# Pianificazione basata su ricerca euristica state-space forward: *Fast Forward (FF)*

Alfonso Gerevini, Alessandro Saetti

Intelligenza Artificiale B, Università degli Studi Brescia

# Bibliografia

Per chi vuole approfondire l'argomento:

- *The FF Planning System: Fast Plan Generation Through Heuristic Search*
  http://www.mpi-sb.mpg.de/~hoffmann/papers/jair01.ps.gz

- *The Metric FF Planning System: Translating "Ignoring Delete Lists" to Numerical State Variables*
  http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/hoffmann03a.ps.Z

- Utilizing Problem Structure in Planning: A Local Search Approach
  http://www.mpi-sb.mpg.de/~hoffmann/papers/lnai03-abstract.ps.gz

# Introduzione

- FF (Fast Forward) è un sistema di pianificazione efficiente per una grande classe di domini benchmark
- pianificazione forward
- ricerca nello spazio degli stati
- utilizza un euristica derivata da Graphplan

# Architettura di Sistema

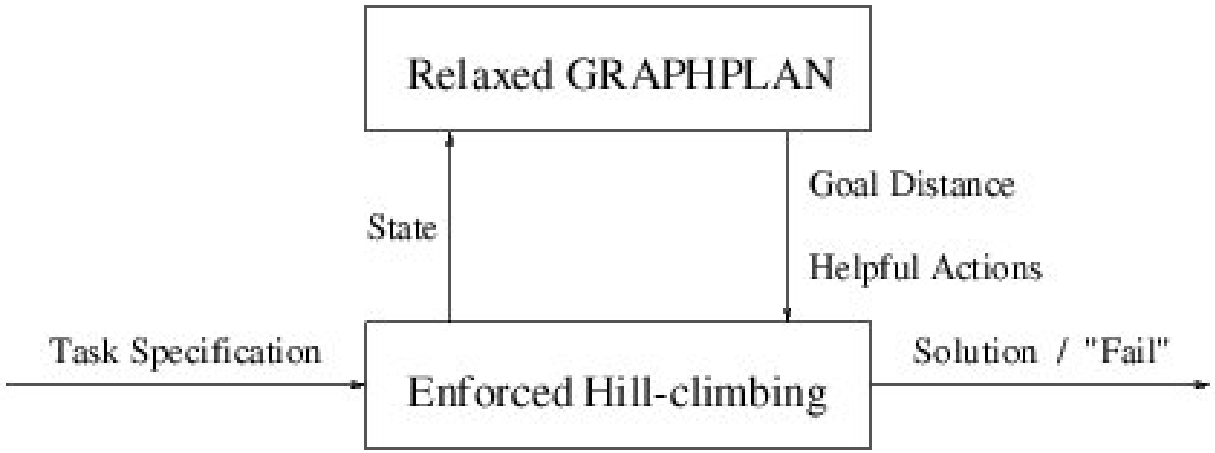



Figure 1: FF's base system architecture.

Se un problema di pianificazione contiene *dead-ends*, *enforced hill-climbing* può fallire. In tal caso si usa una ricerca euristica esaustiva.

# Problema Rilassato

- Una azione STRIPS $o$ è una tripla

$$o = (pre(o), add(o), del(o))$$

tale che

$$Result(S, \langle o \rangle) = \begin{cases} (S \cup add(o) \setminus del(o)) & pre(o) \subseteq S \\ undefined & otherwise \end{cases}$$

- Un **problema di pianificazione** proposizionale $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$ è una tripla dove $\mathcal{O}$ è un insieme di azioni, $\mathcal{I}$ lo stato iniziale e $\mathcal{G}$ i goals

- Dato un problema di pianificazione $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, il **problema di pianificazione rilassato** $\mathcal{P}'$ di $\mathcal{P}$ è definito come $\mathcal{P}' = (\mathcal{O}', \mathcal{I}, \mathcal{G})$, con

$$\mathcal{O}' = \{(pre(o), add(o), \emptyset) \mid (pre(o), add(o), del(o)) \in \mathcal{O}\}$$

# Piano Rilassato

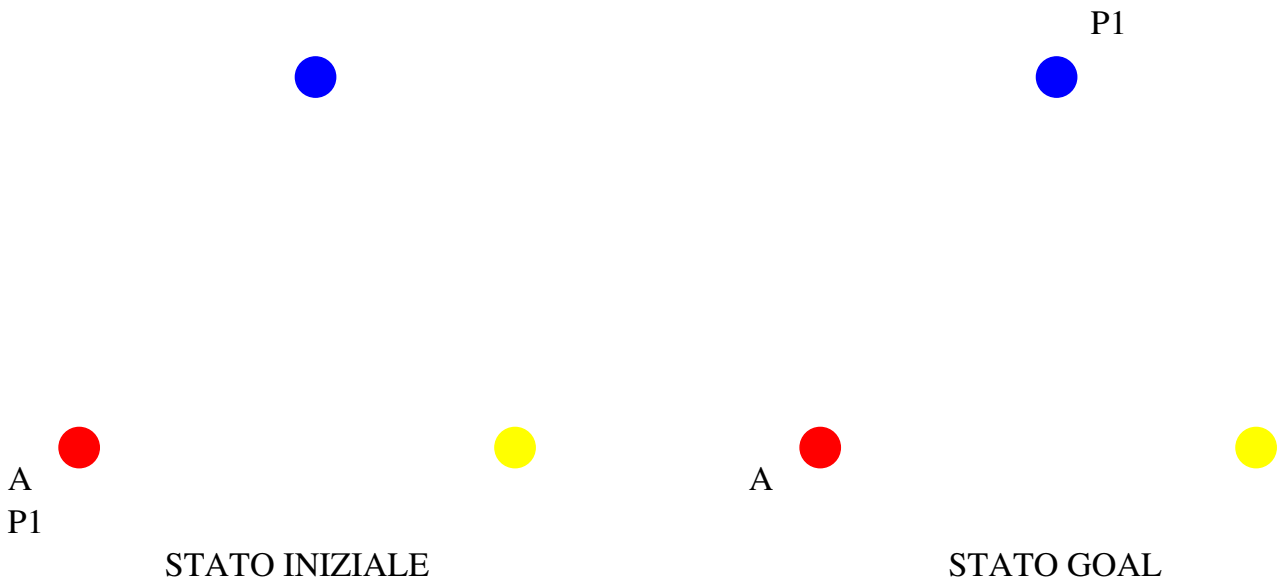
- Il risultato dell'applicazione di una sequenza di azioni

$$Result(S, \langle o_1, \cdots, o_n \rangle) = Result(Result(S, \langle o_1, \cdots, o_{n-1} \rangle), \langle o_n \rangle)$$

- Dato un problema di pianificazione $\mathcal{P} = (\mathcal{O}, \mathcal{I}, \mathcal{G})$, un **piano** è una sequenza di azioni, per cui $\mathcal{G} \subseteq Result(\mathcal{I}, \mathcal{P})$

- un piano è **rilassato** per $\mathcal{P}$, sse risolve il problema rilassato $\mathcal{P}'$ di $\mathcal{P}$

# Esempio di Piano Rilassato




STATO INIZIALE

STATO GOAL

Piano: `load(A,P1,Red)`, `fly(A,Red,Blue)`, `unload(A,P1,Blue)`, `fly(A,Blue,Red)`

Piano Rilassato: `load(A,P1,Red)`, `fly(A,Red,Blue)`, `unload(A,P1,Blue)`

# Euristica di HSP

- piano rilassato utilizzato per fornire una stima euristica
- euristica inspirata a HSP: Heuristic Search Planner (Bonet & Geffner 1998)
- HSP assume l'indipendenza nella realizzazione dei goals

$$h(S) = weight_S(\mathcal{G}) = \sum_{g \in \mathcal{G}} weigth_S(g)$$

$$weight_S(f) = \begin{cases} 0 & se\ f \in S \\ i & se\ min_{o \in \mathcal{O}, f \in add(o)} \sum_{p \in pre(o)} weight_S(p) = i - 1 \\ \infty & altrimenti \end{cases}$$

# Esempio Euristica HSP

Consideriamo il problema di pianificazione, dove lo stato iniziale è vuoto, i goals sono $G_1$ e $G_2$ e ci sono 3 azioni:

| nome | | (pre, | add, | del) |
|---|---|---|---|---|
| op$G_1$ | = | ($\{P\}$, | $\{G_1\}$, | $\emptyset$) |
| op$G_2$ | = | ($\{P\}$, | $\{G_2\}$, | $\emptyset$) |
| op$P$ | = | ($\{\emptyset\}$, | $\{P\}$, | $\emptyset$) |

Distanza dai goals da $S = \{\}$ per HSP: 4

Al fine di tenere in considerazione interdipendenze positive, FF utilizza GRAPHPLAN sul problema $(\mathcal{O}', S, \mathcal{G})$

# GraphPlan per Problemi Rilassati

Sia $\mathcal{P}' = (\mathcal{O}', \mathcal{I}, \mathcal{G})$ un problema rilassato,

- GraphPlan non considera alcuna coppia di azioni e di fatti come *mutuamente esclusiva* per $\mathcal{P}'$
- GraphPlan non necessita backtracking per $\mathcal{P}'$
- GraphPlan trova una soluzione per $\mathcal{P}'$ in un tempo polinomiale in $l$, $\mid \mathcal{O}' \mid$ e $\mid \mathcal{I}' \mid$

# Euristica di FF

Sia $(O_0, \cdots, O_{m-1})$ il piano rilassato, dove $O_i$ è l'insieme di azioni selezionate da Graphplan al livello $i$ e $m$ è il livello contenenti i goals

$$h(S) = \sum_{i=0,\cdots m-1} | O_i |$$

GraphPlan tiene conto di interazioni positive tra i goals.

# Esempio Euristica FF

Consideriamo il problema di pianificazione, dove lo stato iniziale è vuoto, i goals sono $G_1$ e $G_2$ e ci sono 3 azioni:

| nome | | (pre, | add, | del) |
|---|---|---|---|---|
| op$G_1$ | = | ($\{P\}$, | $\{G_1\}$, | $\emptyset$) |
| op$G_2$ | = | ($\{P\}$, | $\{G_2\}$, | $\emptyset$) |
| op$P$ | = | ($\{\emptyset\}$, | $\{P\}$, | $\emptyset$) |

Piano costruito da GraphPlan: $\langle$ { op$P$ }, { op$G_1$, op$G_2$}$\rangle$

Distanza dai goals per FF = 3

# Esempio Euristica FF

- La stima é più accarata se la soluzione é più corta (meno azioni)
- Bisogna modificare GraphPlan per fargli ritornare la soluzione un piano compatto nel numero di azioni (non livelli!)
- Sia $\mathcal{O}', \mathcal{I}, \mathcal{G}$ un problema di pianificazione rilassato. Usando la strategia *noop-first*, il piano di GraphPlan conterrà un'azione al più una volta
- Se non esiste una no-op, si seleziona l'azione le cui precondizioni sono "più facili da supportare"

  $difficulty(o) = \sum_{p \in pre(o)} min\{i \mid p \text{ fa parte del livello fatto } i\}$
- la difficoltà di ciascuna azione può essere ricavata durante la costruzione del grafo di pianificazione

# Piani Sequenziali dal Planning Graph

- FF ricerca piani sequenziali (i relaxed plan devono essere sequenziali), ma GraphPlan costruisce un piano parallelo per il problema rilassato
- Bisogna *linearizzare* il piano parallelo
- Differenti linearizzazioni producono piani di lunghezza differente (inferiore al numero di azioni nel piano parallelo)
- Ad es: $A$ e $B$ nel piano a stesso livello; $A$ ha effetto *aggiuntivo* $f$, $B$ ha precond $f$; $C$ al livello precedente di $B$ supporta $f$: se linearizziamo $A$ prima di $B$, $C$ non serve!
- Trovare la migliore linearizzazione è un problema NP-completo
- Le azioni sono linearizzate nell'ordine in cui sono scelte

# Estrazione Piano Rilassato

**for** $i := 1, \ldots, m$ **do**
    $G_i := \{g \in \mathcal{G} \mid \text{layer-membership}(g) = i\}$
**endfor**
**for** $i := m, \ldots, 1$ **do**
    **for** all $g \in G_i$, $g$ not marked TRUE at time $i$ **do**
        select an action $o$ with $g \in add(o)$ and layer membership $i-1$, $o$'s difficulty being minimal
        **for** all $f \in pre(o)$, layer-membership$(f) \neq 0$, $f$ not marked TRUE at time $i-1$ **do**
            $G_{\text{layer-membership}(f)} := G_{\text{layer-membership}(f)} \cup \{f\}$
        **endfor**
        **for** all $f \in add(o)$ **do**
            mark $f$ as TRUE at times $i-1$ and $i$
        **endfor**
    **endfor**
**endfor**

# Algoritmo di Ricerca

- Hill-Climbing: seleziona il migliore stato successore (ottenuto applicando un'azione allo stato corrente)
- ricerca locale affetta dal problema dei minimi locali
- con la funzione euristica di FF, i minimi locali e plateaus tendono a essere piccoli
- Enforced Hill Climbing = Hill Climbing + Breadth First

# Enforced Hill-Climbing

- per ogni stato $S$ viene invocato un algoritmo breadth-first da $S$
- breadth-first trova il più vicino stato $S'$ con valutazione migliore di $S$ oppure fallisce
- se fallisce, l'algoritmo fallisce
- altrimenti, si aggiunge al piano il percorso da $S$ a $S'$
- quando si raggiunge uno stato goal, la ricerca termina

# Breadth-First in Enforced Hill-Climbing

- gli stati sono conservati in una coda
- il primo stato $S'$ della coda è valutata da GraphPlan
- se la valutazione di $S'$ è migliore di quella di $S$, la ricerca ha successo
- altrimenti, gli stati successori di $S'$ sono messi alla fine della coda
- gli stati ripetuti sono evitati
- se nessun nuovo stato può essere raggiunto, la ricerca breadth-first fallisce

# Enforced Hill-Climbing

initialize the current plan to the empty plan <>
$S := \mathcal{I}$
**while** $h(S) \neq 0$ **do**
    perform breadth first search for a state $S'$ with $h(S') < h(S)$
    **if** no such state can be found **then**
        output "Fail", stop
    **endif**
    add the actions on the path to $S'$ at the end of the current plan
    $S := S'$
**endwhile**

# Dead-Ends

Sia $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ un problema di pianificazione, $S$ è un **dead-end** sse $\exists P, S = Result(\mathcal{I}, P)$ e $\neg\exists P', \mathcal{G} \subseteq Result(S, P')$

Metodo completo solo per problemi non contenenti dead-ends

Decidere se un problema di pianificazione è *Dead-end Free* è PSPACE-completo

- esegui Enforced Hill-Climbing fino a raggiungere uno stato goals oppure fallimento

- se l'Enforced Hill-Climbing fallisce, esegui *best-first search* (A* con $g(S) = 0$ e $h(S)$ euristica FF)

# Helpful Actions

- consentono il pruning della ricerca breadth-first
- derivate dal piano di GraphPlan per il problema rilassato $\mathcal{P}'$
- **helpful actions** (per uno stato $S$): *insieme di azioni applicabili in $S$ che supportano i primi (sub)goal del piano lineare rilassato per $S$*
- il breadth first analizza solo quegli stati che possono essere raggiunti da helpful actions
- il pruning attraverso *Helpful Actions* non preserva la completezza neppure per problemi senza dead-ends (ma spesso utili!)

# Added Goal Deletion

- riguarda l'ordinamento dei goals
- è inutile realizzare $G_1$, se per realizzare $G_2$ è necessario distruggere $G_1$:

  *Se la soluzione del problema rilassato (can stato iniziale $S$) contiene un'azione $o$ che cancella un goal $G$ giá realizzato dalla ricerca BF, allora rimuoviamo $S$ dallo spazio di ricerca della BF*

- Esempio (Blocksworld): Siamo nello stato in cui $A$ è su $B$ e $B$ e $C$ sono sul tavolo (e i goal sono $A$ on $B$ e $B$ on $C$)

  $<$ { unstack(A,B) },
  { pickup(B) },
  { stack(B,C) } $>$

  Il goal $on(A, B)$ è cancellato da unstack A B

- il pruning attraverso *Added Goal Deletion* non preserva la completezza neppure per problemi senza dead-ends

# Goal Agenda

- impone un ordinamento nel raggiungere i goals
- suddivide i goals in una serie di sottoinsiemi $G_1, \ldots, G_n$
- enforced hill-climbing dallo stato iniziale per $G_1$
- enforced hill-climbing dal nuovo stato iniziale a $G_1 \cup G_2$, ecc...
- preserva la completezza per problemi di pianificazione senza dead-ends