Ultima Parte del Corso

- Tipologie di agenti intelligenti e ambienti
- Risoluzione di problemi con ricerca e CSP discreti
- Conoscenza e ragionamento logico (con LP e FOL)
- CSP per domini continui (ragionamento temporale e spaziale)
- Algoritmi basati su grafi per CSP
- **Pianificazione automatica**: linguaggi, algoritmi, sistemi
- **Conoscenza e ragionamento con incertezza:**
  - ragionamento probabilistico e Reti Bayesiane
  - Agenti basati su utilità e processi decisionali: MDP e altro

Pianificazione Automatica:

linguaggi, algoritmi e sistemi

Pianificare…

- ***Scegliere e organizzare azioni per raggiungere determinati obiettivi o "goal" (condizioni desiderate sullo stato del mondo nel dominio applicativo)***
- Alcuni goal richiedono pianificazione altri no (azioni riflessive, piani memorizzati)
- Utile per nuove situazioni/compiti, o compiti complessi (non si possono memorizzare tutti i piani!). Ad es:
  - Situazioni di emergenza, imprevisti di varia natura a cui far fronte
  - Organizzare un viaggio complesso in gruppo con goal, vincoli e preferenze diversi
- Importante in applicazioni di sistemi/agenti (semi)autonomi: ad es., robot (spazio, terra, mare, aria), giochi, agenti internet (composizione servizi web), progettazione/configurazione assistita, ecc…

Tre metodi di pianificazione

- **Domain-dependent**: algoritmi e programmi *specifici* per tipi di pianificazione e *domini specifici* (es torre di Hanoi)
- **Domain-independent**: algoritmi e programmi *generali* utilizzabili per una *ampia classe di domini*
- **Domain-configurable***:* algoritmi e programmi generali *configurabili attraverso conoscenza aggiuntiva* specifica del dominio applicativo *per funzionare efficacemente*

Il modello delle azioni e l'input sono differenti!

Tipologie di planning domain independent

- **Project planning**:
  - **Modello azione**: vincoli temporali tra azioni "senza semantica esplicita" (è nella mente dell'utente!). *Ma cosa è la semantica?*
  - **Input**: *nomi* azioni da svolgere + vincoli temporali (durate, precedenze, scadenze, ecc.) che definiscono un "piano".
  - **Output**: Piano usato *interattivamente* (query, consistenza, cammino critico, ordinamenti impliciti, ecc.) Esempi: alla lavagna…..

Tipologie di planning domain independent

- **Scheduling/allocazione risorse**:
  - **Modello azione:** come project planning + vincoli su risorse disponibili
  - **Input**: *nomi* azioni da svolgere, vincoli globali su risorse e temporali/di ordinamento, criterio di ottimizzazione
  - **Output**: azioni di input ordinate e risorse allocate in modo da soddisfare i vincoli e il criterio di ottimizzazione

Tipologie di planning domain independent

- **Sintesi Automatica di Piani (model based)**:
  - **Modello azione**: ha semantica esplicita: precondizioni e effetti (+ eventuali info come nello scheduling).
  - **Modello in input**: ***descrizione*** *azioni* + *goal del problema* (condizioni sullo stato del mondo desiderato) + *descrizione stato del mondo iniziale* + eventuali vincoli temporali e su risorse, criterio ottimizzazione
  - **Output**: *Sottoinsieme* ordinato delle azioni di input, il cui effetto globale, se le azioni sono eseguite nell'ordine determinato, raggiunge il goal del problema senza violare i vincoli di ingresso ed ottimizzando la qualità del piano prodotto secondo il criterio dato di input

Sintesi Automatica di Piani

- Simile alla risoluzione di problemi di ricerca (problem solving), ma con importanti differenze!

- **Linguaggio di pianificazione**: *Linguaggio formale ben definito (sintassi e semantica) che consente la specifica dell'input di un problema di pianificazione e delle soluzioni prodotte. Ad es.:*
  - STRIPS: STanford Research Institute Planning System
  - ADL: Action Description Language
  - PDDL: Planning Domain Definition Language (standard internazionale!)

- **Pianificatore:** *Collezione di algoritmi e strutture dati che, a partire da un problema di pianificazione specificato con un linguaggio di pianificazione, genera automaticamente una soluzione (cioè un piano valido)*

Varie tipologie di pianificazione

- **Deterministica (o classica)**: *"off-line", lo stato del mondo e gli effetti delle azioni sono conosciuti con certezza*
- **Non-deterministica**: *lo stato del mondo e/o gli effetti delle azioni sono incerti (disgiunzione o modelli probabilistici)*
- **Interattiva:** *il programma e l'utente umano "collaborano" al fine di risolvere il problema (collaborative problem solving)*
- **Con preferenze e utilità** *sui goal e sulla struttura delle soluzioni*
- **"Interleaved"**: *combinata con l'esecuzione (problemi real-time)*
- **Continua**: *durante la pianificazione vengono continuamente aggiunti e modificati i goal*
- **Multi agente:** *pianificazione distribuita cooperativa/competitiva*
- **Mirata alla revisione** (off-line/on-line) *di un piano esistente*
- **Gestione di un data base di piani**: *archiviazione, reperimento*
- **Temporale/concorrente, numerica, ecc…**

Alcuni esempi

- Linguaggio standard PDDL
- Dominio e problema di pianificazione
- Breve dimostrazione di un pianificatore di successo: LPG
  - Vincitore di due competizioni internazionali
  - Sviluppato a Brescia!

Un semplice linguaggio: STRIPS

STRIPS = STanford Research Institute Planning System

- **Descrizione operatori**: *classi* di azioni formate specificate da parametri, precondizioni, effetti +/- (*add-list, delete-list*).

  Esempio di operatore (schema di azioni):

  Fly(?p, ?from, ?to)

  Parametri: ?p,?from,?to

  Prec: Plane(?p) & Airport(?from) & Airport(?to) & At(?p,?from)

  Eff.: not At(?p,?from) & At(?p,?to)

- **Descrizione stati**:
  - Uno stato è un insieme di semplici formule logiche (KB)
  - *Letterali "function free"* (implica numero finito azioni con numero finito di oggetti!)
  - Assunzione "mondo chiuso" (*Closed World Assumption*)

Un semplice linguaggio: STRIPS

- **Altra assunzione: Inerzia dei fatti** per risolvere il "*Frame Problem*"
  - *Una proposizione (o più in generale una formula) P resta vera (falsa) da uno stato s1 al successivo s2 se e solo se non c'è un'azione che eseguita in s1 rende P falsa (vera)*

- Confronto STRIPS e ADL (vedi tabella)

- Molte estensioni e dialetti di STRIPS/ADL hanno portato al linguaggio standard **PDDL** (planning domain definition language) attualmente in uso)

Esempi STRIPS

- Dominio logistico (figura 11.2 R&N II ed.)
- Dominio “ruota di scorta” (figura 11.3 R&N II ed.)
- Dominio “Blocks World”
  - Formalizzazione con 1 operatore
  - Formalizzazione con 2 operatori
  - Formalizzazione con 4 operatori
  - On-line demo (UBC website)

Esempio: dominio logistico

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

Esempio: ruota di scorta (STRIPS?)

$Init(Tire(Flat) \land Tire(Spare) \land At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(obj, loc),$
PRECOND: $At(obj, loc)$
EFFECT: $\neg At(obj, loc) \land At(obj, Ground))$
$Action(PutOn(t, Axle),$
PRECOND: $Tire(t) \land At(t, Ground) \land \neg At(Flat, Axle)$
EFFECT: $\neg At(t, Ground) \land At(t, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \land \neg At(Spare, Axle) \land \neg At(Spare, Trunk)$
$\land \neg At(Flat, Ground) \land \neg At(Flat, Axle) \land \neg At(Flat, Trunk))$

Figure 10.2 The simple spare tire problem.

Blocks World (1 operator)

- Action(Move(b,x,y)

  Precond: On(b,x) & Clear(b) & Clear(y),

  Effect: On(b,y) & Clear(x) &

  Not(On(b.x)) & Not(Clear(y)))

  Clear(b): introduced to represent

  "Not(Exists z, On(z,b))"

  NOTE: b, x, y are **parameters**, z is a **variable**

Blocks world (2 Operators)

$nit(On(A, Table) \wedge On(B, Table) \wedge On(C, A)$
$\wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$
PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge$
$(b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x),$
EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

igure 10.3 A planning problem in the blocks world: building a three-block tower. On
olution is the sequence $[MoveToTable(C, A), Move(B, Table, C), Move(A, Table, B)]$.

Blocks world con effetti condizionali

```
define (operator puton)
 :parameters (?X ?Y ?Z)
 :precondition (and (on ?X ?Z) (clear ?X) (clear ?Y)
                    (neq ?Y ?Z) (neq ?X ?Z)
                    (neq ?X ?Y) (neq ?X Table))
 :effect
 (and (on ?X ?Y) (not (on ?X ?Z))
      (when (neq ?Z Table) (clear ?Z))
      (when (neq ?Y Table) (not (clear ?Y)))))
```

Pianificazione nello spazio degli stati

- **Progressiva** (forward state-space search)
  - Descrizione stati *completa* (con assunzioni mondo chiuso e inerzia)
  - I nodi dell'albero di ricerca sono stati del mondo completi
  - Posso usare A* (ma fattore diramazione enorme!..)
  - Molte azioni *irrilevanti* (esempio dominio logistico)

    10 aereoporti, 5 aerei, 20 casse da spostare

    goal: spostare tutte le casse da JFK a SFO banale ma irrisolvibile da una ricerca "classica" per presenza di numerose azioni irrilevanti
- **Regressiva** (backward state-space search)
  - Più mirata della ricerca in avanti (*le azioni sono rilevanti*)
  - **Azione rilevante** per un insieme di goal: *se i suoi effetti soddisfano almeno un goal.* Es nel dominio logistico:

    UNLOAD(C1,AIRPLANE1,ROMA) rilevante per AT(C1,ROMA)

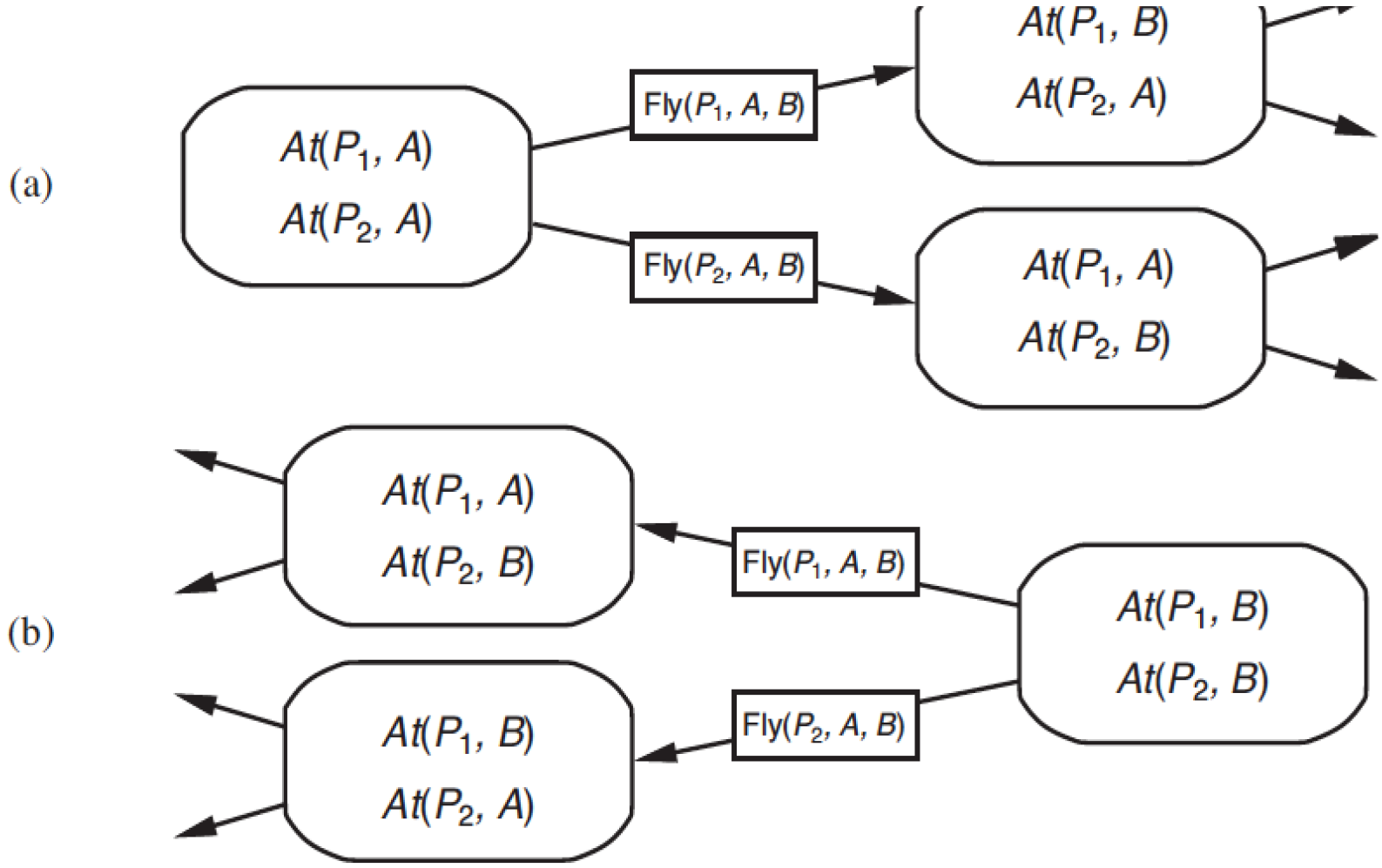

Figure 10.5 Two approaches to searching for a plan. (a) Forward (progression) search

Pianificazione nello spazio degli stati

- **Regressione:** Applicazione di un'azione rilevante e consistente "all'indietro"
  - Azione **consistente** se raggiunge un goal senza falsificarne un altro!

- Se G è una congiunzione di (sotto)goal (= nodo albero ricerca) la *regressione di G attraverso l'azione A* è:

  G – effettipositivi(A) + precondizioni(A)

  Esempio: regressione di At(C1,B) & At(C2,B) & At(C3,B) attraverso Unload(C1,P,B) = ??

- **Termine ricerca**: *il set dei goal correnti sono veri nello stato iniziale.*

Pianificazione nello spazio degli stati

- Sono necessarie delle buone **euristiche**! (domain-[in?]dependent dipende se il planner è "configurable")
  - Forward: costo per raggiungere uno stato che soddisfa i goal
  - Backward: costo per raggiungere un set di goal veri nello stato iniziale
- Se inventate una nuova euristica migliore di quelle attuali diventate famosi!
- Il costo può essere definito come *numero di azioni necessarie* (*minimo* se l'euristica è ammissibile)
- Due euristiche principali domain-independent:
  - *Euristica* ***problemi rilassati***
  - *Euristica indipendenza dei* ***goal congiunti***

Euristica problema rilassato

Togliamo tutte le precondizioni e troviamo un piano rilassato per raggiungere i goal dallo stato corrente

1. Numero azioni piano rilassato = numero goal da soddisfare?
2. Euristica ammissibile?

1. NO: due azioni potrebbero negare ciascuna un effetto utile dell'altra (ad es: se i goal sono due, oltre a queste azioni, ne serve un'altra).
2. NO: un'azione potrebbe soddisfare più di un goal

E se eliminassimo anche gli effetti negativi e contassimo il numero *minimo* di azioni necessarie per raggiungere tutti i goal?

SI! Avremmo una euristica ammissibile, ma calcolarla è un problema **NP-hard**! (riduzione di *minimal set-covering*)

Esempio Riduzione di Minimal Set Covering a Planning (MSC problema molto noto in informatica)

- Goal: A & B & C (= insieme richieste da *coprire*: A,B,C)
- Eff(Az1): A & P (Az1 produce A)
- Eff(Az2): B & C & Q (Az2 produce B e C)
- Eff(Az3): C & P & Q (Az3 produce C)
- Insieme minimo di azioni (produttori) per soddisfare i goal (= *coprire il set*): Az1 e Az2

Euristica Indipendenza dei goal congiunti

→ *Ogni singolo goal può essere soddisfatto ottimamente indipendentemente dagli altri* (poi le singole soluzioni possono essere unite).

- Tecnica **non ammissibile** (in generale, interazioni positive e negative dei sottopiani)
- Esempio "anomalia di Sussman"

  stato iniziale: on(b,table) on(c,a) on(a,table)

  Goal: on(a,b) on(b,c) on(c,table)

  Quante azioni sono necessarie con questa euristica?

  Quante azioni ha il piano ottimo?

Euristica "Empty Delete List" (no effetti negativi nelle azioni)

→ *Ignoriamo gli effetti negativi, ma manteniamo le precondizioni*

- Trovare soluzioni *rilassate ottime* è NP-hard…
- Ma se non sono necessari piani ottimi possiamo facilmente trovare soluzioni rilassate sub-ottime
- *In pratica funziona bene* per pianificatori sub-ottimi di tipo forward state-space come "FF" (Fast Forward), che vedremo…