Multi-Agent Path Finding

UNIVERSITY OF BRESCIA

Intelligenza Artificiale

Mattia Chiari

Swarm Automation (SA)

Definizione

Campo della robotica che riguarda la **coordinazione di grandi gruppi di robot autonomi relativamente semplici** che hanno come obiettivo il raggiungimento di un **goal condiviso.**

Fonte: Toyota

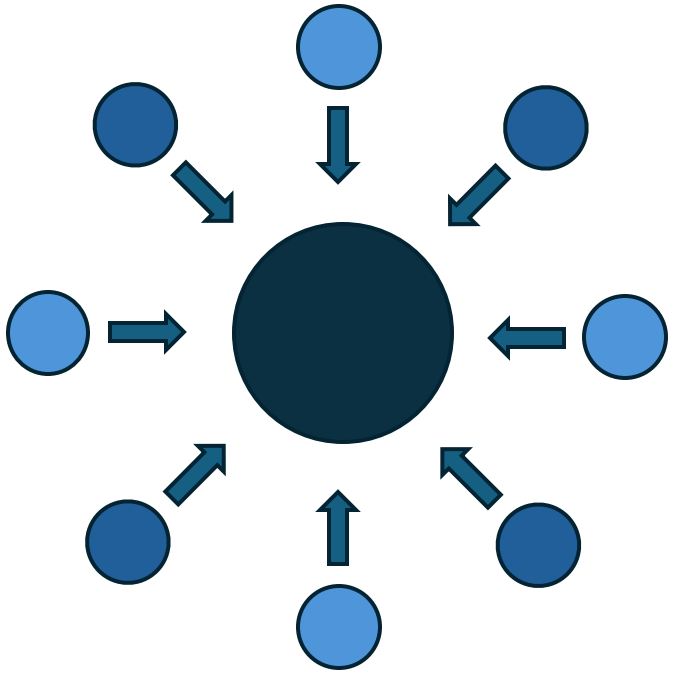
Fonte: Ocado Intelligent Automation

Ha diverse applicazioni tra cui:

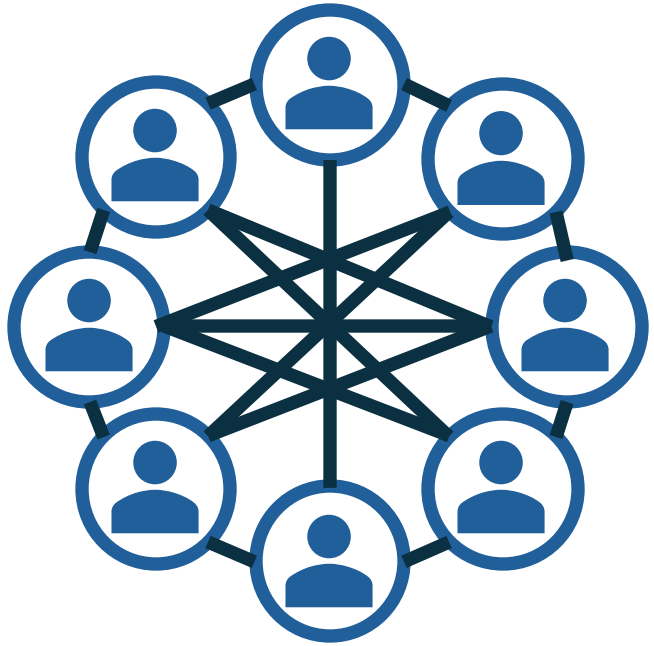
- Warehouse and Logistics
- Operazioni di ricerca e salvataggio
- Salute
- Analisi e costruzione di infrastrutture

Due modi per gestire SA

Centralizzazione



Decentralizzazione



Centralizzazione VS Decentralizzazione

Centralizzazione

Un **coordinatore** gestisce tutti gli agenti e **decide come muovere ciascuno** di essi per raggiungere il goal.

Decentralizzazione

Ogni **agente** esegue il **proprio piano** e comunica con gli agenti vicini per **evitare collisioni**

Centralizzazione VS Decentralizzazione (2)

Centralizzazione

PRO:

- Fornisce garanzie teoriche come ottimalità e completezza

CONTRO:

- Lenta
- Non scalabile

Decentralizzazione

PRO:

- Veloce
- Scalabile

CONTRO:

- Poche o nessuna garanzia teorica

Centralizzazione VS Decentralizzazione (3)

Per casi reali, in cui il numero di agenti supera anche le **500 unità**, la scelta più utilizzata è la (semi) **decentralizzazione**. Questo è dovuto principalmente alla **poca scalabilità** e alla **lentezza** dei **sistemi centralizzati** che impediscono di gestire numeri elevati di agenti rapidamente.

Fonte: YT Intel Automated Material-Handling System

Decentralizzazione

La **decentralizzazione** è uno **strumento molto potente** che noi utilizziamo nella vita di tutti i giorni.

Quando ci spostiamo, per esempio, noi cerchiamo di raggiungere la nostra destinazione interagendo con altre persone che hanno destinazioni diverse per evitare di fare incidenti.

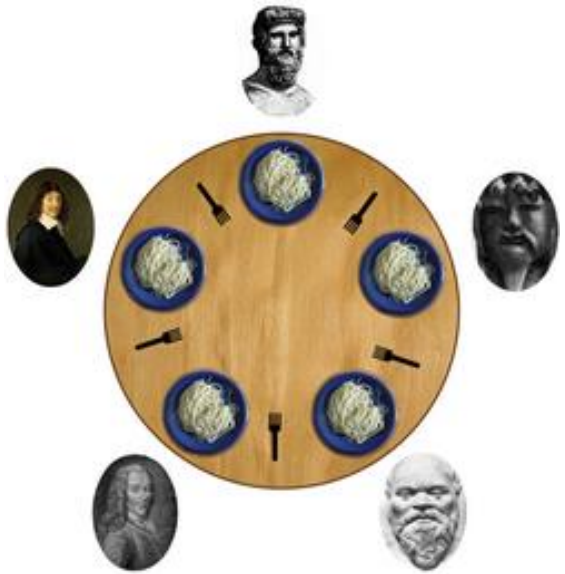
Fonte: Incrocio di Shibuya (Tokyo, JP)

Fonte: Tarifa, SPA

Decentralizzazione (2)

La **decentralizzazione** genera la possibilità di avere una **cattiva coordinazione** tra gli agenti

Fonte: https://kei18.github.io/lacam2/



Fonte: Problema dei filosofi a cena

Decentralizzazione (3)

La **cattiva coordinazione** può, a sua volta, generare **incidenti**.

Spesso questi incidenti potrebbero essere ridotti attraverso l'utilizzo di un sistema **centralizzato**

Fonte: San Paolo, BRA

Ocado Group PLC + Add to myFT

Ocado's largest warehouse shut after robot collision causes fire

Blaze at London site means automated facility will be unable to fulfil online orders for several days

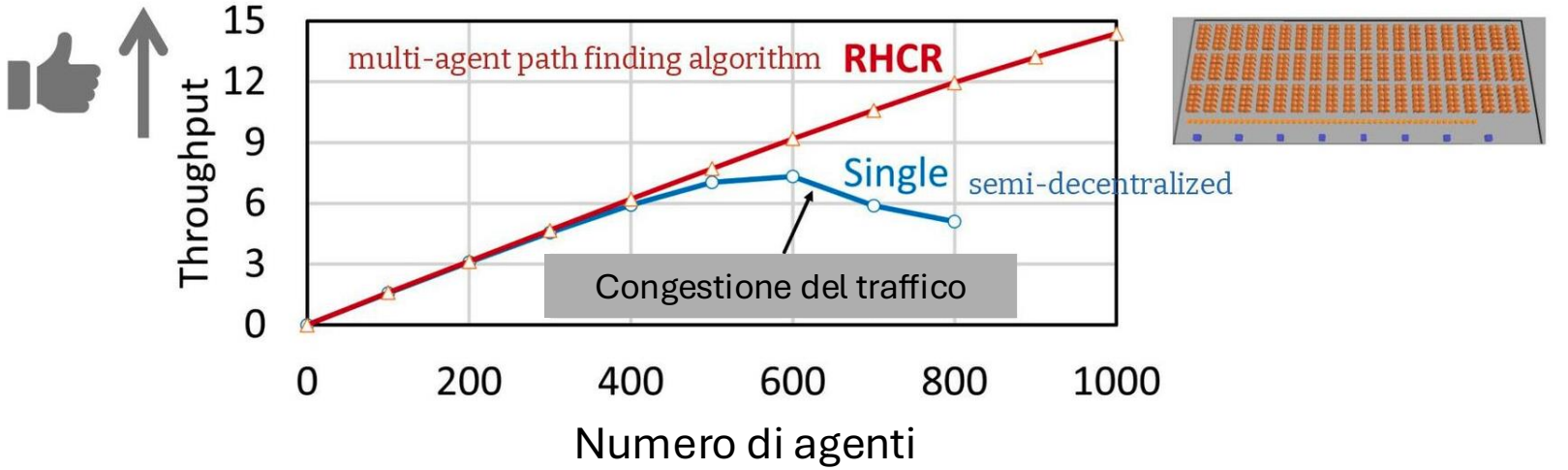
Members of London Fire Brigade at the scene of the blaze at Ocado's fulfilment centre in Erith, south-east London © London Fire Brigade/Twitter

Fonte: Financial Times

Centralizzazione

La **centralizzazione**, oltre ad evitare incidenti, porta anche ad un miglioramento delle prestazioni complessive del sistema

Simulazione per la gestione di un magazzino



Fonte: Varambally, Sumanth, Jiaoyang Li, and Sven Koenig. "Which MAPF model works best for automated warehousing?", SOCS 2022.

Centralizzazione (2)

Domanda

Possiamo creare **algoritmi centralizzati** che siano **scalabili** pur mantenendo alcune **garanzie teoriche**?

Di questo si occupa il **Multi-Agent Path Finding**

Multi-Agent Path Finding (MAPF)

Problema

Coordinare rotte e tempi di molti agenti su uno spazio condiviso, rispettando regole anti-collisione e ottimizzando un criterio scelto.

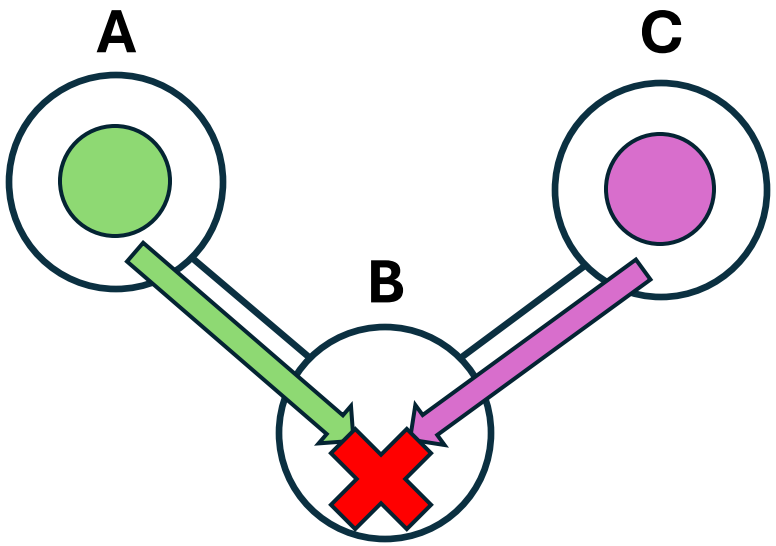
Obiettivi

- Velocità (idealmente **real-time**)
- Scalabilità
- Pochi movimenti inutili (idealmente **ottimalità**)

Tipologie di collisione

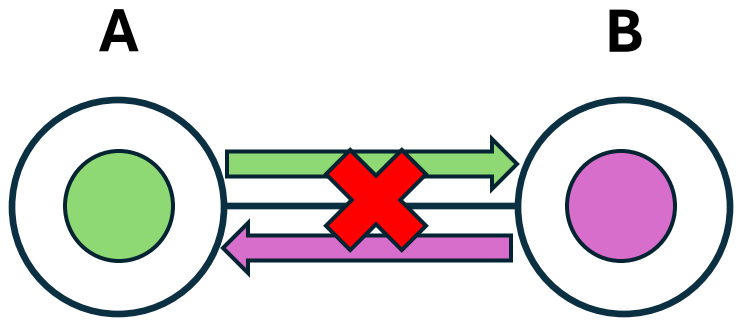
Collisione sul nodo

- L'agente 1 ( ) va dal nodo A al nodo B
- L'agente 2 ( ) va dal nodo C al nodo B



Collisione sul vertice

- L'agente 1 ( ) va dal nodo A al nodo B
- L'agente 2 ( ) va dal nodo B al nodo A



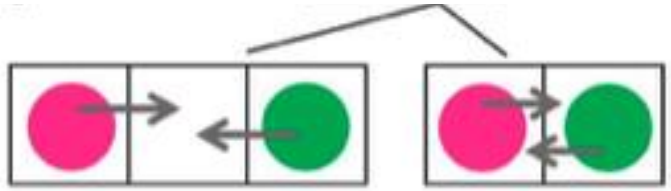
Esempio di MAPF

Dati

- grafo
- agenti
- goal

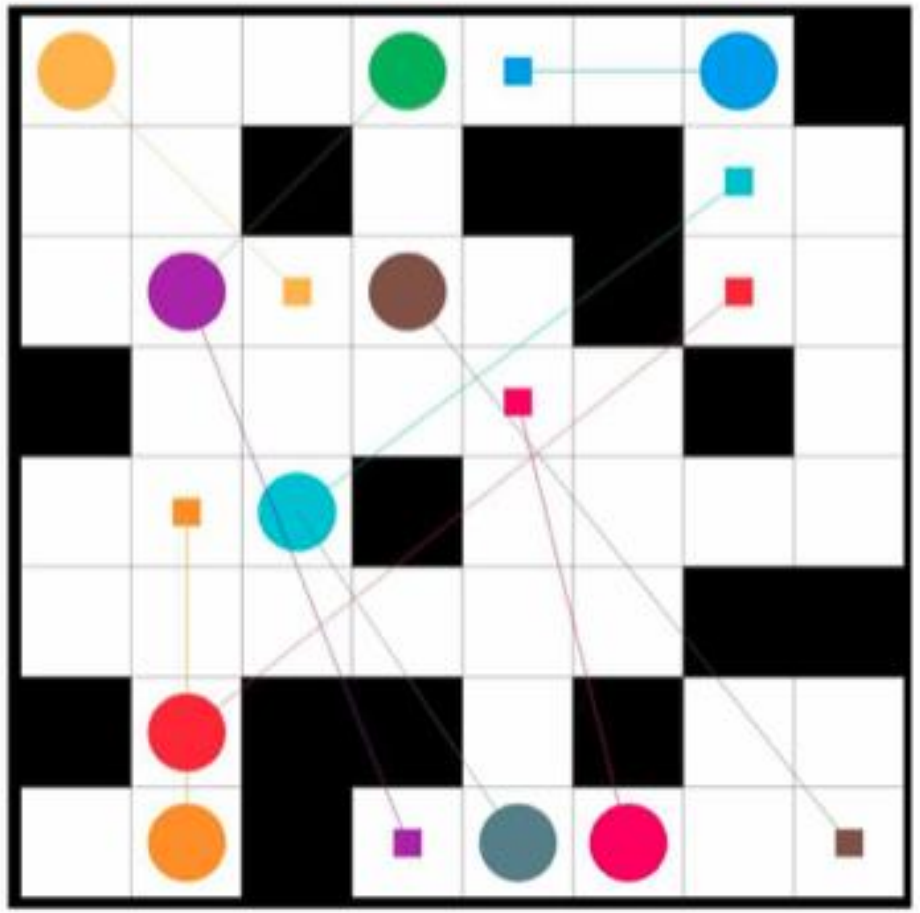
Soluzione

percorsi senza **collisioni**



Costi

tempo totale, distanza, throughput



Fonte: https://kei18.github.io/lacam2/

Applicazioni del MAPF: Videogiochi

Il MAPF è un componente fondamentale dell'intelligenza artificiale per la navigazione dei Non-Player Characters (NPC,), specialmente in scenari in cui un **gruppo di agenti** deve muoversi in un ambiente affollato **senza bloccarsi a vicenda**.

Le tecniche implementate nei videogiochi sono solitamente:

- A* con ottimizzazioni
- CBS su mesh (versione semplificata della mappa)

Fonte: Starcraft II

Fonte: Age of Empire

Applicazioni del MAPF: Logistica

La **logistica** è probabilmente l'applicazione industriale più diffusa e di maggior successo del MAPF.
Alcuni esempi sono:

- **Sistemi Kiva/Amazon Robotics:** I centri di smistamento di Amazon utilizzano centinaia o migliaia di robot autonomi per sollevare e spostare scaffali fino alle postazioni dei lavoratori. Gli algoritmi MAPF sono fondamentali per orchestrare i movimenti di questi agenti.

- **Sistemi di Smistamento Pacchi:** Robot in centri di smistamento merci utilizzano il MAPF per muoversi su nastri o aree dedicate e depositare i pacchi.

Fonte: Toyota

Fonte: Ocado Intelligent Automation

Lifelong Multi Agent Path Finding (LMAPF)

Problema

Coordinare rotte e tempi di molti agenti su uno spazio condiviso, rispettando regole anti-collisione e ottimizzando un criterio scelto. Gli agenti **ricevono continuamente nuovi goal** man mano che completano quelli precedenti.

Obiettivi

Massimizzare il **throughput** (produttività), ovvero il numero medio di compiti completati per unità di tempo dall'intero sistema.

Algoritmi MAPF

Domanda

È vero che gli algoritmi per il MAPF non sono scalabili?

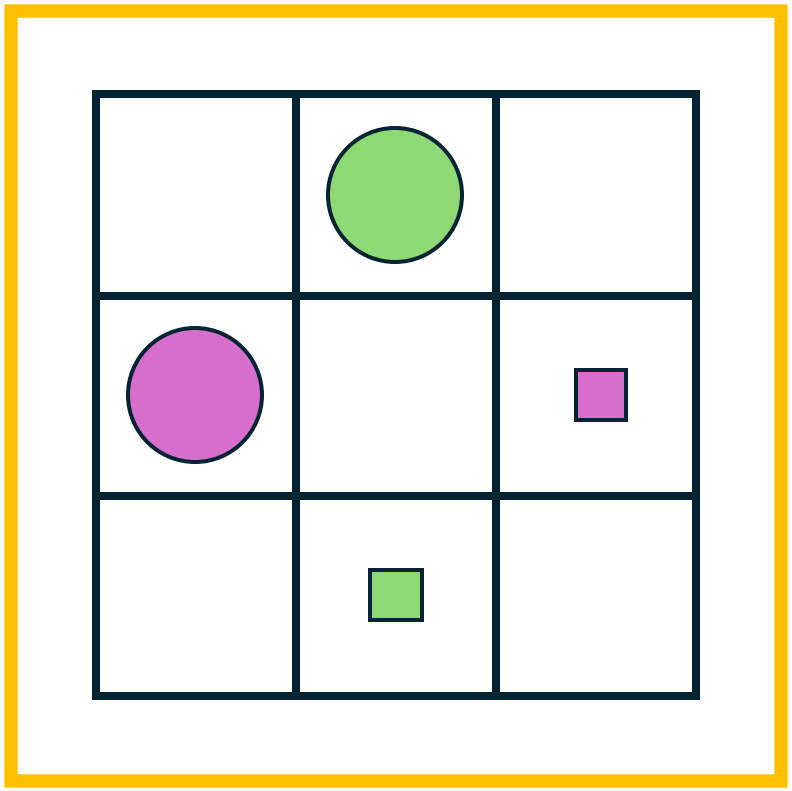
A* per MAPF


```
Function Best-First-Search(start state s_init):
    OPEN ← {s_init};
    while OPEN ≠ ∅ do
        best ← chooseNode(OPEN);
        OPEN = OPEN \ {best};
        if best ∈ G then
            return best;
        end
        OPEN ← OPEN ∪ succ(n)
    end
```

- se `chooseNode` usa $h(n)$: GBFS
- se usa $f(n) = g(n) + h(n)$: $A^\star$;
  - se $h(n)$ **ammissibile**, $A^\star$ produce soluzione **ottima**.

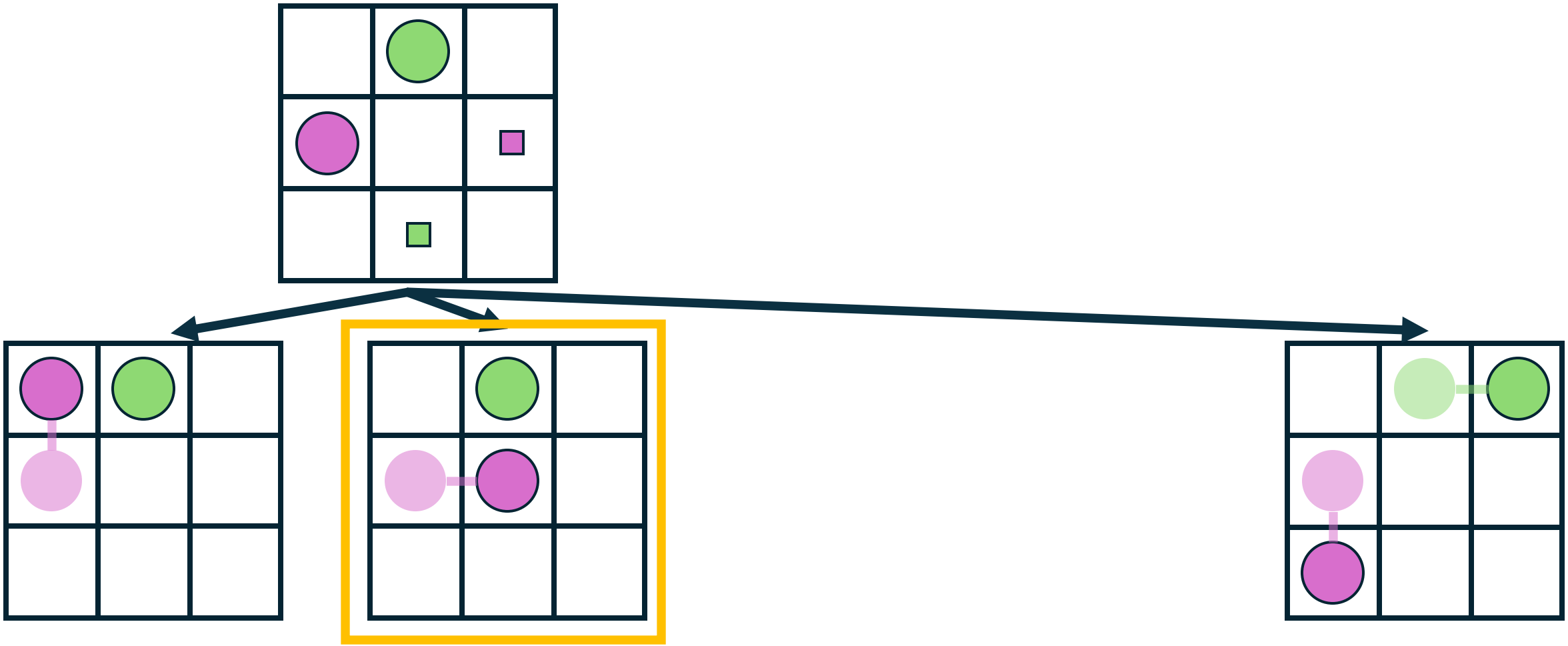
In aggiunta si calcolano i successori *succ(n)* facendo in modo che le **transizioni non ammissibili** (collisioni sul nodo e sul vertice) vengano **escluse**.

A* per MAPF: Esempio

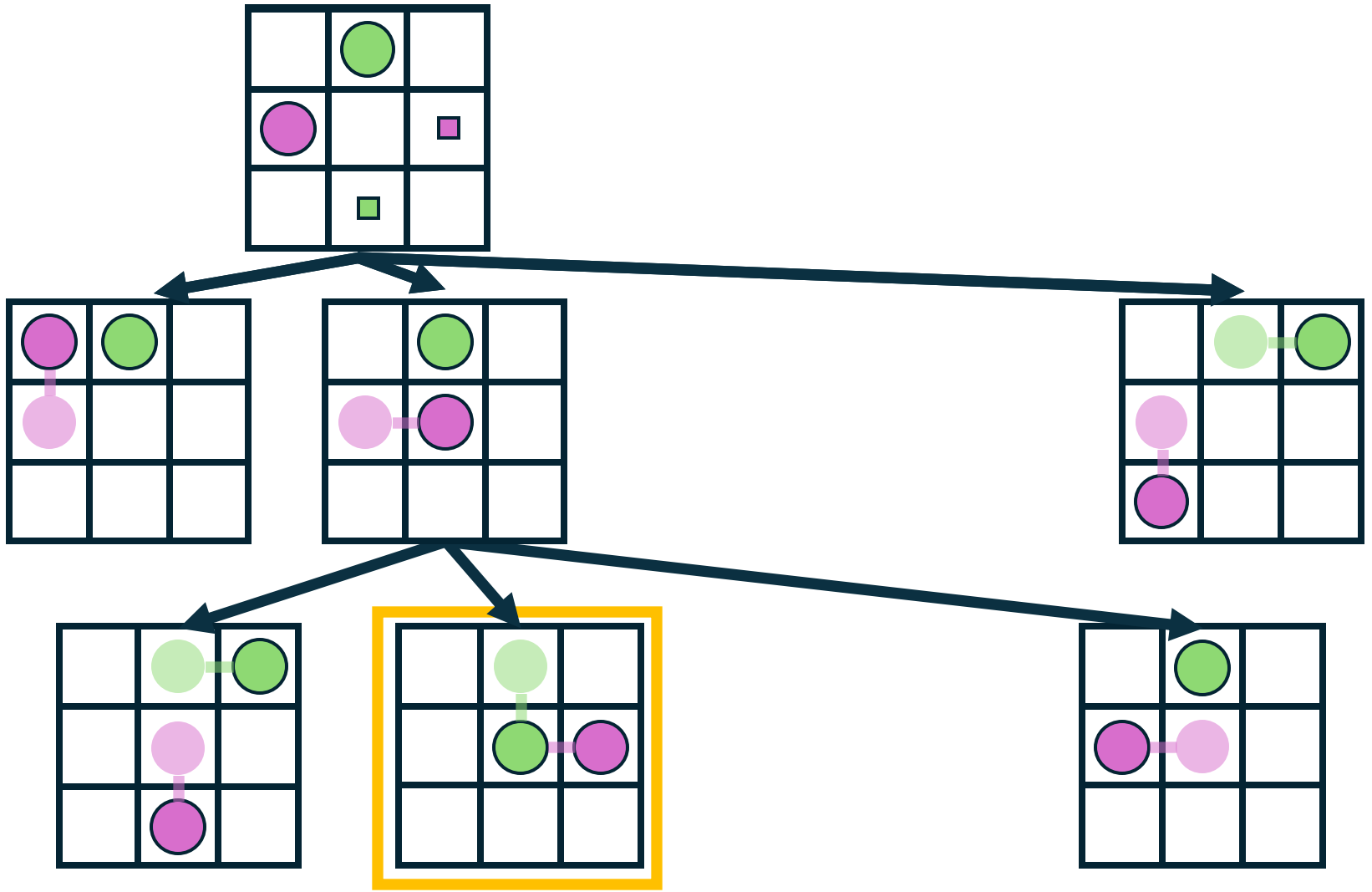


Espando il nodo iniziale

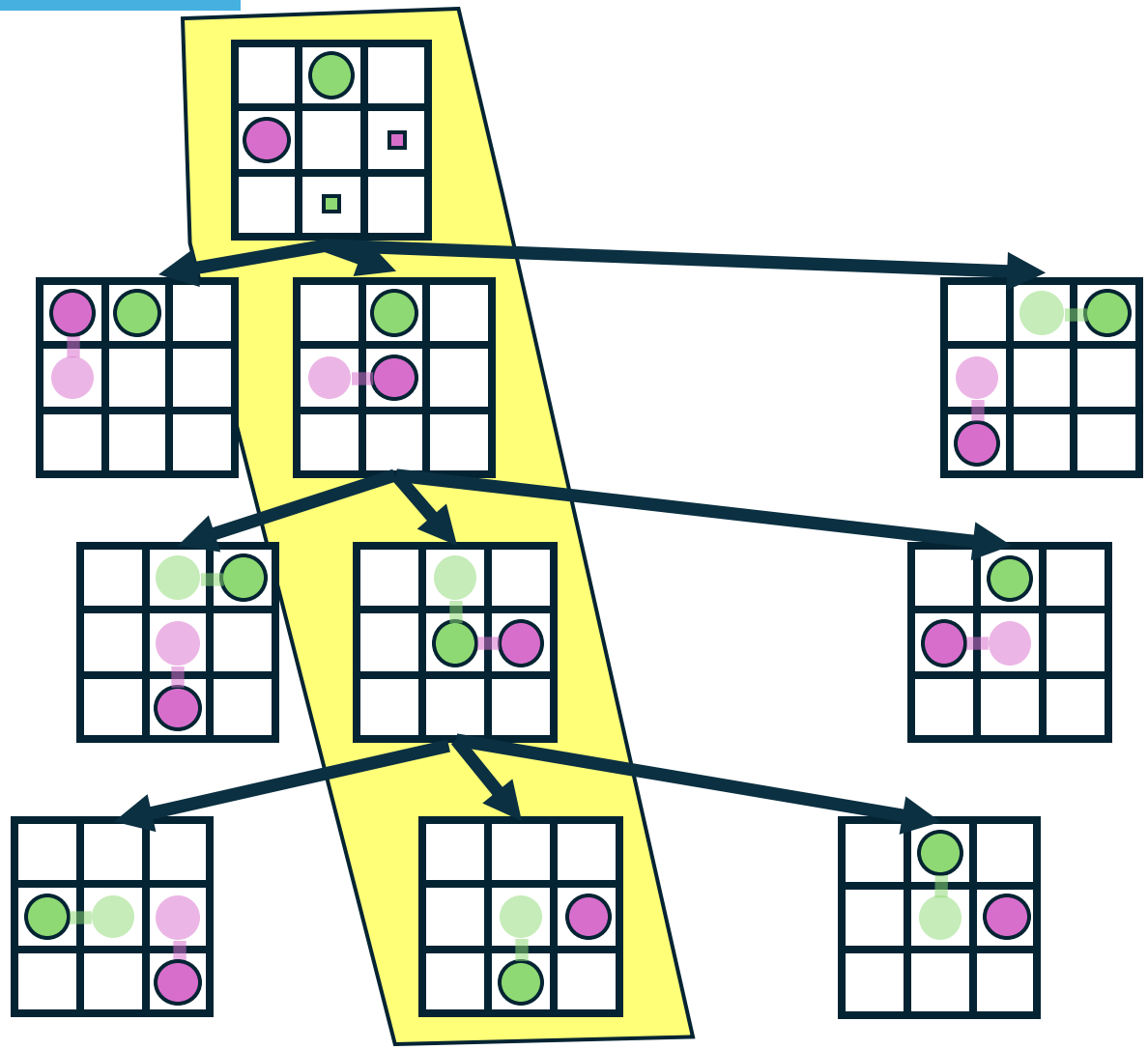
A* per MAPF: Esempio (2)



A* per MAPF: Esempio (3)



A* per MAPF: Esempio (4)



A* per MAPF: Proprietà

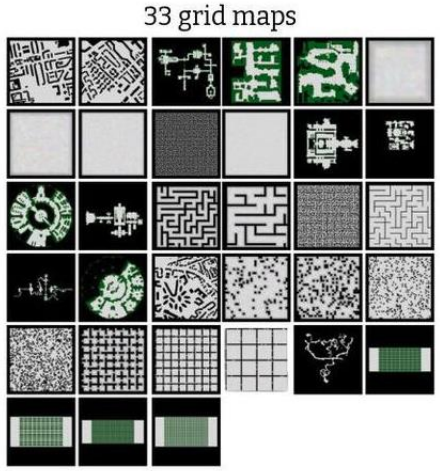
Completezza: l'algoritmo ritorna soluzioni in un tempo finito per tutte le istanze risolvibili; altrimenti segnala la che la soluzione non esiste

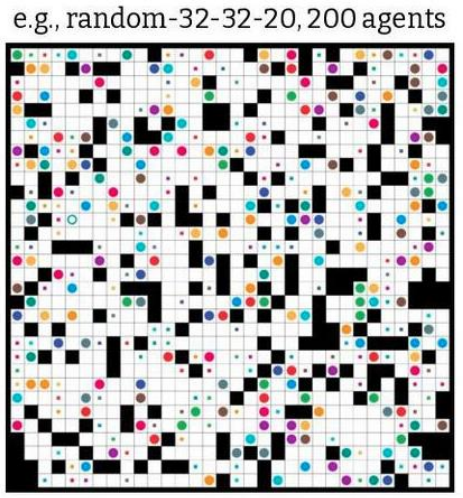
Ottimalità: l'algoritmo ritorna sempre la soluzione che ha costo minimo

A* per MAPF: Risultati su benchmark

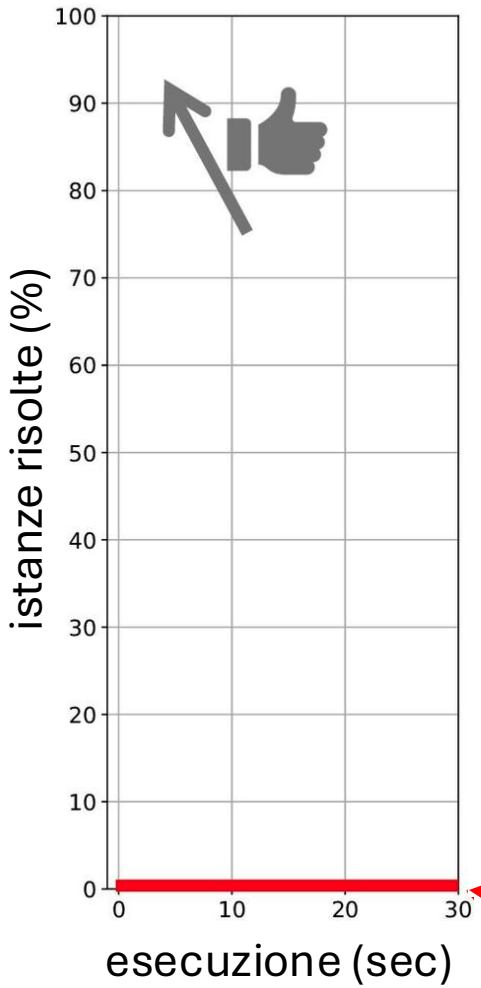
Osserviamo i risultati su un **benchmark** di MAPF composto da:

- 13900 istanze
- 33 mappe a griglia
- Ogni mappa è testata con un numero di agenti che va da 50 a 1000 ad intervalli di 50 agenti
- Il test è effettuato su un PC desktop





A* per MAPF: Risultati su benchmark



0.0% A* [Hart+68]	completo	ottimo

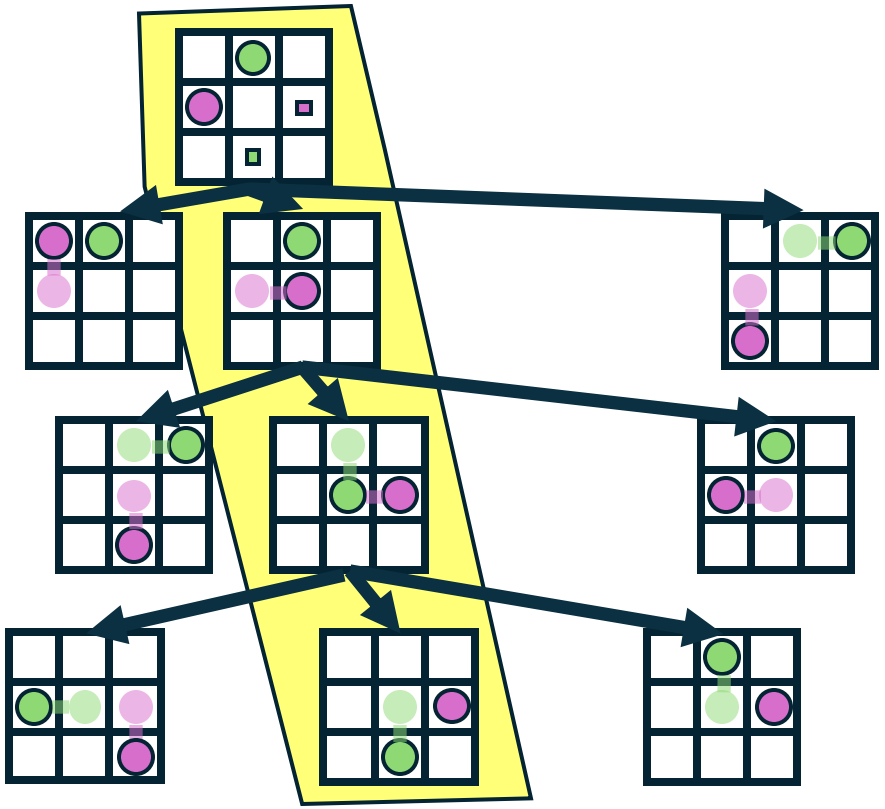
Perché A* non funziona?

Definizione: Branching factor

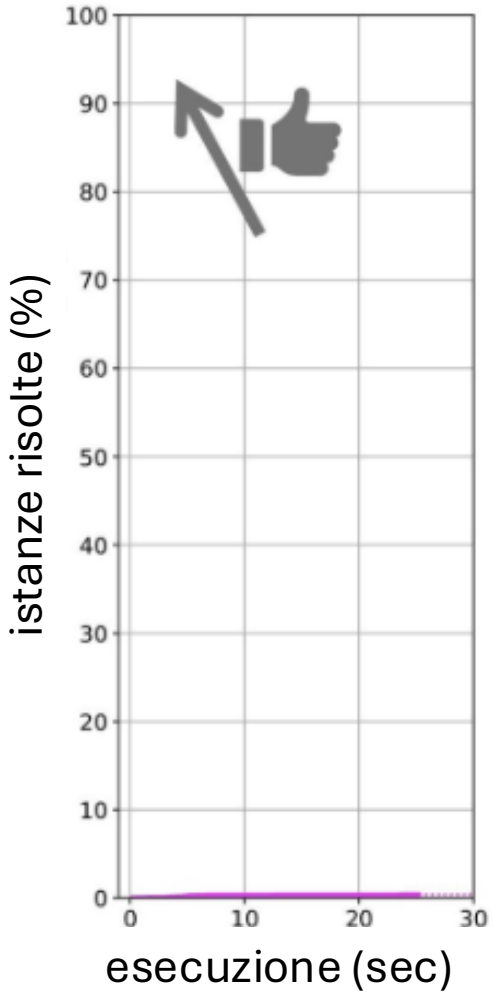
Il numero di nodi successori per un dato nodo

MAPF per mappe a griglia ha un branching factor enorme: O(5^N) dove N è il numero di agenti. Ogni agente infatti può eseguire 5 azioni (su, giù, sinistra, destra e fermo)

Ad esempio con 20 agenti il branching factor è 95367631640625.



A* per MAPF: Risultati su benchmark



Gli algoritmo completi e ottimi **non riescono a scalare**.

Trovare una soluzione ottima è un problema NP-hard.

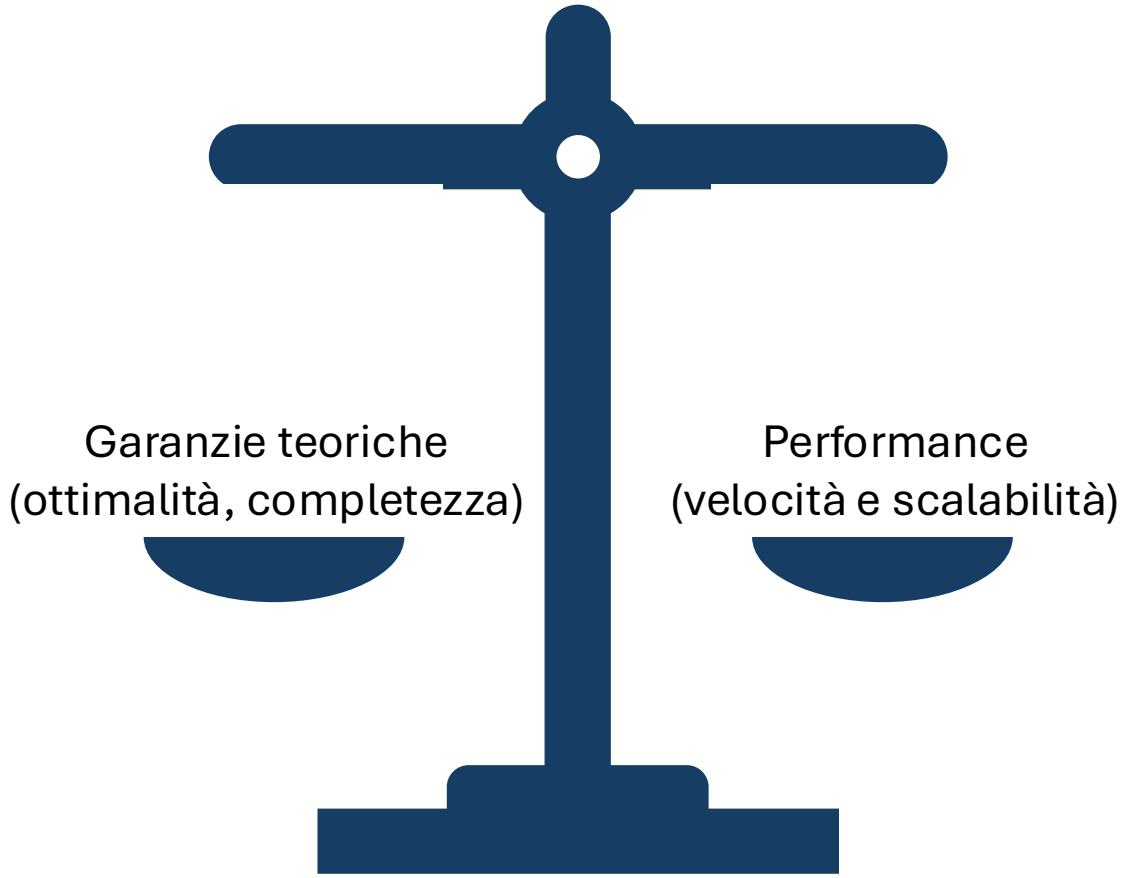
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

Tradeoff per gli algoritmi di MAPF

Gli algoritmi di MAPF devono bilanciare le **garanzie fornite** e le **performance.** Come abbiamo visto, A* ha delle garanzie teoriche molto forti che però impattano pesantemente sulle performance.

Domanda

Possiamo alleggerire queste garanzie e migliorare le performance?

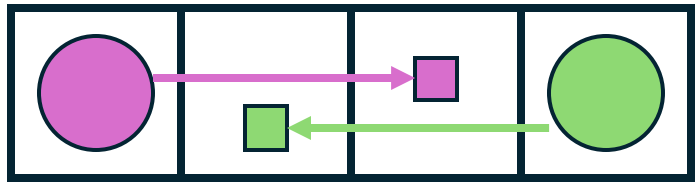


Rilassare la completezza

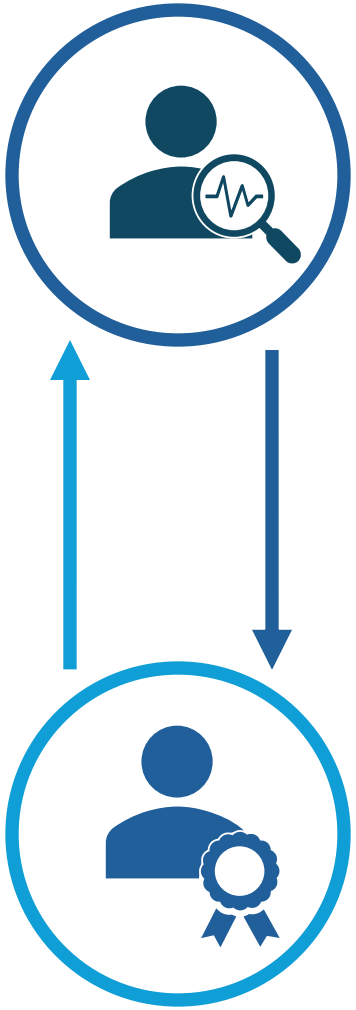
Completezza: l'algoritmo ritorna soluzioni in un tempo finito per tutte le istanze risolvibili; ~~altrimenti segnala lo che la soluzione non esiste~~

Ottimalità: l'algoritmo ritorna sempre la soluzione che ha costo minimo

Perdiamo la capacità di identificare le istanze irrisolvibili



Conflict-based Search (CBS)

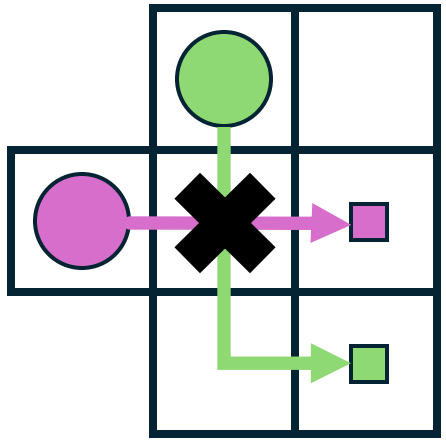


Ricerca ad alto livello

Ricerca a basso livello

Fonte: Sharon et al. Conflict-based search for optimal multi-agent pathfinding. *Artificial intelligence*, 2015.

Ricerca ad alto livello



Ricerca ad alto livello

Identifica i conflitti nella soluzione candidata

Richiede il percorso di un **singolo agente** che **evita i conflitti** identificati

Ricerca a basso livello

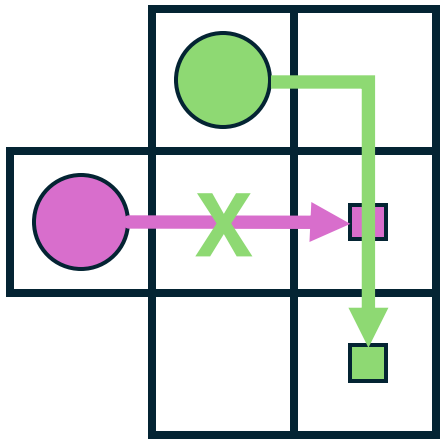
Ricerca a basso livello

Ricerca ad alto livello

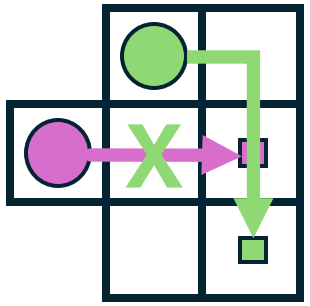
Ritorna il percorso trovato

Ricerca a basso livello

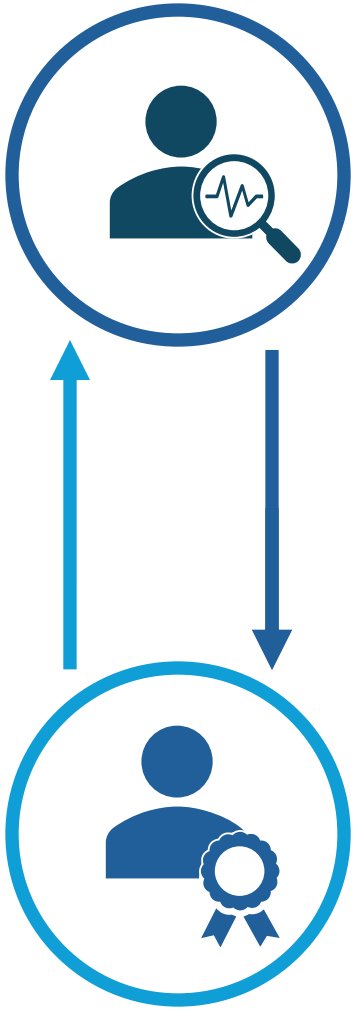
Trova un percorso che **soddisfa i vincoli** (evita i conflitti).



Conflict-based Search (CBS)

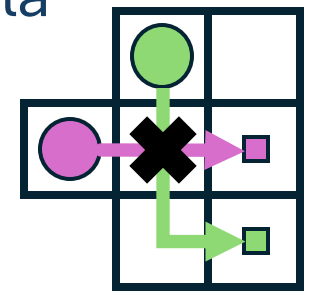


Ritorna il percorso trovato



Ricerca ad alto livello

Identifica i conflitti nella soluzione candidata

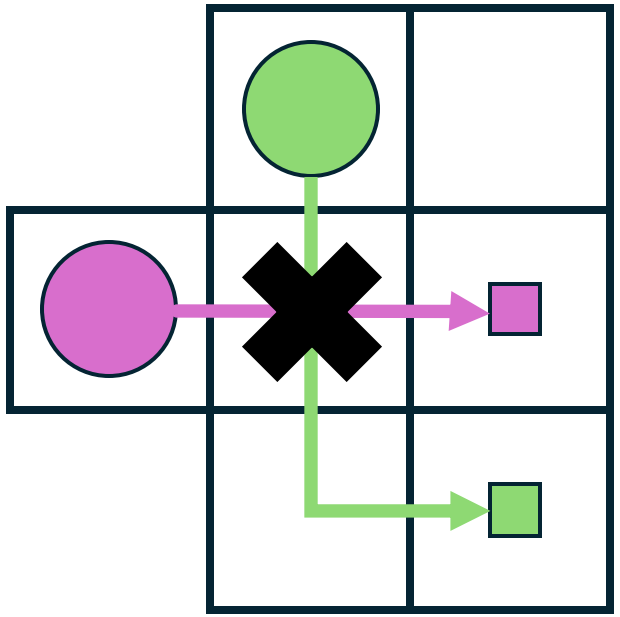


Richiede il percorso di un **singolo agente** che **evita i conflitti** identificati

Ricerca a basso livello

Trova un percorso che **soddisfa i vincoli** (evita i conflitti).

CBS: Esempio

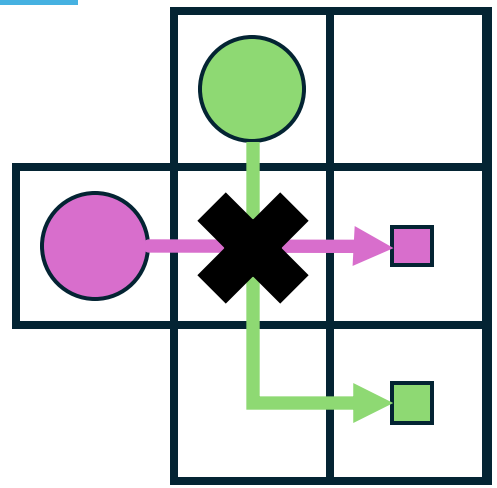


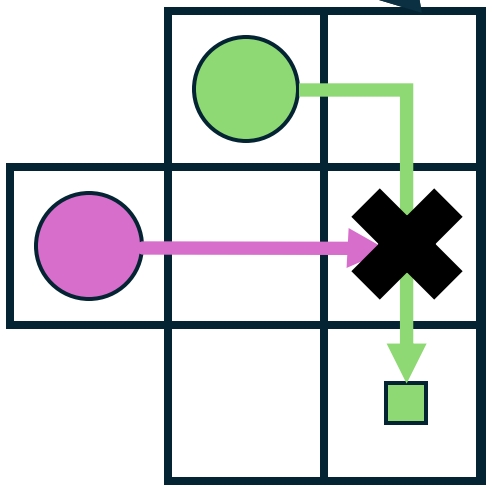
Piano con conflitti
Costo = 5

CBS: Esempio (2)

Replan Agente 1

All'istante t=1 l'agente 1 non può trovarsi nella cella che ha causato il conflitto (c4)

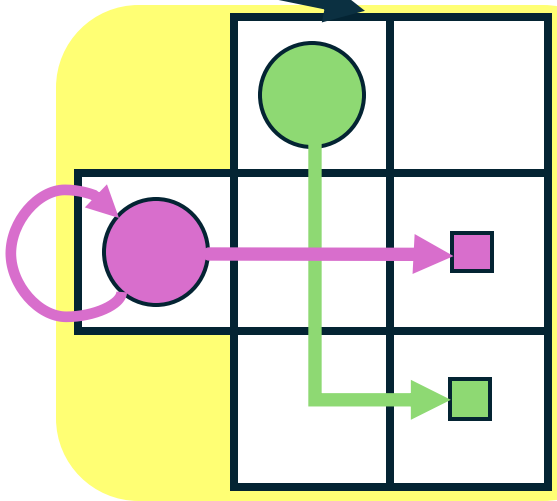




Costo = 5

Replan Agente 2

All'istante t=1 l'agente 2 non può trovarsi nella cella che ha causato il conflitto (c4)

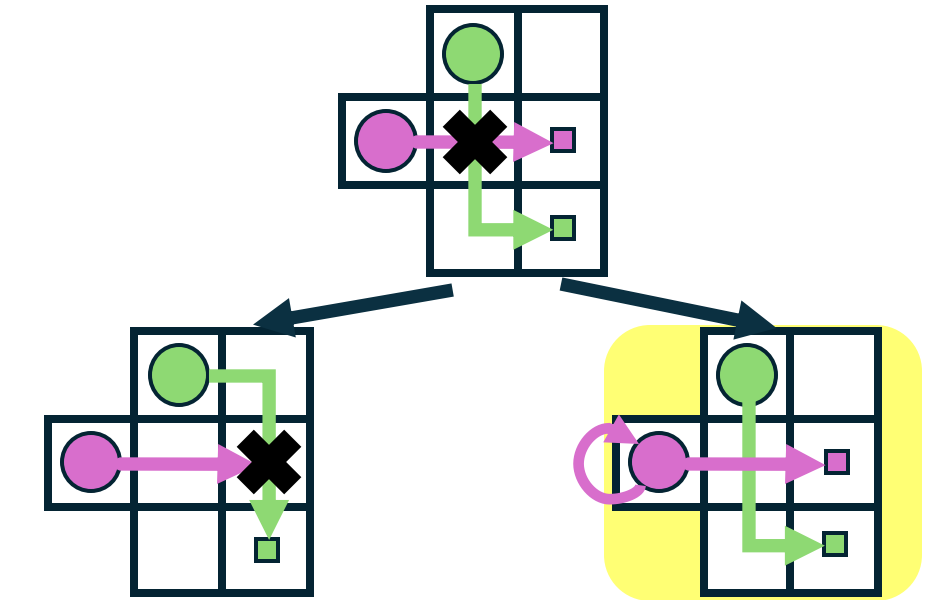


Costo = 6

CBS: Esempio (3)

Replan Agente 1

All’istante t=1 l’agente 1 non può trovarsi in c4 e a t=2 non può trovarsi in c5



Costo = 6

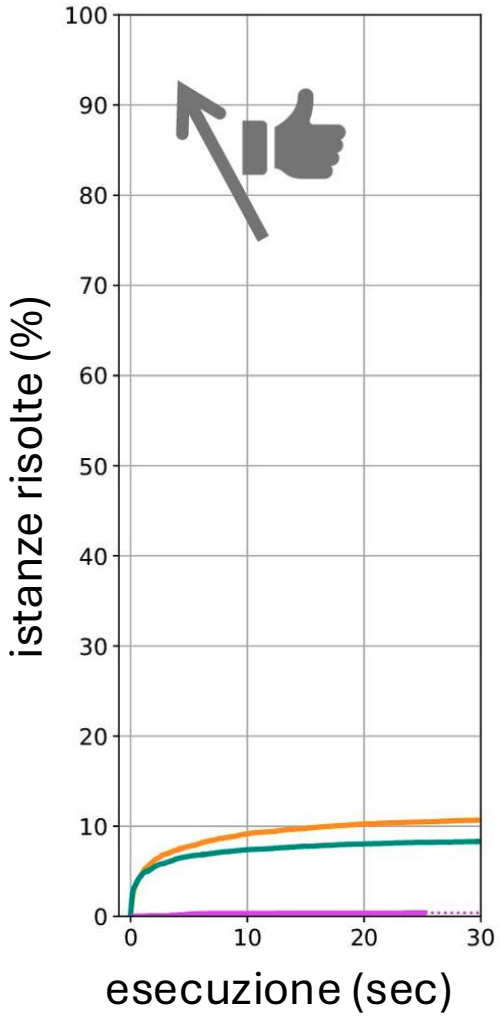
Costo = 6

Costo = 6

Replan Agente 2

All’istante t=1 l’agente 1 non può trovarsi in c4 e all’istante t=2 l’agente 2 non può trovarsi in c5

CBS: Risultati su benchmark



10.7% BCP [lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

Rilassare l'ottimalità

Completezza: l'algoritmo ritorna soluzioni in un tempo finito per tutte le istanze risolvibili; altrimenti segnala la che la soluzione non esiste

Ottimalità: l'algoritmo ritorna sempre la soluzione che ha ~~costo minimo~~

Permettiamo soluzioni che hanno un costo sub-ottimo.
In particolare accettiamo soluzioni con un **costo sub-ottimo vincolato**:

$$c(path) \leq w \cdot C_{opt} \quad (w \geq 1)$$

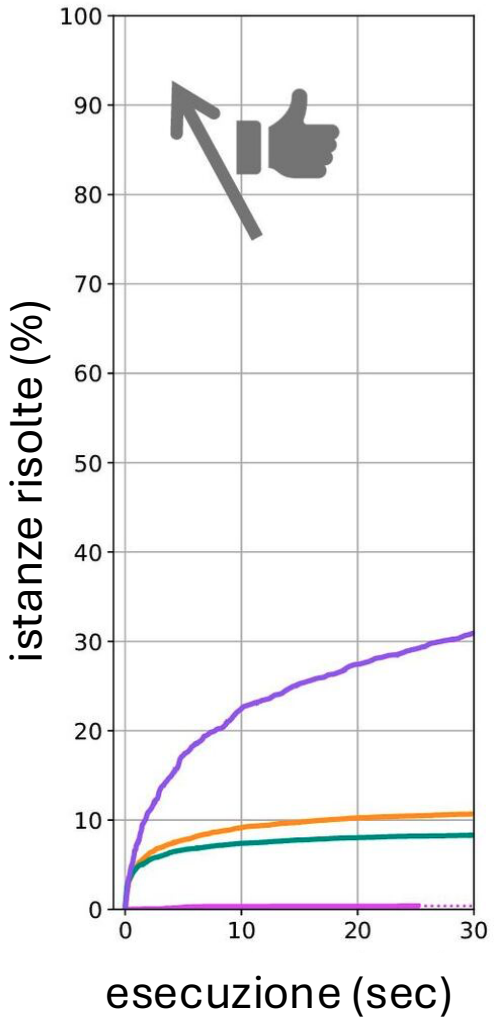
wA*

- $wA^\star$ **generalizza** $A^\star$ cambiando la funzione impiegata per decidere quale nodo espandere prioritariamente; dato $w \geq 1$:

$$f(n) = g(n) + w \cdot h(n)$$

  - le soluzioni prodotte sono w-ammissibili;
  - se $w = 1$, $wA^\star = A^\star$;
  - se $w \to \infty$, $wA^\star \to$ GBFS (i nodi prioritarizzati unicamente secondo $h(n)$)

wA* per MAPF: Risultati su benchmark



Variante di wA*

30.9% I-ODrM*-5 [Wagner+AIJ-15]	completo	w-subottimo
10.7% BCP [lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

Rilassare la completezza e l'ottimalità

Completezza: l'algoritmo ritorna soluzioni in un tempo finito per tutte le istanze risolvibili; ~~altrimenti segnala la che la soluzione non esiste~~

Ottimalità: l'algoritmo ritorna sempre la soluzione che ha ~~costo minimo~~

Perdiamo la capacità di identificare le istanze irrisolvibili
e accettiamo soluzioni con un costo sub-ottimo vincolato

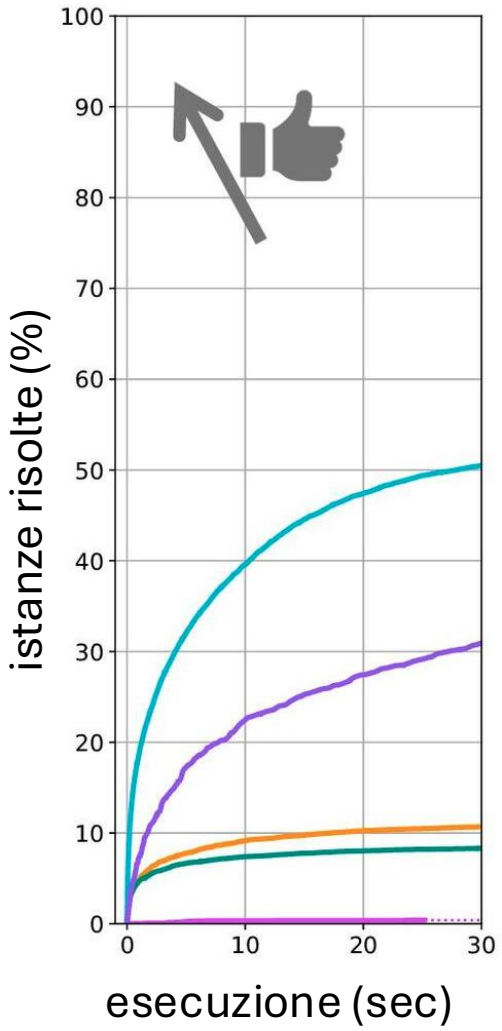
Explicit Estimation Search (EES)

- Ingredienti per EES:
  - $h(n)$, euristica ammissibile *cost-to-go*;
  - $\hat{h}(n)$, euristica non ammissibile *cost-to-go*;
  - $\hat{d}(n)$, euristica non ammissibile *distance-to-go*.

- Per selezionare quale nodo espandere vengono usate le funzioni:

costo : $f(n) = g(n) + h(n)$;
costo : $\hat{f}(n) = g(n) + \hat{d}(n)$;
distanza : $\hat{d}(n)$;

EES+CBS: Risultati su benchmark



EES + CBS

50.5% EECBS-5 [Li+AAAI-21]	completo per soluzioni	w-subottimo
30.9% I-ODrM*-5 [Wagner+AIJ-15]	completo	w-subottimo
10.7% BCP [lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

Rinunciare ad ogni garanzia

~~**Completezza:** l'algoritmo ritorna soluzioni in un tempo finito per tutte le istanze risolvibili; altrimenti segnala che la soluzione non esiste~~

~~**Ottimalità:** l'algoritmo ritorna sempre la soluzione che ha costo minimo~~

Prioritized Planning (PP)

Algorithm 1 PP

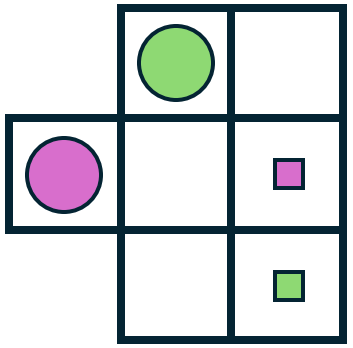
Input:

- $G = \langle V, E \rangle$: il grafo
- s_i: stato iniziale
- s_G: goal
- $P = [a_1, a_2, \ldots, a_m]$: ordine di priorità di m agenti

Output:

- $\{\pi_1, \pi_2, ..., \pi_m\}$ (piano senza collisioni)

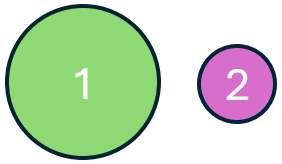
```
1: R ← reservation table vuota
2: for k = 1..m(seguendo P) do
3:     π_k ← A* su spazio-tempo che evita R
4:     if A* su spazio-tempo fallisce then return null
5:     end if
6:     update(R, π_k)
7: end for
```

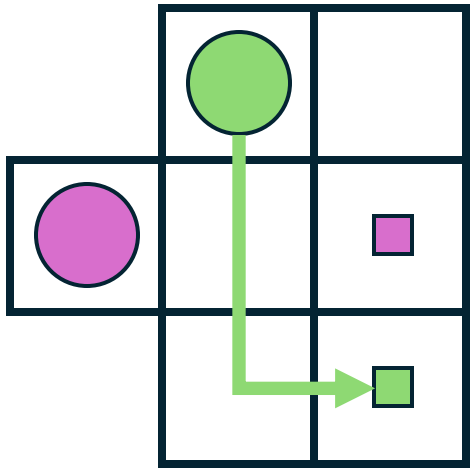


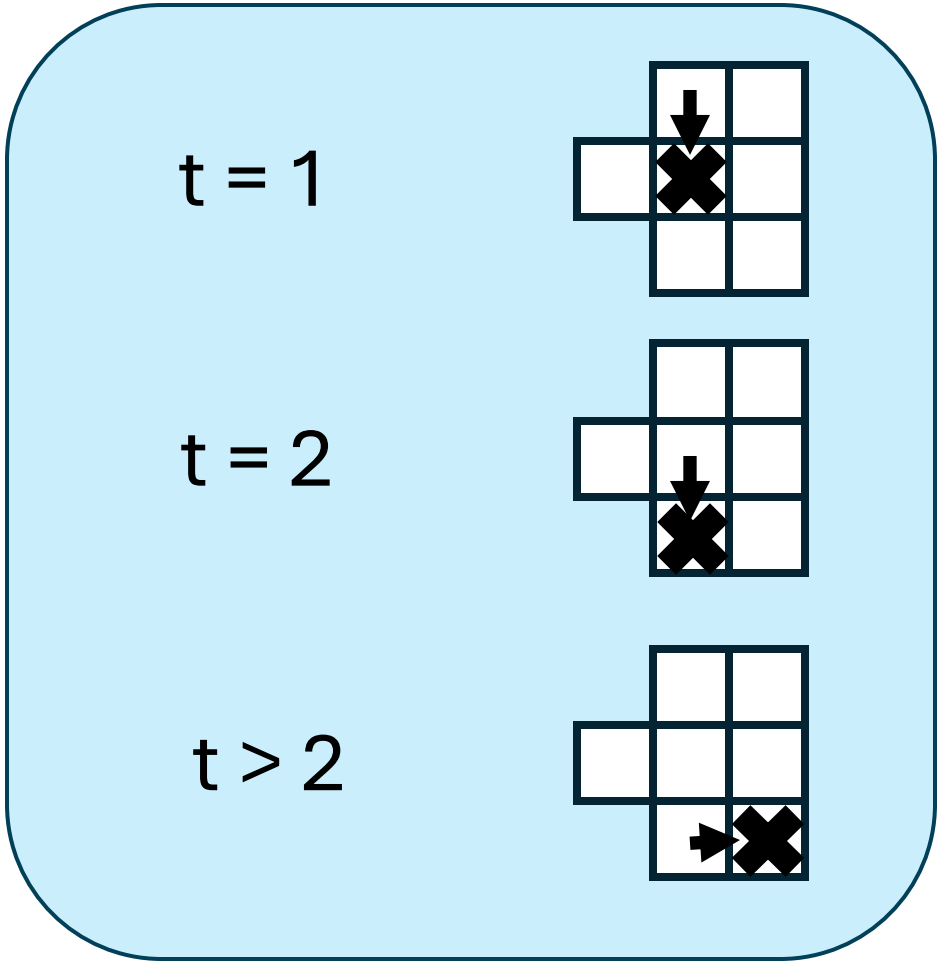
P = 1 2

$update(R, \pi_k)$ inserisce una tupla $\langle arco, vertice, istante \rangle$ per ogni passo del percorso π_k.

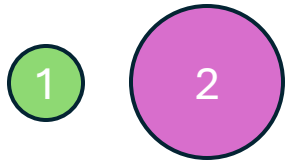
PP: Esempio

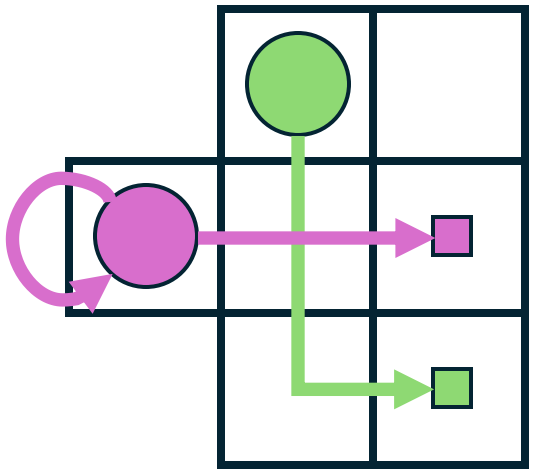
P = 

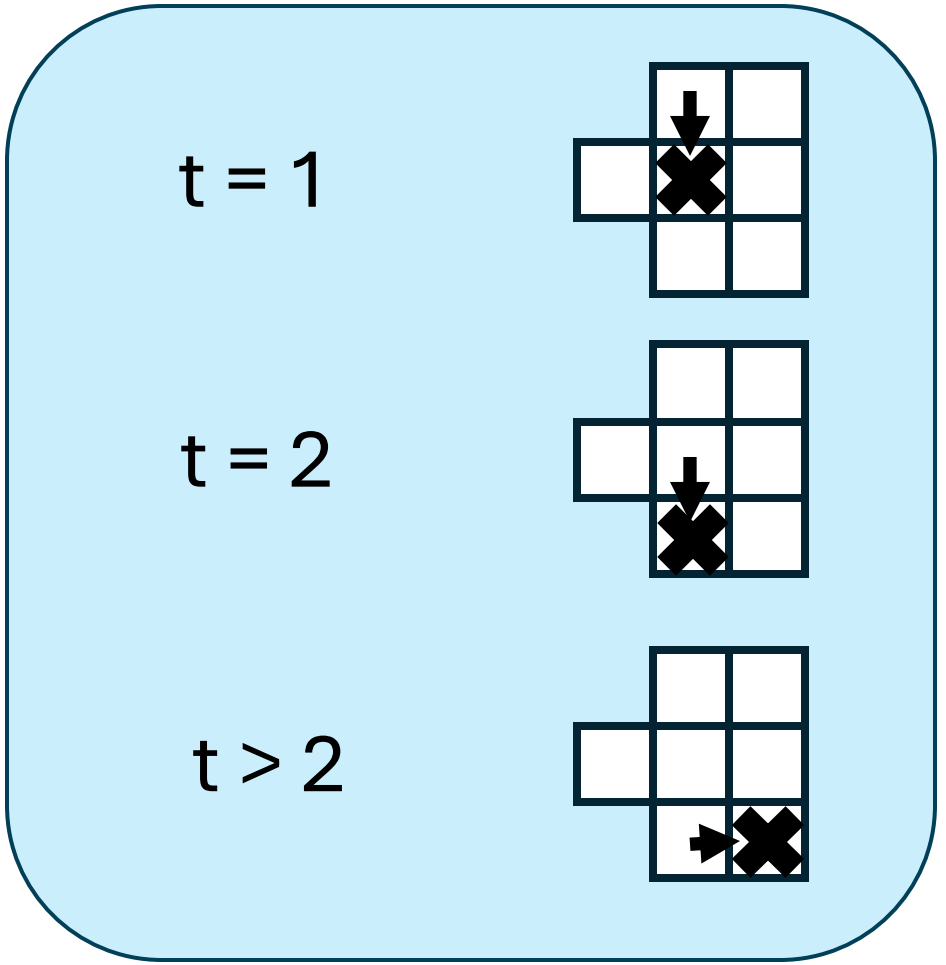


R = 

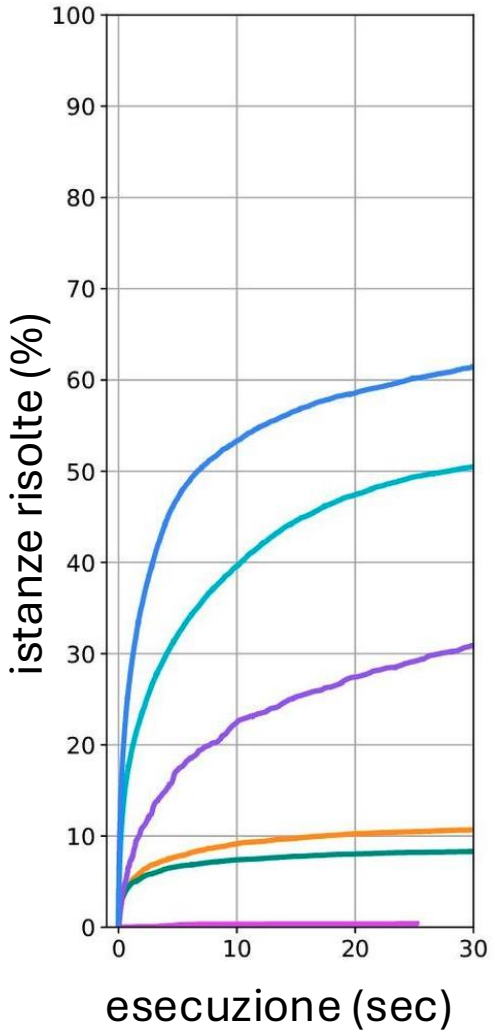
PP: Esempio

P = 



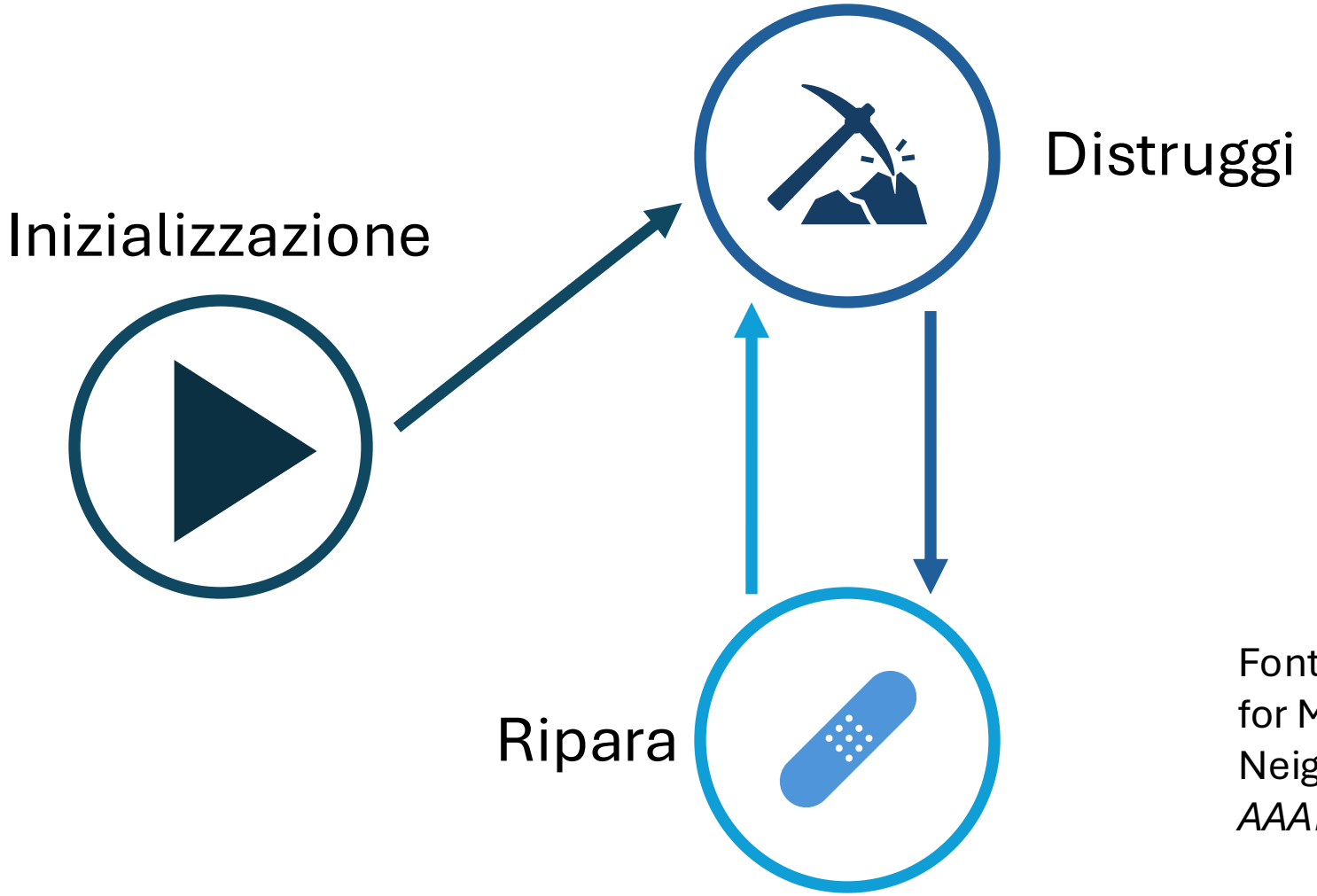
R = 

PP: Risultati su benchmark



61.4% PP [Silver AIIDE-05]	non competo	subottimo
50.5% EECBS-5 [Li+AAAI-21]	completo per soluzioni	w-subottimo
30.9% I-ODrM*-5 [Wagner+AIJ-15]	completo	w-subottimo
10.7% BCP [lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

Large Neighborhood Search: MAPF-LNS2



Fonte: Li et al. MAPF-LNS2: Fast Repairing for Multi-Agent Path Finding via Large Neighborhood Search. *AAAI-2022*.

MAPF-LNS2: Inizializzazione

Inizializzazione

All’inizio, MAPF-LNS2 chiama un **algoritmo di MAPF** per risolvere l’istanza e ottiene un piano che **può contenere dei conflitti**.
Se per qualche agente non è stato calcolato un piano, MAPF-LNS2 calcola il **piano** che **minimizza il numero di collisioni** con i percorsi esistenti per ognuno di questi agenti.

MAPF-LNS2: Selezione degli agenti

MAPF-LNS2 seleziona un **sottoinsieme degli agenti** utilizzando un **metodo di selezione del vicinato** di dimensione *N* per i quali ricalcolare i piani

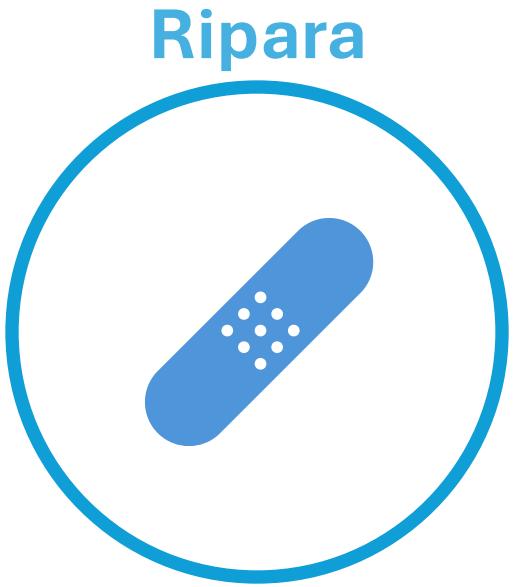
Metodi di selezione del vicinato

- **Collision-based**
- **Failure-based**
- **Random**

Distruggi

MAPF-LNS2: Ricalcolo dei piani

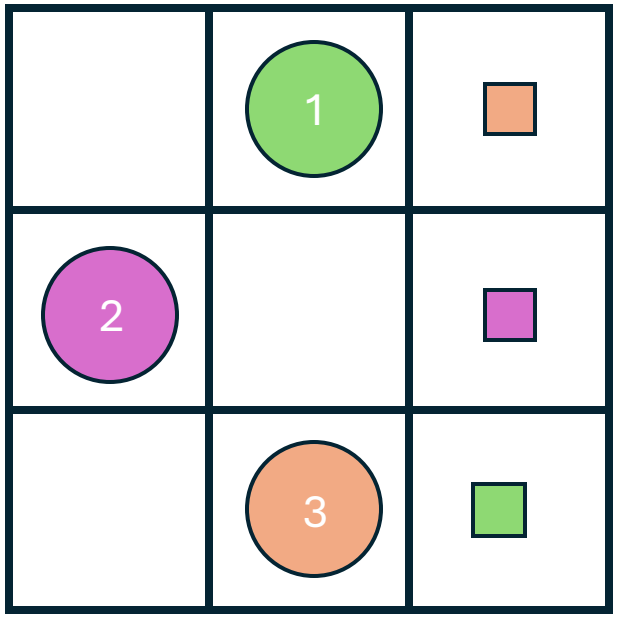
Una volta selezionato il vicinato, MAPF chiama una **versione modificata di Prioritized Planning** in cui assegna una probabilità random a ciascun agente nel vicinato e **ricalcola i piani** cercando di **minimizzare** sia il numero delle collisioni tra i nuovi piani che il numero di collisioni tra i nuovi piani e quelli che non sono stati selezionati nel vicinato.



MAPF-LNS2: Grafo delle collisioni

Dato un insieme di agenti A e l'insieme dei loro percorsi P, possiamo definire il **grafo delle collisioni**:

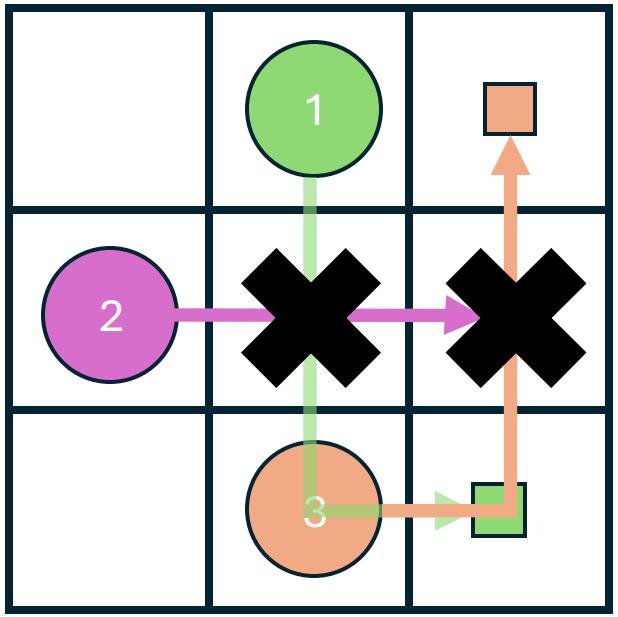
$$G_c = \langle V_c, E_c \rangle$$
$$V_c = \{i | a_i \in A\}$$
$$E_c = \{(i,j) | p_i \in P \text{ collide con } p_j \in P\}$$



MAPF-LNS2: Grafo delle collisioni

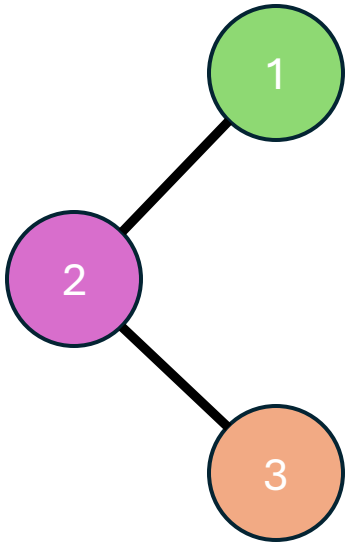
Dato un insieme di agenti A e l'insieme dei loro percorsi P, possiamo definire il **grafo delle collisioni**:

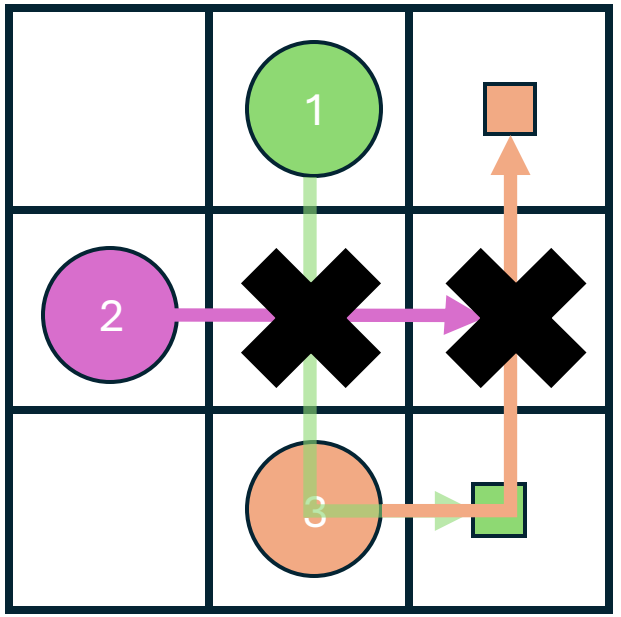
$$G_c = \langle V_c, E_c \rangle$$
$$V_c = \{i | a_i \in A\}$$
$$E_c = \{(i,j) | p_i \in P \text{ collide con } p_j \in P\}$$



MAPF-LNS2: Grafo delle collisioni (2)

Dato un insieme di agenti A e l'insieme dei loro percorsi P, possiamo definire il **grafo delle collisioni**:

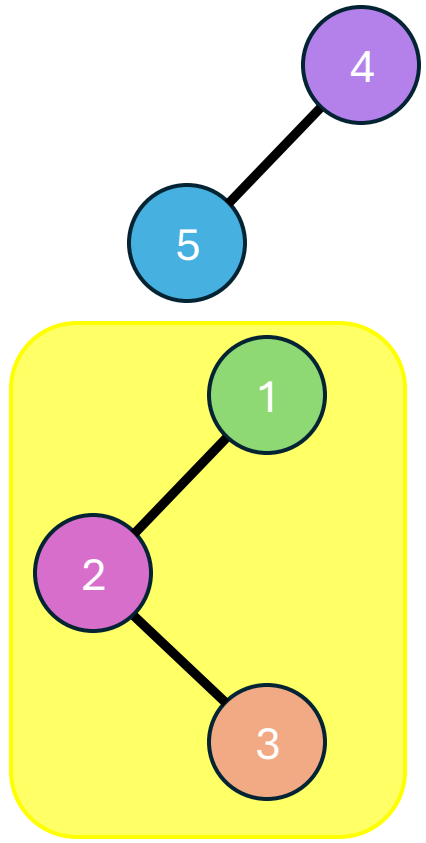




MAPF-LNS2: Selezione collision-based

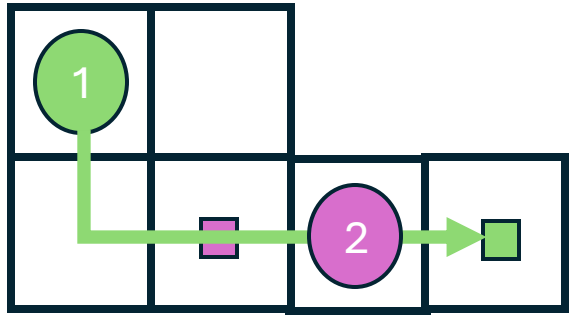
Selezioniamo un vertice casuale $v_i \in V_c$ con $\deg(i) > 0$ e troviamo il più grande sottografo connesso $G'_c = \langle V'_c, E'_c \rangle$

- Se $|V'_c| \leq N$ allora selezioniamo tutti gli agenti $a_v | \, v \in V'_c$. Poi, iterativamente, selezioniamo un nodo u $\in V'_c$ ed effettuiamo una random-walk sul grafo delle collisioni G_c fino a che non troviamo un nodo che non appartiene a V'_c e lo aggiungiamo alla lista.
- Altrimenti effettuiamo un random-walk su G'_c fino a che non visitiamo *N* nodi

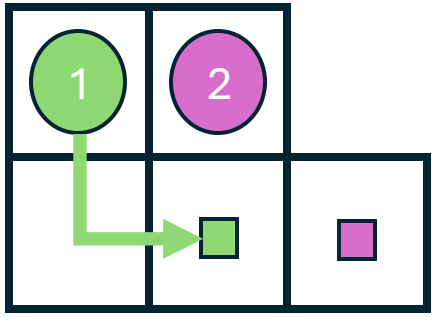


MAPF-LNS2: Selezione failure-based

Analizziamo due casi noti per cui uno specifico agente ha fallito:



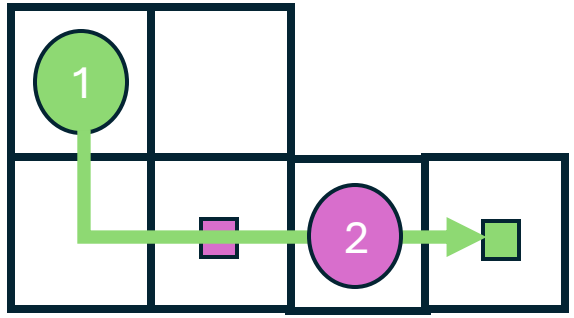
- L'agente a_i è **bloccato** da altri agenti che hanno già raggiunto il proprio goal e **non può raggiungere** il proprio goal g_i



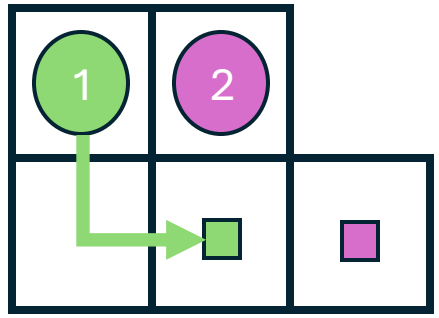
- L'agente a_i è **"scavalcato"** da altri percorsi che gli **impediscono di muoversi** dalla sua posizione s_i

MAPF-LNS2: Selezione failure-based (2)

Selezioniamo un nodo con un nodo dal grafo delle collisioni v_i con $\deg(i)$ alto e per questo **creiamo due liste** che identificano i due tipi di fallimento:



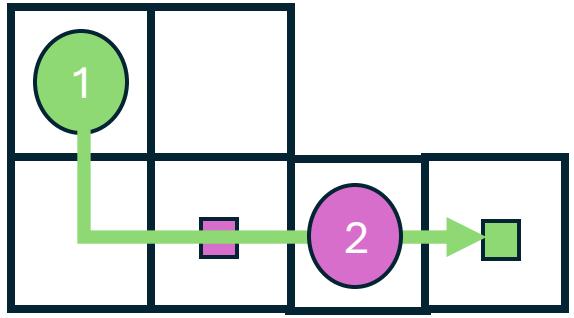
- $A^s = \{a_j \in A | p_j \in P \text{ visita } s_i\}$
- $A^g = \{a_j \in A | p \text{ visita } g_i\}$ dove p è il percorso da s_i a g_i che minimizza $|A^g|$

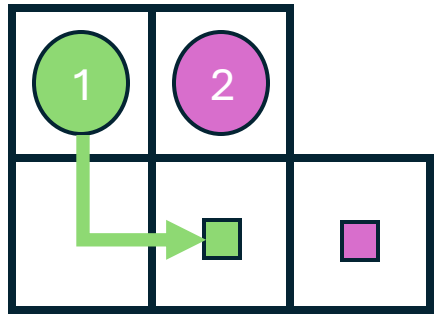


MAPF-LNS2: Selezione failure-based (3)

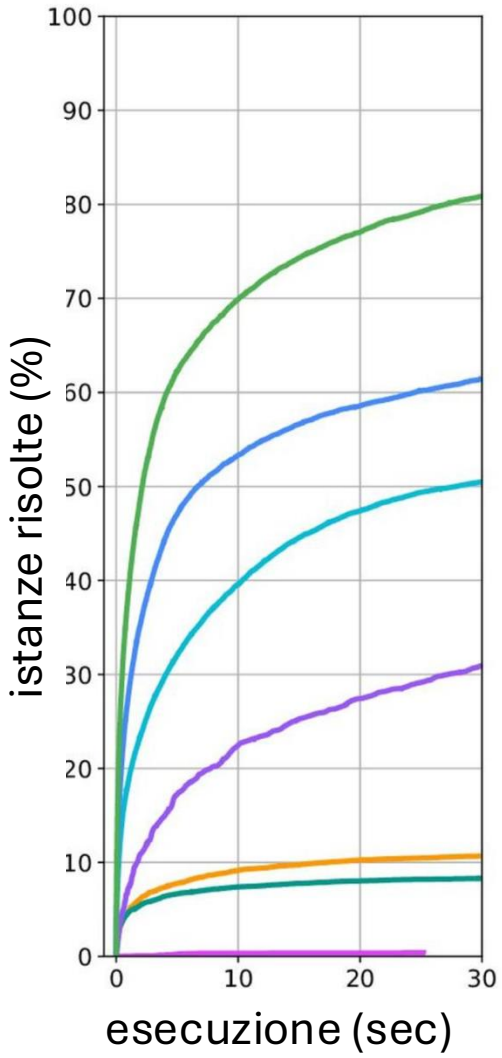
Effettuiamo l'unione delle due liste. Avremo 3 casi diversi:

- $|A^s \cup A^g| = 0$: Esiste un percorso che non causa conflitti: l'agente a_i aspetta che tutti gli altri abbiano finito e raggiunge il suo goal seguendo il percorso p.
- $|A^s \cup A^g| < N - 1$: Ritorniamo tutti gli agenti nell'unione e aggiungiamo agenti il cui goal è presente in almeno uno dei percorsi degli agenti nella lista unita.
- $|A^s \cup A^g| \geq N - 1$: Aggiungiamo agenti random dalla lista dando la precedenza a quelli in A^g





MAPF-LNS2: Risultati su benchmark

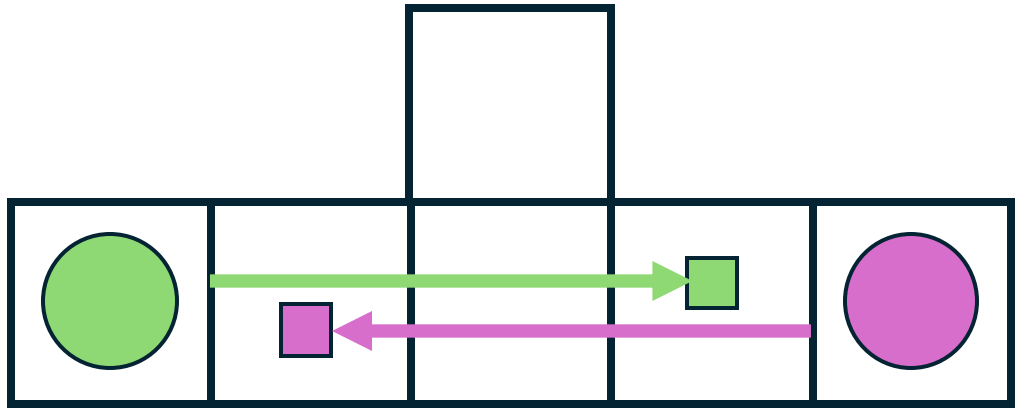


80.9% MAPF-LNS2 [Li+ AAAI-22]	non completo	subottimo
61.4% PP [Silver AIIDE-05]	non competo	subottimo
50.5% EECBS-5 [Li+AAAI-21]	completo per soluzioni	w-subottimo
30.9% I-ODrM*-5 [Wagner+AIJ-15]	completo	w-subottimo
10.7% BCP [Lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

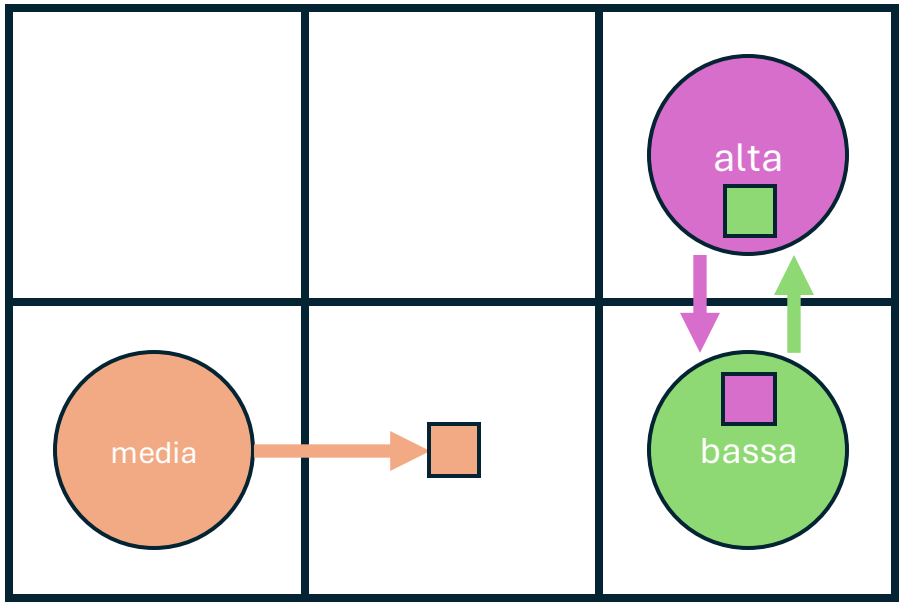
PP e MAPF-LNS2: Debolezze

Problema

PP e MAPF-LNS2 non riescono a risolvere alcune istanze che possono sembrare molto semplici



Priority Inheritance with BackTracking (PIBT)

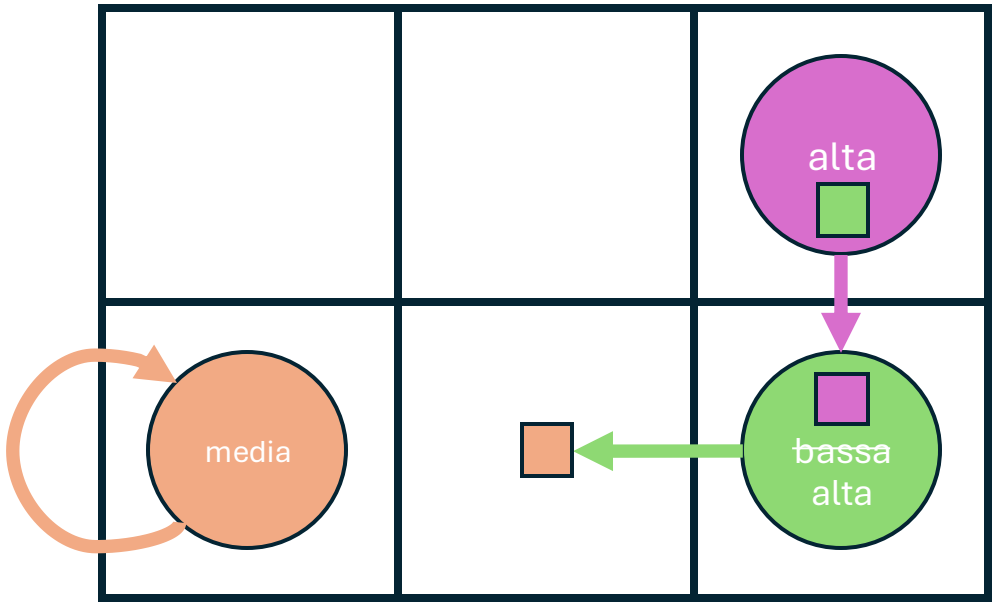


In questo caso, seguendo l'ordine di priorità, l'agente con **priorità alta** decide di rimanere fermo aspettando che si liberi la cella sotto, l'agente con **priorità media** raggiunge il proprio goal mentre l'agente con **priorità bassa** rimane **bloccato**.

PIBT: Algoritmo

PIBT pianifica **un passo per volta** per tutti gli agenti. Ad ogni iterazione PIBT:

1. assegna una **priorità** a ciascun agente;
2. gli agenti provano a scegliere il **prossimo nodo**
3. se il prossimo nodo è occupato da un agente di priorità più bassa, scatta la **priority inheritance**: il più importante "chiede strada" e l'altro pianifica prima;
4. se una scelta porta in stallo, l'algoritmo fa **backtracking** e prova un'alternativa



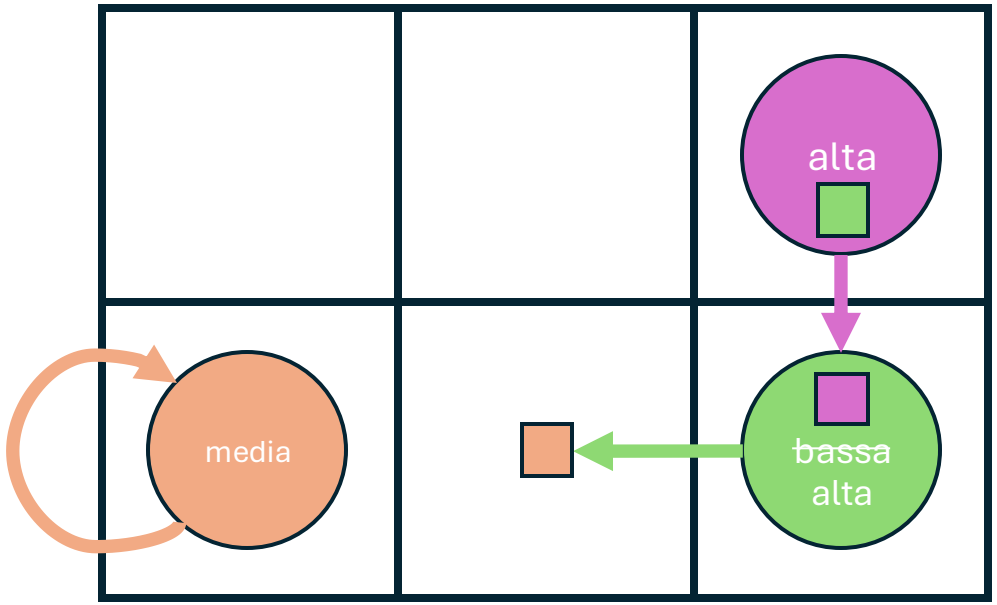
PIBT: Priorità

Come primo passo PIBT assegna una priorità a ciascun agente calcolata come:

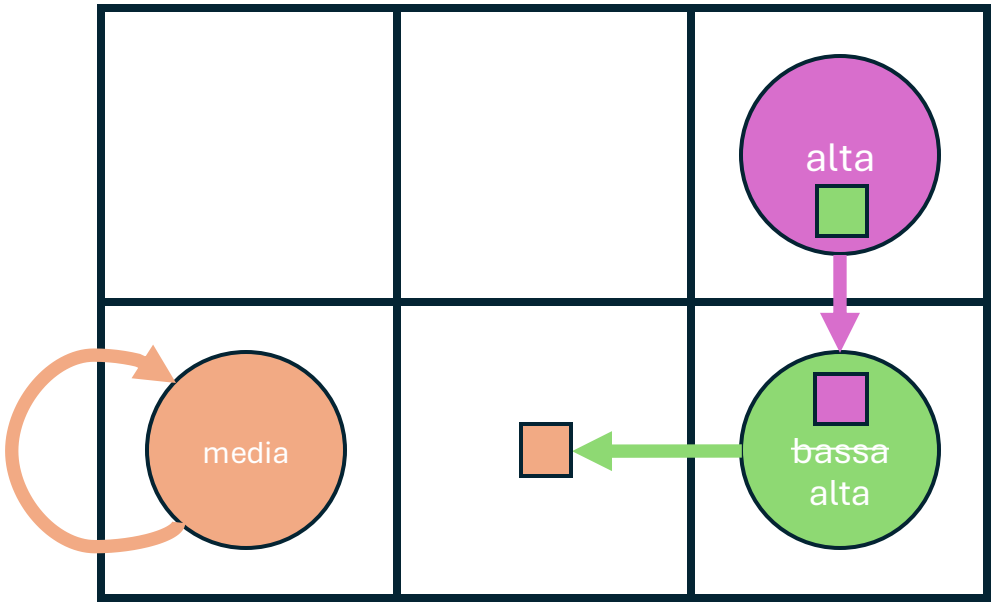
$$p_i(t) = \eta_i(t) + \epsilon_i$$

Dove:

- $\eta_i(t)$ è **l'età del task** dell'agente i ovvero quanti istanti sono trascorsi dall'assegnamento del task
- ϵ_i è un **valore unico** per l'agente i che serve per risolvere i pareggi tra priorità



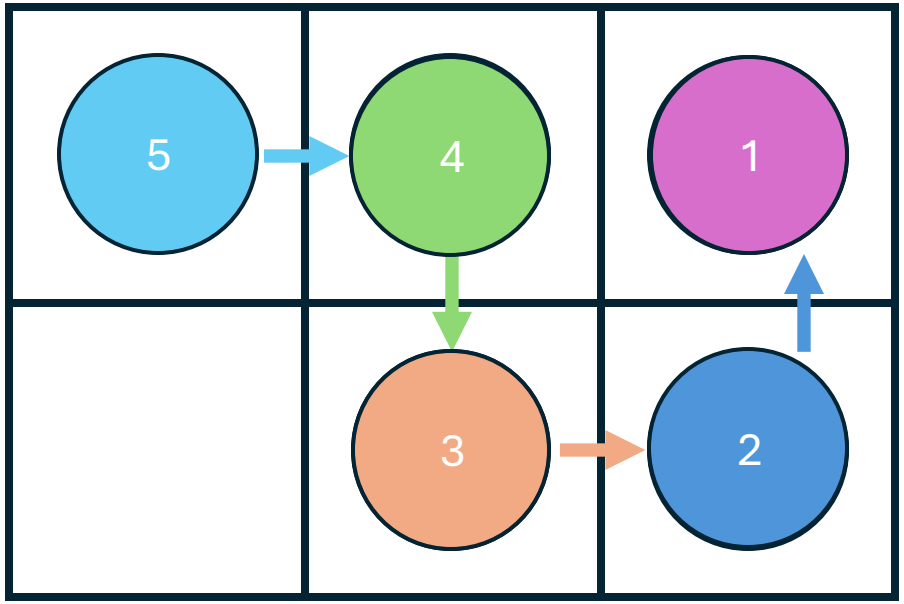
PIBT: Priority inheritance



Con PIBT, un agente con priorità bassa che blocca un agente con priorità maggiore **eredita** la priorità (**priority inheritance**) di quest'ultimo e esegue la propria azione **prima di lui**

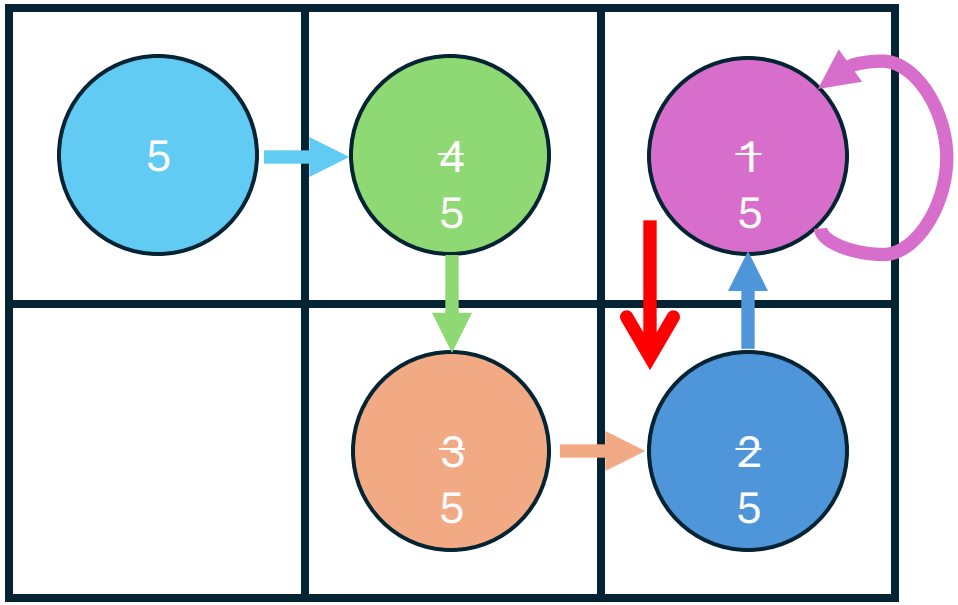
PIBT: Backtracking

L'ereditarietà della priorità non è sempre sufficiente a risolvere i **deadlock**. Per questo PIBT, quando un agente non ha **nessuna mossa disponibile**, implementa un meccanismo di **backtracking**.



PIBT: Backtracking (2)

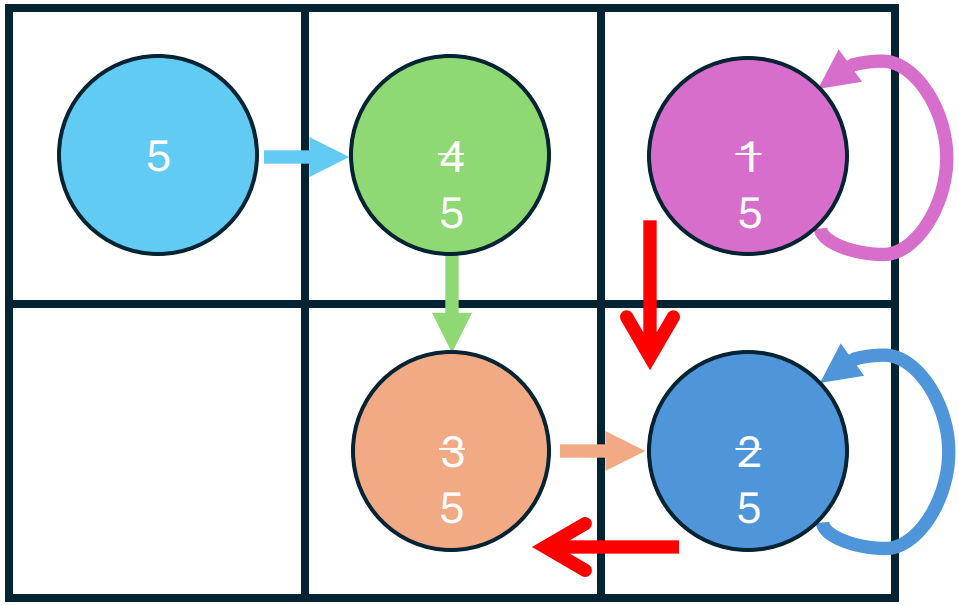
L'agente con **priorità 1** non ha nessuna mossa disponibile quindi ritorna il valore «**invalid**» al suo **predecessore** e **si ferma** nella sua posizione.



↓ = **invalid**, nessuna mossa valida, mi fermo e aspetto

PIBT: Backtracking (3)

Anche l'agente con **priorità 2** non ha nessuna mossa disponibile quindi ritorna il valore «**invalid**» al suo **predecessore** e **si ferma** nella sua posizione.

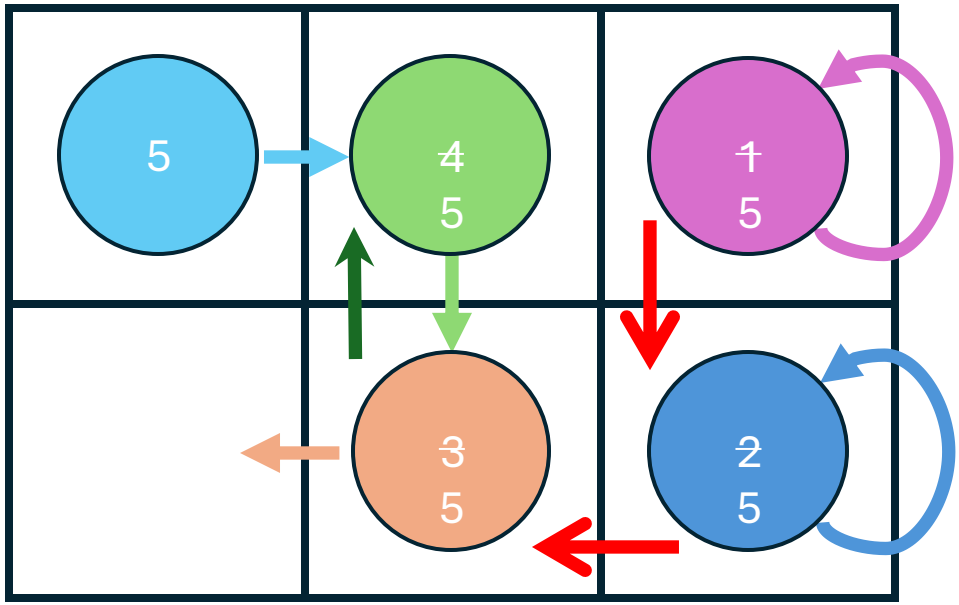


↓ = **invalid**, nessuna mossa valida, mi fermo e aspetto

PIBT: Backtracking (4)

L'agente con **priorità 3** ha ricevuto invalid ma può selezionare una **mossa possibile**. Dato che l'agente con **priorità 3** si può spostare, comunica «**valid**» al proprio **predecessore** e **si sposta**.

↓	=	**invalid**, nessuna mossa valida, mi fermo e aspetto
↓	=	**valid**, mi sono spostato, puoi spostarti nella cella



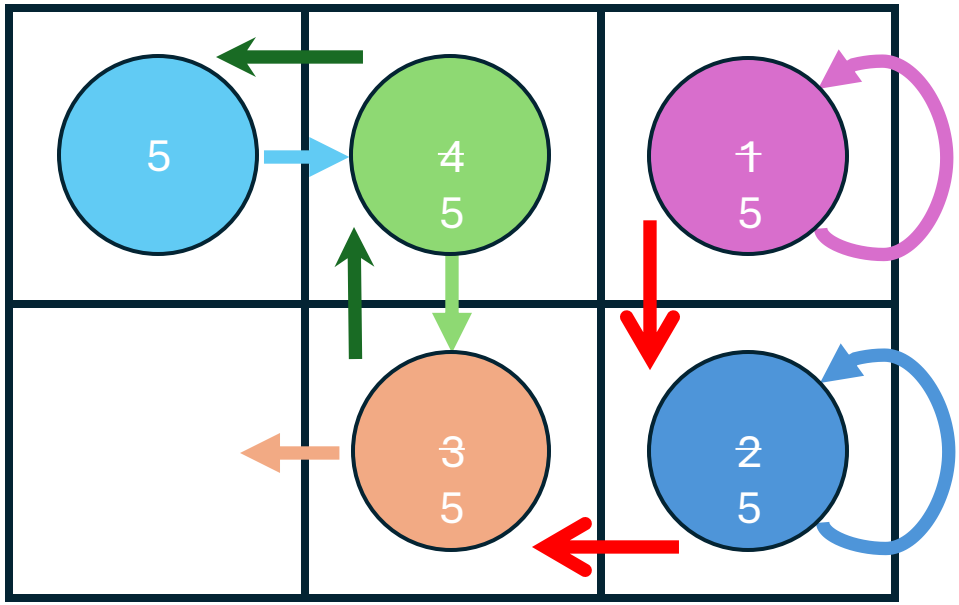
PIBT: Backtracking (5)

L'agente con **priorità 4** ha ricevuto valid e può fare la propria mossa. Comunica valid al proprio **predecessore** e **si sposta**.

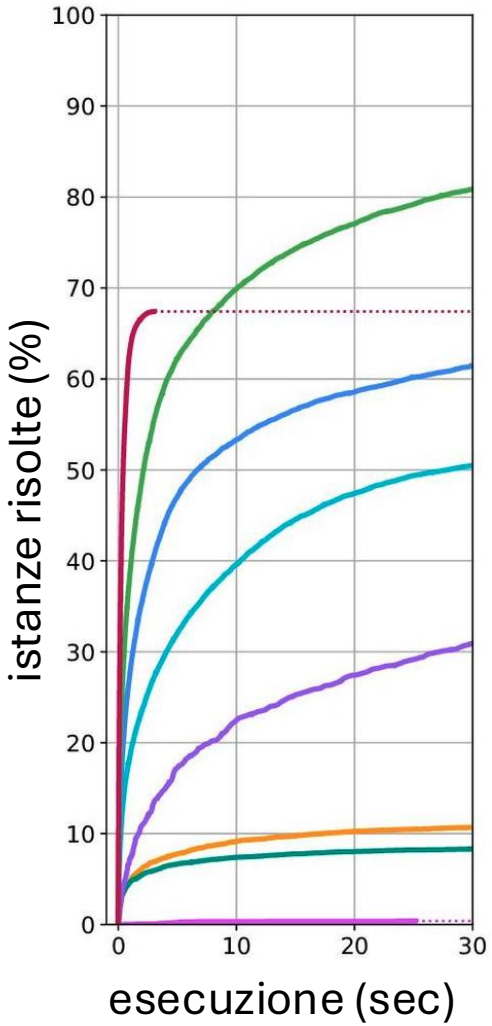
L'agente con **priorità 5** ha ricevuto valid e esegue la propria mossa.

↓ = **invalid**, nessuna mossa valida, mi fermo e aspetto

↓ = **valid**, mi sono spostato, puoi spostarti nella cella



PIBT: Risultati su benchmark

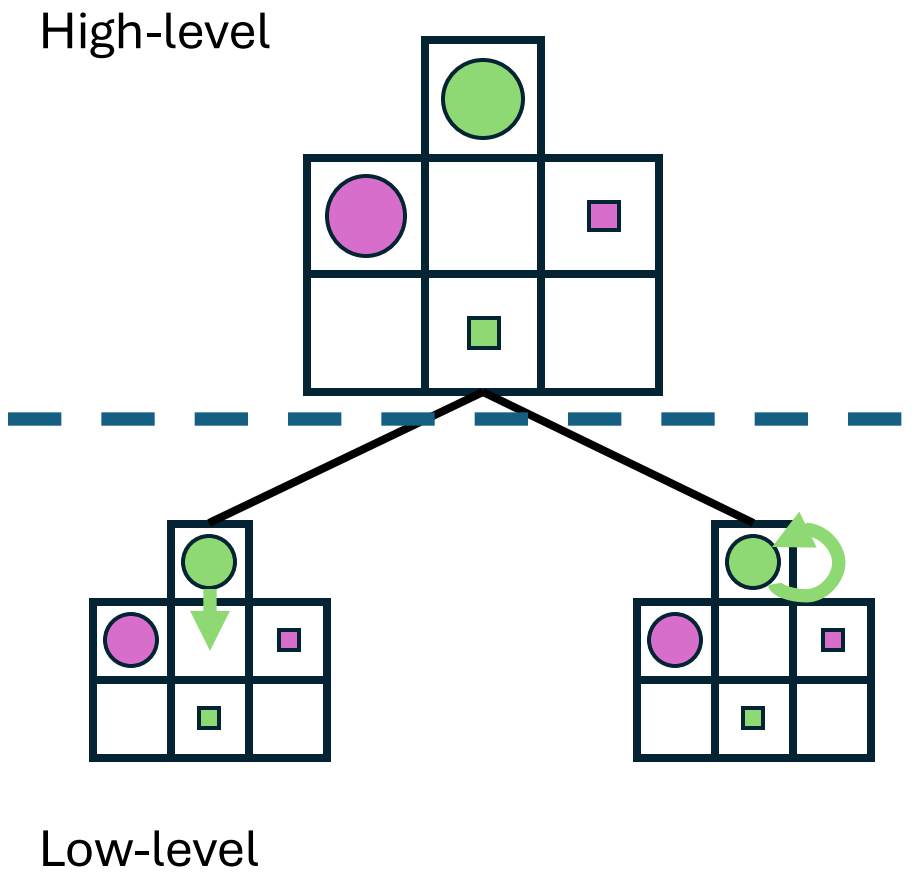


80.9% MAPF-LNS2 [Li+ AAAI-22]	non completo	subottimo
67.4% PIBT [Okumuts+ AIJ-22]	non completo	subottimo
61.4% PP [Silver AIIDE-05]	non completo	subottimo
50.5% EECBS-5 [Li+AAAI-21]	completo per soluzioni	w-subottimo
30.9% I-ODrM*-5 [Wagner+AIJ-15]	completo	w-subottimo
10.7% BCP [Lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

LaCAM

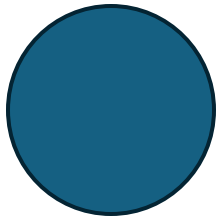
LACAM fa una **ricerca a due livelli** su **configurazioni**:

- **High-level:** esplora una *sequenza di configurazioni*; ogni nodo HL è una configurazione.
- **Low-level:** costruisce, in modo **lazy**, un piccolo albero di **vincoli** che dicono “l’agente i nel **prossimo** timestep deve andare in a (oppure in b, …)”. Da ogni foglia LL viene generata **una** nuova configurazione adiacente (un passo avanti per tutti) che rispetta quei vincoli.

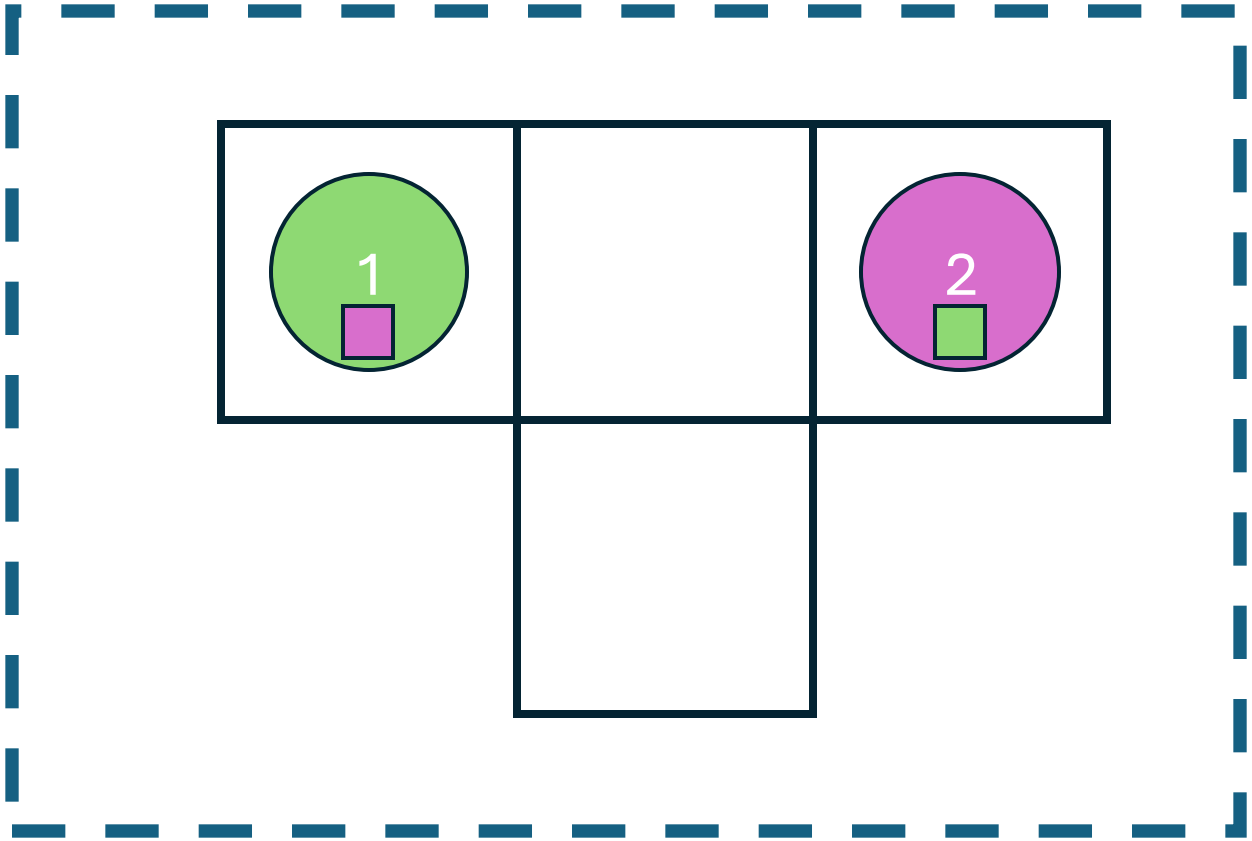


LaCAM: Esempio (1)

High-level:

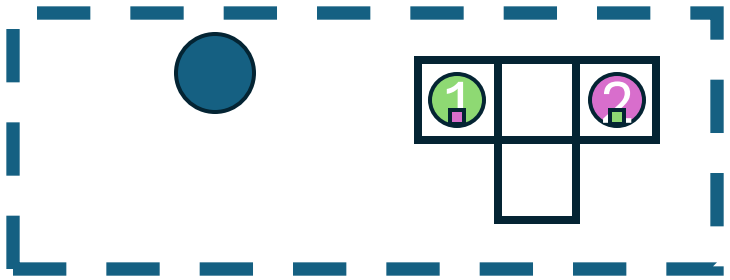


Situazione iniziale. Ho un solo nodo HN.
Effettuo una ricerca low-level

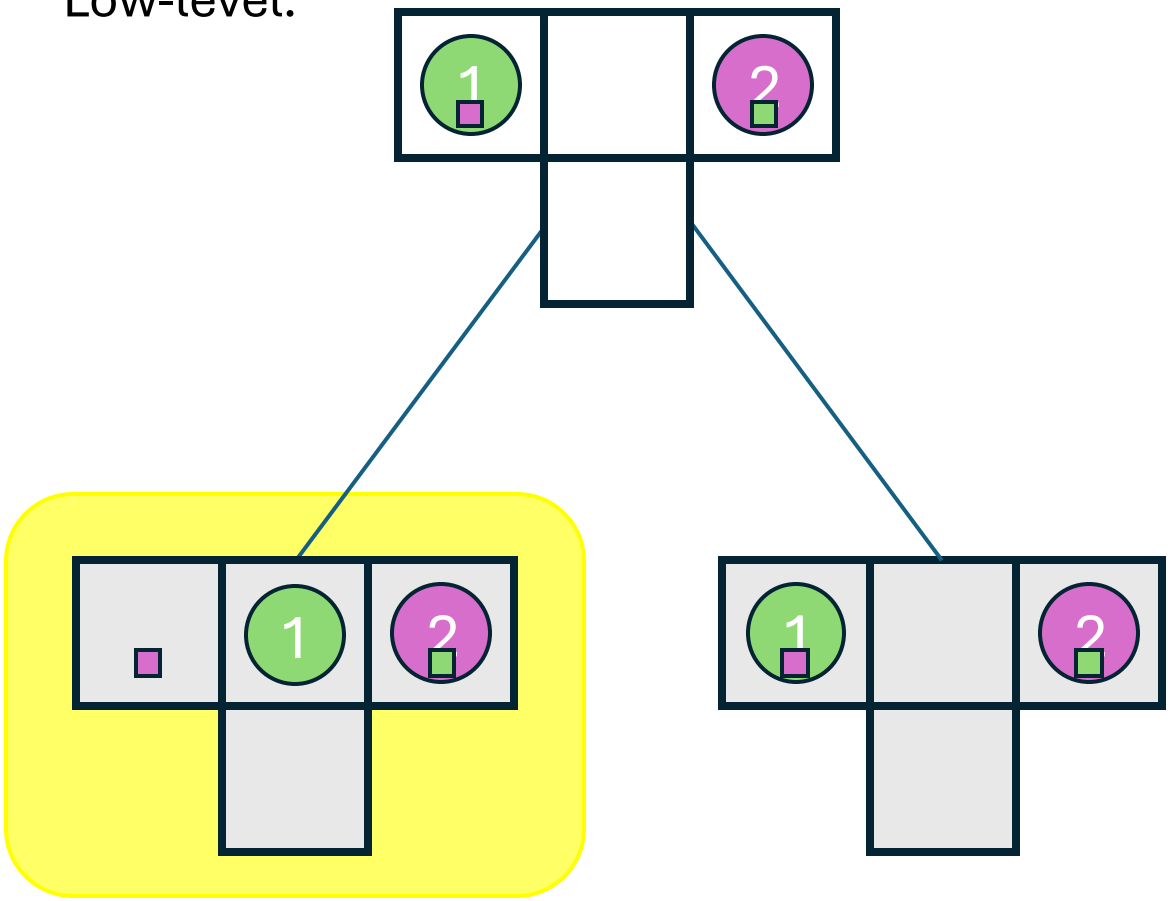


LaCAM: Esempio (2)

High-level:



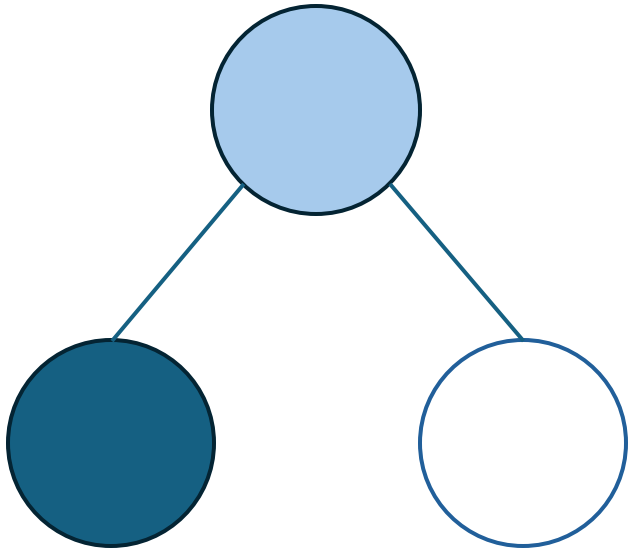
Low-level:



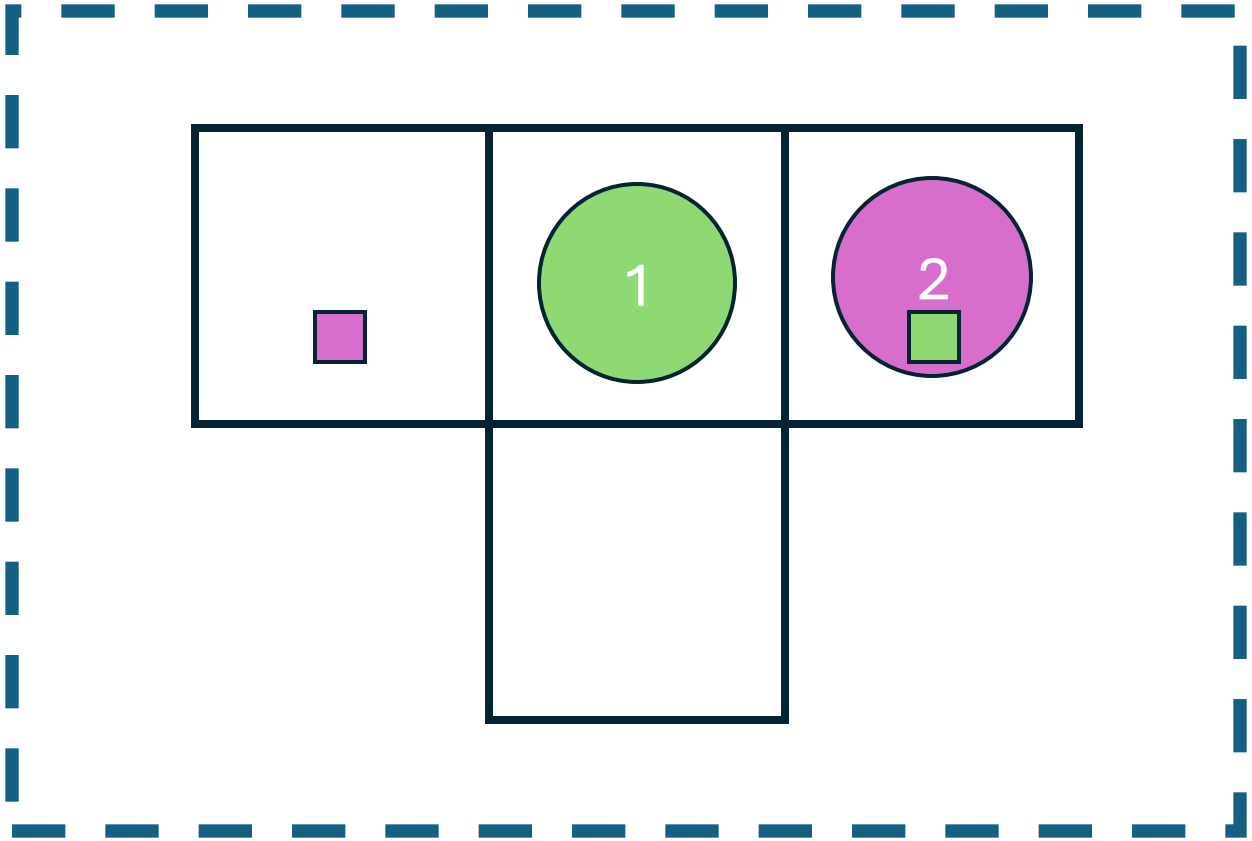
Non ho vincoli nel nodo ad alto livello. Scelgo un agente casuale da espandere (agente 1). Genero i **successori** per l'agente 1 ed esploro il primo utilizzando un euristica **breadth-first**. Dato che il primo figlio è una **configurazione nuova**, lo **aggiungo ai nodi high-level** e mi fermo.

LaCAM: Esempio (3)

High-level:

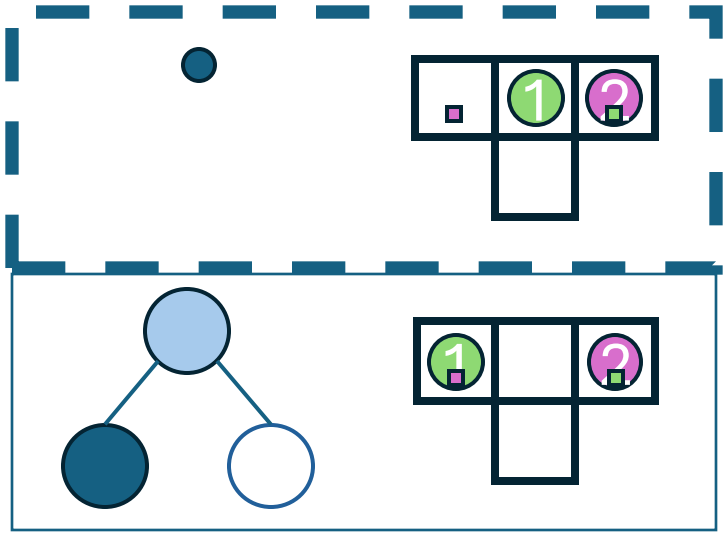


Espando la nuova configurazione trovata

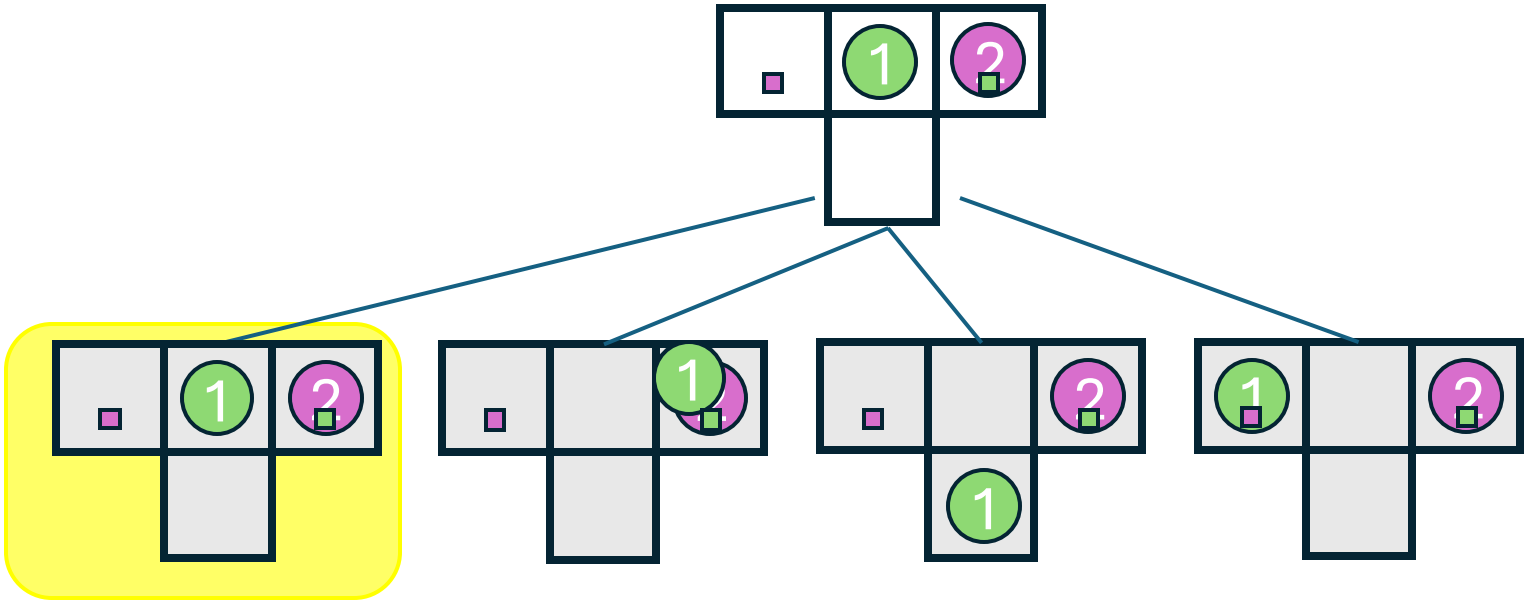


LaCAM: Esempio (4)

High-level:



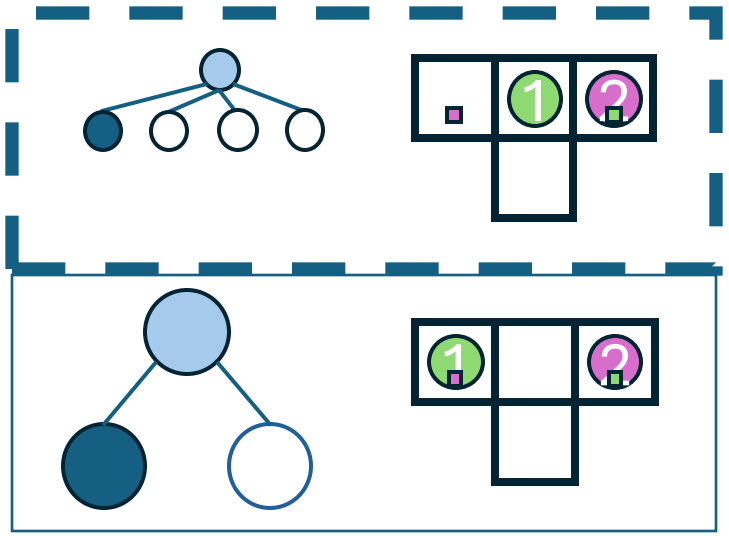
Low-level:



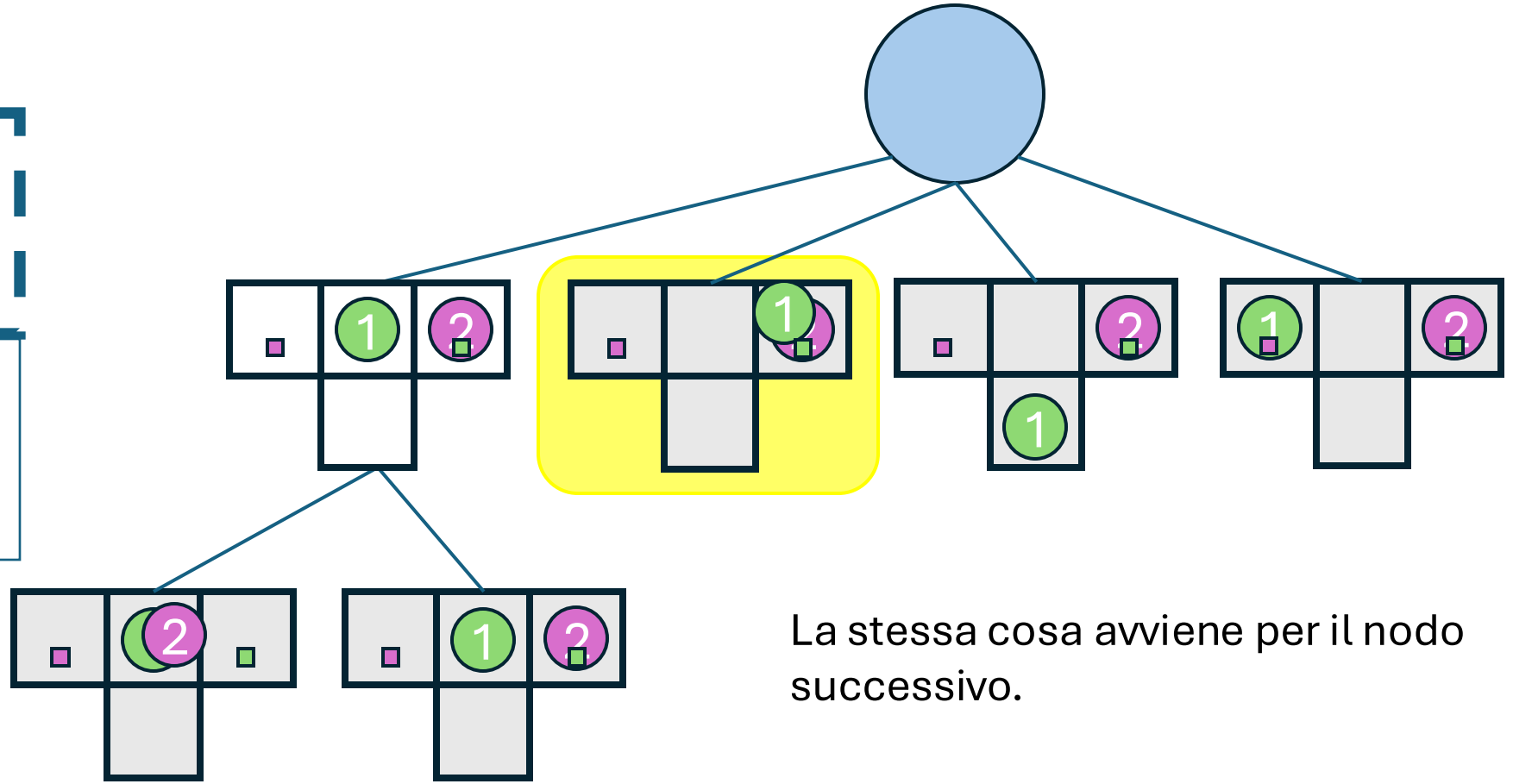
Come prima genero i successori dell'agente 1 e seleziono il primo. Dato che la configurazione generata è già apparsa nella ricerca, non apro un nuovo nodo HL e passo al nodo successivo.

LaCAM: Esempio (5)

High-level:



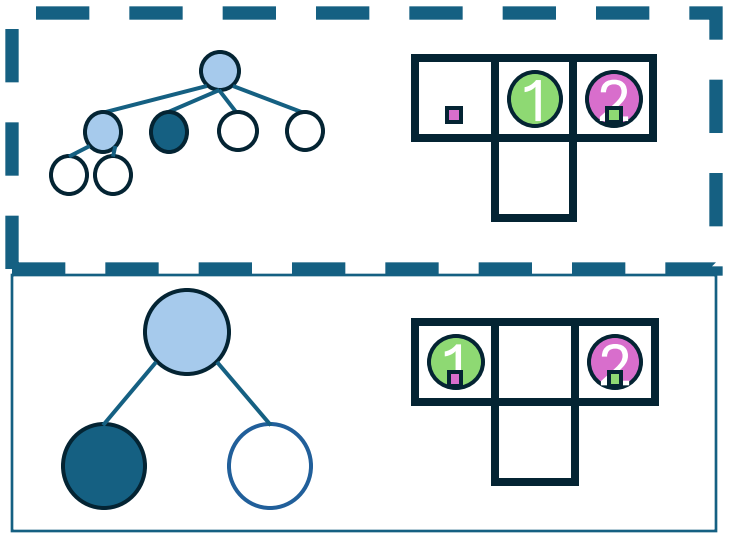
Low-level:



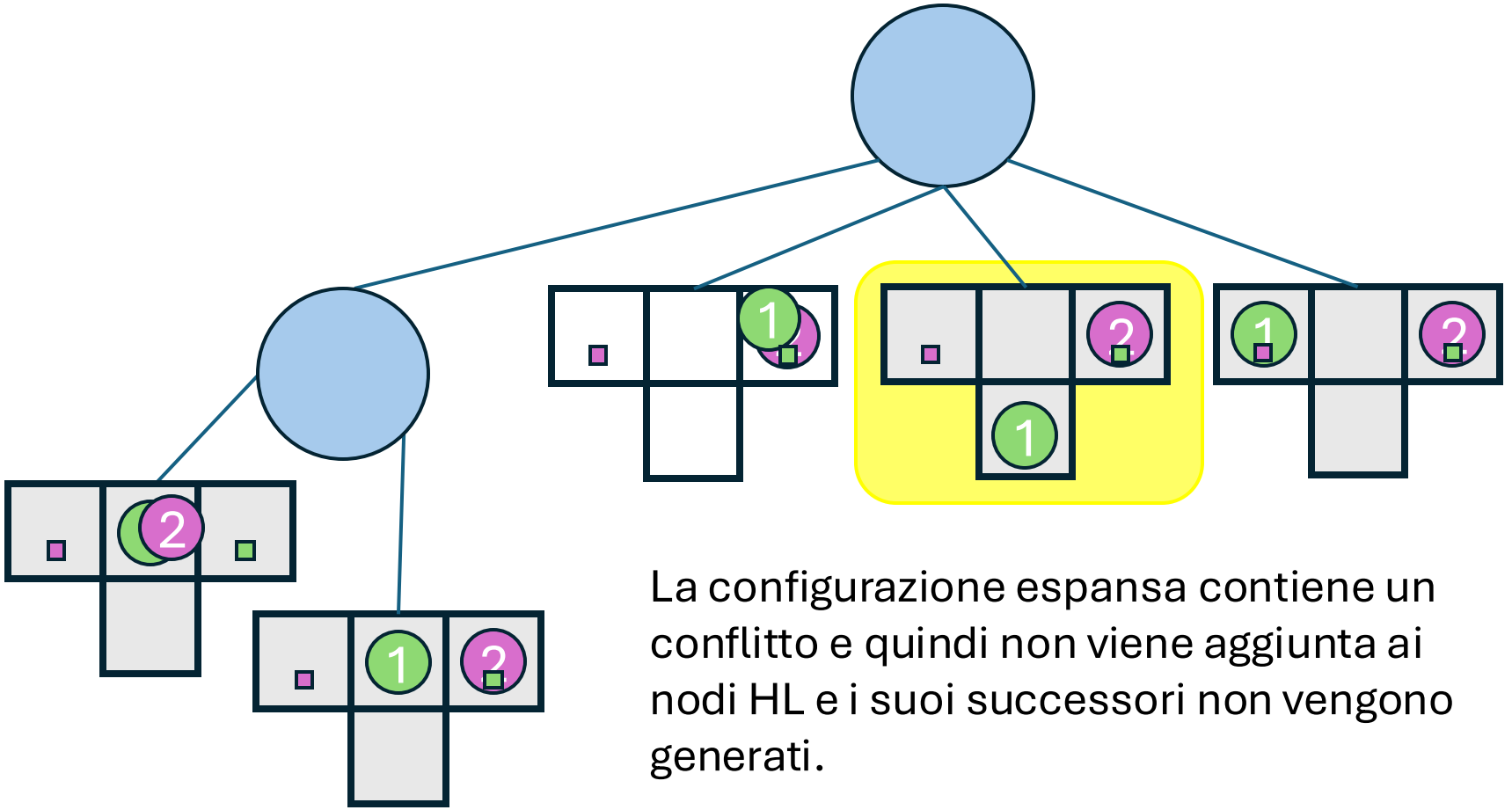
La stessa cosa avviene per il nodo successivo.

LaCAM: Esempio (6)

High-level:



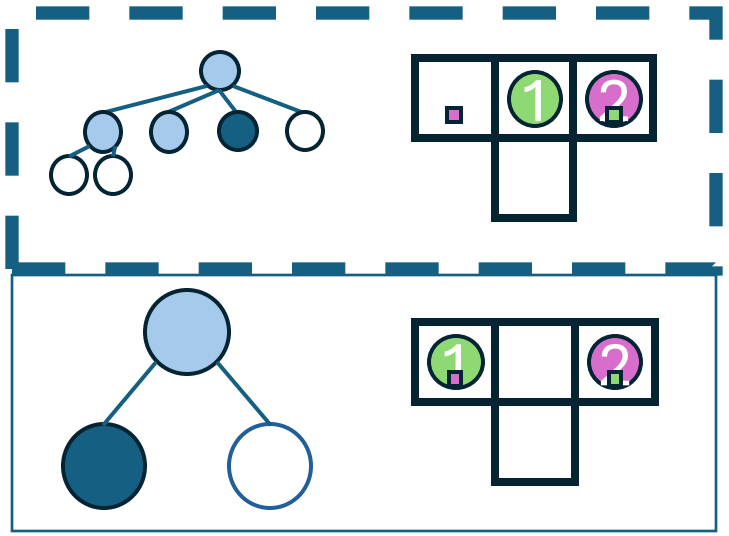
Low-level:



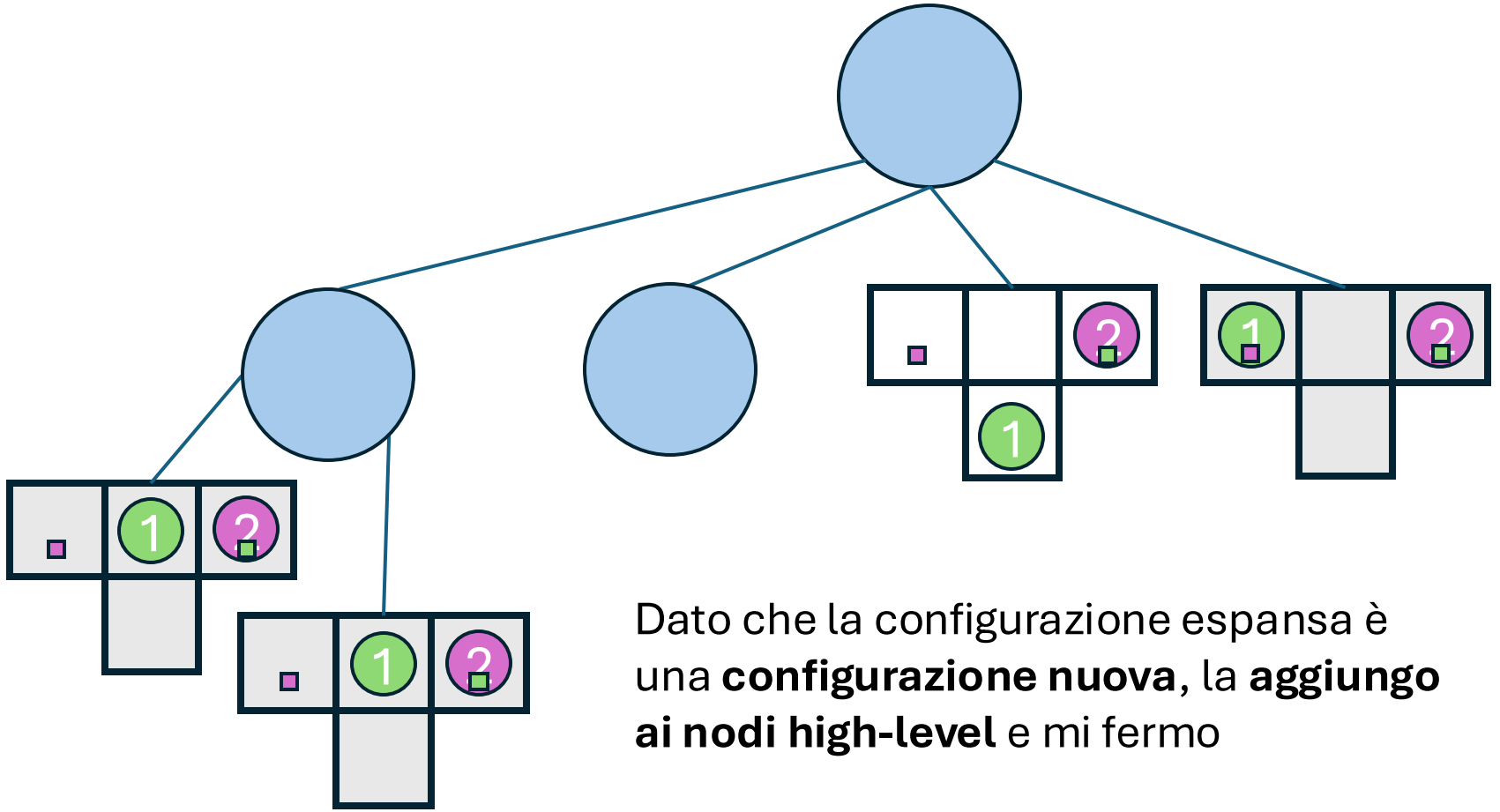
La configurazione espansa contiene un conflitto e quindi non viene aggiunta ai nodi HL e i suoi successori non vengono generati.

LaCAM: Esempio (7)

High-level:



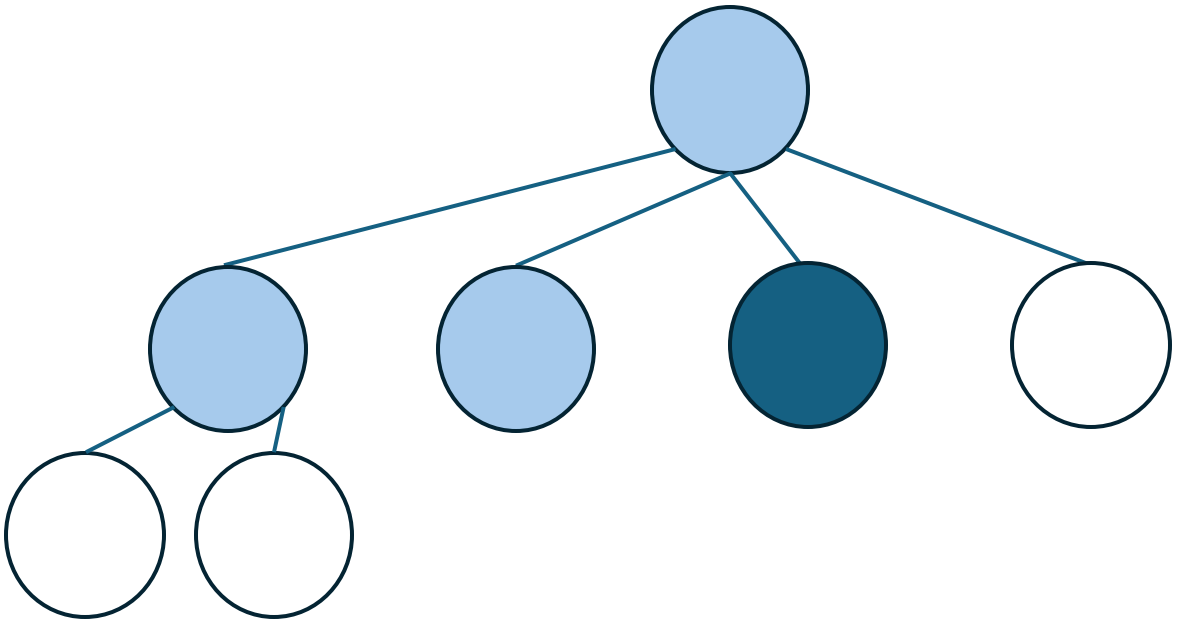
Low-level:



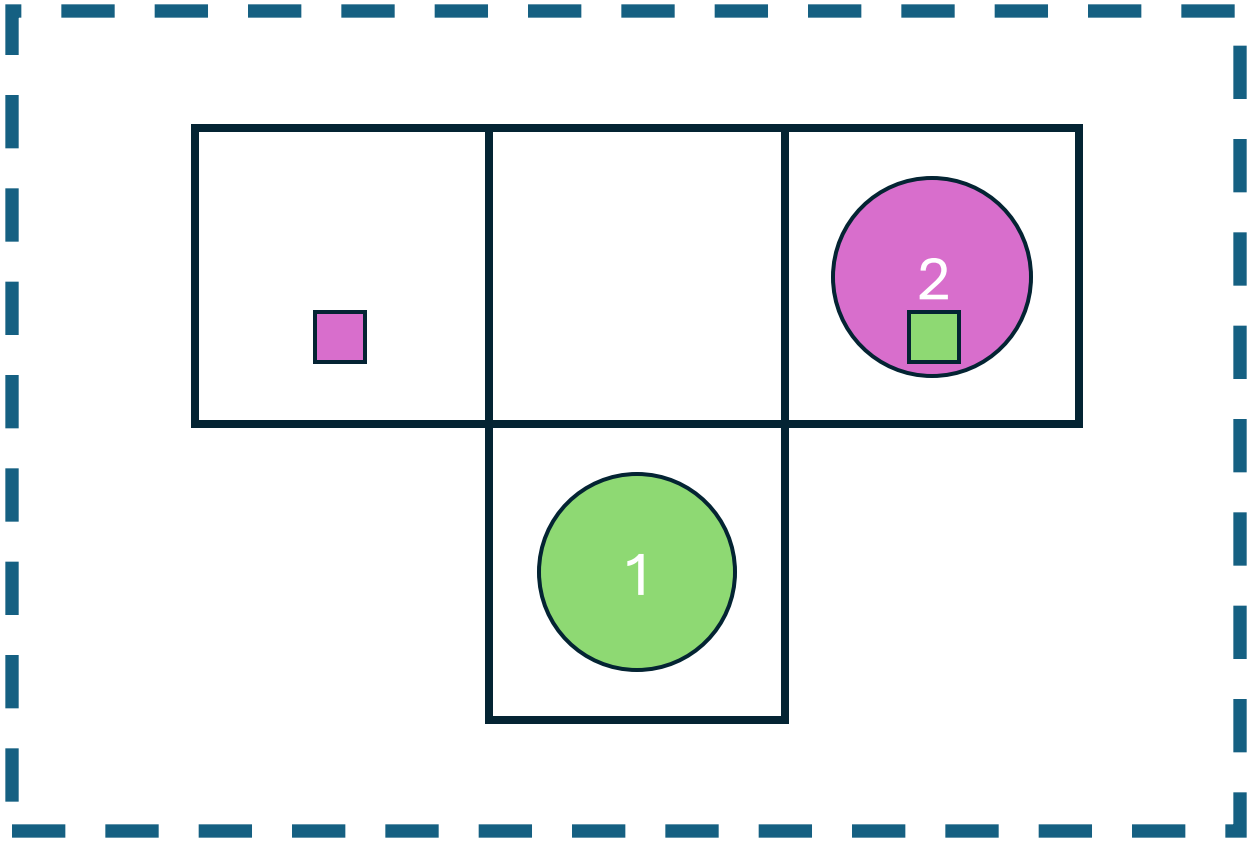
Dato che la configurazione espansa è una **configurazione nuova**, la **aggiungo ai nodi high-level** e mi fermo

LaCAM: Esempio (8)

High-level:

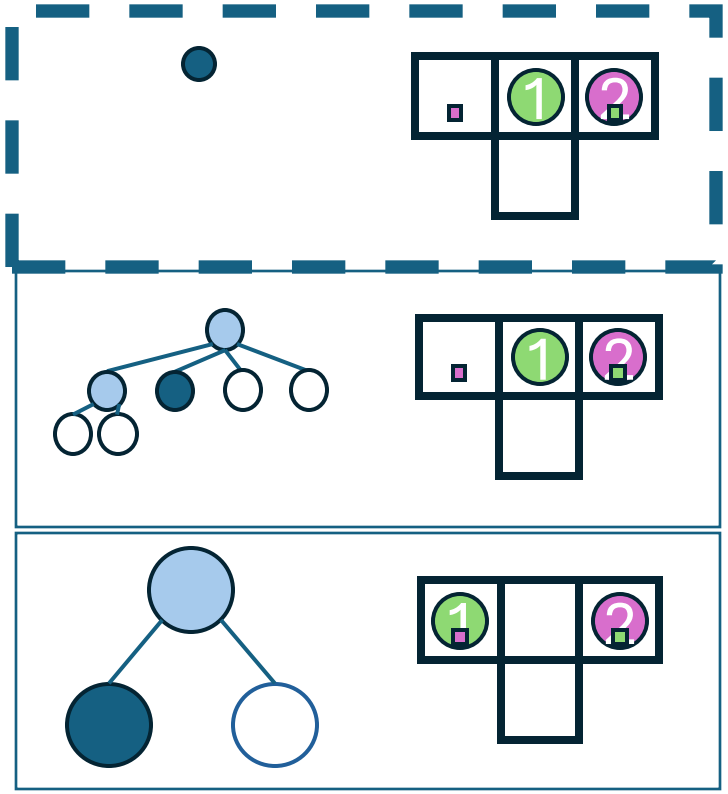


Espando la nuova configurazione trovata

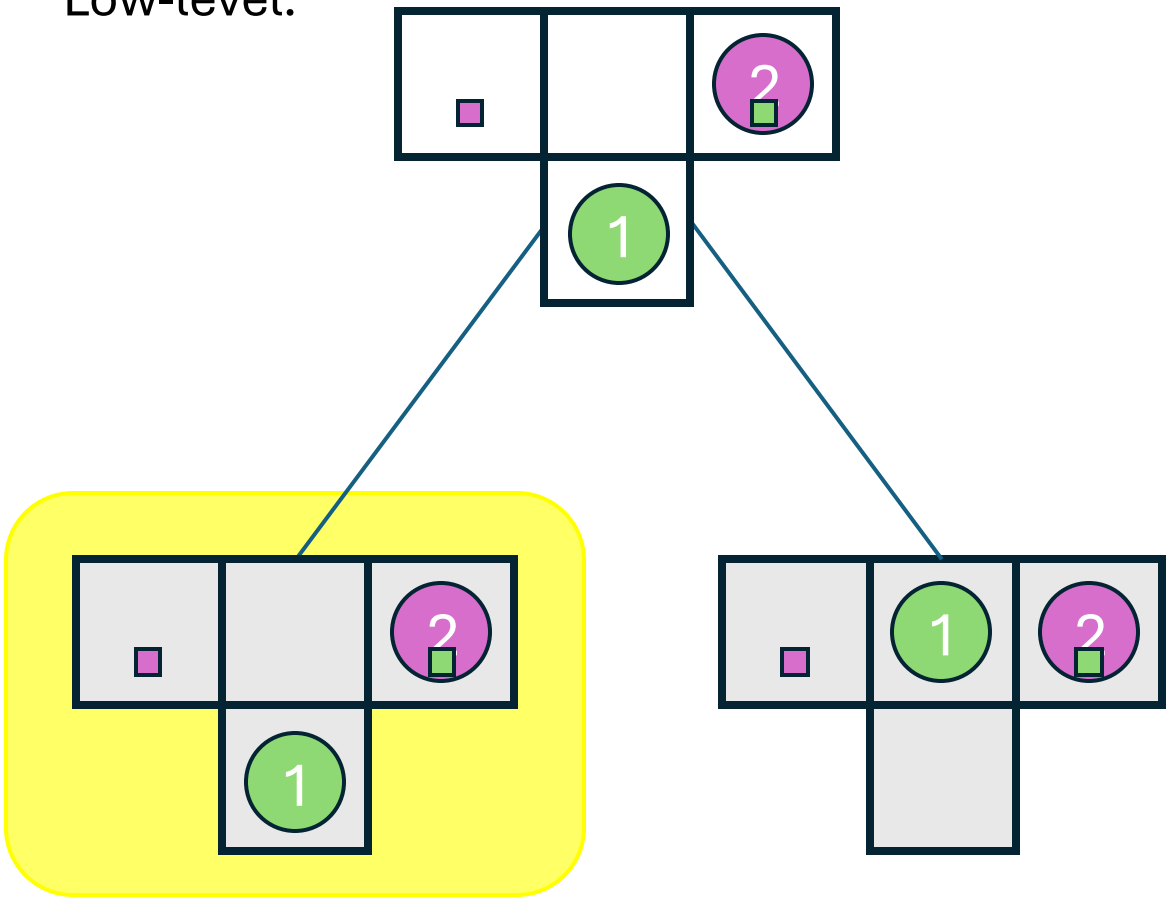


LaCAM: Esempio (9)

High-level:

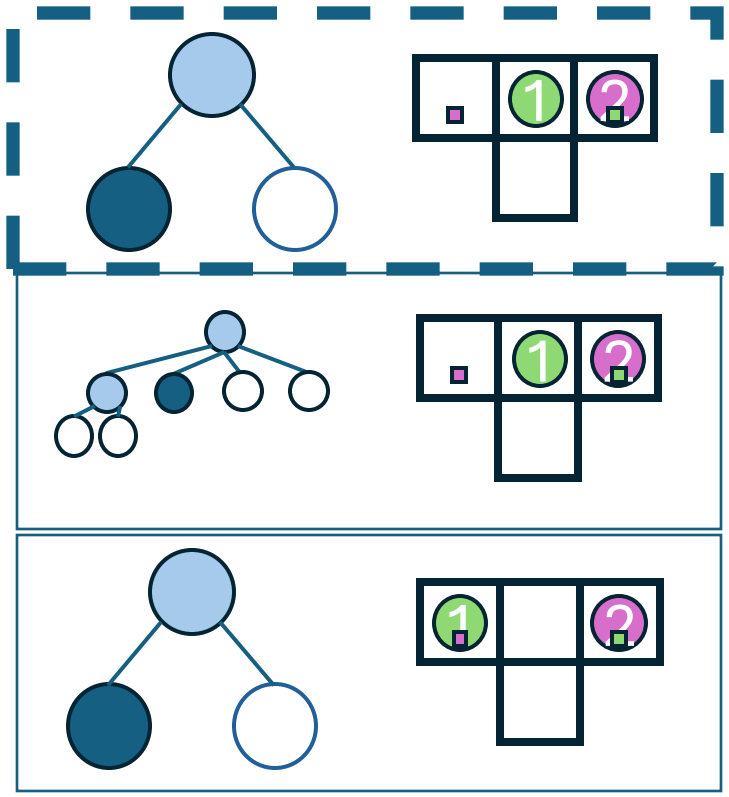


Low-level:

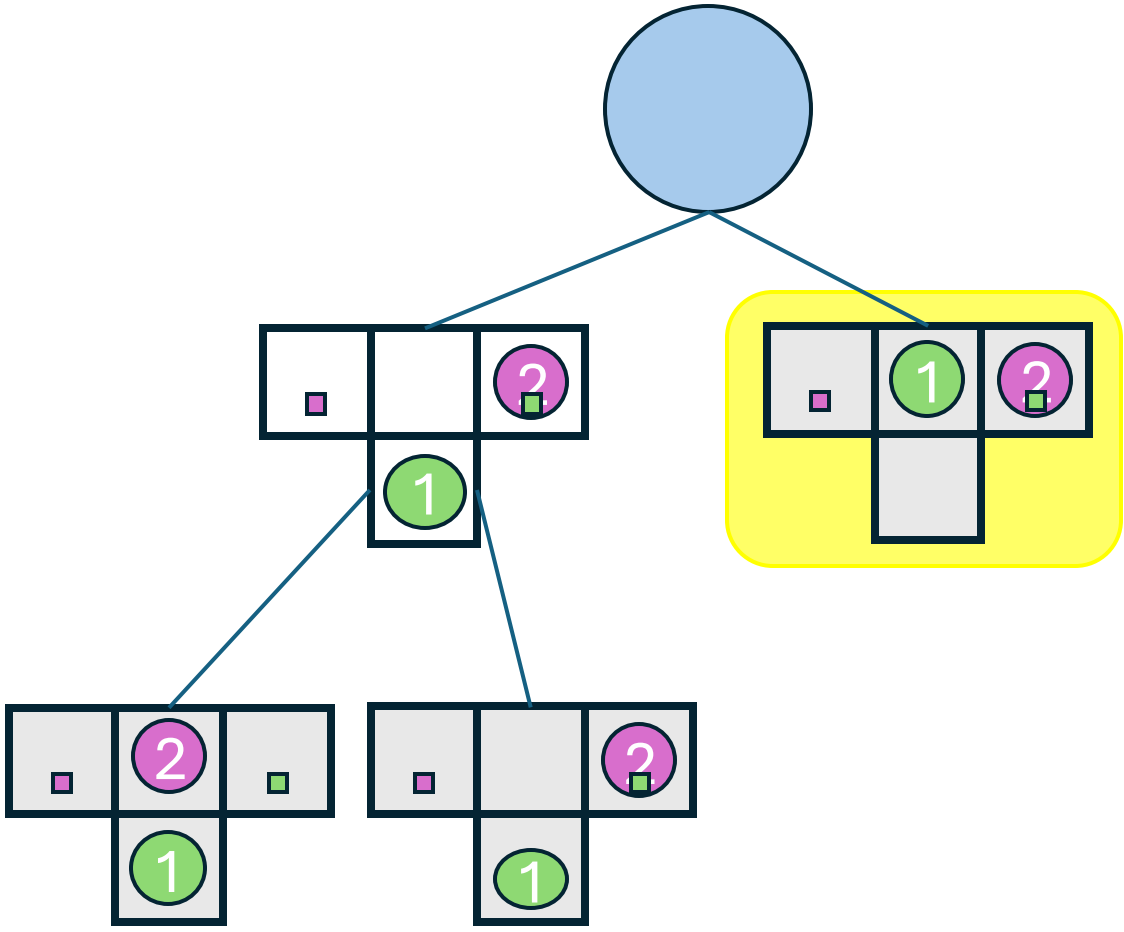


LaCAM: Esempio (10)

High-level:

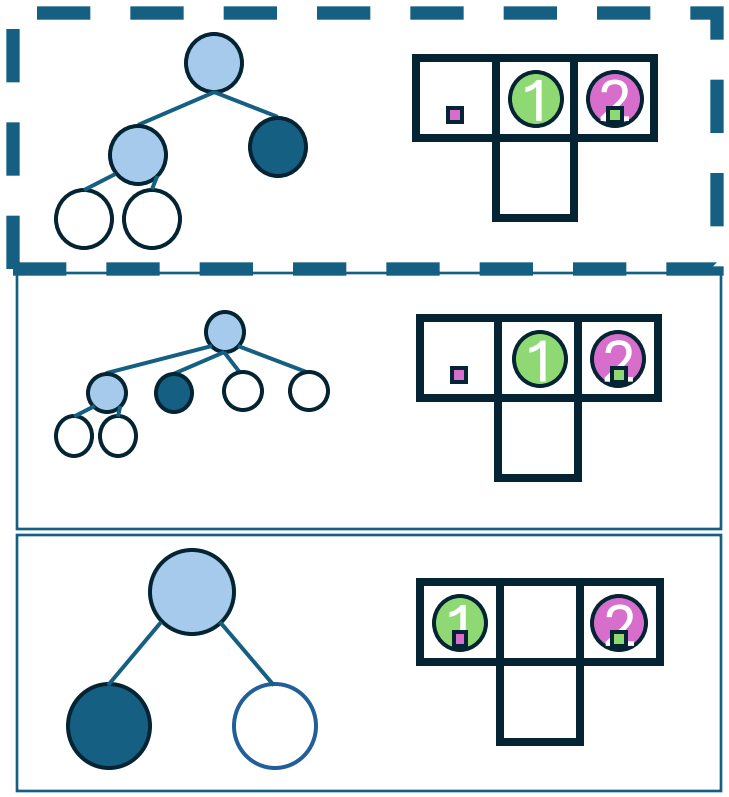


Low-level:

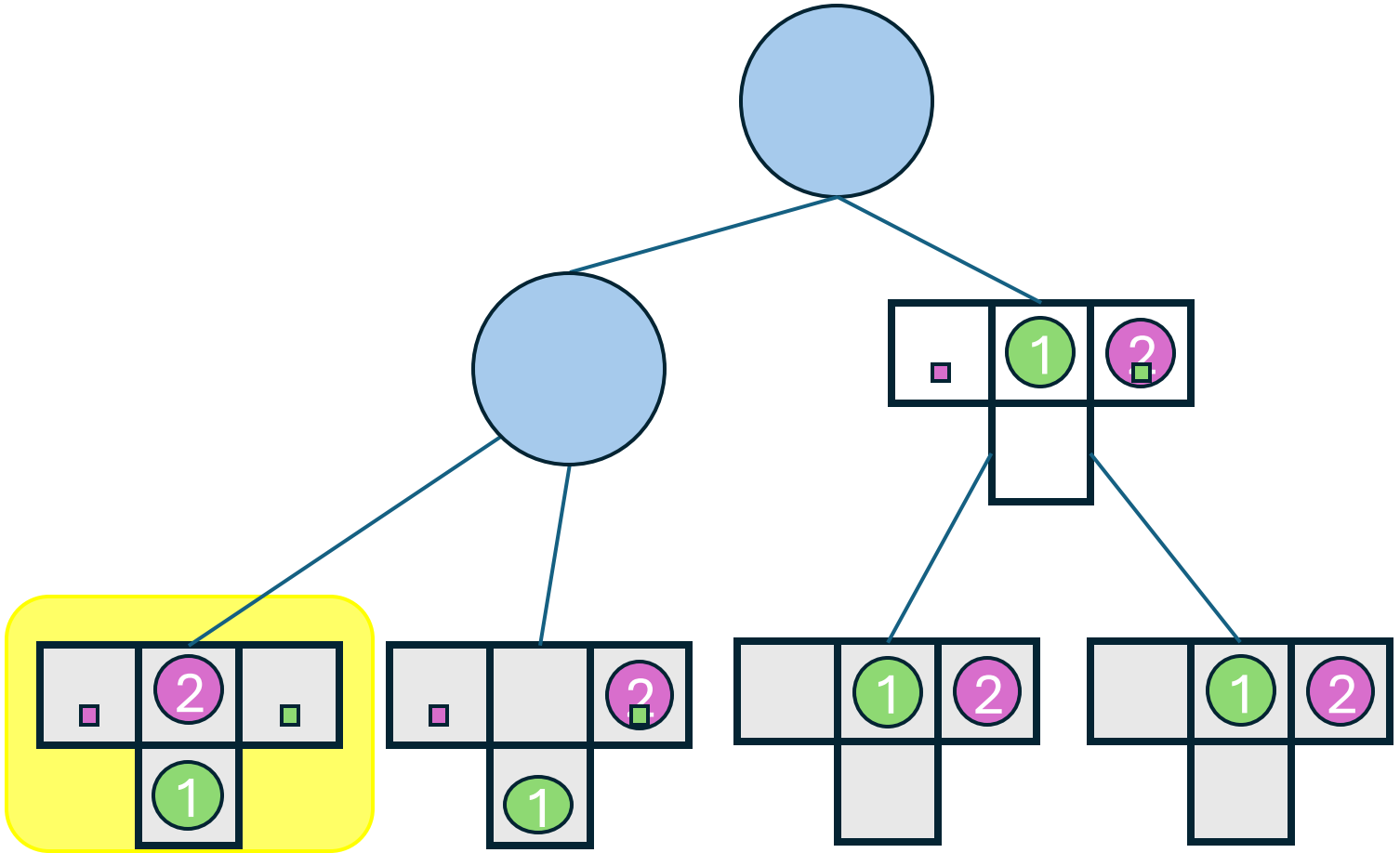


LaCAM: Esempio (11)

High-level:

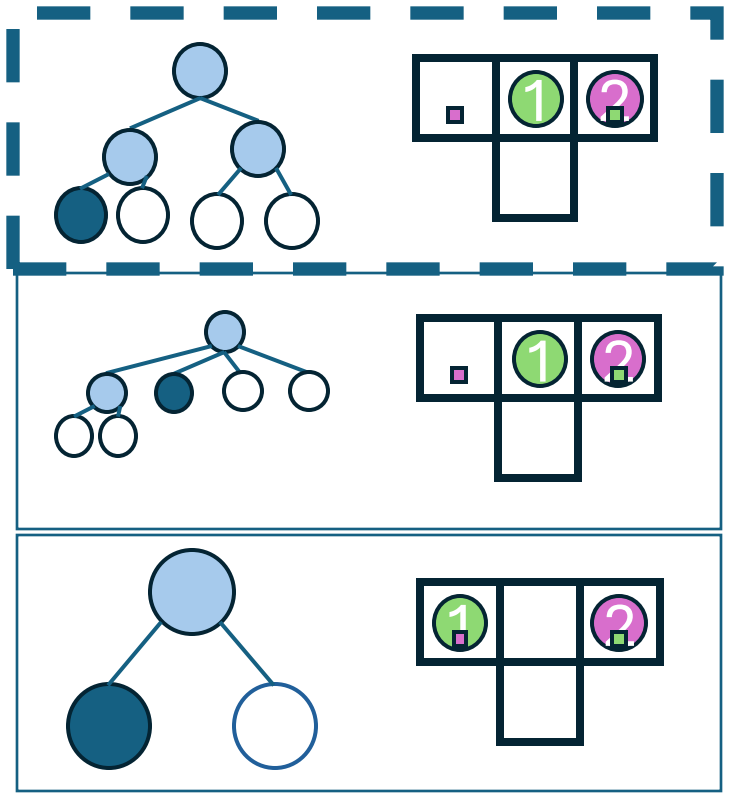


Low-level:

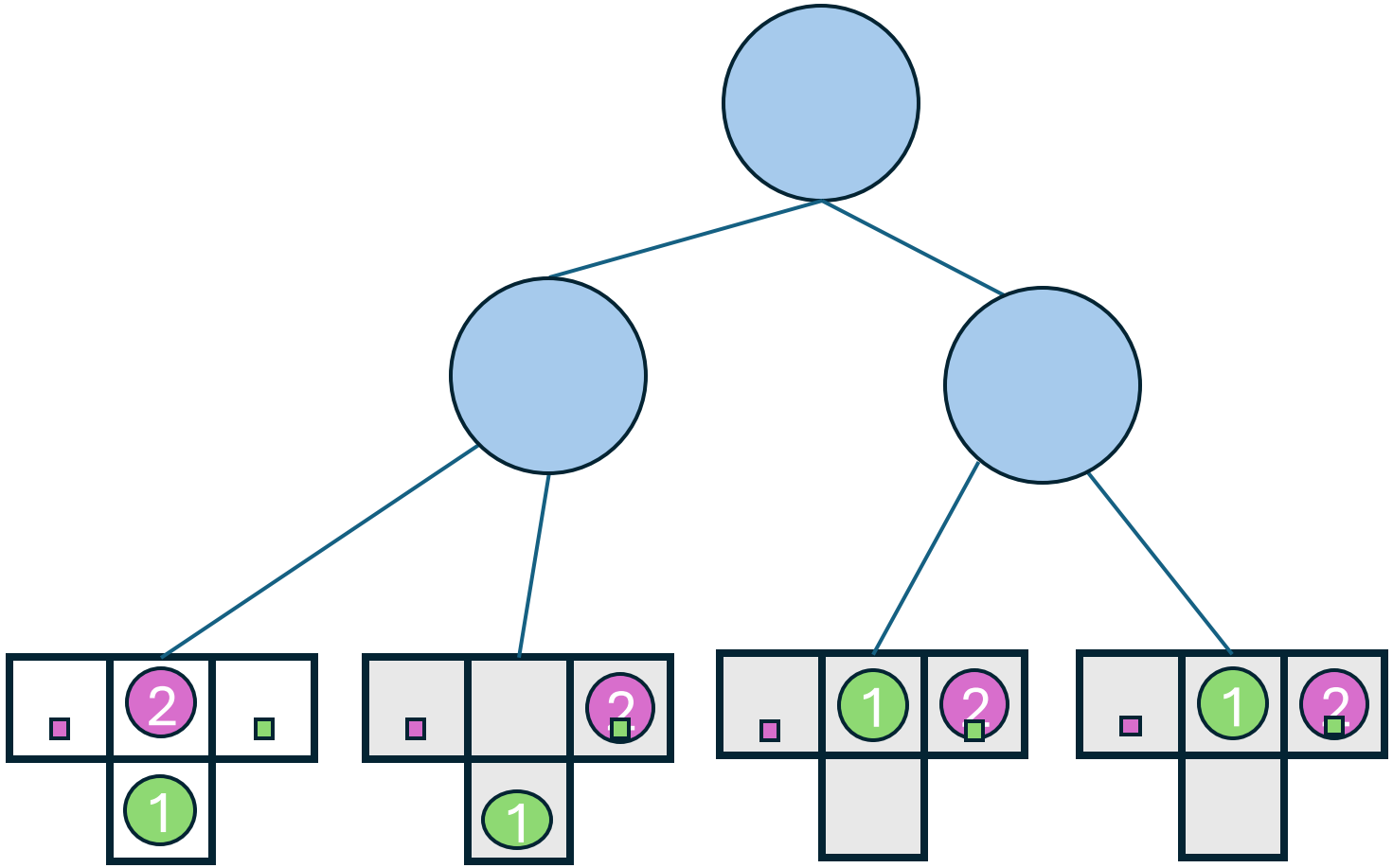


LaCAM: Esempio (12)

High-level:

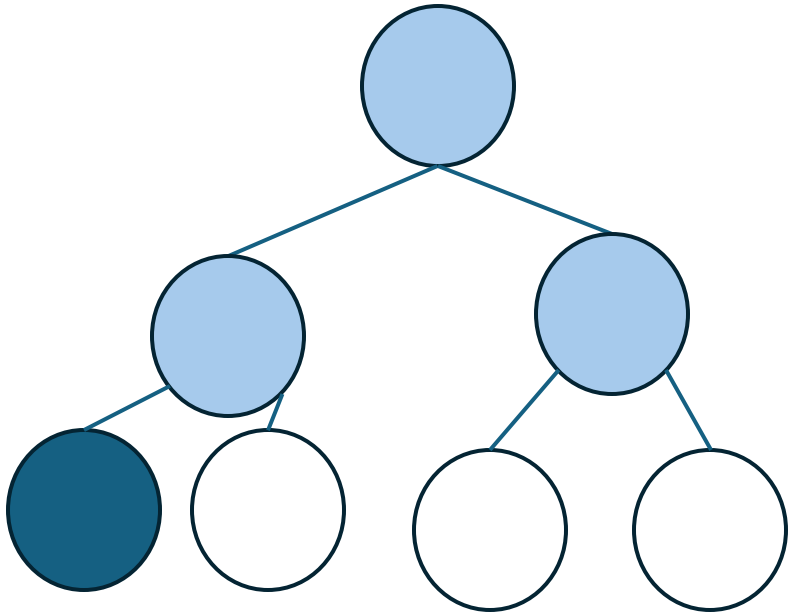


Low-level:

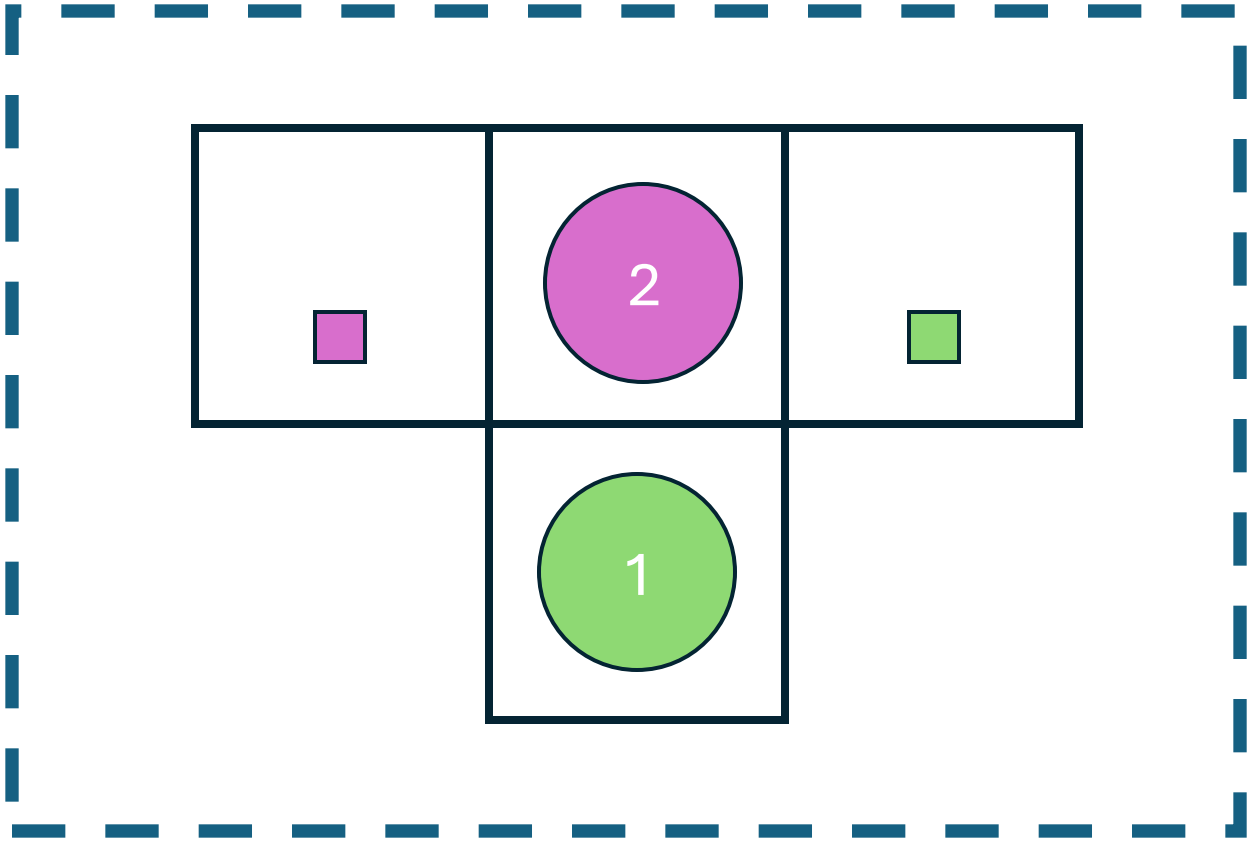


LaCAM: Esempio (13)

High-level:

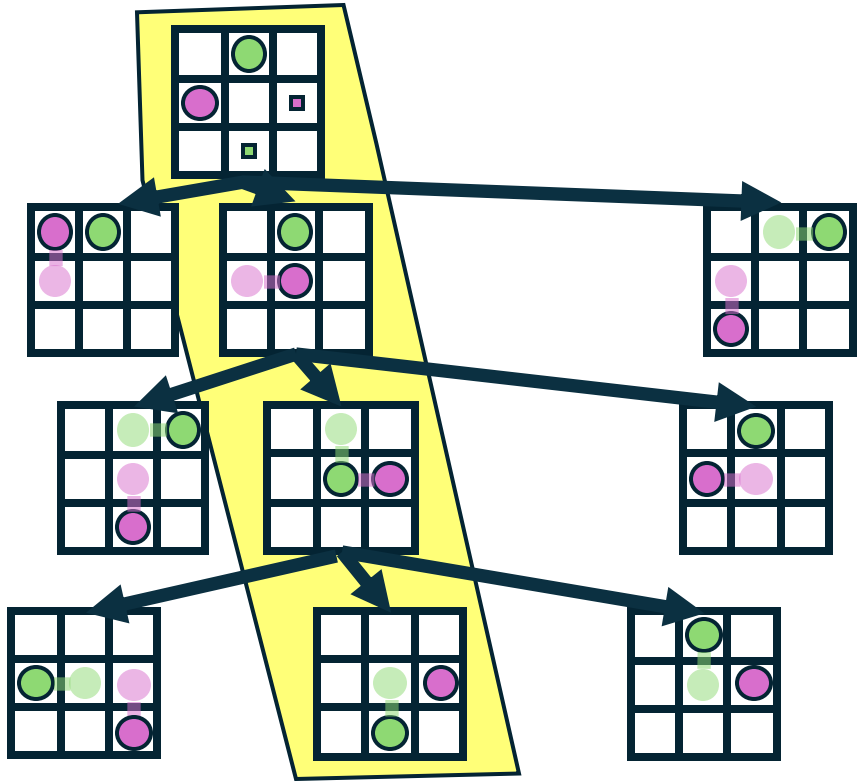


Espando la nuova configurazione trovata e proseguo fino alla soluzione



LaCAM: Idea

LaCAM è simile ad A* ma riduce il branching-factor generando successori in maniera **lazy** (solo quando necessario).



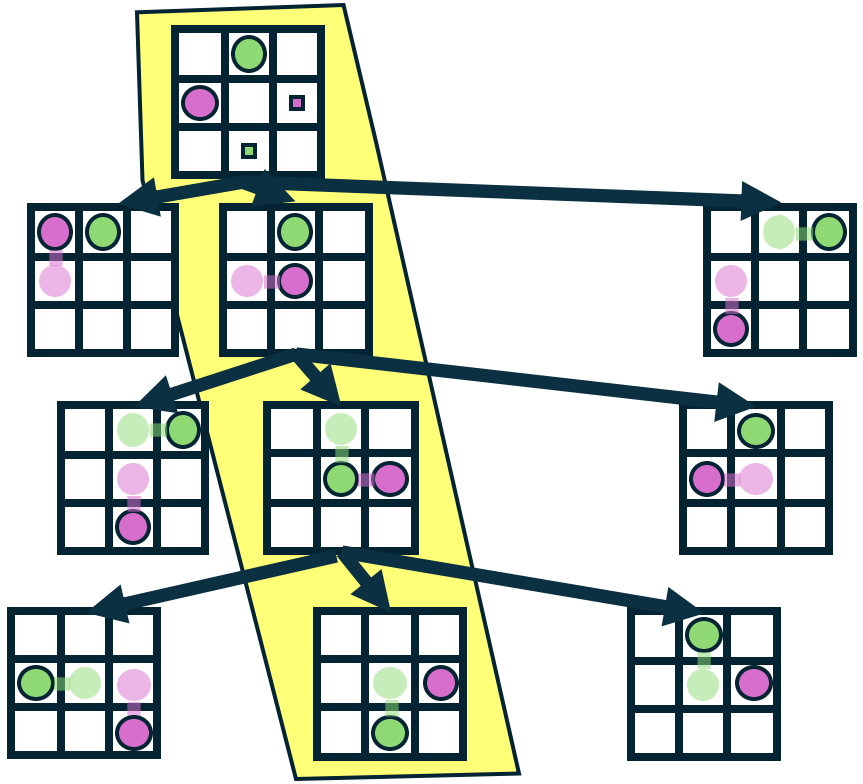
LaCAM: Idea (2)

Problema

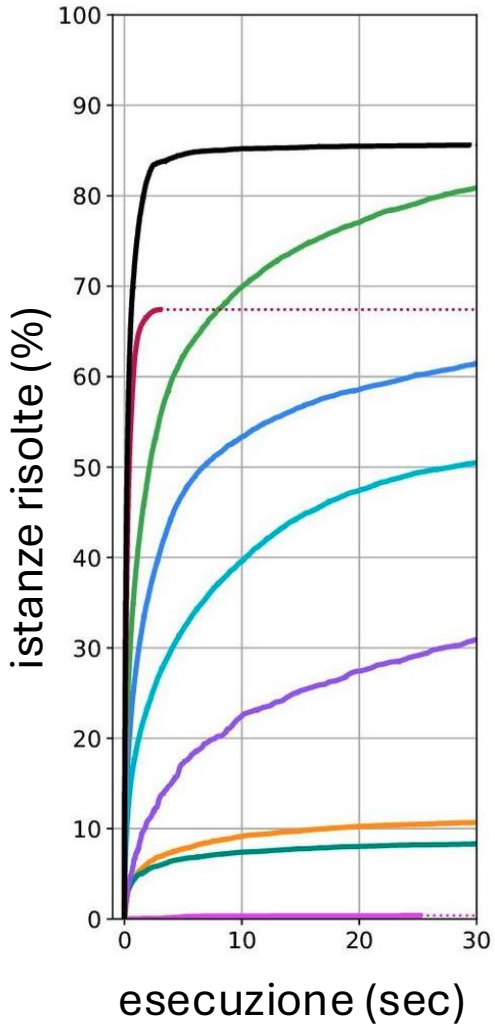
Affinché questo approccio funzioni è necessario che i pochi successori che genero siano di buona qualità

Soluzione

Utilizzo altri approcci di MAPF (e.g. PIBT, PP) per generare dei successori promettenti.

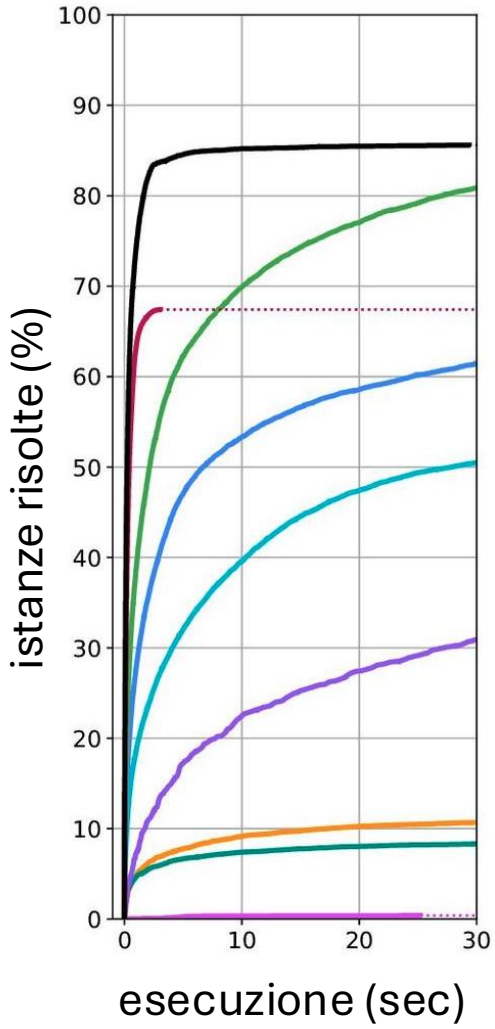


LaCAM: Risultati su benchmark



85.6% LaCAM [Okumura AAAI-23]	completo	subottimo
80.9% MAPF-LNS2 [Li+ AAAI-22]	non completo	subottimo
67.4% PIBT [Okumura+ AIJ-22]	non completo	subottimo
61.4% PP [Silver AIIDE-05]	non completo	subottimo
50.5% EECBS-5 [Li+AAAI-21]	completo per soluzioni	w-subottimo
30.9% I-ODrM*-5 [Wagner+AIJ-15]	completo	w-subottimo
10.7% BCP [Lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

PIBT: Risultati su benchmark



85.6% LaCAM [Okumura AAAI-23]	completo	subottimo
80.9% MAPF-LNS2 [Li+ AAAI-22]	non completo	subottimo
67.4% PIBT [Okumura+ AIJ-22]	non completo	subottimo
61.4% PP [Silver AIIDE-05]	non completo	subottimo
50.5% EECBS-5 [Li+AAAI-21]	completo per soluzioni	w-subottimo
30.9% I-ODrM*-5 [Wagner+AIJ-15]	completo	w-subottimo
10.7% BCP [Lam+ COR-22]	completo per soluzioni	ottimo
8.3% CBS [Sharon+ AIJ-26, Li+ AIJ-21]	completo per soluzioni	ottimo
0.4% ODrM* [Wagner+AIJ-15]	completo	ottimo
0.0% A* [Hart+68]	completo	ottimo

The League of Robot Runners

Il MAPF è un campo di ricerca molto attivo.

Ci sono delle competizioni che cercano di migliorare lo stato dell'arte della ricerca attraverso delle sfide basate sull'implementazione in ambienti reali.

Un esempio è

The League of Robot Runners

(possibilità di progetto per il corso)