Graph-based Planning

Pianificazione STRIPS basata su grafi

Michele Lombardi <michele.lombardi2@unibo.it>

La lezione/esercitazione di oggi

Pianificazione basata su grafi & derivati

Approccio “pratico”: presentazione di pianificatori

- Graphplan (ne parleremo a lungo)
  - Planning graph
  - Pianificazione come ricerca sul planning graph
- Blackbox (accenni)
  - Ricerca nel planning graph come problema di soddisfacimento
- FF (Fast Forward) (accenni)
  - Ricerca su planning graph come euristica

Graphplan

Pianificazione basata su grafi

Graphplan

Perché Graphplan?

La pianificazione basata su grafi nasce con Graphplan!

Cos'è Graphplan?

- E' un pianificatore per problemi tipo STRIPS introdotto da Blum & Furst (CMU) nel 1995
- Basato su una struttura dati detta planning graph, che viene espansa ad ogni passo dell'algoritmo
- Pianificazione come ricerca sul planning graph

Graphplan

Un po' di caratteristiche

- Introduce esplicitamente il fattore tempo nella costruzione di un piano
- E' completo e produce sempre il piano più corto possibile
- Esegue early commitment (come i pianificatori lineari)
- Produce piani parzialmente ordinati (come per i pianificatori non lineari – partial order)
- Graph plan usa la Closed World Assumption quindi rientra nella categoria dei pianificatori off line

Pianificazione STRIPS like

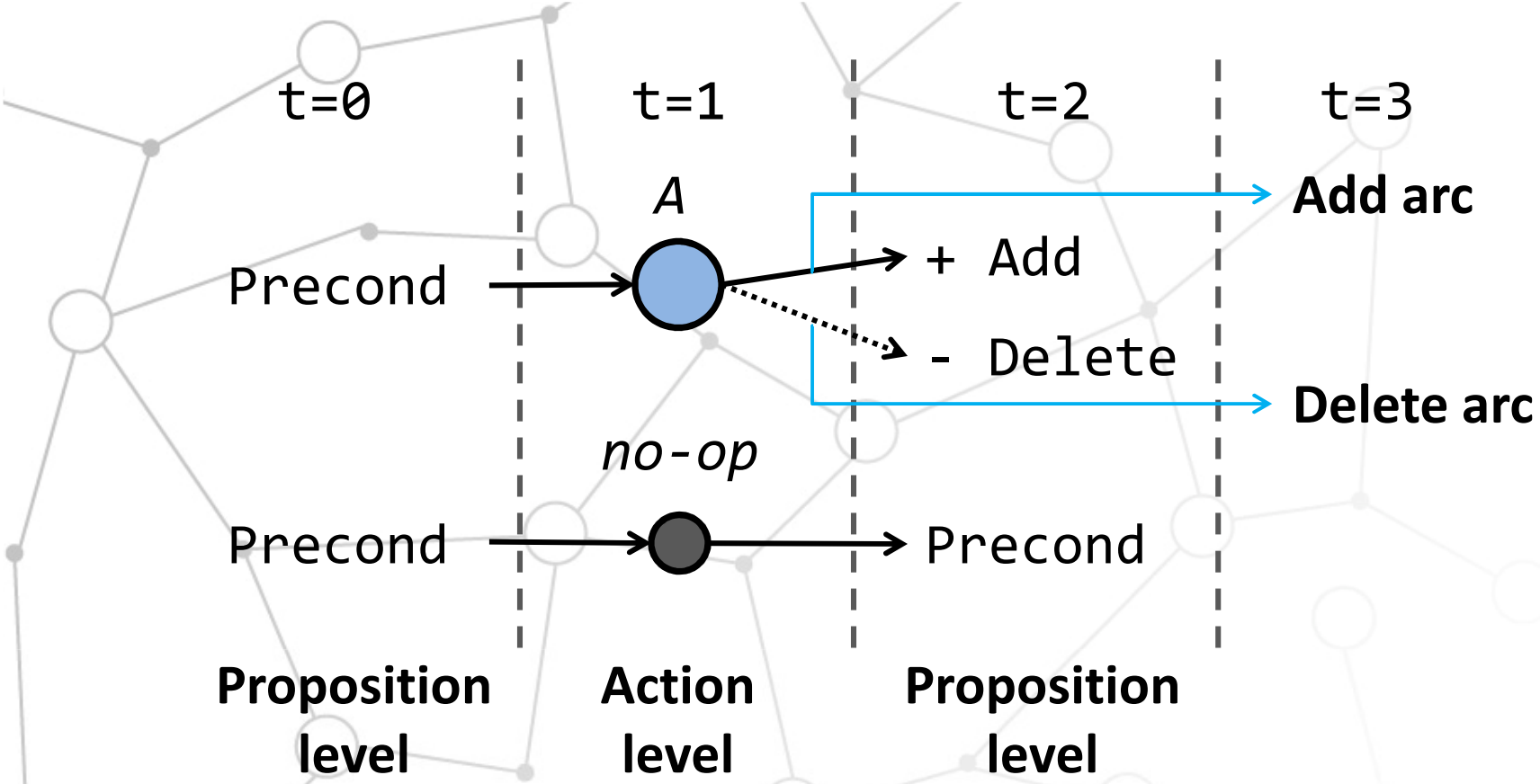
- **Le azioni si rappresentano come quelle di STRIPS** con

  PRECONDIZIONI
  ADD LIST
  DELETE LIST

- E' implicitamente definita una azione **no-op** che non modifica lo stato (ha come precondizione qualunque predicato e come effetto lo stesso)
- **Stato = oggetti + insieme di predicati**
  Gli oggetti hanno un tipo (typed)!

Planning graph

- Il planning graph è un **grafo diretto a livelli:**
  - i nodi sono raggruppati in livelli
  - gli archi connettono nodi a livelli adiacenti.
- Il planning graph alterna:
  - Proposition levels, contenenti proposition nodes
  - Action levels, contenti action nodes
- Il livello 0 corrisponde allo stato iniziale (è un proposition level)
- Gli archi si dividono in:
  - Archi *precodizione* (proposition → action)
  - Archi *add* (action→ proposition)
  - Archi *delete* (action→ proposition)



- A un certo time step si può inserire una azione se al time step precedente sono presenti le sue precondizioni
- Le azioni fittizie “no-op” traslano le proposizioni di un time step al successivo (*soluzione al frame problem*)

Planning graph

- Ogni action level contiene
  - tutte le azioni che sono applicabili in quel time step
  - vincoli che specificano quali coppie di azioni non possono essere eseguite contemporaneamente (inconsistenze o mutue esclusioni)
- Ogni proposition level contiene:
  - Tutte le proposizioni del livello precedente (attraverso i no-ops)
  - Gli add-effects delle azioni dell'action level precedente

NOTA: il processo di costruzione del planning graph non prevede che si faccia alcuna scelta per selezionare le azioni che costruiscono il piano.

Inconsistenze

Durante la costruzione del planning graph vengono individuate eventuali inconsistenze, in particolare

- Due **azioni** possono essere inconsistenti nello stesso time step
- Due **proposizioni** possono essere incosistenti nello stesso time step

In questo caso le azioni/proposizioni sono **mutuamente esclusive**

- Non possono comparire insieme in un piano
- **Ma possono comparire** nello stesso livello **nel planning graph**

Inconsistenze

Inconsistenza tra **azioni**:

- **Interferenza (interference):** una azione nega un effetto o una precondizione di un'altra
  ex. L'azione move(part,dest) ha come effetto not at(part) mentre l'azione no-op su at(part) ha questo come effetto
  ex. L'azione move(part,dest) ha come effetto not at(part) mentre l'azione no-op su at(part) ha questo come precondizione
- **Requisiti in competizioni (competing needs):** due azioni hanno precondizioni mutuamente esclusive
  ex. L'azione load(carico,mezzo) ha come precondizione in(carico, mezzo) mentre l'azione unload(carico, mezzo) ha come precondizione not in(carico, mezzo)

Inconsistenze

Due **proposizioni** sono mutuamente esclusive:

- Se sono uno la negazione del'altro
- Inconsistenze sul dominio

  ex. esistono regole quali ad esempio il fatto che un oggetto non può trovarsi in due luoghi contemporaneamente in uno stesso time step.
- Se tutti i possibili modi per raggiungerli coinvolgono azioni mutuamente esclusive

Costruzione del planning graph

Algoritmo per la costruzione passo a passo del planning graph:

1. **INIZIALIZZAZIONE:**
   Tutte le proposizioni vere nello stato iniziale sono inserite nel primo proposition level
2. **CREAZIONE DI UN ACTION LEVEL:**
   1. ∀operatore ∀ogni modo di unificare le sue precondizioni a proposizioni non mutuamente esclusive nel livello precedente, inserisci un action node
   2. Per ogni proposizione nel proposition level precedente, inserisci un operatore no-op
   3. Identifica le relazioni di mutua esclusione tra gli operatori appena costruiti

Costruzione del planning graph

Algoritmo per la costruzione passo a passo del planning graph:

3. **CREAZIONE DI UN PROPOSITION LEVEL:**
    1. Per ogni action node nel action level precedente, aggiungi le proposizioni nella sua add list tramite archi non tratteggiati
    2. Per ogni "no-op" nel livello precedente, aggiungi la proposizione corrispondente
    3. Per ogni action node nel action level precedente, collega proposizioni nella sua delete list tramite archi tratteggiati
    4. Marca come esclusive due proposizioni tali per cui tutti i modi per raggiungere la prima siano incompatibili con tutti i modi per raggiungere la seconda.

Un esempio classico (rivisitato!)

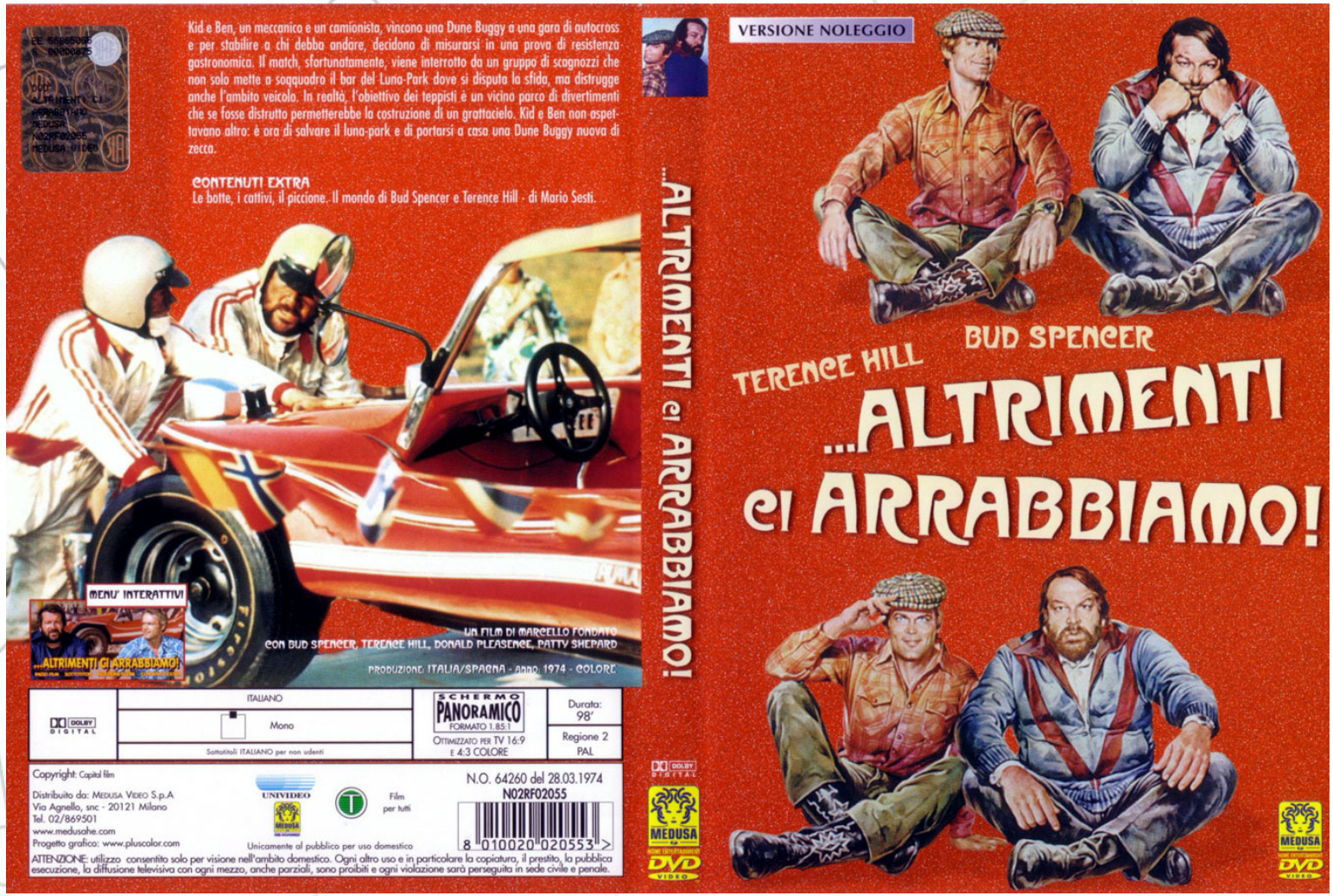
1. Due soggetti da trasportare

2. Un luogo di partenza

3. Un carrello motorizzato

4. Un luogo di destinazione

Un esempio classico (rivisitato!)



Un esempio classico (rivisitato!)

Azioni:

- **MOVE(R, PosA, PosB)**
  PRECONDIZIONI: at(R,PosA), hasFuel(R)
  ADD LIST: at(R,PosB)
  DELETE LIST: at(R,PosA), hasFuel(R)
- **LOAD(Oggetto, R, Pos)**
  PRECONDIZIONI: at(R,Pos), at(Oggetto,Pos)
  ADD LIST: in(R,Oggetto)
  DELETE LIST: at(Oggetto,Pos)
- **UNLOAD(Oggetto, Pos)**
  PRECONDIZIONI: in(R,Oggetto), at(R,Pos)
  ADD LIST: at(Oggetto,Pos)
  DELETE LIST: in(R,Oggetto)

Un esempio classico (rivisitato!)

Tipi, stato, goal:

- **OGGETTI:**

```
carrello: Carriola (c)
oggetti: Bud (b), Terence (t)
locazioni: Lunapark (l), HQ del Capo (chq)
```

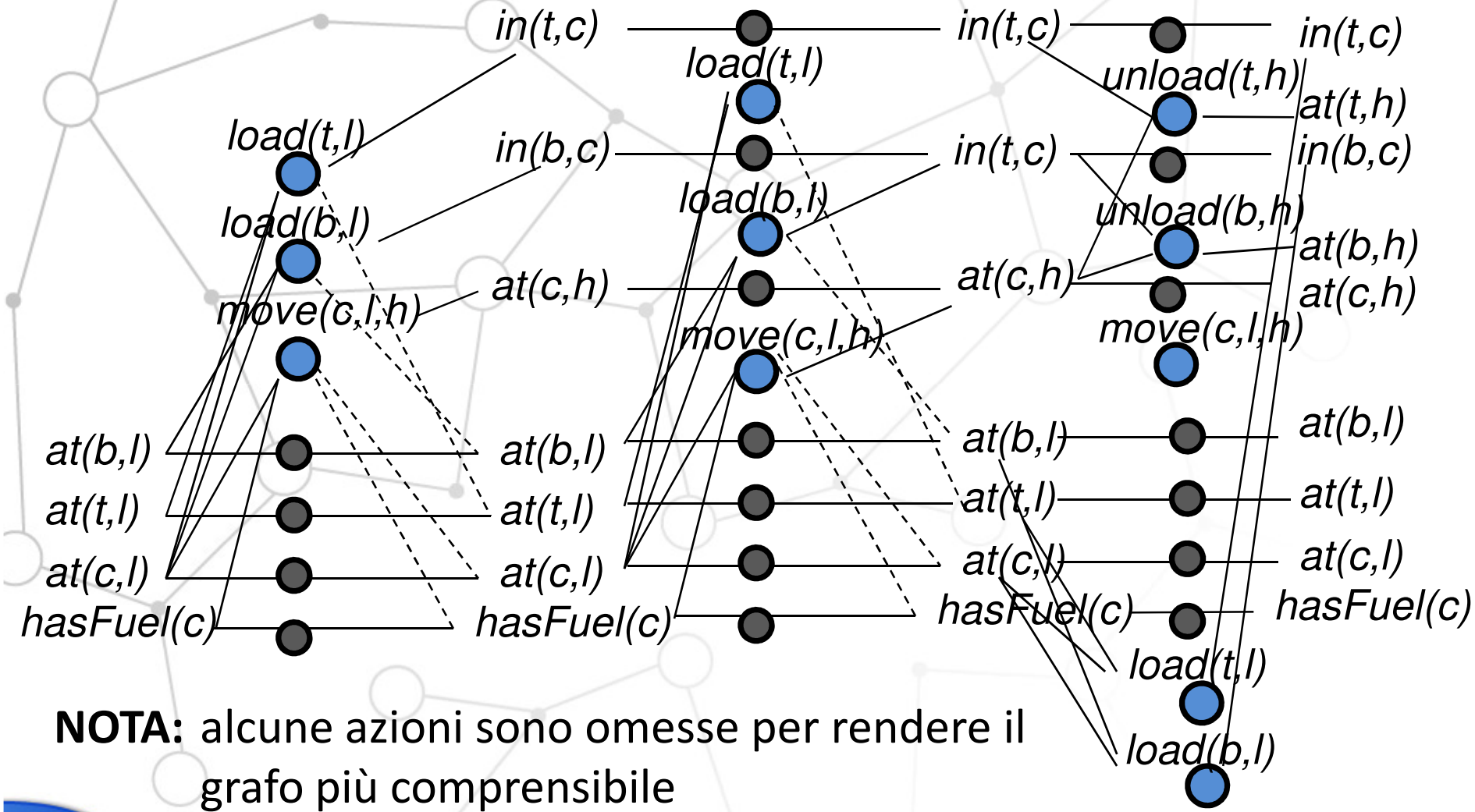
- **STATO INIZIALE:**

```
at(b,l)
at(t,l)
at(c,l)
hasFuel(c)
```

- **GOAL:**

```
at(b, chq)
at(t, chq)
```

Per ogni operatore/proposizione ad ogni time step viene calcolata una **lista di mutua esclusione**



NOTA: alcune azioni sono omesse per rendere il grafo più comprensibile

Esercizio

Vediamo la costruzione del grafo in azione

- Scaricare graphplan dal sito del corso
  `http://www.lia.deis.unibo.it/Courses/AI/applicationsAI2008-2009/Lucidi/Xesercitazioni/graphplan.gentoo.zip`
- Decomprimere l'archivio
  `unzip graphplan.gentoo.zip`
- Eseguire `graphplan` (o `graphplan -h`)
  ```
  ops file: cart.gp.ops
  fact file: cart.gp.facts
  ```
- Livello di informazioni (opzione `-i` da riga di comando)
  ```
  0: il piano e poco più
  1: grafo e passi della ricerca
  2: grafo e liste di mutua esclusione
  ```

Estrazione di un piano

Valid plan

un sottografo connesso e consistente del planning graph

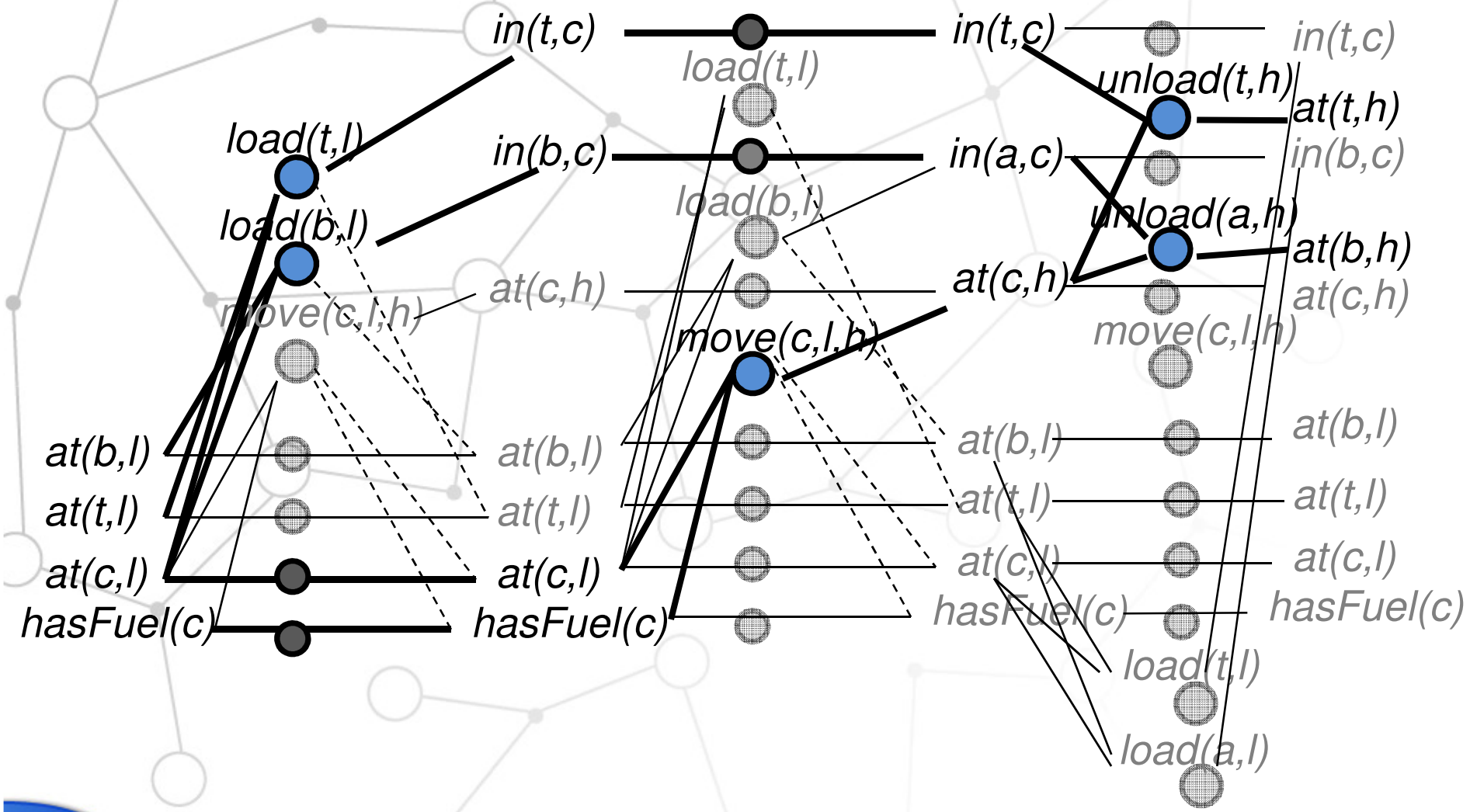
- Azioni allo stesso time step del valid plan possono essere eseguite in qualunque ordine (non interferiscono)
- Proposizioni allo stesso time step del valid plan sono non mutuamente esclusive
- L'ultimo time step contiene tutti i letterali del goal e questi non sono marcati come mutuamente esclusivi

Teoremi

1. Se esiste un piano valido allora questo è un sottografo del planning graph.
2. In un planning graph due azioni sono mutuamente esclusive in un time step se non esiste un valid plan che le contiene entrambe.
3. In un il planning graph due proposizioni sono mutualmente esclusive in un time step se sono inconsistenti ossia una nega carsi dell'altra.

Conseguenza importante:
Le inconsistenza trovate dall'algoritmo permettono di eliminare strade nell'albero di ricerca

Un esempio di **valid plan**



Graphplan: algoritmo

```
function GRAPHPLAN(problema):
         grafo = GRAFO_INIZIALE(problema)
         obiettivi = GOAL(problema)
         loop do:
            if obiettivi non mutex nell'ultimo step:
               Sol = ESTRAI_ SOLUZIONE(grafo, obiettivi)
               if sol Sol ≠ fail: return Sol
               else if LEVEL_OFF(grafo): return fail
            grafo = ESPANDI_GRAFO(grafo, problema)
```

- Il primo nodo contiene il planning graph iniziale. Questo contiene solo un time step (proposition level) con le proposizioni vere nello stato iniziale.
  Il grafo iniziale è estratto da `GRAFO_INIZIALE(problema)`

Graphplan: algoritmo

- Il goal da raggiungere è estratto dalla funzione `GOAL(problema)`
- Se gli obiettivi sono non mutuamente esclusivi nell'ultimo livello il planning graph potrebbe contenere un piano, ossia un valid plan. Il valid plan è estratto da `ESTRAI_SOLUZIONE(grafo, obiettivi)` che fornisce una soluzione o un fallimento
  - Si procede livello dopo livello per meglio sfruttare i vincoli di mutua esclusione
  - Metodo ricorsivo: dato un insieme di goals a tempo t si cerca un insieme di azioni a tempo t-1 che abbiano tali goals come add effects. **Le azioni devono essere non mutuamente esclusive.**
  - La ricerca è ad albero, ibrida breadth/depth first e completa

Graphplan: algoritmo

- **Memoization** (non è un errore tipografico!)
  Se ad un certo step della ricerca si determina che un sottoinsieme di goals non è soddisfacibile, graphplan salva questo risultato in una hash table. Ogni volta che lo stesso sottoinsieme di goals verrà selezionato in futuro quel ramo di ricerca fallirà automaticamento

Esercizio

Vediamo la ricerca di un valid plan in azione

- Eseguire `graphplan` (o `graphplan -h`)

```
ops file: cart.gp.ops
fact file: cart.gp.facts
```

1. Guardate gli step del processo di ricerca variando il livello di informazione
2. Provate a disabilitare l'individuazione delle mutue esclusioni: cosa vi aspettate che succeda? Cosa succede realmente?
3. Riuscite a vedere quando la memoization viene utilizzata? In quali casi questo succede per l'esempio considerato?

Blackbox

Unificare pianificazione basata su grafi e pianificazione come soddisfacibilità

SATPLAN

SATPLAN

Nel 92 Selman e Kautz introducono un approccio alla pianificazione basato sulla ricerca di un assegnamento di variabili logiche indicanti una selezione di azioni che soddisfi una serie di vincoli, piuttosto che sulla deduzione

APPROCCIO DEDUTTIVO (ex. situation calculus)

- Data una serie di regole di deduzione in first order logic...
- ...e un insieme di fluent inizialmente validi...
- ...si cerca di dedurre i goals

Questo approccio non può essere modellato direttamente come problema di soddisfacimento di vincoli. A causa di un certo numero di problemi...

SATPLAN

1. **Le regole di deduzione sono generali** (ex. sono parametriche, quantificate universalmente)

$$\forall x, y: clear(x,s) \wedge clear(y,s) \Rightarrow on(x,y,move(x,y,s)) \wedge clear(x,move(x,y,s))$$

...ma si applicano ad un numero finito di oggetti! Potremmo introdurre una variabile logica per ogni possibile unificazione di una azione con le sue precondizioni

VARIABILE

$$clear(b1,s) \wedge clear(b2,s) \wedge move(b1,b2,s) \Rightarrow on(b1,b2,move(b1,b2,s)) \wedge clear(b1, move(b1,b2,s))$$
$$clear(b1,s) \wedge clear(b3,s) \wedge move(b1,b3,s) \Rightarrow on(b1,b3,move(b1,b3,s)) \wedge clear(b1, move(b1,b3,s))$$
...

SATPLAN

2. Le regole di deduzione sono ricorsive

$$\text{clear(b1,s)} \wedge \text{clear(b2,s)} \wedge \text{move(b1,b2,s)} \Rightarrow \underline{\textbf{on(b1,b2,move(b1,b2,s))}} \wedge \underline{\textbf{clear(b1, move(b1,b2,s))}}$$

...ma a noi interessano piani con un numero finito di azioni!
Potremmo introdurre una variabile logica per ogni possibile unificazione di una azione con le sue precondizioni ad ogni time step.

$$\text{clear(b1,t0)} \wedge \text{clear(b2,t0)} \wedge \text{move(b1,b2,t0)} \Rightarrow \text{on(b1,b2,t1)} \wedge \text{clear(b1, t1)}$$
$$\text{clear(b1,t1)} \wedge \text{clear(b3,t1)} \wedge \text{move(b1,b2,t1)} \Rightarrow \text{on(b1,b3,t2)} \wedge \text{clear(b1, t2)}$$
...

SATPLAN

3. **Le regole possono ammettere soluzioni spurie** perché “falso” implica qualunque cosa. Ex. Se “move” viene selezionata ed uno dei “clear” è falso, gli effetti possono essere ancora veri!

$$\text{clear}(b1,t0) \wedge \text{clear}(b2,t0) \wedge \text{move}(b1,b2,t0) \Rightarrow \text{on}(b1,b2,t1) \wedge \text{clear}(b1, t1)$$

...ma precondizioni ed effetti possono essere trattati in modo simmetrico:

$$\text{move}(b1,b2,t0) \Rightarrow \text{clear}(b1,t0) \wedge \text{clear}(b2,t0) \wedge \text{on}(b1,b2,t1) \wedge \text{clear}(b1, t1)$$

SATPLAN → BLACKBOX

- Con questi accorgimenti, per un certo numero di time step k:
  1. `problema = variabili logiche + vincoli logici(∧∨¬⇒)`
  2. `piano = un assegnamento delle variabili logiche`

SAT (SATisfiability problem): data una formula logica, trovare un assegnamento consistente delle sue variabili. Esistono risolutori molto efficienti per SAT.

SATPLAN nella sua prima versione effettuava questa conversione con k crescente, e risolveva il problema mediante un SAT solver.

BLACKBOX (Selman, Kautz 1999) migliora l'approccio costruendo mano a mano un planning graph, e convertendo il planning graph in un problema SAT.

Blackbox: algoritmo

```
function BLACKBOX(problema):
        grafo = GRAFO_INIZIALE(problema)
        obiettivi = GOAL(problema)
        loop do:
            if obiettivi non mutex nell'ultimo step:
                converti il grafo in un problema SAT
                Sol = RICERCA(problema SAT, risolutore)
                if sol Sol ≠ fail: return Sol
                else if LEVEL_OFF(grafo): return fail
            grafo = ESPANDI_GRAFO(grafo, problema)
```

- Rispetto a graphplan cambia solo il modo di fare ricerca (e la conversione, chiaramente)
- Si può usare qualunque SAT solver, o anche graphplan stesso

Esercizio

- Scaricare balckbox dal sito del corso
  `http://www.lia.deis.unibo.it/Courses/AI/applicationsAI2008-2009/Lucidi/Xesercitazioni/blackbox.gentoo.zip`
- Decomprimere l'archivio
  `unzip blackbox.gentoo.zip`
- Eseguire `blackbox` (viene visualizzato l'help)
  Blackbox risolve problemi specificati in PDDL (Planning Domain Definition Language). E' abbastanza intuitivo, ma una guida breve è comunque disponibile su:
  `http://www.ida.liu.se/~TDDA13/labbar/planning/2003/writing.html`
- L'esempio di Bud&Terence in PDDL è incluso nell'archivio

Esercizio

Provate:

1. Ad eseguire blackbox sull'esempio
2. Ad eseguire blackbox usando come solver Graphplan
3. Ad eseguire blackbox usando come solver CHAFF
4. A vedere la WFF (Well Formed Formula) in cui viene convertito il planning graph all'ultimo stadio (per questo destreggiatevi un po' tra la varie opzioni disponibili)

FF: Fast Forward

Pianificazione basata su grafi come euristica (accenni)

FF: Fast Forward

Fast Forward

FF è un pianificatore euristico estremamente efficiente introdotto da Hoffmann nel 2000

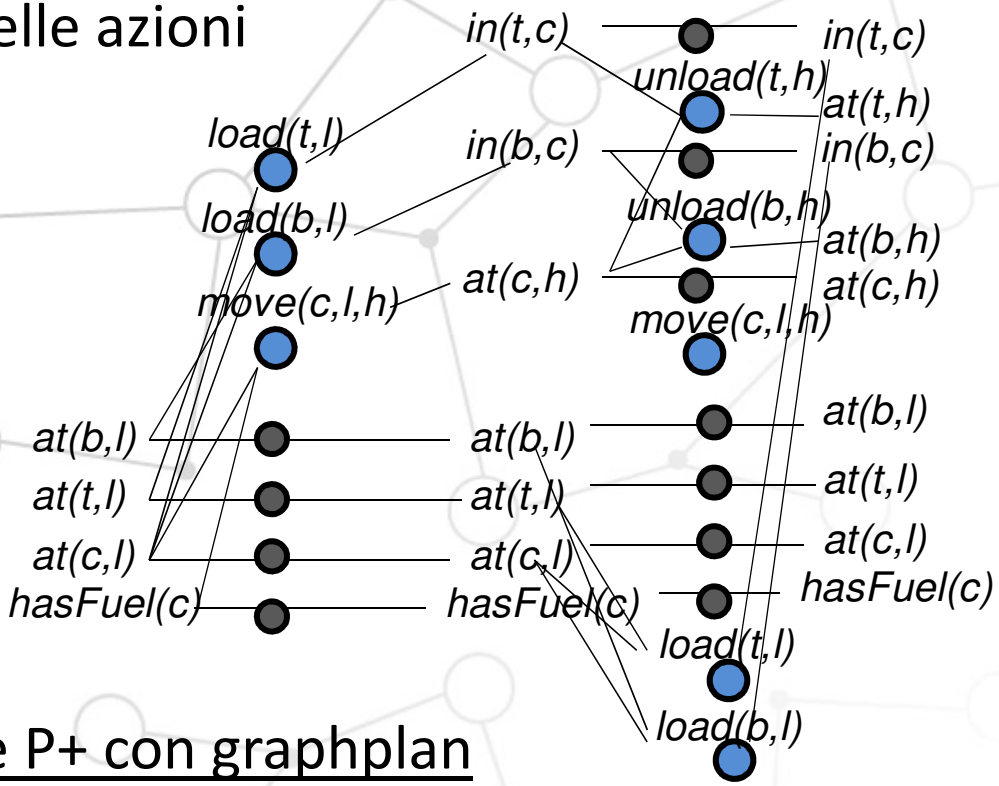
- Euristico = ad ogni stato S è una valutazione della distanza dal goal mediante una funzione euristica

Funzionamento base: hill climbing + A*

1. A partire da uno stato S, si esaminano tutti i successori S'
2. Se si individua uno stato successore S* migliore di S, ci si sposta su di esso e torna al punto 1
3. Se non si trova alcuno stato con valutazione migliore, viene eseguita una ricerca completa A*, usando la stessa euristica

La funzione euristica

- Dato un problema P, uno stato S ed un goal G, FF considera il problema rilassato P+ che si ottiene da P trascurando i delete effects delle azioni



- FF risolve P+ con graphplan
- Il numero di azioni nel piano risultante è utilizzato come euristica

Enforced hill climbing

FF utilizza in realtà un cosiddetto “enforced hill climbing”

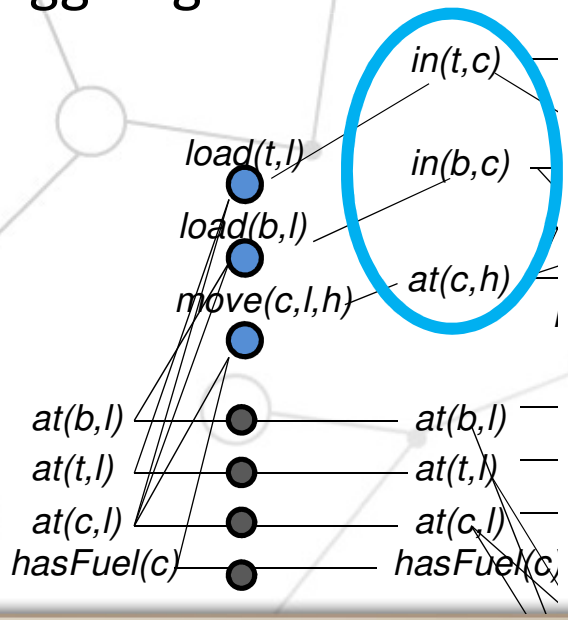
```
function FF(problema): return soluzione or fails
         S = STATO_INIZIALE(problema)
         k = 1
         loop do:
            explore all states S’ at k steps
            if a better state S* is found: S = S*
            else if k can be increased: k = k+1
            else perform complete A* search
```

- In pratica è una ricerca completa breadth first
- Una soluzione viene sempre trovata, a meno che lo stato corrente non sia un vicolo cieco

Helpful actions

Per muoversi da uno stato S ad uno adiancente non si considerano tutte le azioni, ma solo le cosiddette **azioni utili (helpful actions)**

- Sia G1 l'insieme delle proposizioni al time step 1 della soluzione del problema rilassato P+: $H(S) = \{pre(o) \subseteq S, add(o) \cap G1 \neq \emptyset\}$
- Ossia H(S) contiene le azioni applicabili nello stato corrente (S), che aggiungono almeno una delle proposizioni in G1




```
H(S0) = {load(t,l),
         load(b,l),
         move(c,l,h)}
```

Esercizi

- Scaricare FF dal sito del corso
  `http://www.lia.deis.unibo.it/Courses/AI/applicationsAI2008-2009/Lucidi/Xesercitazioni/ff.gentoo.zip`
- Decomprimere l'archivio
- Provare ad eseguire il programma sull'esempio del carrello in pddl
- Guardate come varia la distanza dal goal
- Provate a modificare il modello in modo da rendere il problema impossibile: come varia la distanza dal goal?

Riferimenti

- GRAPHPLAN
  - http://www.cs.cmu.edu/~avrim/graphplan.html
  - A. Blum and M. Furst, "Fast Planning Through Planning Graph Analysis", Artificial Intelligence, 90:281--300 (1997).
- Blackbox
  - http://www.cs.rochester.edu/u/kautz/satplan/blackbox/index.html
  - SATPLAN: http://www.cs.rochester.edu/u/kautz/satplan/index.htm
  - Henry Kautz and Bart Selman, “Planning as Satisfiability”, Proceedings ECAI-92.
  - Henry Kautz and Bart Selman, “Unifying SAT-based and Graph-based Planning”, Proc. IJCAI-99, Stockholm, 1999.
- Fast Forward
  - http://members.deri.at/~joergh/ff.html
  - J. Hoffmann, “FF: The Fast-Forward Planning System”, in: AI Magazine, Volume 22, Number 3, 2001, Pages 57 - 62

Esercizi

Modellazione: esercizio 1

Modellare in pddl l'**eight puzzle**

`Stato iniziale`

7	3	8
2		1
8	6	5

`Goal`

1	2	3
4	5	6
7	8	

- Provare a risolverlo con i vari pianificatori
- Come variano le performance "mescolando" un altro po'?

Modellazione: esercizio 2

Modellare per graphplan il classico indovinello “capra-lupo-cavolo” sulla riva del fiume. ATTENZIONE: è molto difficile!

- Lupo, capra e cavolo, barca e pastore sono sulla sponda A
- Il lupo se lasciato da solo con la capra la mangia
- La capra se lasciata sola con il cavolo lo mangia
- Il pastore può portare solo un oggetto alla volta con la barca
- Tutti devono arrivare sulla sponda B

Valutazione: esercizio 1

Sul sito del corso sono disponibili alcuni pacchetti di istanze dall'International Planning Competition, edizione 3.

- Scaricate le istanze dal sito
- Due domini:
  - Rovers: gestire le comunicazione tra robot per l'esplorazione di un pianeta
  - Freecell: AKA il buon vecchio solitario di windows!
- Provate a vedere che performance hanno Graphplan (Blackbox + Graphplan), Blackbox + Chaff e FF sulle istanze

Valutazione: esercizio 2

Sul sito del corso sono disponibili alcuni pacchetti di istanze dall'International Planning Competition, edizione 5.

- Scaricate le istanze dal sito
- Un dominio:
  - TPP (Traveling Purchaser Problem)
  - Attenzione: per avere una modellazione STRIPS classica occorre una file di dominio per ogni file di istanza
- Provate a vedere che performance hanno Graphplan (Blackbox + Graphplan), Blackbox + Chaff e FF sulle istanze