Graphplan

Sommario
Il pianificatore Graphplan è stato proposto da Blum e Furst[1] per costruire piani in modo efficiente in domini STRIPS-like. L'algoritmo usato dal pianificatore inizia costruendo una struttura compatta chiamata *planning graph*. Il planning graph codifica il problema di planning in modo da rendere esplicitamente disponibili molti vincoli inerenti al problema. Il planning graph può essere costruito in tempo polinomiale, quindi in modo efficiente rispetto al problema di pianificazione che di solito ha complessità almeno esponenziale. Il pianificatore Graphplan usa il planning graph per guidare la ricerca di un piano. Questa ricerca viene effettuata all'indietro, ossia guidata dal goal, mentre invece la costruzione del planning graph procede in avanti. Graphplan genera piani parzialmente ordinati in cui *più azioni* possono essere eseguite in parallelo, se non interferiscono fra di loro.

Domini STRIPS-like
L'algoritmo si applica a domini STRIPS-like, ossia domini costituiti da:
- un insieme di *oggetti* (ad es. i blocchi b1, b2, ...)
- un insieme di *operatori* (pickup(X), stack(X,Y), ...)
- un insieme di *proposizioni* (ontable(X), clear(X), ...)

Le operazioni hanno *precondizioni*, *effetti di tipo add* ed *effetti di tipo delete*, tutti rappresentati come insiemi di proposizioni.

Ad esempio, l'azione pickup(X) ha:
precondizioni: {ontable(X), clear(X), handempty}
effetti delete: {ontable(X), clear(X), handempty}
effetti add: {holding(X)}

L'algoritmo richiede che tutte le proposizioni e tutte le operazioni siano istanziate (ossia le variabili devono essere sostituite da oggetti). Es. clear(b1), holding(b2), pickup(b1), ...
Chiamiamo **azioni** le operazioni completamente istanziate.
Uno **stato** viene rappresentato con un *insieme di proposizioni* (istanziate). L'esecuzione di una azione in uno stato aggiunge tutte le proposizioni che sono suoi effetti add e cancella tutte le proposizioni che sono effetti delete. Le proposizioni non cancellate persistono nel nuovo stato. Questo risolve il cosiddetto *frame problem*, ossia il problema della persistenza delle proposizioni che non sono influenzate dall'esecuzione di una azione.
Graphplan viceversa tratta in modo esplicito il *frame problem* usando delle azioni particolari, chiamate azioni *no-op*, che "non fanno niente". Assumiamo quindi di avere una azione **persists(P)** per ogni proposizione P. Questa azione ha come precondizione P e come effetto di tipo add P: in pratica fa passare P da uno stato a quello successivo.

Come è fatto un piano
Un piano è una *sequenza di insiemi di azioni* (comprese azioni *no-op*). Le azioni che compaiono nello stesso insieme possono essere eseguite in parallelo e non devono interferire fra di loro.
Ad esempio nel modo dei blocchi, dato uno stato iniziale
{ *on(a,b), clear(a), ontable(b), handempty*}
è possibile eseguire il piano di due passi
{*unstack(a,b), persists(ontable(b))*} {*putdown(a), persists(ontable(b)), persists(clear(b))*}
per mettere il blocco a sul tavolo.
Eseguendo il primo insieme di azioni si ottiene lo stato

[1] A.L. Blum e M.L. Furst, Fast planning through planning graph analysis, Artificial Intelligence 90 (1997) 281-300.

{ *holding(a), clear(b), ontable(b)* }

dove le prime due proposizioni sono l'effetto dell'azione *unstack(a,b)*, mentre la proposizione *ontable(b)* è l'effetto della persistenza. Analogamente, eseguendo il secondo insieme di azioni si ottiene lo stato

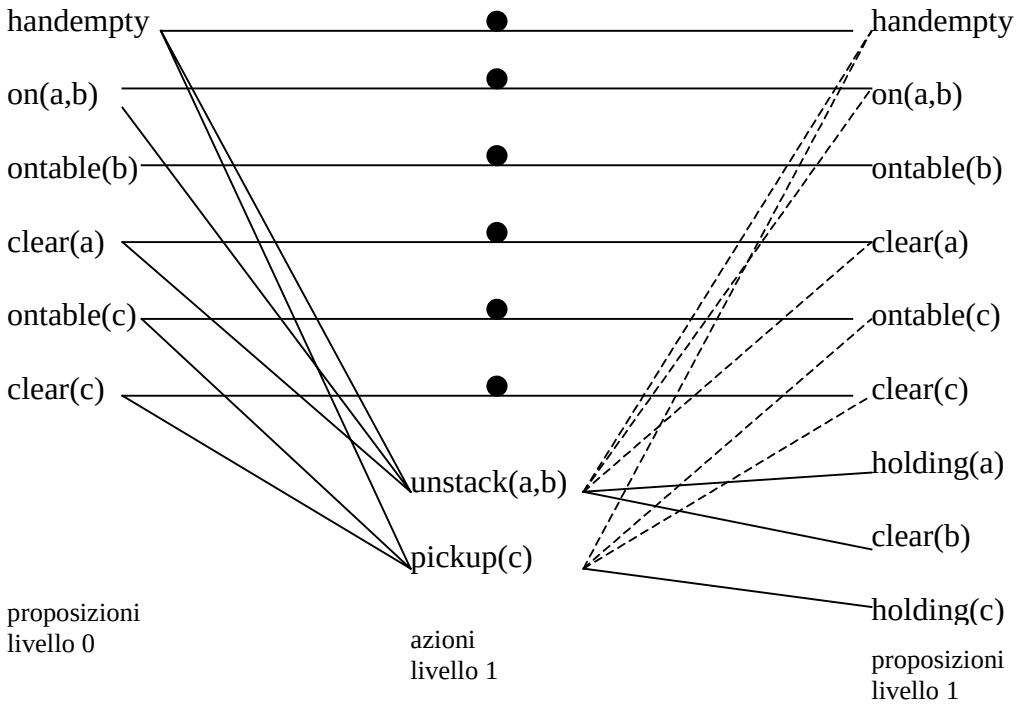
{ *ontable(a), clear(a), handempty, ontable(b), clear(b)*}

Come è fatto un planning graph

Un planning graph è un grafo a livelli con due tipi di nodi e tre tipi di archi. I livelli del grafo si alternano fra *livelli di proposizioni*, i cui nodi sono etichettati con proposizioni, e *livelli di azioni*, i cui nodi sono etichettati con azioni. Il livello 0 è un livello proposizionale e contiene un nodo per ogni proposizione nelle stato iniziale. Successivamente ci sarà un livello 1 di azioni seguito da un livello 1 di proposizioni, e così via. Il livello 1 di azioni contiene tutte le azioni (comprese le *no-op*) che possono essere eseguite avendo come prerequisiti le proposizioni del livello 0. Viceversa il livello 1 di proposizioni contiene tutte le proposizioni che sono effetto add di una qualche azione del livello 1 di azioni (notare che, siccome ci sono anche azioni *no-op*) il livello 1 di proposizioni conterrà tutte le proposizioni del livello 0.

Gli archi rappresentano relazioni fra azioni e proposizioni. I nodi azione al livello i sono collegati con archi precondizione alle loro precondizioni al livello i-1, mentre sono collegati con archi add o delete ai loro effetti add o delete al livello i.

La figura seguente contiene un esempio per il mondo dei blocchi.



I pallini rappresentano azioni no-op. Le linee tratteggiate rappresentano archi delete.

E' possibile vedere che, dato un planning graph di n livelli, se esiste un piano con un numero di passi ≤ n, questo piano sarà un sottografo del planning graph.

Mutua esclusione fra nodi
Due azioni o due proposizioni allo stesso livello sono *mutuamente esclusive* se non può esistere nessun piano che le contiene entrambe. La procedura di costruzione del planning graph registra le relazioni di mutua esclusione fra azioni o proposizioni in modo da rendere più efficiente la ricerca di un piano.
Ci sono due cause che fanno sì che due azioni a e b mutuamente esclusive:

- *Interferenza*. Se una delle due azioni cancella una precondizione o un effetto add dell'altra.
- *Precondizioni incompatibili*. Se ci sono una precondizione di a e una di b mutuamente esclusive fra loro.

Due proposizioni p e q vengono marcate come mutuamente esclusive se tutti i modi di creare p sono esclusivi con tutti i modi di creare q. Più precisamente se ogni azione che ha p come effetto add è mutuamente esclusiva con ogni azione che ha q come effetto add.

Ad esempio nella figura precedente, assumendo che tutte le proposizioni al livello 0 siano compatibili, le due azioni *unstack(a,b)* e *pickup(c)* risultano esclusive per interferenza, perché una cancella la precondizione *handempty* dell'altra. Analogamente *pickup(c)* e *persists(clear(c))* sono mutuamente esclusive perché l'azione *pickup(c)* cancella *clear(c)* che è l'effetto dell'altra azione.
Le due proposizioni di livello 1 *holding(a)* e *holding(c)* sono mutuamente esclusive perché le azioni che le generano sono mutuamente esclusive.

Descrizione dell'algoritmo
L'algoritmo procede per livelli, iniziando con un planning graph di livello 0. Supponendo di avere un planning graph di livello i-1, l'algoritmo estende il grafo di un livello, costruendo il livello i di azioni ed il livello i di proposizioni. A questo punto l'algoritmo inizia la ricerca di un piano di lunghezza i muovendosi all'indietro a partire dal livello i. Se non si trova una soluzione, si itera estendendo il planning graph di un ulteriore livello.
In sostanza l'algoritmo adotta una tecnica di iterative deepening, che trova sempre il piano di lunghezza minima.

a) Come estendere il planning graph
Dato un planning graph con i-1 livelli, il livello i di azioni ed il livello i di proposizioni sono ottenuti come segue. Il livello di azioni i contiene un nodo per ogni azione (comprese le azioni no-op) le cui precondizioni valgono nel livello i-1, purché non ci siano due di queste precondizioni mutuamente esclusive fra loro. Occorre inoltre calcolare tutte le coppie di azioni mutuamente esclusive.
Il livello i delle proposizioni è costituito da tutti gli effetti add di tutte le azioni (comprese le azioni no-op) del livello i. Occorre inoltre calcolare tutte le coppie di proposizioni mutuamente esclusive.
La costruzione di questo grafo richiede un tempo polinomiale.

b) Come cercare un piano in un planning graph
Dato un planning graph di i livelli, Graphplan cerca una soluzione con una procedura di ricerca in propfondità che usa una strategia backward (ossia partendo dal *goal*), muovendosi per livelli. Per prima cosa l'algoritmo verifica che il goal da raggiungere (un insieme di proposizioni) sia contenuto nelle proposizioni di livello i, e che non ci siano due proposizioni del goal mutuamente esclusive. Se questo non si verifica, certamente non c'è soluzione.
Supponiamo che la ricerca, muovendosi all'indietro, sia arrivata al livello k con un certo goal G_k, costituito da un insieme di proposizioni. L'algoritmo determina (nondeterministicamente) un insieme di azioni (comprese le azioni no-op) del livello k che abbiano le proposizioni del goal fra i propri effetti add e che non siamo mutuamente esclusive. Le precondizioni di queste azioni

costituiscono il nuovo goal G_{k-1} su cui la procedura viene chiamata ricorsivamente, fino a quando raggiunge il livello 0.
Naturalmente la procedura può raggiungere un livello da cui non riesce più a continuare. In questo caso fa backtracking fino al punto in cui esiste la possibilità di scegliere un altro insieme di azioni.
E' facile vedere che questa fase di ricerca può richiedere un tempo esponenziale.

NOTA Possibile ottimizzazione
Supponiamo che l'algoritmo di ricerca scopra che un certo goal G ad certo livello i non ha soluzione. Prima di fare backtracking l'algoritmo potrebbe memorizzare questa informazione (in Prolog con una assert), in modo che se in una iterazione successiva si ritrova nella stessa situazione possa fallire immediatamente senza rifare tutta la ricerca.

ESEMPI DI DOMINI

Blocchi
Mondo dei blocchi con le azioni pickup, putdown, stack e unstack.
Visto che Graphplan genera dei piani paralleli, può essere interessante estendere il dominio in modo da aver più di un braccio per muovere i blocchi.

Rocket
Gli oggetti sono:
- un insieme di posizioni
- un insieme di carichi
- un insieme di rocket per spostare i carichi da una posizione all'altra

Le proposizioni sono
- at(O,P) dove O può essere o un rocket o un carico e P è una posizione
- in(C,R) il carico C è nel rocket R
- has_fuel(R) il rocket R ha carburante
- no_fuel(R) il rocket R non ha carburante

Le azioni sono:
- load(R,P,C) carica C nel rocket R alla posizione P
- unload(R,P,C) scarica C dal rocket R alla posizione P
- move(R,From,To) sposta il rocket R dalla posizione From alla posizione To
- refuel(R) fai il pieno a R

Precondizioni, effetti add e delete possono essere definiti in modo intuitivo, tenendo presente che quando un carico si trova in un rocket non è in nessuna posizione e viceversa, e che un pieno serve per un solo spostamento.

Shakey (Russell e Norvig, 11.8)
Il dominio consiste in un insieme di stanze collegate da un corridoio, un insieme di scatole che possono essere spostate, un interruttore in ogni stanza e un robot (anche in questo caso sarebbe più interessante averne più di uno). Per eseguire l'azione di accendere o spegnere la luce il robot deve salire su una scatola.