set price of 0 for the leaf state that holds in the initial state
- Saturate the leaves: reachability analysis (Dijkstra) on each leaf using leaf-only actions whose center preconditions hold

Example:



Explicit initial state

	l_1	t	l_2	l_3
p_1	0	1	∞	∞

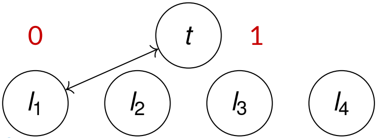
	l_1	t	l_2	l_3
p_2	0	1	∞	∞

$t = l_1$

	l_1	t	l_2	l_3
p_3	∞	∞	∞	0

	l_1	t	l_2	l_3
p_4	∞	∞	∞	0

Decoupled initial state



High-level Idea

Decoupled state space is a transition system (TS) where:

High-level Idea

Decoupled state space is a transition system (TS) where:

- States: decoupled states (**saturated** w.r.t. reachable leaf states),

High-level Idea

Decoupled state space is a transition system (TS) where:

- States: decoupled states (**saturated** w.r.t. reachable leaf states),
- Transitions: induced only by **center actions**, **saturate** successor,

High-level Idea

Decoupled state space is a transition system (TS) where:

- States: decoupled states (**saturated** w.r.t. reachable leaf states),
- Transitions: induced only by **center actions**, **saturate** successor,
- Initial state: **saturated** explicit initial state,

High-level Idea

Decoupled state space is a transition system (TS) where:

- States: decoupled states (**saturated** w.r.t. reachable leaf states),
- Transitions: induced only by **center actions**, **saturate** successor,
- Initial state: **saturated** explicit initial state,
- Goal states: all goals are **reached** in decoupled state (goal member state).

High-level Idea

Decoupled state space is a transition system (TS) where:

- States: decoupled states (**saturated** w.r.t. reachable leaf states),
- Transitions: induced only by **center actions**, **saturate** successor,
- Initial state: **saturated** explicit initial state,
- Goal states: all goals are **reached** in decoupled state (goal member state).

$\rightarrow$ **Run** – in principle – **any (heuristic) search algorithm** on this TS.

High-level Idea

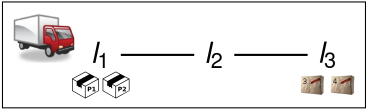
Decoupled state space is a transition system (TS) where:

- States: decoupled states (**saturated** w.r.t. reachable leaf states),
- Transitions: induced only by **center actions**, **saturate** successor,
- Initial state: **saturated** explicit initial state,
- Goal states: all goals are **reached** in decoupled state (goal member state).

→ **Run** – in principle – **any (heuristic) search algorithm** on this TS.

(Optimal planning: minor modifications required)

Decoupled Search – Fork (Truck-centered)



	l_1	t	l_2	l_3
p_1	0	1	∞	∞

	l_1	t	l_2	l_3
p_2	0	1	∞	∞

$t = l_1$

	l_1	t	l_2	l_3
p_3	∞	∞	∞	0

	l_1	t	l_2	l_3
p_4	∞	∞	∞	0

drive(l_1, l_2)

	l_1	t	l_2	l_3
p_1	0	1	**2**	∞

	l_1	t	l_2	l_3
p_2	0	1	**2**	∞

$t = l_2$

	l_1	t	l_2	l_3
p_3	∞	∞	∞	0

	l_1	t	l_2	l_3
p_4	∞	∞	∞	0

drive(l_2, l_1)

	l_1	t	l_2	l_3
p_1	0	1	**2**	∞

	l_1	t	l_2	l_3
p_2	0	1	**2**	∞

$t = l_1$

	l_1	t	l_2	l_3
p_3	∞	∞	∞	0

	l_1	t	l_2	l_3
p_4	∞	∞	∞	0

drive(l_2, l_3)

	l_1	t	l_2	l_3
p_1	0	1	2	**2**

	l_1	t	l_2	l_3
p_2	0	1	2	**2**

$t = l_3$

	l_1	t	l_2	l_3
p_3	∞	**1**	∞	0

	l_1	t	l_2	l_3
p_4	∞	**1**	∞	0

Solution Reconstruction

For every member state $s \in [s^{\mathcal{F}}]$ of a decoupled state $s^{\mathcal{F}}$, we can construct a **global plan** reaching s from the initial state I.

Solution Reconstruction

For every member state $s \in [s^{\mathcal{F}}]$ of a decoupled state $s^{\mathcal{F}}$, we can construct a **global plan** reaching s from the initial state I.

Approach:

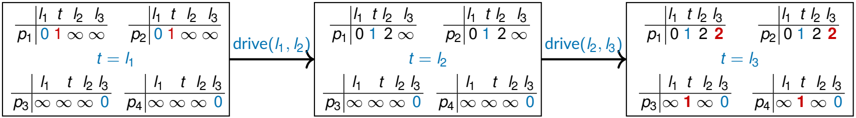
1. Extract global plan following parent pointers

Solution Reconstruction

For every member state $s \in [s^{\mathcal{F}}]$ of a decoupled state $s^{\mathcal{F}}$, we can construct a **global plan** reaching s from the initial state I.

Approach:

1. Extract global plan following parent pointers
2. For every step in the global plan, each leaf adds actions (by independence, actions of different leaves can be applied in any order)



Solution Reconstruction

For every member state $s \in [s^{\mathcal{F}}]$ of a decoupled state $s^{\mathcal{F}}$, we can construct a **global plan** reaching s from the initial state I.

Approach:

1. Extract global plan following parent pointers
2. For every step in the global plan, each leaf adds actions (by independence, actions of different leaves can be applied in any order)

Decoupled state 1 ($t = l_1$)

	l_1	t	l_2	l_3
p_1	0	1	∞	∞
p_2	0	1	∞	∞
p_3	∞	∞	∞	0
p_4	∞	∞	∞	0

$\xrightarrow{\text{drive}(l_1, l_2)}$

Decoupled state 2 ($t = l_2$)

	l_1	t	l_2	l_3
p_1	0	1	2	∞
p_2	0	1	2	∞
p_3	∞	∞	∞	0
p_4	∞	∞	∞	0

$\xrightarrow{\text{drive}(l_2, l_3)}$

Decoupled state 3 ($t = l_3$)

	l_1	t	l_2	l_3
p_1	0	1	2	2
p_2	0	1	2	2
p_3	∞	1	∞	0
p_4	∞	1	∞	0

unload(p_1, l_3)

unload(p_2, l_3)

load(p_3, l_3)

load(p_4, l_3)

Solution Reconstruction

For every member state $s \in [s^{\mathcal{F}}]$ of a decoupled state $s^{\mathcal{F}}$, we can construct a **global plan** reaching s from the initial state I.

Approach:

1. Extract global plan following parent pointers
2. For every step in the global plan, each leaf adds actions (by independence, actions of different leaves can be applied in any order)

$t = l_1$

	l_1	t	l_2	l_3
p_1	0	1	∞	∞
p_2	0	1	∞	∞
p_3	∞	∞	∞	0
p_4	∞	∞	∞	0

$\xrightarrow{drive(l_1, l_2)}$

$t = l_2$

	l_1	t	l_2	l_3
p_1	0	1	2	∞
p_2	0	1	2	∞
p_3	∞	∞	∞	0
p_4	∞	∞	∞	0

$\xrightarrow{drive(l_2, l_3)}$

$t = l_3$

	l_1	t	l_2	l_3
p_1	0	1	2	**2**
p_2	0	1	2	**2**
p_3	∞	**1**	∞	0
p_4	∞	**1**	∞	0

load(p_1, l_1)

load(p_2, l_1)

unload(p_1, l_3)

unload(p_2, l_3)

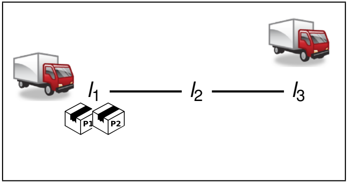
load(p_3, l_3)

load(p_4, l_3)

Initial Decoupled State – Inv Fork (Package–centered)

- Center variables get their value from the explicit state

Example:



Explicit initial state

	l_1	l_2	l_3
t_1	∞	∞	∞

	l_1	l_2	l_3
t_2	∞	∞	∞

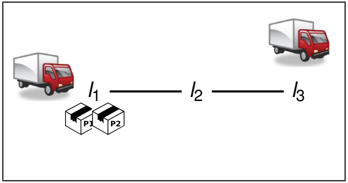
$p_1 = l_1, p_2 = l_1$

Decoupled initial state

Initial Decoupled State – Inv Fork (Package-centered)

- Center variables get their value from the explicit state
- Set price of 0 for the leaf state that holds in the initial state

Example:



Explicit initial state

$$\begin{array}{c|ccc} & l_1 & l_2 & l_3 \\ \hline t_1 & 0 & \infty & \infty \end{array} \qquad \begin{array}{c|ccc} & l_1 & l_2 & l_3 \\ \hline t_2 & \infty & \infty & 0 \end{array}$$

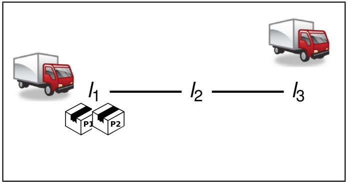
$$p_1 = l_1, p_2 = l_1$$

Decoupled initial state

Initial Decoupled State – Inv Fork (Package–centered)

- Center variables get their value from the explicit state
- Set price of 0 for the leaf state that holds in the initial state
- Saturate the leaves: reachability analysis (Dijkstra) on each leaf using leaf-only actions whose center preconditions hold

Example:



Explicit initial state

	l_1	l_2	l_3
t_1	0	1	

	l_1	l_2	l_3
t_2		1	0

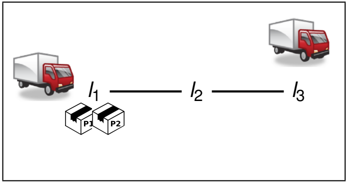
$p_1 = l_1, p_2 = l_1$

Decoupled initial state

Initial Decoupled State – Inv Fork (Package-centered)

- Center variables get their value from the explicit state
- Set price of 0 for the leaf state that holds in the initial state
- Saturate the leaves: reachability analysis (Dijkstra) on each leaf using leaf-only actions whose center preconditions hold

Example:



Explicit initial state

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	1	0

$p_1 = l_1, p_2 = l_1$

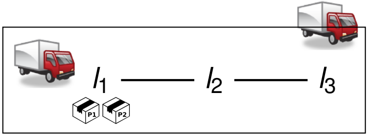
Decoupled initial state

Decoupled Search – InvFork (Package-centered)

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	1	0

$p_1 = l_1, p_2 = l_1$



$\text{load}(p_1, t_2, l_1)$

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	1	0

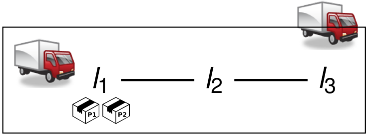
$p_1 = t_2, p_2 = l_1$

Decoupled Search – InvFork (Package-centered)

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	1	0

$p_1 = l_1, p_2 = l_1$



$\xrightarrow{\text{load}(p_1, t_2, l_1)}$

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	1	0

$p_1 = t_2, p_2 = l_1$

Decoupled Search – InvFork (Package-centered)

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	1	0

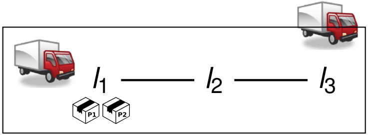
$p_1 = l_1, p_2 = l_1$

$\text{load}(p_1, t_2, l_1)$

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	∞	∞

$p_1 = t_2, p_2 = l_1$



Decoupled Search – InvFork (Package-centered)

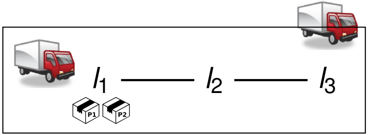
$$\begin{array}{c|ccc} & l_1 & l_2 & l_3 \\ \hline t_1 & 0 & 1 & 2 \end{array} \qquad \begin{array}{c|ccc} & l_1 & l_2 & l_3 \\ \hline t_2 & 2 & 1 & 0 \end{array}$$

$p_1 = l_1, p_2 = l_1$

$\xrightarrow{\text{load}(p_1, t_2, l_1)}$

$$\begin{array}{c|ccc} & l_1 & l_2 & l_3 \\ \hline t_1 & 0 & 1 & 2 \end{array} \qquad \begin{array}{c|ccc} & l_1 & l_2 & l_3 \\ \hline t_2 & 2 & 3 & 4 \end{array}$$

$p_1 = t_2, p_2 = l_1$

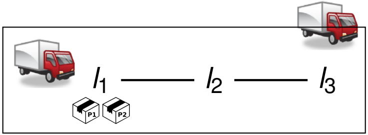


Decoupled Search – InvFork (Package-centered)

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	2	1	0

$p_1 = l_1, p_2 = l_1$



load(p_1, t_2, l_1)cost=3

	l_1	l_2	l_3
t_1	0	1	2

	l_1	l_2	l_3
t_2	0	1	2

$p_1 = t_2, p_2 = l_1$

Ensuring optimallity

All search algorithms can directly be applied in the decoupled search space

- Complete
- Optimal

Minor technical detail: in optimal planning, stop when min_f open $\geq$ current solution cost

Ensuring optimallity

All search algorithms can directly be applied in the decoupled search space

- Complete
- Optimal

Minor technical detail: in optimal planning, stop when min_f open $\geq$ current solution cost

A^* cannot stopped when expanding a goal decoupled state

Ensuring optimallity

All search algorithms can directly be applied in the decoupled search space

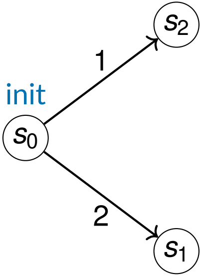
- Complete
- Optimal

Minor technical detail: in optimal planning, stop when min_f open $\geq$ current solution cost

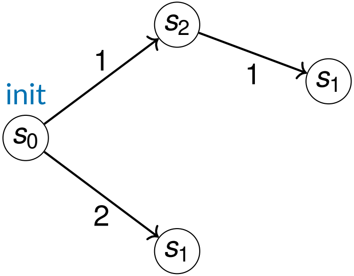
A^* cannot stopped when expanding a goal decoupled state

Reason: decoupled states contain multiple states, so the state with minimum f and the goal state could be two different ones

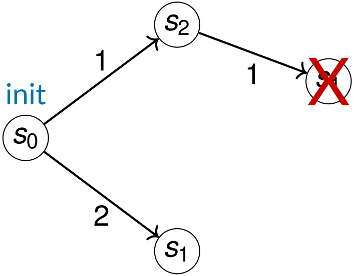
How to Eliminate Previously Seen States?



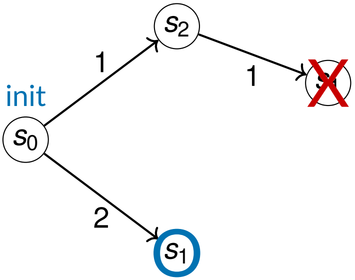
How to Eliminate Previously Seen States?



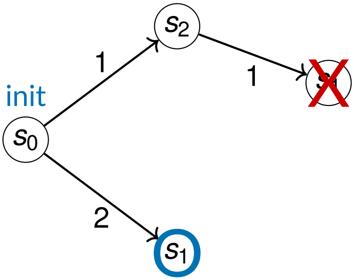
How to Eliminate Previously Seen States?

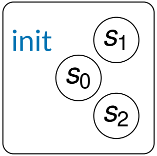


How to Eliminate Previously Seen States?

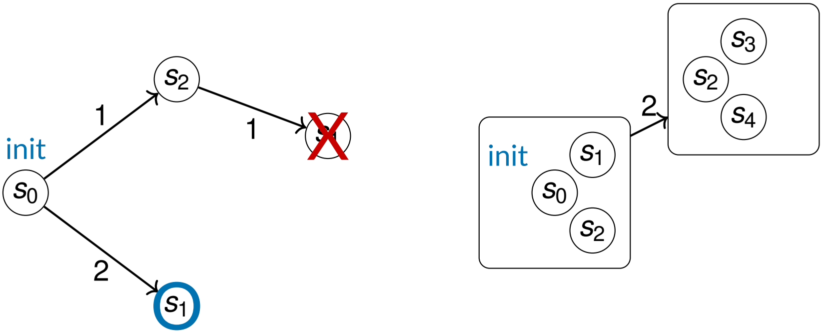


How to Eliminate Previously Seen States?

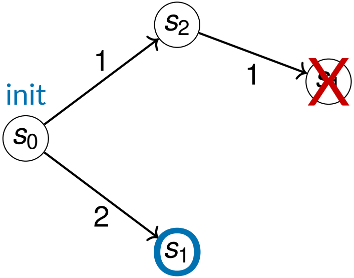


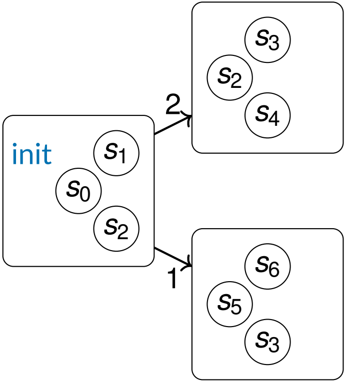


How to Eliminate Previously Seen States?

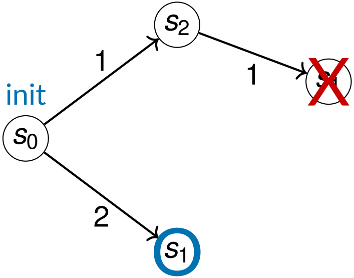


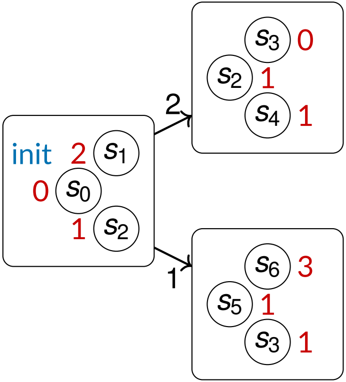
How to Eliminate Previously Seen States?



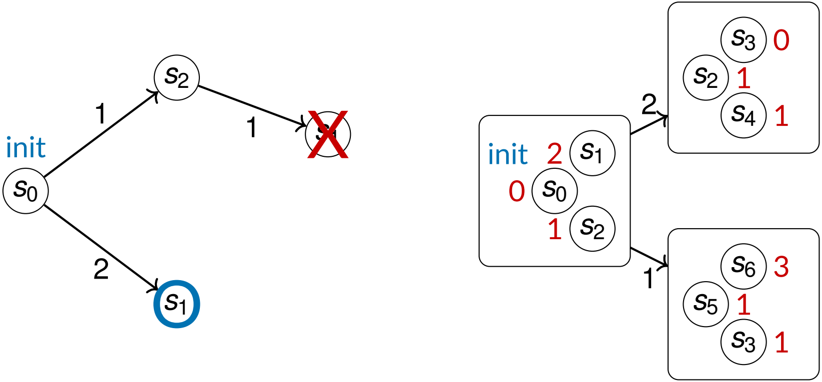


How to Eliminate Previously Seen States?





How to Eliminate Previously Seen States?



How powerful is exact duplicate checking for decoupled search?

Dominance Pruning for Decoupled States

Definition Dominance Pruning. *A decoupled state* $s^{\mathcal{F}}$ *dominates another state* $t^{\mathcal{F}}$*, denoted* $t^{\mathcal{F}} \preceq s^{\mathcal{F}}$*, if the center state is the same, i.e.* $s^{C}(s^{\mathcal{F}}) = s^{C}(t^{\mathcal{F}})$*, and* ***for all leaf states*** s^{L}*:*
$prices(s^{\mathcal{F}})[s^{L}] \leq prices(t^{\mathcal{F}})[s^{L}]$.

Dominance Pruning for Decoupled States

Definition Dominance Pruning. *A decoupled state $s^{\mathcal{F}}$ dominates another state $t^{\mathcal{F}}$, denoted $t^{\mathcal{F}} \preceq s^{\mathcal{F}}$, if the center state is the same, i.e. $s^C(s^{\mathcal{F}}) = s^C(t^{\mathcal{F}})$, and **for all leaf states** s^L:*
$prices(s^{\mathcal{F}})[s^L] \leq prices(t^{\mathcal{F}})[s^L]$.

1. Dominance pruning can be exponentially stronger than exact duplicate checking.
2. Optimality is preserved when comparing new state $t^{\mathcal{F}}$ only to other states with lower g-value (A^*).

Dominance Pruning for Decoupled States

Definition Dominance Pruning. *A decoupled state $s^{\mathcal{F}}$ dominates another state $t^{\mathcal{F}}$, denoted $t^{\mathcal{F}} \preceq s^{\mathcal{F}}$, if the center state is the same, i.e. $s^{C}(s^{\mathcal{F}}) = s^{C}(t^{\mathcal{F}})$, and **for all leaf states** s^{L}: $prices(s^{\mathcal{F}})[s^{L}] \leq prices(t^{\mathcal{F}})[s^{L}]$.*

1. Dominance pruning can be exponentially stronger than exact duplicate checking.
2. Optimality is preserved when comparing new state $t^{\mathcal{F}}$ only to other states with lower g-value (A^*).

Practical Issues?

Dominance Pruning for Decoupled States

Definition Dominance Pruning. *A decoupled state $s^{\mathcal{F}}$ dominates another state $t^{\mathcal{F}}$, denoted $t^{\mathcal{F}} \preceq s^{\mathcal{F}}$, if the center state is the same, i.e. $s^C(s^{\mathcal{F}}) = s^C(t^{\mathcal{F}})$, and* ***for all leaf states*** *s^L:*
$prices(s^{\mathcal{F}})[s^L] \leq prices(t^{\mathcal{F}})[s^L]$.

1. Dominance pruning can be exponentially stronger than exact duplicate checking.
2. Optimality is preserved when comparing new state $t^{\mathcal{F}}$ only to other states with lower g-value (A^*).

Practical Issues?

Exact duplicate checking is extremely efficient $\rightarrow$ hashing.

Dominance Pruning for Decoupled States

Definition Dominance Pruning. *A decoupled state $s^{\mathcal{F}}$ dominates another state $t^{\mathcal{F}}$, denoted $t^{\mathcal{F}} \preceq s^{\mathcal{F}}$, if the center state is the same, i.e. $s^{C}(s^{\mathcal{F}}) = s^{C}(t^{\mathcal{F}})$, and **for all leaf states** s^{L}: $prices(s^{\mathcal{F}})[s^{L}] \leq prices(t^{\mathcal{F}})[s^{L}]$.*

1. Dominance pruning can be exponentially stronger than exact duplicate checking.
2. Optimality is preserved when comparing new state $t^{\mathcal{F}}$ only to other states with lower g-value (A^*).

Practical Issues?

Exact duplicate checking is extremely efficient $\rightarrow$ hashing.

$\rightarrow$ For dominance pruning, we need to compare a new decoupled state to all previously seen states with the same center state.

Heuristics for Decoupled States

Heuristic: $h(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$

Heuristics for Decoupled States

Heuristic: $h(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

Decoupled State $s^{\mathcal{F}}$

$tl_3, p_1 t, p_2 l_3, p_3 l_3, p_4 l_3$ 7
$tl_3, p_1 l_3, p_2 l_3, p_3 t, p_4 t$ 6
$tl_3, p_1 t, p_2 t, p_3 t, p_4 l_3$ 5
$tl_3, p_1 l_3, p_2 t, p_3 t, p_4 l_3$ 6
$tl_3, p_1 t, p_2 l_3, p_3 t, p_4 t$ 5
...

Hypercube $[s^{\mathcal{F}}]$ (144 member states!)

Heuristics for Decoupled States

Heuristic: $h(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

Decoupled State $s^{\mathcal{F}}$

$tl_3, p_1 t, p_2 l_3, p_3 l_3, p_4 l_3$ 7
$tl_3, p_1 l_3, p_2 l_3, p_3 t, p_4 t$ 6
$tl_3, p_1 t, p_2 t, p_3 t, p_4 l_3$ 5
$tl_3, p_1 l_3, p_2 t, p_3 t, p_4 l_3$ 6
$tl_3, p_1 t, p_2 l_3, p_3 t, p_4 t$ 5
...

Hypercube $[s^{\mathcal{F}}]$ (144 member states!)

Definition (Decoupled Heuristic). $h : S^{\mathcal{F}} \mapsto \mathbb{R} \cup \{\infty\}$

Star-perfect heuristic: $h^*_{\mathcal{F}}(s^{\mathcal{F}}) := \min_{s \in [s^{\mathcal{F}}]} prices(s^{\mathcal{F}}, s) + h^*(s)$

$h_{\mathcal{F}}$ is star-admissible if $h_{\mathcal{F}} \leq h^*_{\mathcal{F}}$

Heuristics for Decoupled States

Heuristic: $h(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

Decoupled State $s^{\mathcal{F}}$

$tl_3, p_1 t, p_2 l_3, p_3 l_3, p_4 l_3$ 7
$tl_3, p_1 l_3, p_2 l_3, p_3 t, p_4 t$ 6
$tl_3, p_1 t, p_2 t, p_3 t, p_4 l_3$ 5
$tl_3, p_1 l_3, p_2 t, p_3 t, p_4 l_3$ 6
$tl_3, p_1 t, p_2 l_3, p_3 t, p_4 t$ 5
$\dots$

Hypercube $[s^{\mathcal{F}}]$ (144 member states!)

Definition (Decoupled Heuristic). $h : S^{\mathcal{F}} \mapsto \mathbb{R} \cup \{\infty\}$

Star-perfect heuristic: $h^*_{\mathcal{F}}(s^{\mathcal{F}}) := \min_{s \in [s^{\mathcal{F}}]} prices(s^{\mathcal{F}}, s) + h^*(s)$

$h_{\mathcal{F}}$ is star-admissible if $h_{\mathcal{F}} \leq h^*_{\mathcal{F}}$

$\rightarrow$ Pricing function is taken into account.

Planning Heuristics I: Naive Method

Given any planning heuristic $h_\Pi(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$,
How to use $h_\Pi(s)$ to compute $h_\mathcal{F}(s^\mathcal{F})$?

Planning Heuristics I: Naive Method

Given any planning heuristic $h_\Pi(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$,
How to use $h_\Pi(s)$ to compute $h_\mathcal{F}(s^\mathcal{F})$?

$$\min_{s \in [s^\mathcal{F}]} prices(s^\mathcal{F}, s) + h(s)$$

Planning Heuristics I: Naive Method

Given any planning heuristic $h_\Pi(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$,
How to use $h_\Pi(s)$ to compute $h_\mathcal{F}(s^\mathcal{F})$?

$$\min_{s \in [s^\mathcal{F}]} prices(s^\mathcal{F}, s) + h(s)$$

Pros:

- As informative as it gets (makes the most out of h)

Planning Heuristics I: Naive Method

Given any planning heuristic $h_\Pi(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$,
How to use $h_\Pi(s)$ to compute $h_\mathcal{F}(s^\mathcal{F})$?

$$\min_{s \in [s^\mathcal{F}]} prices(s^\mathcal{F}, s) + h(s)$$

Pros:

- As informative as it gets (makes the most out of h)

Cons:

- Decompresses the decoupled state, losing all the gains that decoupled search has

Planning Heuristics I: Naive Method

Given any planning heuristic $h_\Pi(s) : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$,
How to use $h_\Pi(s)$ to compute $h_\mathcal{F}(s^\mathcal{F})$?

$$\min_{s \in [s^\mathcal{F}]} prices(s^\mathcal{F}, s) + h(s)$$

Pros:

- As informative as it gets (makes the most out of h)

Cons:

- Decompresses the decoupled state, losing all the gains that decoupled search has

$\rightarrow$ So, we need better ways to compute or approximate this

Planning Heuristics II: Buy-leaves compilation

Buy-Leaves compilation: compute $h_{\Pi'}(s)$ on a different planning task Π', which is equal to Π but with additional actions:

- buy-p1-l1: eff: $p_1 = l_1$, cost=0,
- buy-p1-t: eff: $p_1 = t$, cost=1,
- buy-p1-l2: eff: $p_1 = l_2$, cost=2,
- ...

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

plus additional machinery so that exactly one leaf state is bought per leaf

Planning Heuristics II: Buy-leaves compilation

Buy-Leaves compilation: compute $h_{\Pi'}(s)$ on a different planning task Π', which is equal to Π but with additional actions:

- buy-p1-l1: eff: $p_1 = l_1$, cost=0,
- buy-p1-t: eff: $p_1 = t$, cost=1,
- buy-p1-l2: eff: $p_1 = l_2$, cost=2,
- ...

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

plus additional machinery so that exactly one leaf state is bought per leaf

Pros:

- Limited overhead (the new task is not much bigger)
- Can use any admissible heuristic (e.g., LM-cut)

Planning Heuristics II: Buy-leaves compilation

Buy-Leaves compilation: compute $h_{\Pi'}(s)$ on a different planning task Π', which is equal to Π but with additional actions:

- buy-p1-l1: eff: $p_1 = l_1$, cost=0,
- buy-p1-t: eff: $p_1 = t$, cost=1,
- buy-p1-l2: eff: $p_1 = l_2$, cost=2,
- ...

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

plus additional machinery so that exactly one leaf state is bought per leaf

Pros:

- Limited overhead (the new task is not much bigger)
- Can use any admissible heuristic (e.g., LM-cut)

Cons:

- Buy-actions change per state, so h cannot be precomputed (huge overhead for abstraction heuristics, PDBs, etc.)

Planning Heuristics II: Buy-leaves compilation

Buy-Leaves compilation: compute $h_{\Pi'}(s)$ on a different planning task Π', which is equal to Π but with additional actions:

- buy-p1-l1: eff: $p_1 = l_1$, cost=0,
- buy-p1-t: eff: $p_1 = t$, cost=1,
- buy-p1-l2: eff: $p_1 = l_2$, cost=2,
- ...

	l_1	t	l_2	l_3	l_4
p_1	0	1	2	2	∞

	l_1	t	l_2	l_3	l_4
p_2	0	1	∞	2	2

$t = l_3$

	l_1	t	l_2	l_3	l_4
p_3	∞	1	∞	2	0

	l_1	t	l_2	l_3	l_4
p_4	∞	1	∞	2	0

plus additional machinery so that exactly one leaf state is bought per leaf

Pros:

- Limited overhead (the new task is not much bigger)
- Can use any admissible heuristic (e.g., LM-cut)

Cons:

- Buy-actions change per state, so h cannot be precomputed (huge overhead for abstraction heuristics, PDBs, etc.)
- Heuristic may approximate buying leaf states

Heuristics Part III: Precomputed heuristics

Given a precomputed abstraction heuristic (PDB, ADD, M&S) can we compute $h_{\mathcal{F}}(s^{\mathcal{F}})$ efficiently?

- Single PDBs: yes

Heuristics Part III: Precomputed heuristics

Given a precomputed abstraction heuristic (PDB, ADD, M&S) can we compute $h_{\mathcal{F}}(s^{\mathcal{F}})$ efficiently?

- Single PDBs: yes
- ADDs/M&S: not in general (NP-complete), but yes for compliant data-structures (Gnad *et al.* (2023))

  $\rightarrow$ align data-structure with the factoring has no cost

Heuristics Part III: Precomputed heuristics

Given a precomputed abstraction heuristic (PDB, ADD, M&S) can we compute $h_{\mathcal{F}}(s^{\mathcal{F}})$ efficiently?

- Single PDBs: yes
- ADDs/M&S: not in general (NP-complete), but yes for compliant data-structures (Gnad *et al.* (2023))

  $\rightarrow$ align data-structure with the factoring has no cost
- Multiple PDBs (max or sum): not in general (NP-complete)
  $\rightarrow$ For PDBs that only affect a single leaf, we can approximate their sum (Sievers *et al.* (2022))

Heuristics Part III: Precomputed heuristics

Given a precomputed abstraction heuristic (PDB, ADD, M&S) can we compute $h_{\mathcal{F}}(s^{\mathcal{F}})$ efficiently?

- Single PDBs: yes
- ADDs/M&S: not in general (NP-complete), but yes for compliant data-structures (Gnad *et al.* (2023))

  $\rightarrow$ align data-structure with the factoring has no cost
- Multiple PDBs (max or sum): not in general (NP-complete)
  $\rightarrow$ For PDBs that only affect a single leaf, we can approximate their sum (Sievers *et al.* (2022))

Open Question: How to approximate additive abstractions in more general ways?

Pruning Methods

- Symmetry breaking (Gnad *et al.* (2017))
  $\rightarrow$ Permute prices and/or center state
- Dominance pruning with dominance analysis (for forks) (Torralba *et al.* (2016))
  $\rightarrow$ Propagate prices from better to worse leaf states
- Partial order reduction (Gnad *et al.* (2019))
  $\rightarrow$ over global actions

Recharging Robots

- Running Example from (Gnad *et al.* (2022))
- Submitted to the International Planning Competition
- IPC Organizers improved the domain (so, the version here is substantially different from the IPC version).

Decoupled Search – The Story so far..

- **Beating LM-cut with hmax (Sometimes) – Fork-Decoupled State-Space Search**
  G, Hoffmann, ICAPS'15.

$$\begin{tikzcd} & C \arrow[dl] \arrow[d] \arrow[dr] & \\ L_1 & L_2 & L_3 \end{tikzcd}$$

Decoupled Search – The Story so far..

- **Beating LM-cut with hmax (Sometimes) – Fork-Decoupled State-Space Search**
  G, *Hoffmann*, ICAPS'15.

$$\begin{tikzcd} & C \arrow[dl] \arrow[d] \arrow[dr] & \\ L_1 & L_2 & L_3 \end{tikzcd}$$

- **From Fork Decoupling to Star-Topology Decoupling**
  G, *Hoffmann*, *Domshlak*, SOCS'15.

$$\begin{tikzcd} L_1 & & L_2 \arrow[dl] \\ & C \arrow[ul] \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\ L_3 \arrow[rr, leftrightarrow, dashed] & & L_4 \end{tikzcd}$$

Decoupled Search – The Story so far..

- **Beating LM-cut with hmax (Sometimes) – Fork-Decoupled State-Space Search**
  G, *Hoffmann*, ICAPS'15.

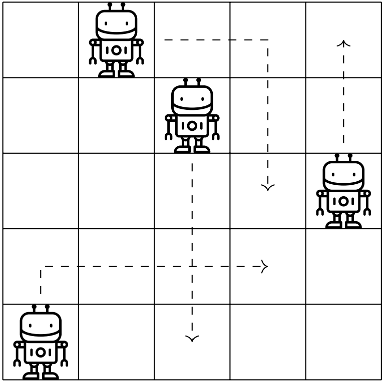
$$
\begin{tikzcd}
& C \arrow[dl] \arrow[d] \arrow[dr] & \\
L_1 & L_2 & L_3
\end{tikzcd}
$$

- **From Fork Decoupling to Star-Topology Decoupling**
  G, *Hoffmann*, *Domshlak*, SOCS'15.

$$
\begin{tikzcd}
L_1 & & L_2 \arrow[dl] \\
& C \arrow[ul] \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] & \\
L_3 \arrow[rr, leftrightarrow, dashed] & & L_4
\end{tikzcd}
$$

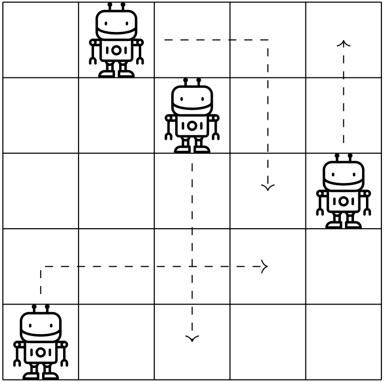
- How to obtain Star Factorings? IJCAI'17, ICAPS'19.

Collaborative Robots – Where is the center?



Robots (R_i) move freely in world, no collisions, battery usage (B_i).

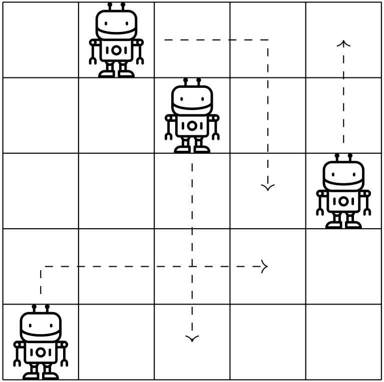
Collaborative Robots – Where is the center?



Robots (R_i) move freely in world, no collisions, battery usage (B_i).

Actions: $move(R_i, B_i, l_x, l_y)$: moving consumes battery; robots can $charge(R_i, B_i, R_j, B_j)$ each other.

Collaborative Robots – Where is the center?



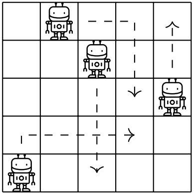
Causal Graph:

$$\begin{tikzcd}
& R_1 \leftrightarrow B_1 \arrow[dl, leftrightarrow] \arrow[dr, leftrightarrow] \arrow[dd, leftrightarrow] & \\
R_2 \leftrightarrow B_2 \arrow[rr, leftrightarrow] \arrow[dr, leftrightarrow] & & R_3 \leftrightarrow B_3 \arrow[dl, leftrightarrow] \\
& R_4 \leftrightarrow B_4 &
\end{tikzcd}$$

Robots (R_i) move freely in world, no collisions, battery usage (B_i).

Actions: $move(R_i, B_i, l_x, l_y)$: moving consumes battery; robots can $charge(R_i, B_i, R_j, B_j)$ each other.

Factoring in the Recharging Robots

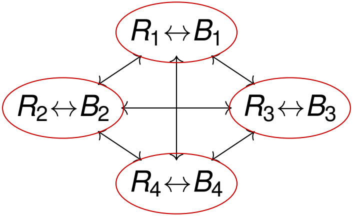


$move(R_i, B_i, l_x, l_y)$:

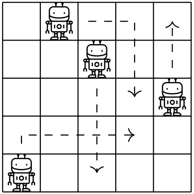
$$pre = \{R_i = l_x, B_i = b\},$$
$$eff = \{R_i = l_y, B_i = b - 1\}$$

$charge(R_i, B_i, R_j, B_j)$:

$$pre = \{R_i = R_j = l_x, B_i = b, B_j = c\},$$
$$eff = \{B_i = b - 1, B_j = c + 1\}$$



Factoring in the Recharging Robots



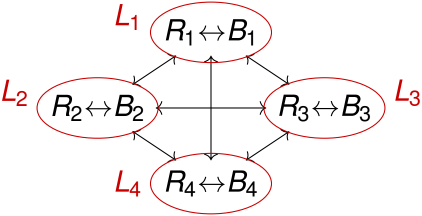
$move(R_i, B_i, l_x, l_y)$: $\rightarrow$ **internal(leaf-only) actions**

$$pre = \{R_i = l_x, B_i = b\},$$
$$eff = \{R_i = l_y, B_i = b - 1\}$$

$charge(R_i, B_i, R_j, B_j)$: $\rightarrow$ **global actions**

$$pre = \{R_i = R_j = l_x, B_i = b, B_j = c\},$$
$$eff = \{B_i = b - 1, B_j = c + 1\}$$

$\mathbf{C} = \emptyset$



Finding Generalized Factorings

- Formulate factoring process as **integer linear program** (ILP).
- Any **partition of the state variables** is a valid factoring.

Finding Generalized Factorings

- Formulate factoring process as **integer linear program** (ILP).
- Any **partition of the state variables** is a valid factoring.
- **Optimize important properties** of the factoring:

Finding Generalized Factorings

- Formulate factoring process as **integer linear program** (ILP).
- Any **partition of the state variables** is a valid factoring.
- **Optimize important properties** of the factoring:
  - Number of leaves,
  - Mobility: number of leaf-only actions (sum over leaves),
  - Balanced mobility: # leaf-only actions (product over leaves),
  - Flexibility: ratio of leaf-only actions (sum over facts).

Finding Generalized Factorings

- Formulate factoring process as **integer linear program** (ILP).
- Any **partition of the state variables** is a valid factoring.
- **Optimize important properties** of the factoring:
  - Number of leaves,
  - Mobility: number of leaf-only actions (sum over leaves),
  - Balanced mobility: # leaf-only actions (product over leaves),
  - Flexibility: ratio of leaf-only actions (sum over facts).
- **Require minimum flexibility** $\{0\%, 5\%, \ldots 100\%\}$.

Finding Generalized Factorings

- Formulate factoring process as **integer linear program** (ILP).
- Any **partition of the state variables** is a valid factoring.
- **Optimize important properties** of the factoring:
  - Number of leaves,
  - Mobility: number of leaf-only actions (sum over leaves),
  - Balanced mobility: # leaf-only actions (product over leaves),
  - Flexibility: ratio of leaf-only actions (sum over facts).
- **Require minimum flexibility** $\{0\%, 5\%, \ldots 100\%\}$.
- Leaf candidates: **action effect schemas** $vars(\mathit{eff}\, a)$ and **SCCs of CG**.

Factoring Properties

What are important properties of a factoring that influence search-space reduction?

Factoring Properties

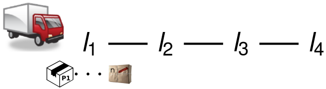
What are important properties of a factoring that influence search-space reduction?

- **Number of leaf factors:**

Factoring Properties

What are important properties of a factoring that influence search-space reduction?

- **Number of leaf factors:**
  2^N different states resulting from only loading all packages at l_1!



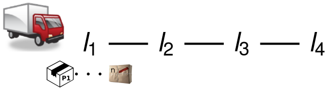
Factoring Properties

What are important properties of a factoring that influence search-space reduction?

- **Number of leaf factors:**

  2^N different states resulting from only loading all packages at l_1!

  $\rightarrow$ This is a single decoupled state.



The reduction is exponential in the number of leaves. (Gnad and Hoffmann (2018))

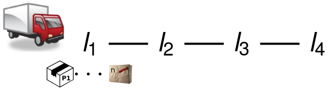
Factoring Properties

What are important properties of a factoring that influence search-space reduction?

- **Number of leaf factors:**

  2^N different states resulting from only loading all packages at l_1!

  $\rightarrow$ This is a single decoupled state.



  The reduction is exponential in the number of leaves. (Gnad and Hoffmann (2018))

- **Leaf mobility:**

  A leaf factor $L \in \mathcal{L}$ is *mobile*, if it has only-leaf actions

  $\rightarrow$ Leaves that are not mobile do not contribute to the search-space reduction

Maximizing the Number of Leaves – Complexity

Theorem (Maximize Number of Leaf Factors). *Given a planning task Π, it is* **NP**-*hard to decide if there exists a factoring with N leaves.*

Maximizing the Number of Leaves – Complexity

Theorem (Maximize Number of Leaf Factors). *Given a planning task Π, it is* **NP***-hard to decide if there exists a factoring with N leaves.*

Proof sketch. Reduction from maximum independent set (MIS)
Compute a MIS of $CG(\Pi)$. By construction, no connection between variables in the maximum independent set.
$\rightarrow$ Each of these variables forms a leaf factor, the rest is the center.

Maximizing the Number of Leaves – Complexity

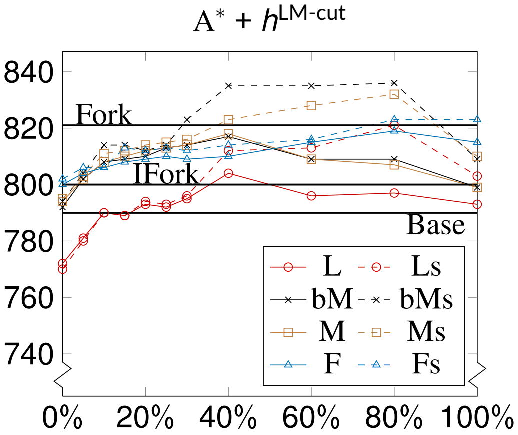
Theorem (Maximize Number of Leaf Factors). *Given a planning task Π, it is* **NP***-hard to decide if there exists a factoring with N leaves.*

Proof sketch. Reduction from maximum independent set (MIS)
Compute a MIS of $CG(\Pi)$. By construction, no connection between variables in the maximum independent set.
$\rightarrow$ Each of these variables forms a leaf factor, the rest is the center.

Practical Approaches:

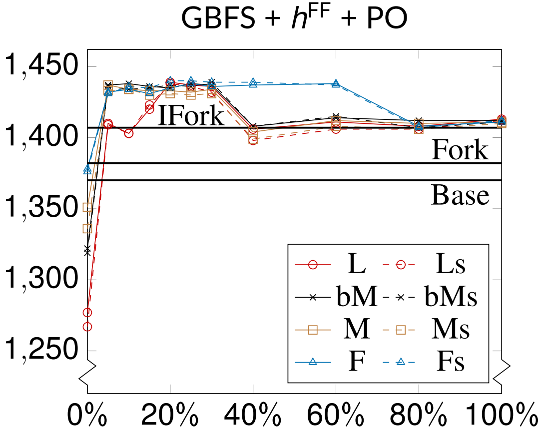
- Compute MIS of CG $\rightarrow$ strict-star factorings,
- Analyze strongly-connected components in CG
  $\rightarrow$ (inverted-)fork factorings,
- Greedy selection of center variables based on CG connectivity
  $\rightarrow$ strict-star factorings,
- Encode factoring as Integer Linear Program $\rightarrow$ star factorings.

Enforce minimum Leaf Fact Flexibility



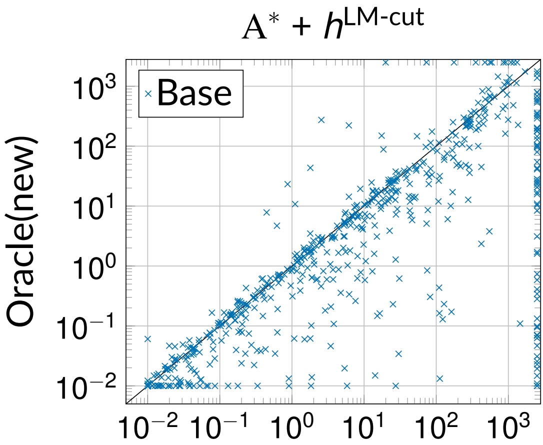
Fork: fork factorings, IFork: inverted-forks, Base: explicit-state search.

Enforce minimum Leaf Fact Flexibility

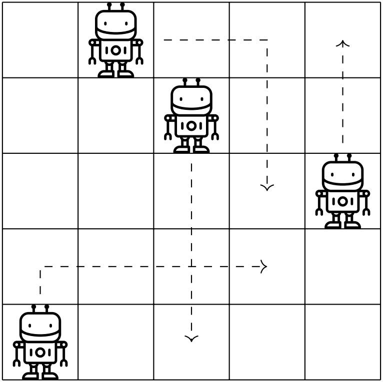


Fork: fork factorings, IFork: inverted-forks, Base: explicit-state search.

Runtime Scatterplot – LM-cut

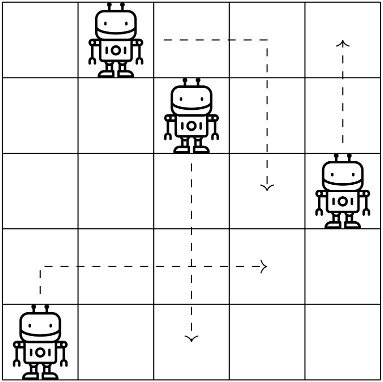


Collaborative Robots – Deja vu?



Robots (R_i) move freely in world, no collisions, battery usage (B_i).

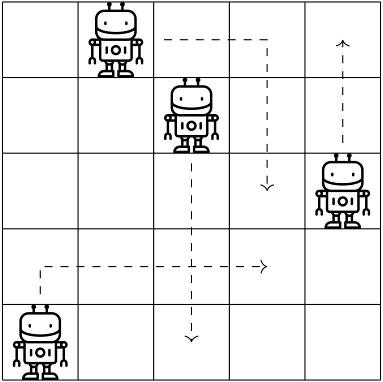
Collaborative Robots – Deja vu?



Robots (R_i) move freely in world, no collisions, battery usage (B_i).

Actions: $move(R_i, B_i, l_x, l_y)$: moving consumes battery;
robots can $charge(R_i, B_i, R_j, B_j)$ each other.

Multi-Agent Pathfinding

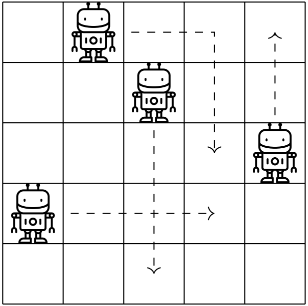


Actions: $move(R_i, l_x, l_y)$

- Constraint: Two agents cannot be in the same cell at the same time
- Metric: Minimize Makespan

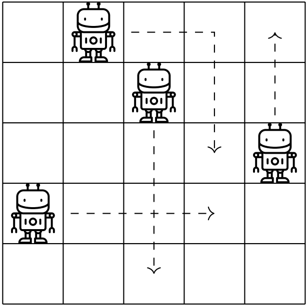
Conflict-Based Search (Sharon *et al.* (2015))

1. Each agent plans their own shortest path
2. If there is no conflict, done



Conflict-Based Search (Sharon *et al.* (2015))

1. Each agent plans their own shortest path
2. If there is no conflict, done
3. If there is a conflict, branch adding constraints that resolve it



R2 cannot be at $(3, 2)$ at $t = 2$

$\emptyset$

R1 cannot be at $(3, 2)$ at $t = 2$

Questions

- Can decoupled search be applied to multi-agent pathfinding?
- What is the relation to Conflict-based Search (CBS)?

Questions

- Can decoupled search be applied to multi-agent pathfinding?
- What is the relation to Conflict-based Search (CBS)?

Challenge: Representing MAPF as a Planning Task is not Straightforward

- How to represent the constraint that two robots cannot be in the same cell at the same time?
- How to make sure that robots move simultaneously?

Questions

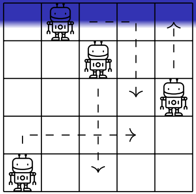
- Can decoupled search be applied to multi-agent pathfinding?
- What is the relation to Conflict-based Search (CBS)?

Challenge: Representing MAPF as a Planning Task is not Straightforward

- How to represent the constraint that two robots cannot be in the same cell at the same time?
- How to make sure that robots move simultaneously?

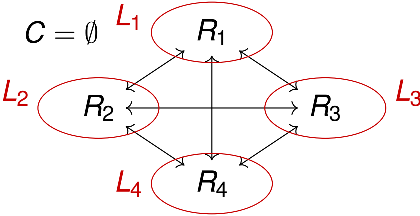
$\rightarrow$ Let's Ignore the Details

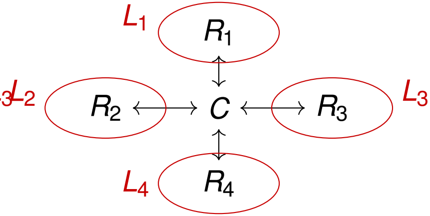
Let's Ignore the Details



$move(R_i, B_i, l_x, l_y)$: $\rightarrow$ **internal(leaf-only) actions**

$book\_space(R_i, l_y, t)$: $\rightarrow$ **global actions**





Parallels between Decoupled Search and CBS

Decoupled Search for planning:	CBS for MAPF:
Leafs are conditionally independent	Agents are conditionally independent
Search over center actions	Search over conflict resolution
Handle conflicts eagerly	Handle conflicts lazily

Parallels between Decoupled Search and CBS

Decoupled Search for planning:	CBS for MAPF:
Leafs are conditionally independent	Agents are conditionally independent
Search over center actions	Search over conflict resolution
Handle conflicts eagerly	Handle conflicts lazily

- Typically more conflicts in planning
- Conflicts in planning are more complex to represent and resolve (e.g. the plans of the leaves may need to be interleaved in a specific way)
- Extensions of CBS handle conflicts more eagerly when needed

Parallels between Decoupled Search and CBS

Decoupled Search for planning:	CBS for MAPF:
Leafs are conditionally independent	Agents are conditionally independent
Search over center actions	Search over conflict resolution
Handle conflicts eagerly	Handle conflicts lazily

- Typically more conflicts in planning
- Conflicts in planning are more complex to represent and resolve (e.g. the plans of the leaves may need to be interleaved in a specific way)
- Extensions of CBS handle conflicts more eagerly when needed

→ Can we transfer ideas?

Conclusion

- The success of heuristic search heavily depends on the definition of the search space

Conclusion

- The success of heuristic search heavily depends on the definition of the search space
- Decoupled Search:
  - State-space reduction method (reduce the search space by orders of magnitude)
  - Define the search space
  - Exploit the task structure (conditional independence)
  - Each search node in the new search space represents many states of the planning task
- Properties:
  - captures the reachability of all states of a planning task and preserves optimality for any optimal search algorithm
  - Decoupled search can be combined with (in principle) any known AI Planning heuristic, making available highly informed search guidance techniques for decoupled states.
- Still lots of things to do!

References I

Daniel Gnad and Jörg Hoffmann. Star-topology decoupled state space search. *Artificial Intelligence*, 257:24–60, 2018.

Daniel Gnad, Álvaro Torralba, Alexander Shleyfman, and Jörg Hoffmann. Symmetry breaking in star-topology decoupled search. In Laura Barbulescu, Jeremy Frank, Mausam, and Stephen F. Smith, editors, *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS 2017)*, pages 125–134. AAAI Press, 2017.

Daniel Gnad, Jörg Hoffmann, and Martin Wehrle. Strong stubborn set pruning for star-topology decoupled state space search. *Journal of Artificial Intelligence Research*, 65:343–392, 2019.

Daniel Gnad, Álvaro Torralba, and Daniel Fišer. Beyond stars - generalized topologies for decoupled search. In Sylvie Thiébaux and William Yeoh, editors, *Proceedings of the Thirty-Second International Conference on Automated Planning and Scheduling (ICAPS 2022)*, pages 110–118. AAAI Press, 2022.

References II

Daniel Gnad, Silvan Sievers, and Álvaro Torralba. Efficient evaluation of large abstractions for decoupled search: Merge-and-shrink and symbolic pattern databases. In Sven Koenig, Roni Stern, and Mauro Vallati, editors, *Proceedings of the Thirty-Third International Conference on Automated Planning and Scheduling (ICAPS 2023)*. AAAI Press, 2023.

Daniel Gnad. *Star-Topology Decoupled State-Space Search in AI Planning and Model Checking*. PhD thesis, Saarland University, 2021.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

Guni Sharon, Roni Stern, Ariel Felner, and Nathan Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

Silvan Sievers, Daniel Gnad, and Álvaro Torralba. Additive pattern databases for decoupled search. In *Proceedings of the 15th Annual Symposium on Combinatorial Search (SoCS 2022)*, pages 180–189. AAAI Press, 2022.

References III

Álvaro Torralba, Daniel Gnad, Patrick Dubbert, and Jörg Hoffmann. On state-dominance criteria in fork-decoupled search. In Subbarao Kambhampati, editor, *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016)*, pages 3265–3271. AAAI Press, 2016.