Constraint Programming & Planning

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

Udine, Marzo 2011

Constraint Programming

CSP/COP

Definizioni

Spazio di ricerca

CSP o COP?

Complessità di CSP

Alcune Applicazioni

Planning

Action Description Languages

ASP and CLP encoding of $\mathcal{B}$

Officials

Sommario

Constraint Programming

CSP/COP
- Definizioni
- Spazio di ricerca
- CSP o COP?
- Complessità di CSP

Alcune Applicazioni

Planning
- Action Description Languages
- ASP and CLP encoding of $\mathcal{B}$

Officials

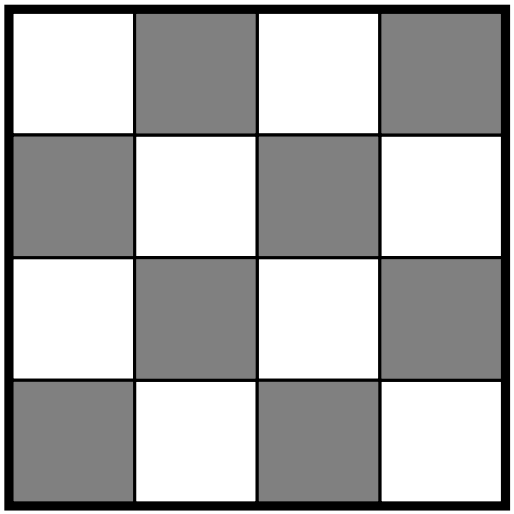
Introduzione

- Il *Constraint Programming* è una metodologia di programmazione dichiarativa, nata in ambito AI, adatta alla formalizzazione di problemi combinatorici ed alla loro risoluzione
- Le fasi di codifica (dichiarativa) e di ricerca sono nettamente separate
- I linguaggi permettono di scrivere codice molto chiaro e facilmente modificabile/estendibile
- Non ci sono restrizioni a priori sul *tipo* di vincoli
- La ricerca di soluzioni è naturalmente parallelizzabile
- È anche naturale inserire *euristiche* per guidare la ricerca

Introduzione

Il problema delle N regine

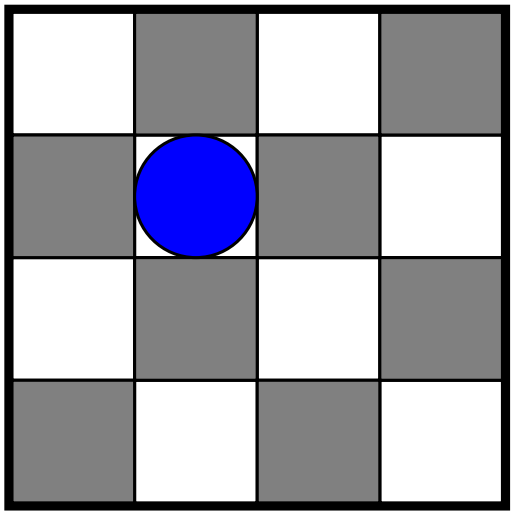
4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



Introduzione

Il problema delle N regine

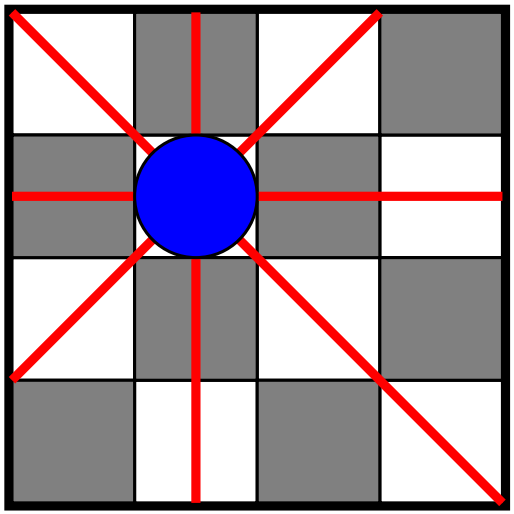
4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



Introduzione

Il problema delle N regine

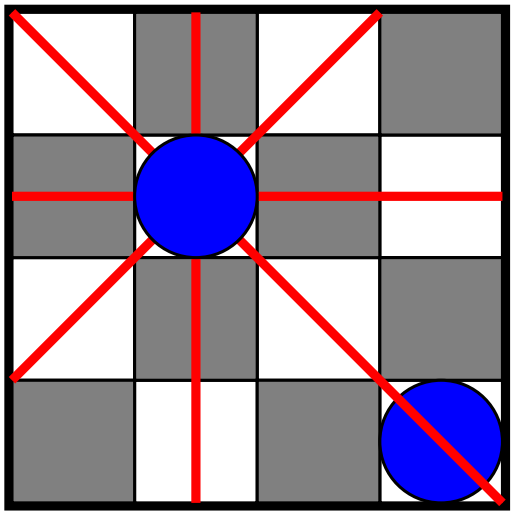
4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



Introduzione

Il problema delle N regine

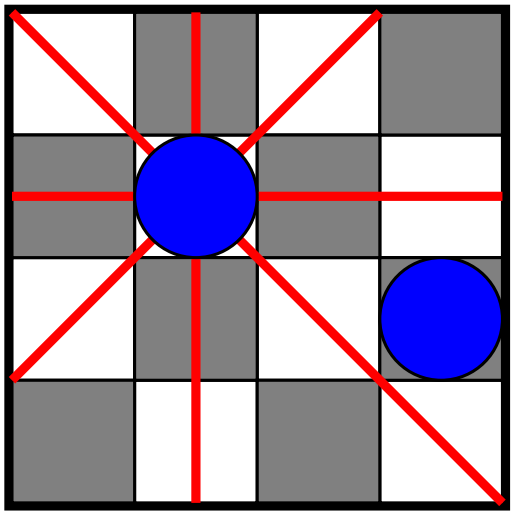
4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



Introduzione

Il problema delle N regine

4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



MODELLO DEL PROBLEMA

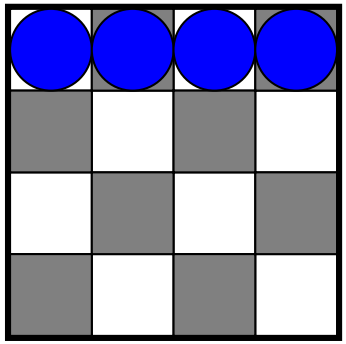
- Variabili
$$X_1, \ldots, X_4$$
$X_i = j$ significa "la regina sta in colonna i, riga j"
- Domini
$$D(X_i) = \{1, \ldots, 4\} \text{ per } i = 1..4$$
- Constraints (per $i, j = 1..4$ e $i < j$)
$$X_i \neq X_j \quad \text{(attacco orizzontale)}$$
$$|X_j - X_i| \neq j - i \quad \text{(attacco in diagonale)}$$

INTRODUZIONE

POSIZIONAMENTI

$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$

Posizionamento errato

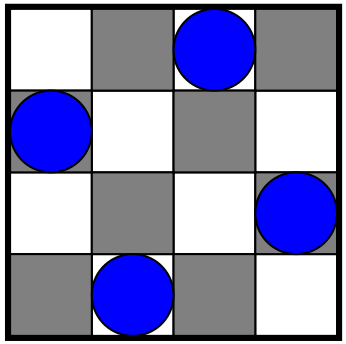


$$X_i \neq X_j \qquad \text{(attacco orizzontale)}$$

$$|X_j - X_i| \neq j - i \qquad \text{(attacco in diagonale)}$$

$$X_1 = 2, X_2 = 4, X_3 = 1, X_4 = 3$$

Posizionamento corretto



$$X_i \neq X_j \qquad \text{(attacco orizzontale)}$$
$$|X_j - X_i| \neq j - i \qquad \text{(attacco in diagonale)}$$

Constraint Satisfaction Problem

CSP (Constraint Satisfaction Problem)

- Insieme di variabili $\mathcal{V} = \{V_1, \ldots, V_n\}$
- Insieme di domini $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$
- Insieme di Vincoli (Constraints) C sulle variabili $\mathcal{V}$.
- Si vuole trovare una (o tutte) soluzione ammissibile, ovvero una funzione $\sigma : \mathcal{V} \longrightarrow \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n$ che rispetti i domini ed i vincoli imposti.

Nessun limite ai domini e al tipo di vincoli.

Constraint Optimization Problem

COP (Constraint Optimization Problem)

- Insieme di variabili $\mathcal{V} = \{V_1, \ldots, V_n\}$
- Insieme di domini $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$
- Insieme di Vincoli C su $\mathcal{V}$
- Una funzione $f : \mathcal{V} \longrightarrow \mathbb{A}$
- Si vuole trovare una (o tutte) soluzione ammissibile del CSP che minimizza (massimizza) la funzione f.

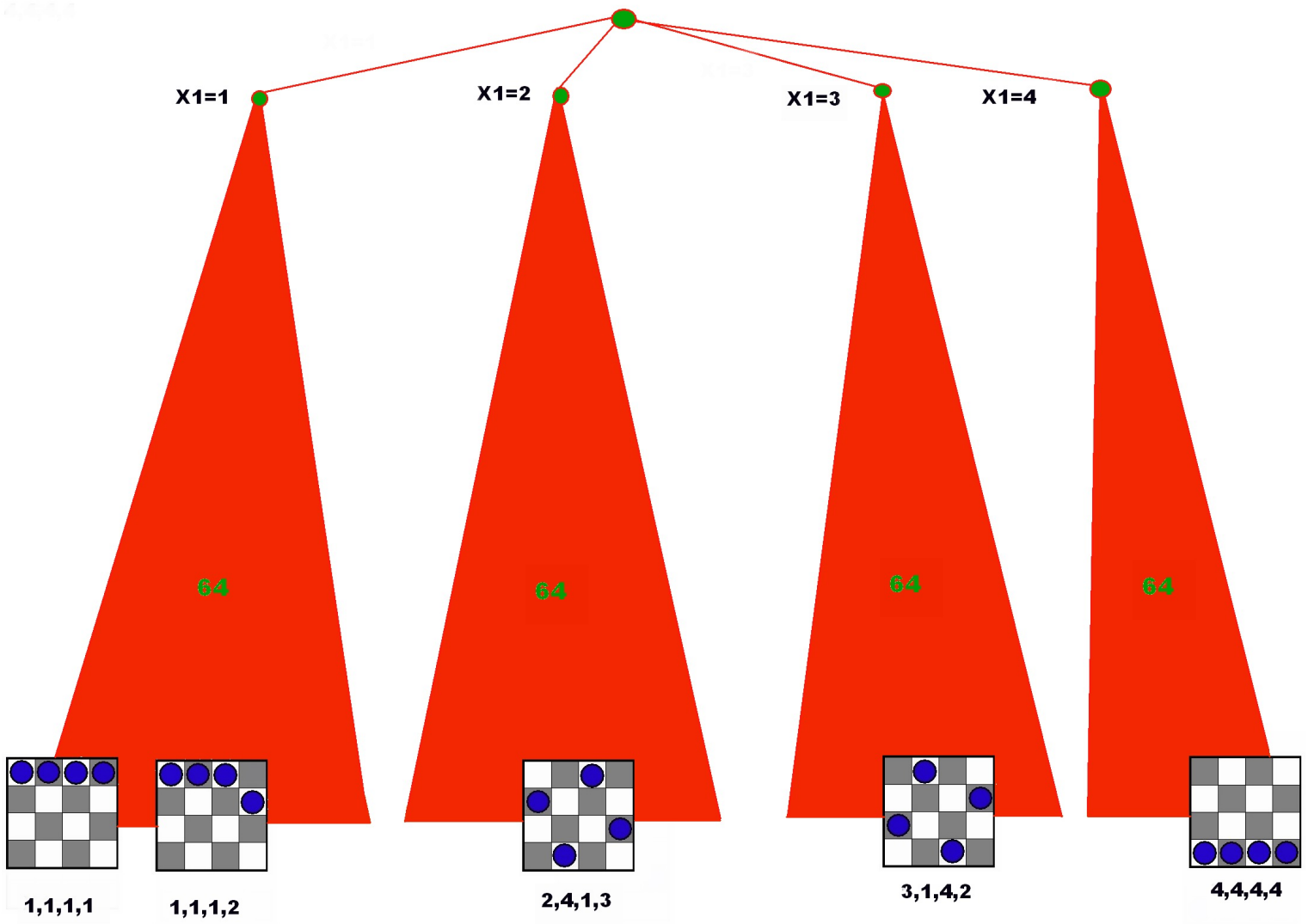
Nessun limite ai domini, al tipo di vincoli, e alla funzione.

Spazio di ricerca

- Dato un CSP $\langle X_1 \in \mathcal{D}_1, \ldots, X_n \in \mathcal{D}_n; C\rangle$ (con C denotiamo l'insieme di tutti i vincoli)
- lo *spazio di ricerca* (search space) è l'insieme delle n-uple
$$\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$$
(i punti $\langle d_1, \ldots, d_n\rangle$ che soddisfano i domini)
- Bisogna trovare tra questi i punti che verificano il constraint C
- Questo insieme viene comunemente rappresentato mediante un albero (search tree).
- I vari $\mathcal{D}_i$ potrebbero essere anche infiniti!

Spazio di ricerca

Esempio delle 4-regine

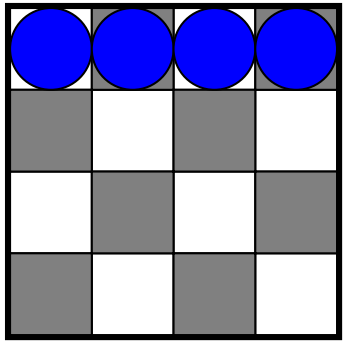


CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

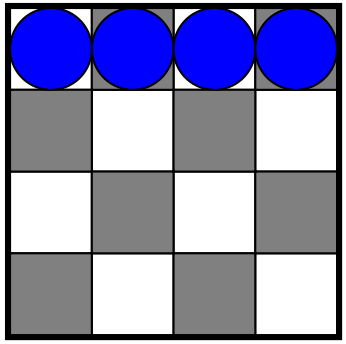
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$



inconsistente

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$



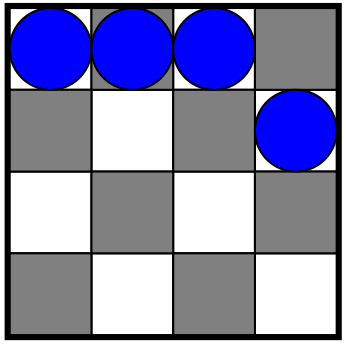
inconsistente

CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

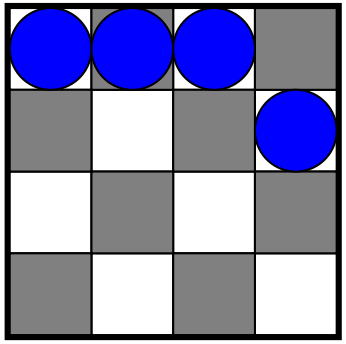
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$



inconsistente

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$



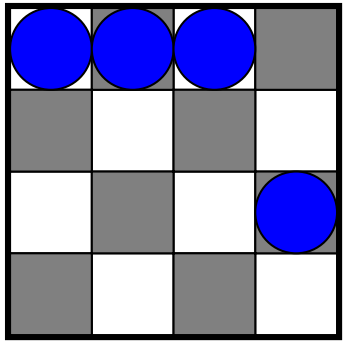
inconsistente

CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$



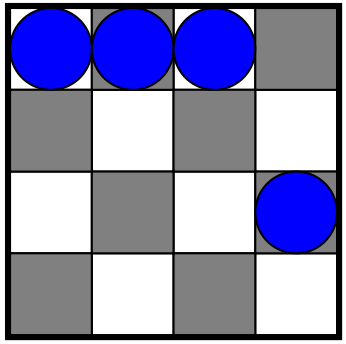
inconsistente

CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$



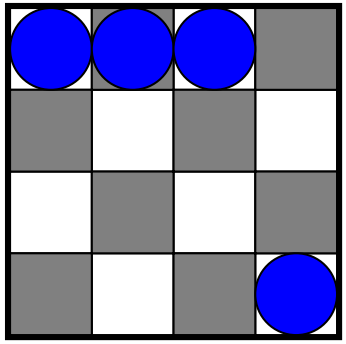
inconsistente

CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$



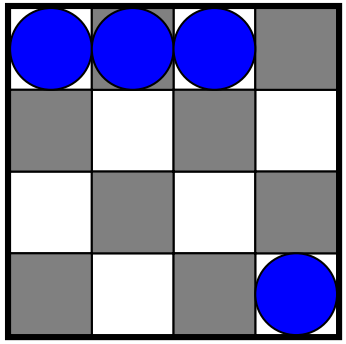
inconsistente

CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$



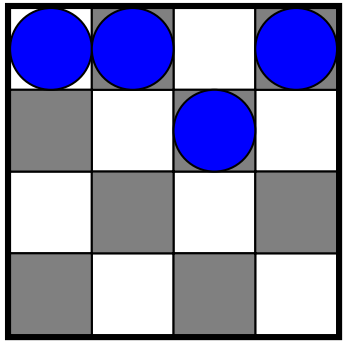
inconsistente

CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$



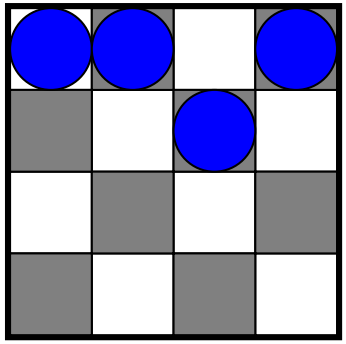
inconsistente

CSP/COP

Ricerca soluzioni Naive: Generate and Test

Tecnica non-deterministica in cui si generano una ad una le possibili soluzioni e poi si verificano.

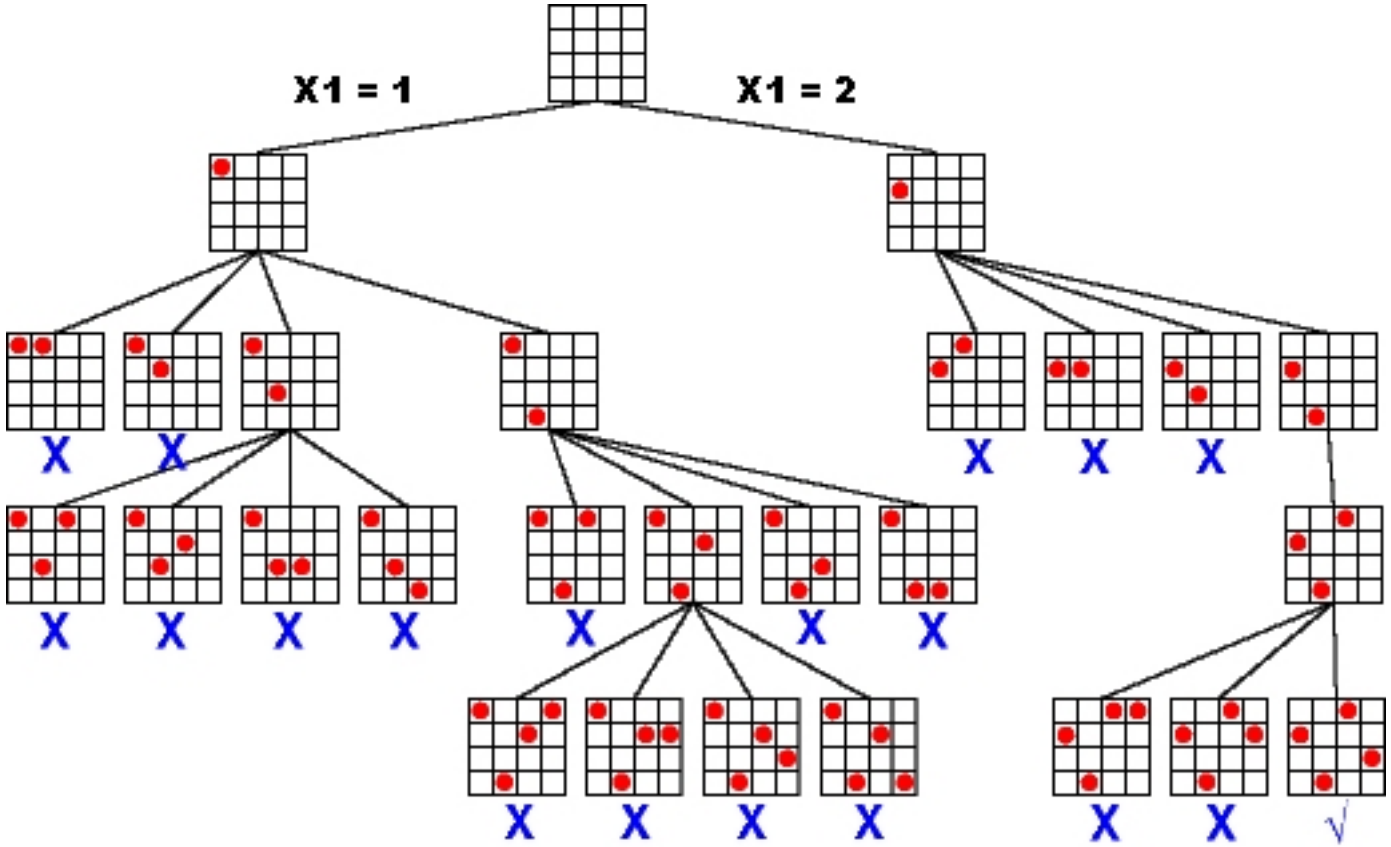
$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$



inconsistente ... Si visita tutto l'albero.
Rischio $4^4 = 256$ tentativi

CSP/COP

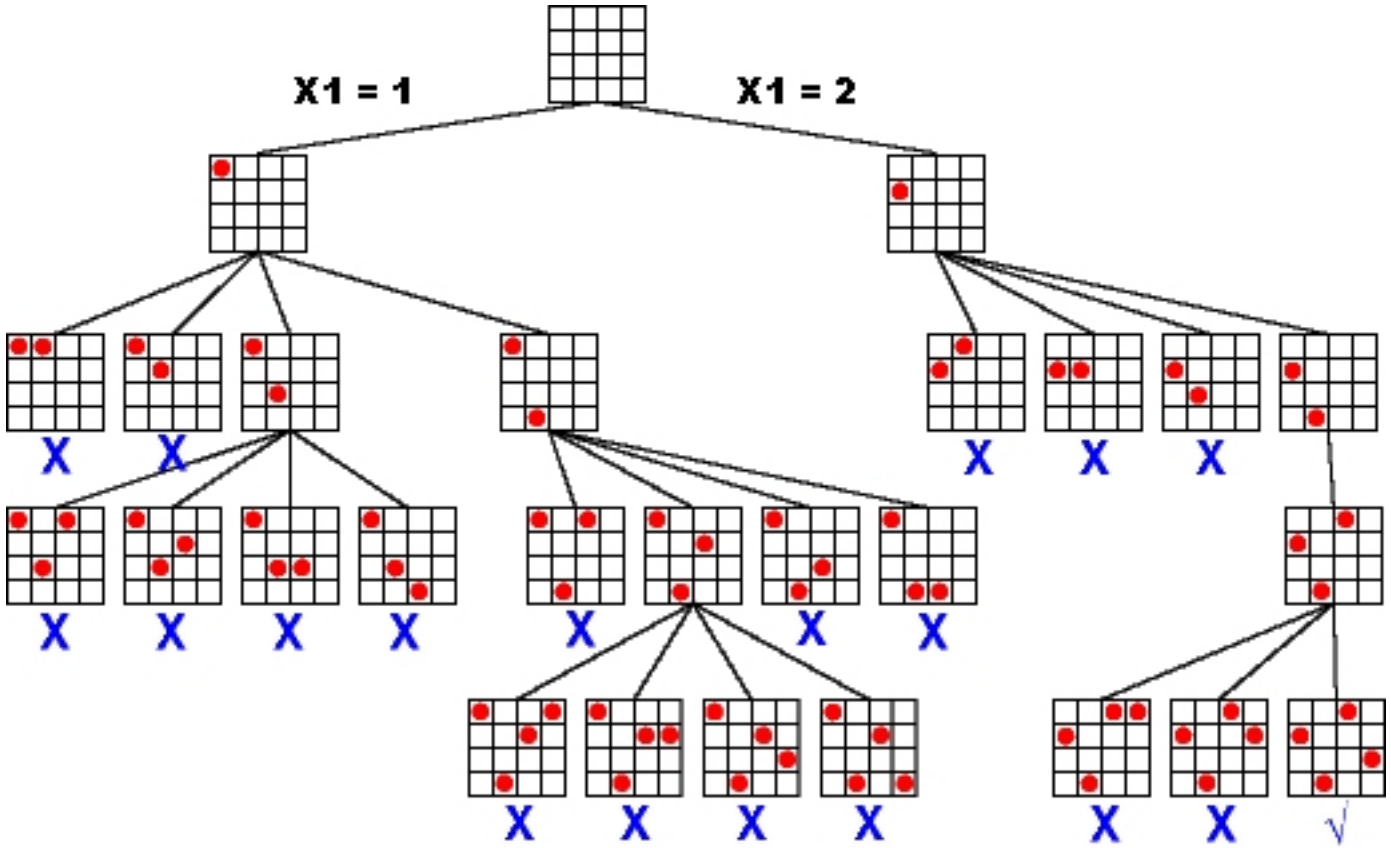
Ricerca soluzioni Naive: Assegnamenti parziali e backtracking



19 tentativi. Gli assegnamenti con $X_1 = 3$ e $X_1 = 4$ sono simmetrici a quelli già presenti nell'albero.

CSP/COP

Ricerca soluzioni Naive: Assegnamenti parziali e backtracking



19 tentativi. Gli assegnamenti con $X_1 = 3$ e $X_1 = 4$ sono simmetrici a quelli già presenti nell'albero.

CSP/COP

Principali tecniche usate per la ricerca delle soluzioni

- Local Search
- Programmazione Lineare (Intera)
- Traduzione a SAT
- Constraint Programming (useremo Constraint Logic Programming)
- Answer Set Programming

CSP o COP?

COP $\Rightarrow$ CSP

- Sia dato un CSP $P = \langle X_1 \in \mathcal{D}_1, \ldots, X_n \in \mathcal{D}_n; C \rangle$
- E una funzione f che vogliamo minimizzare — ovvero trovare $\langle d_1, \ldots, d_n \rangle$ soluzione del CSP tale che $f(d_1, \ldots, d_n) = \min\{f(\vec{x}) \ : \ \vec{x} \in Sol(P)\}$
- È evidente che la soluzione del COP implica la soluzione del CSP.

CSP o COP?

CSP $\Rightarrow$ COP

- Supponiamo ora di saper risolvere il CSP $P' = \langle X_1 \in \mathcal{D}_1, \ldots, X_n \in \mathcal{D}_n; C, f(X_1, \ldots, X_k) \leq k \rangle$
- Sia M un limite superiore (anche largo) della funzione. Dato un problema di solito è facile scoprire un valore di M.
- Per bisezione, con un numero di chiamate limitato da $\log_2 M$ riesco a risolvere il COP (se il minimo non è intero, dipendera' anche dall'approssimazione).
- Ad esempio, supponiamo il minimo sia 15 e $M = 205$. Proviamo P' con $k = 205, 102, 51, 25, 12$. Con 12 si ha il primo no. Dunque il minimo sta tra 13 e 25. Proviamo dunque $12 + 6$ (yes), $12 + 3$ (yes), $12 + 1$ (no). Rimane il dubbio se il minimo sia 14 o 15. Lanciamo P' con $k = 14$ e lo scopriamo.

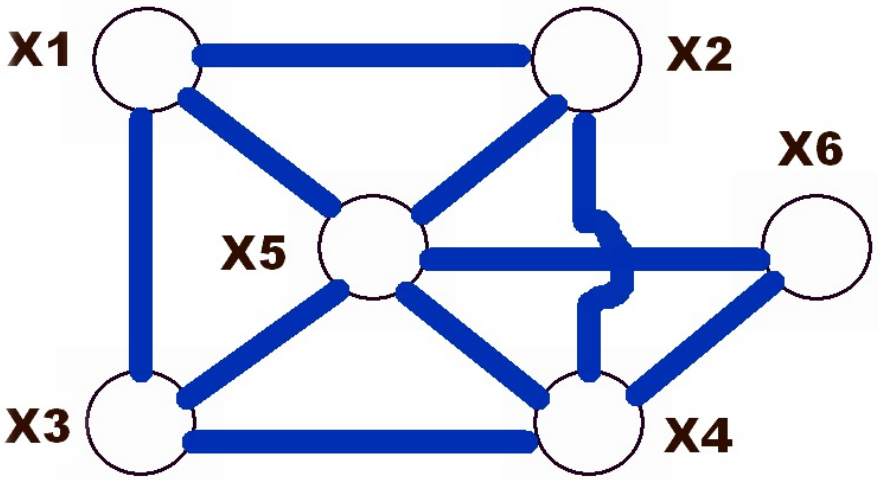
CSP o COP?

COP $\simeq$ CSP

- Risolvere CSP e COP non è dunque troppo diverso computazionalmente.
- COP implica CSP senza nessun aggravio, tuttavia
- trovare l'ottimo in problemi di grosse dimensioni implica la visita (anche se in modo intelligente) dell'intero spazio di ricerca.
- Spesso ci si accontenta di una soluzione buona del CSP decisionale associato.

NP-HARDNESS DI CSP

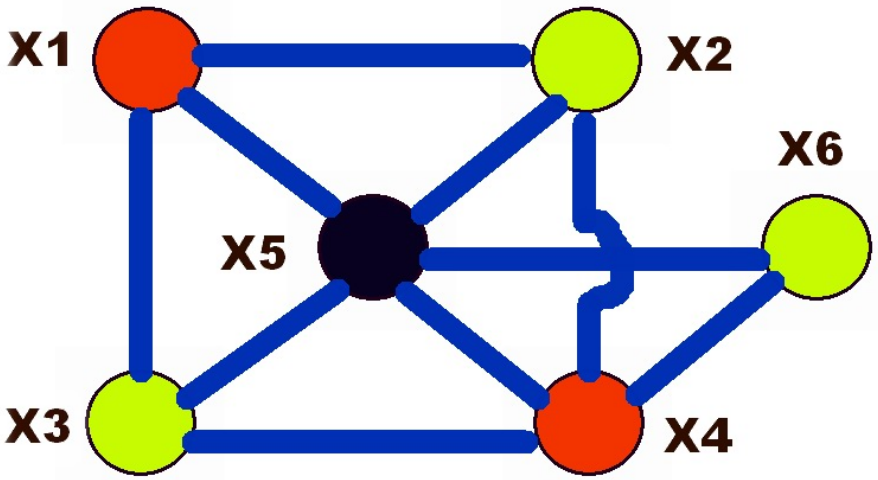
Problema del 3-coloring. Esiste una colorazione dei nodi del grafo usando 3 colori in modo tale che nodi adiacenti abbiano colore diverso?



$D_1 = \cdots = D_6 = \{\text{verde}, \text{rosso}, \text{nero}\}$,
$X_1 \neq X_2, X_1 \neq X_3, X_1 \neq X_5, \ldots, X_5 \neq X_6$

NP-HARDNESS DI CSP

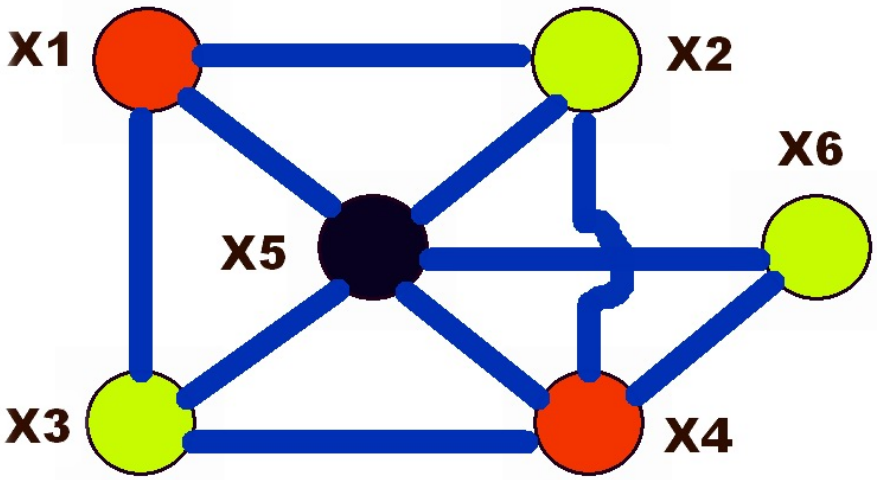
Problema del 3-coloring. Esiste una colorazione dei nodi del grafo usando 3 colori in modo tale che nodi adiacenti abbiano colore diverso?



$D_1 = \cdots = D_6 = \{\text{verde}, \text{rosso}, \text{nero}\}$,
$X_1 \neq X_2, X_1 \neq X_3, X_1 \neq X_5, \ldots, X_5 \neq X_6$

NP-HARDNESS DI CSP

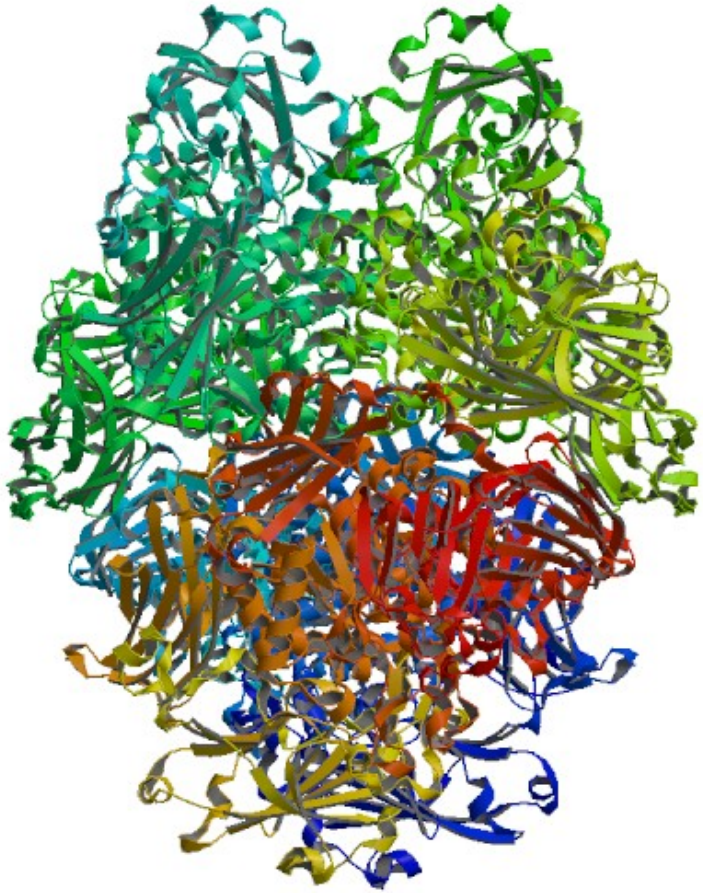
Problema del 3-coloring. Esiste una colorazione dei nodi del grafo usando 3 colori in modo tale che nodi adiacenti abbiano colore diverso?



$D_1 = \cdots = D_6 = \{\text{verde}, \text{rosso}, \text{nero}\}$,
$X_1 \neq X_2, X_1 \neq X_3, X_1 \neq X_5, \ldots, X_5 \neq X_6$

Alcune Applicazioni

Il problema del protein folding



Protein 1DP0, E. COLI BETA-GALACTOSIDASE

www.pdb.org

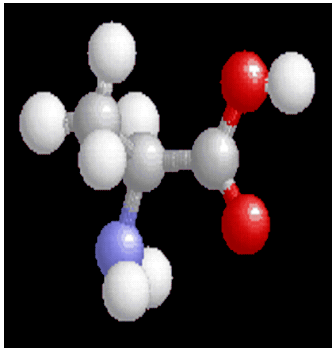
Il problema del protein folding

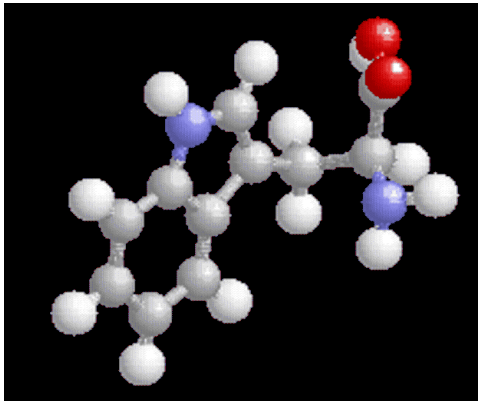
La struttura primaria

- La struttura primaria di una proteina è una sequenza di monomeri, detti aminoacidi, di lunghezza variabile da poche unità ad alcune centinaia.
- Gli aminoacidi sono di 20 tipi: **Ala**nine (A), **Cys**teine (C), **Asp**artic Acid (D), **Glu**tamic Acid (E), **Phe**nylalanine (F), **Gly**cine (G), **His**tidine (H), **I**so**le**ucine (I), **Lys**ine (K), **Leu**cine (L), **Met**hionine (M), **As**paragi**n**e (N), **Pro**line (P), **Gl**utami**n**e (Q), **Arg**inine (R), **Ser**ine (S), **Thr**eonine (T), **Val**ine (V), **Tryp**tophan (W), **Tyr**osine (Y).
- Per noi la struttura primaria di una proteina è una stringa di lettere nell'alfabeto $\{A, \ldots, Z\} \setminus \{B, J, O, U, X, Z\}$.

Il problema del protein folding

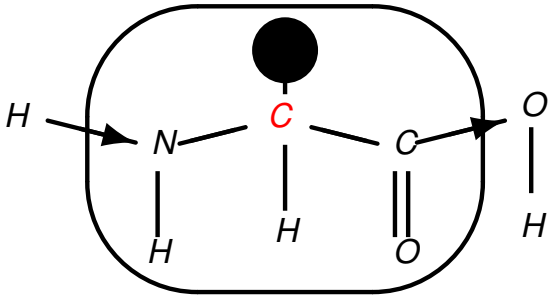
Due aminoacidi (Alanina, Triptofano)





$C_3H_7NO_2 \rightarrow 13$ atoms $\quad C_{11}H_{12}N_2O_2 \rightarrow 27$ atoms

White	=	H
Blue	=	N
Red	=	O
Grey	=	C



Il problema del protein folding

La struttura terziaria

- La struttura terziaria è la forma 3D che assume la proteina.
- La proteina si folda a condizioni di temperatura standard raggiungendo l'unico stato nativo.
- Tale stato è quello che minimizza l'energia libera della proteina (*Anfinsen*'s hypothesis)
- e determina la funzione della proteina.
- Il problema del protein structure prediction (folding) è il problema di predire la struttura terziaria (ed il processo per raggiungerla) data la struttura primaria.

Il problema del protein folding

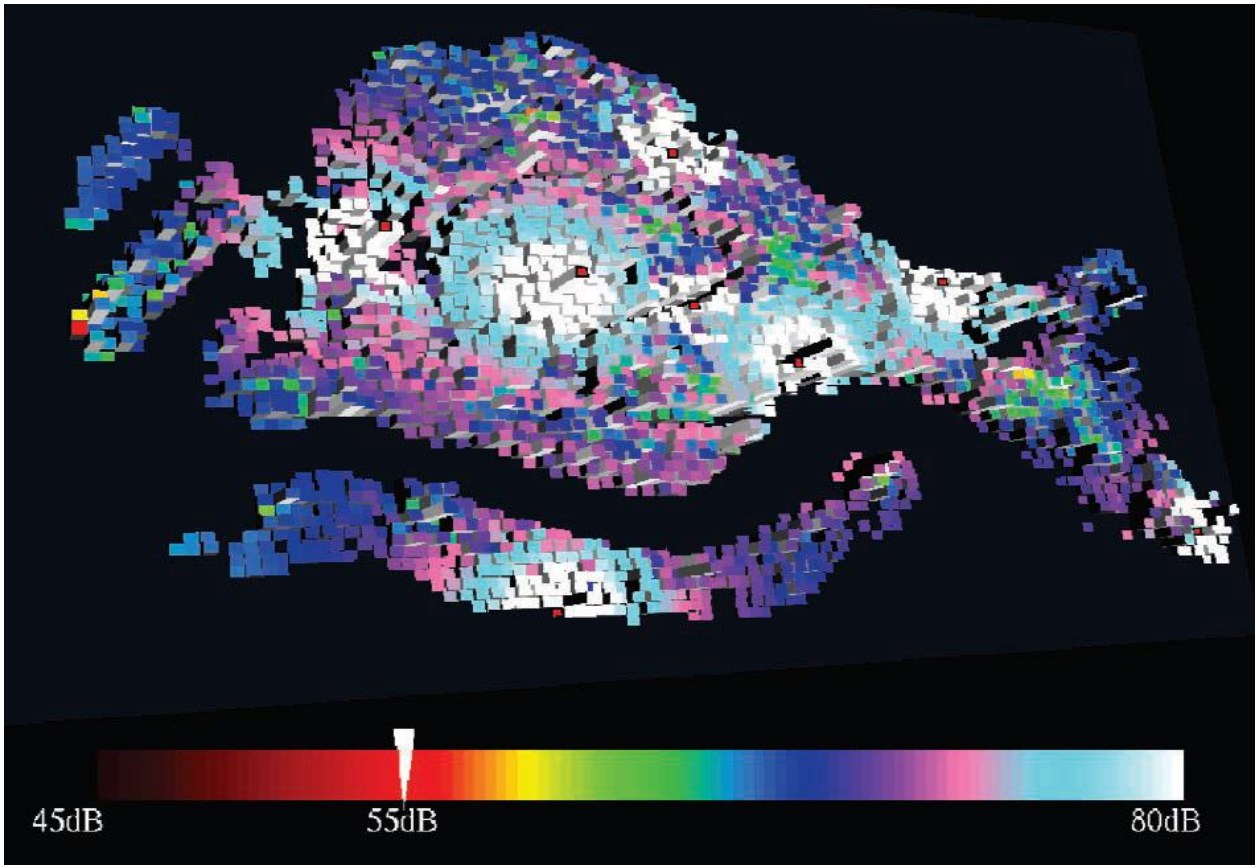
Alcuni risultati

Attualmente prediciamo ab-initio proteine di lunghezza

- circa 100 su reticolo FCC (spazio $\sim 6^{100}$)
- circa 100 anche fuori reticolo ma usando l'idea del fragment assembly.

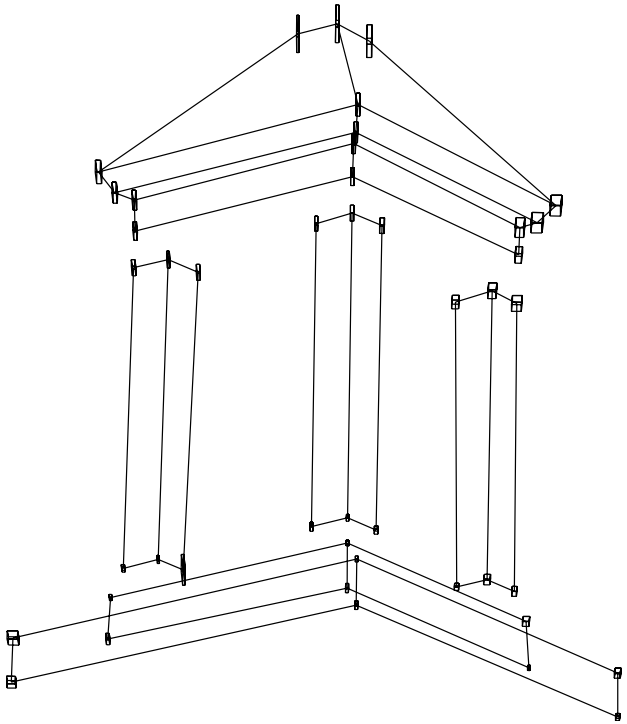
Alcune Applicazioni

Designing an Urban-Scale Auditory Alert System.
IEEE Computer 37(9):55–61 (2004).
(allocazione ottimale sirene acqua alta a Venezia)



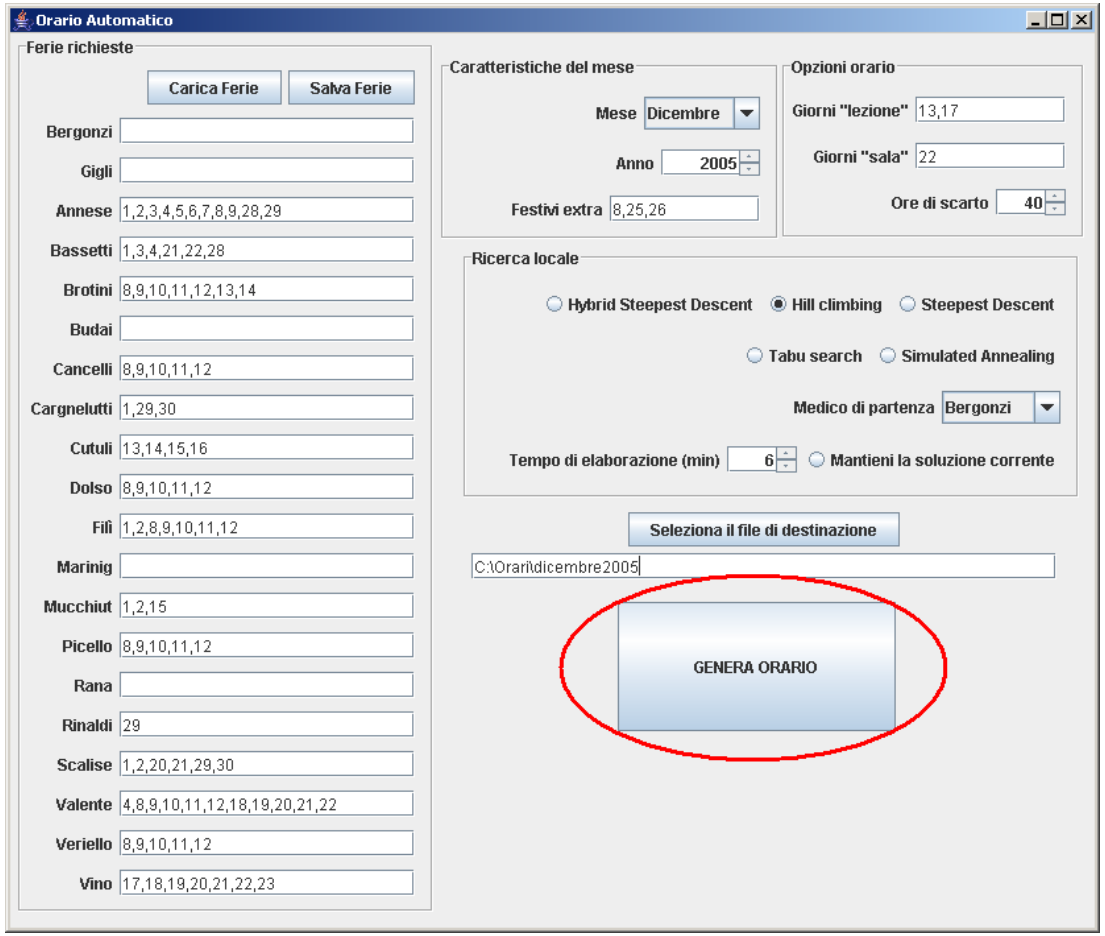
Alcune Applicazioni

Reconstruction with Interval Constraints Propagation. Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition. New York, Giugno 2006.



Alcune Applicazioni

Hybrid Approaches for Rostering: a Case Study in the Integration of Constraint Programming and Local Search. Int. Workshop on Hybrid Metaheuristics. Gran Canaria, 2006. (tool per i turni del reparto di neurologia, ospedale di Udine)



Un CSP divertente

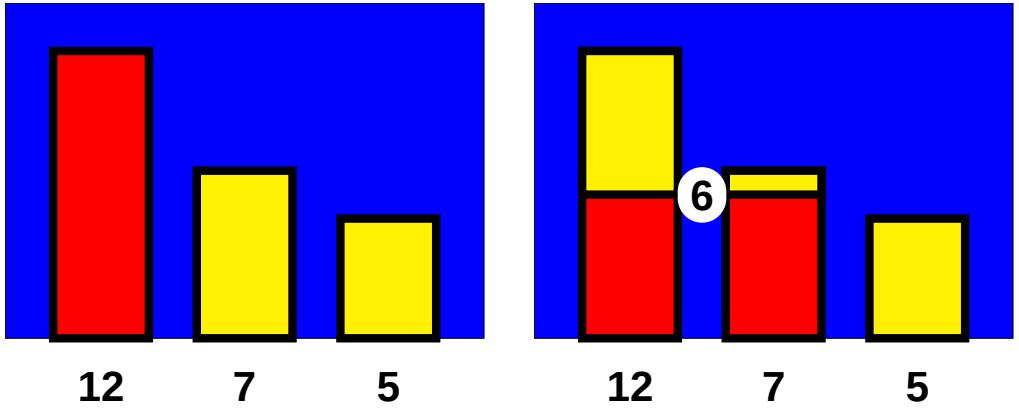
Sudoku

		1						
		2		3				4
			5			6		7
5			1	4				
	7						2	
				7	8			9
8		7			9			
4				6		3		
						5		

$\text{domain}([X_{1,1}, \ldots, X_{9,9}], 1, 9)$
$X_{1,3} = 1, X_{2,3} = 2, X_{2,5} = 3, X_{2,9} = 4, \ldots, X_{9,7} = 5$
$\text{alldifferent}(X_{1,1}, \ldots, X_{1,9}) \ldots \text{alldifferent}(X_{9,1}, \ldots, X_{9,9})$
$\text{alldifferent}(X_{1,1}, \ldots, X_{9,1}) \ldots \text{alldifferent}(X_{1,9}, \ldots, X_{9,9})$
$\text{alldifferent}(X_{1,1}, \ldots, X_{3,3}) \ldots \text{alldifferent}(X_{7,7}, \ldots, X_{9,9})$

Example: the Three-barrels problem

"There are three barrels of capacity N (even number), $N/2+1$, and $N/2-1$, resp. At the beginning the largest barrel is full of Taylor's Porto, the other two are empty. We wish to reach a state in which the two larger barrels contain the same amount of porto. The only permissible action is to pour porto from one barrel to another, until the latter is full or the former is empty."



Initial state Final state

Action description languages

Basics

Formal models to represent knowledge on actions and change (e.g., $\mathcal{A}$ and $\mathcal{B}$ [Gelfond and Lifschitz])
Specifications are given through declarative assertions that permit

- to describe actions and their effects on states
- to express queries on the underlying transition system

Action description languages

Planning

A *planning problem* can be described through an action description, which defines the notions of

Fluents i.e., *variables* describing the state of the world, and whose value can change

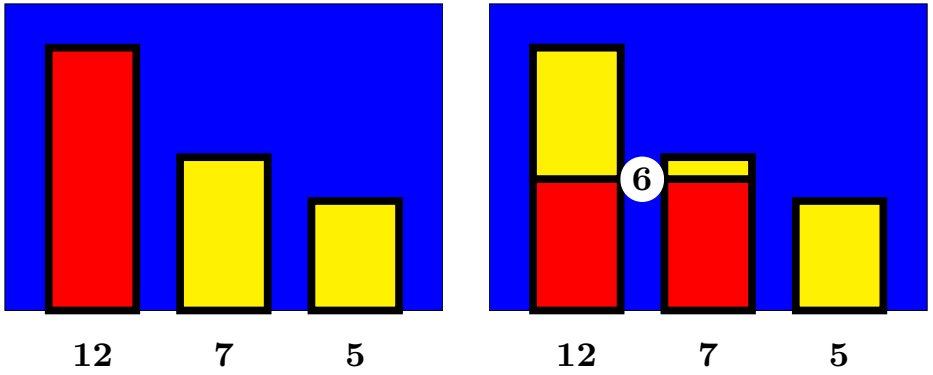
States i.e., possible configurations of the domain of interest: an assignment of values to the fluents

Actions that affect the state of the world, and thus cause the transition from a state to another

A complete (or partial) description of the initial and final states is given in input as a query.

Example: the Three-barrels problem

"There are three barrels of capacity N (even number), $N/2+1$, and $N/2-1$, resp. At the beginning the largest barrel is full of Taylor's Porto, the other two are empty. We wish to reach a state in which the two larger barrels contain the same amount of porto. The only permissible action is to pour porto from one barrel to another, until the latter is full or the former is empty."

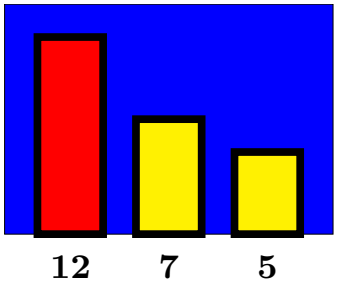


Initial state Final state

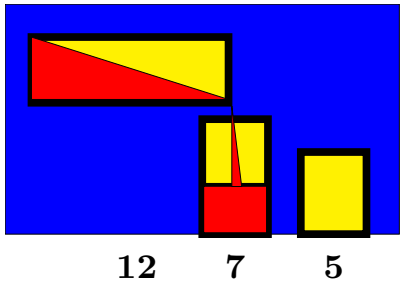
Example: the Three-barrels problem

fluents, states, and action

State 1

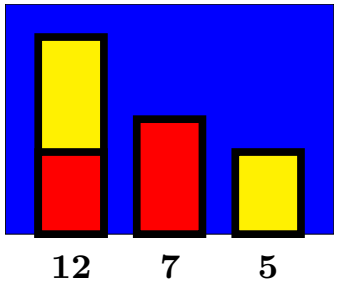


contains(12,12)
contains(7,0)
contains(5,0)
¬ contains(12,0)
...



pour(12,7)

State 2



contains(12,5)
contains(7,7)
contains(5,0)
¬ contains(12,0)
...

The language $\mathcal{B}$

Action Description

Let `a` be an action and `f` be a Boolean fluent. We have:

- Executability conditions:
  `executable(a, [list-of-preconditions])`
  asserting that the given preconditions have to be satisfied in the current state for the action `a` to be executable
- Dynamic causal laws:
  `causes(a, f, [list-of-preconditions])`
  describes the effect (the fluent literal `f`) of the execution of action `a` in a state satisfying the given preconditions
- Static causal laws:
  `caused([list-of-preconditions], f)`
  describes the fact that the fluent literal `f` is true in a state satisfying the given preconditions

The language $\mathcal{B}$

Barrels and Porto

```
executable(pour(X,Y),
      [contains(X,LX),contains(Y,LY)]) :-
   action(pour(X,Y)),
   fluent(contains(X,LX)),
   fluent(contains(Y,LY)),
   LX > 0, LY < Y.
caused([contains(X,LX) ],neg(contains(X,LY))):-
   fluent(contains(X,LX)),
   fluent(contains(X,LY)),
   barrel(X),liters(LX),liters(LY),
   neq(LX,LY).
```

The language $\mathcal{B}$

Barrels and Porto

```
executable(pour(X,Y),
      [contains(X,LX),contains(Y,LY)]) :-
   action(pour(X,Y)),
   fluent(contains(X,LX)),
   fluent(contains(Y,LY)),
   LX > 0, LY < Y.
caused([contains(X,LX) ],neg(contains(X,LY))):-
   fluent(contains(X,LX)),
   fluent(contains(X,LY)),
   barrel(X),liters(LX),liters(LY),
   neq(LX,LY).
```

The language $\mathcal{B}$

Goal

- *Initial state*

  `initially(f)`

  asserts that `f` holds in the initial state.

- *Goal*

  `goal(f)`

  asserts that `f` is required to hold in the final state.

Answer Set Programming

- Prolog has a unique minimal semantics if definite clauses are used
- This is no longer true in presence of negation
- Several minimal models can be defined
- For several reasons stable models are particularly interesting.
- Answer Set Programming is basically programming in Prolog, with negation, reasoning with the stable model semantics.

Modeling $\mathcal{B}$ in ASP and CLP(FD)

- We will show how encoding $\mathcal{B}$ in Answer Set Programming (traditional encoding)
- We will show how encoding $\mathcal{B}$ in CLP(FD)
- We will solve some problems using $\mathcal{B}$

Dettagli sul corso

- Testo: dispense Dovier-Formisano
- Linguaggi da usare: BProlog (o SICStus o GNU o ECLIPSE o SWI) Prolog per la parte di constraint programming
- Clingo (o lparse/smodels) nella parte di KR
- Guardate home page del corso per links
- Guardate anche: `clp.dimi.uniud.it` e `www.logicprogramming.org`
- Modalità d'esame: progetto e orale