Constraint Programming e sue applicazioni alla Predizione di Struttura

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

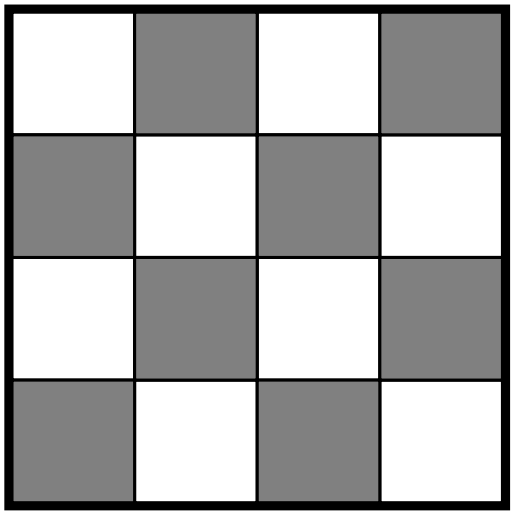
Udine, 6 Maggio 2008

Sommario

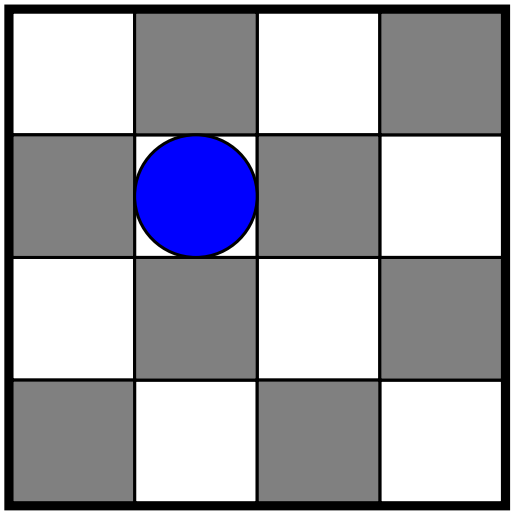
Introduzione

- Il *Constraint Programming* (CP) 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 di CP 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

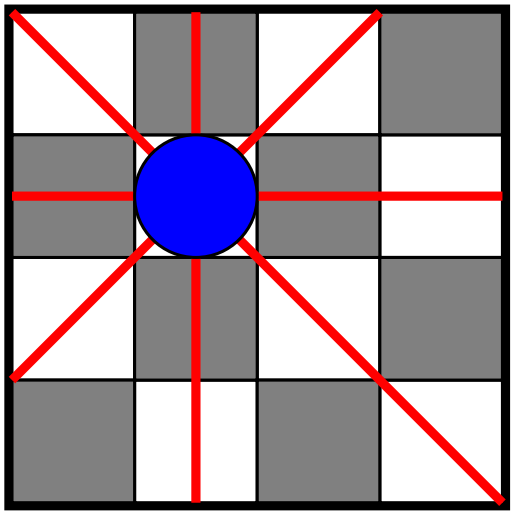
4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



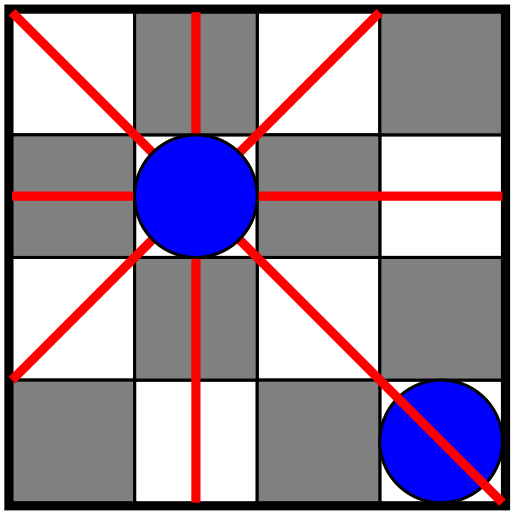
4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



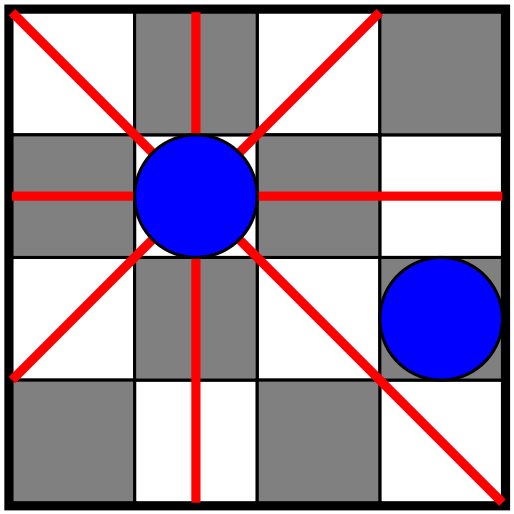
4-Regine: posizionare 4 regine su una scacchiera 4×4 in modo tale che non si attacchino vicendevolmente.



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



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

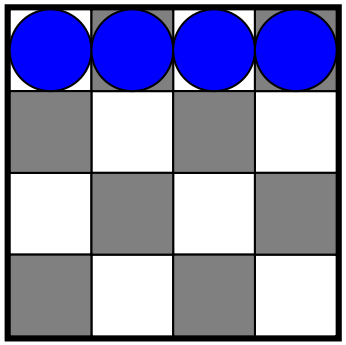


MODELLO DEL PROBLEMA

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

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

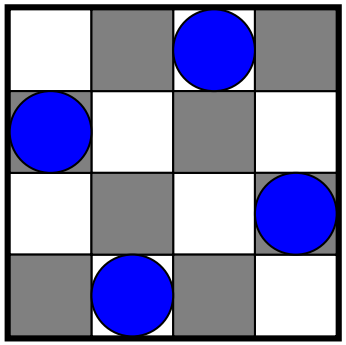
Posizionamento errato



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

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

Posizionamento corretto



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

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 Satisfaction Problem
Più precisamente

- Date le variabili $\mathcal{V} = \{V_1, \ldots, V_n\}$ e i domini $\{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$
- Un vincolo (constraint) C sulle variabili $V_{i_1}, \ldots, V_{i_m}$ è una relazione su $\mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_m}$, ovvero $C \subseteq \mathcal{D}_{i_1} \times \cdots \times \mathcal{D}_{i_m}$
- Una soluzione è una funzione $\sigma : \mathcal{V} \longrightarrow \mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n$ tale che
  - per ogni i, $\sigma(V_i) \in \mathcal{D}_i$ e
  - per ogni vincolo C sulle variabili $V_{i_1}, \ldots, V_{i_m}$, si ha che $\langle \sigma(V_{i_1}), \ldots, \sigma(V_{i_m}) \rangle \in C$

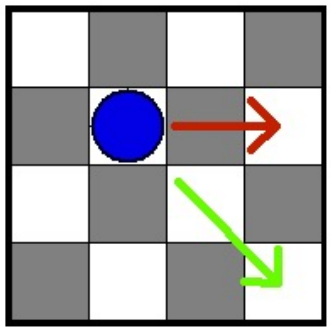
La definizione relazionale è matematicamente rigorosa ed indipendente dalla sintassi.
Ovviamente, quando il contesto è chiaro, scriveremo (ad esempio) $X \neq Y$ anziché l'insieme delle tuple.

CSP/COP
Esempio

- $\mathcal{V} = \{X_1, X_2, X_3, X_4\}$
- $\mathcal{D}_1 = \mathcal{D}_2 = \mathcal{D}_3 = \mathcal{D}_4 = \{1, 2, 3, 4\}$
- Vincoli di attacco orizzontale: $X_i \neq X_j$ sta per:

  $\{(1,2),(1,3),(1,4), \quad (2,1),(2,3),(2,4),$
  $(3,1),(3,2),(3,4), \quad (4,1),(4,2),(4,3)\}$
- Vincoli di attacco diagonale: $|X_j - X_i| \neq j - i$. Ad esempio, per $i = 2, j = 4$ sta per

  $\{(1,1),(1,2),(1,4), \quad \underline{(2,1),(2,2),(2,3),}$
  $(3,2),(3,3),(3,4), \quad (4,1),(4,3),(4,4)\}$



$(2,2)$ e $(2,4)$ non sono ammessi

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.

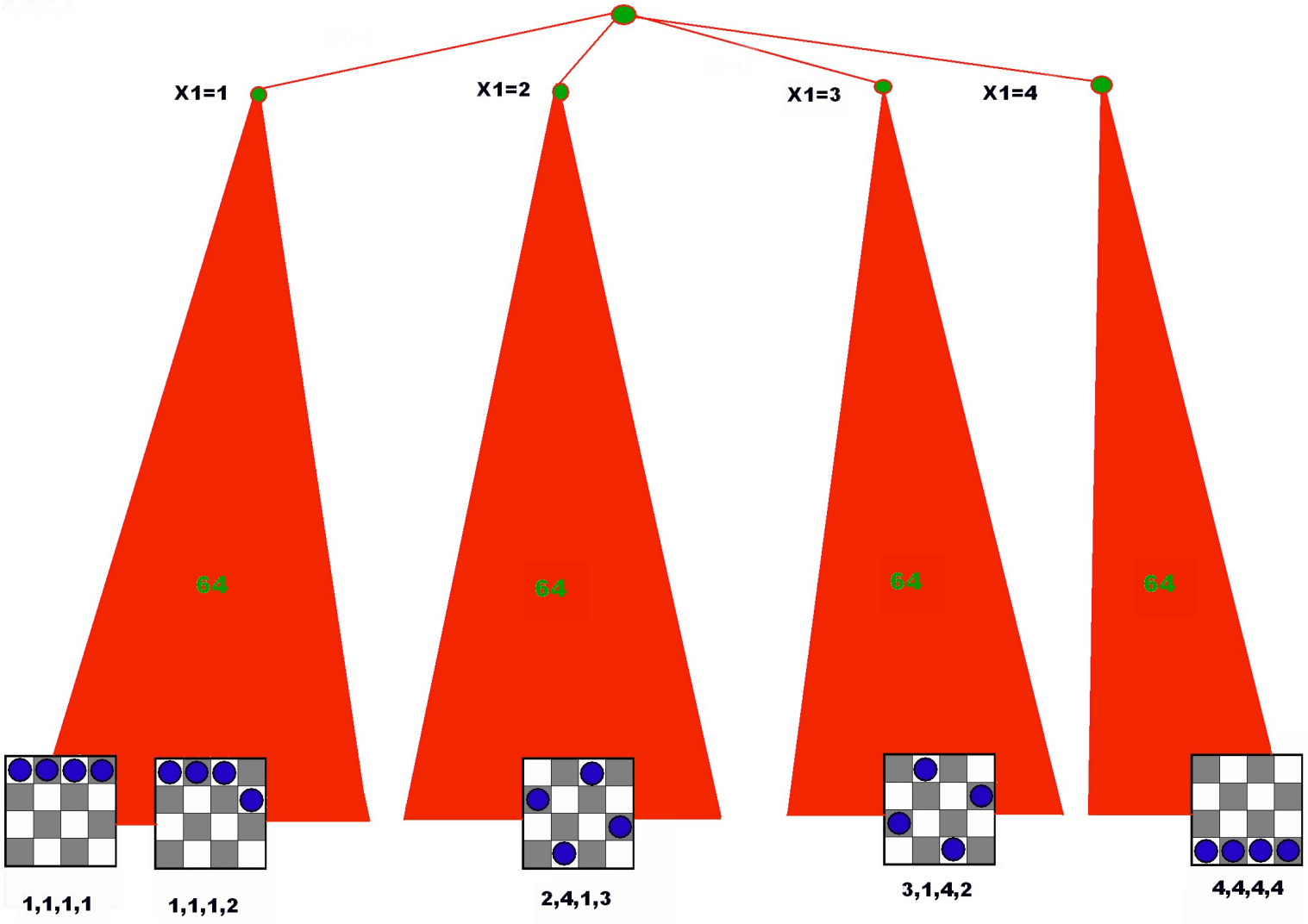
Spazio di ricerca

- 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

$$\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$$

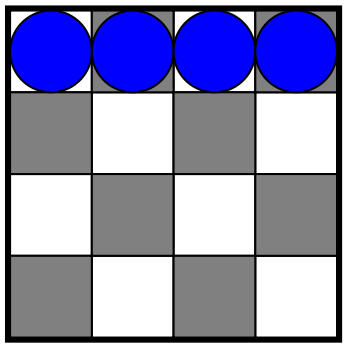
  (i punti $\langle d_1, \ldots, d_n \rangle$ che soddisfano i domini)
- Bisogna trovare tra questi i punti che verificano il constraint C
- Questo insieme viene comunemente rappresentato mediante un albero (search tree).
- I vari $\mathcal{D}_i$ potrebbero essere anche infiniti!

Spazio di ricerca
Esempio delle 4-regine



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

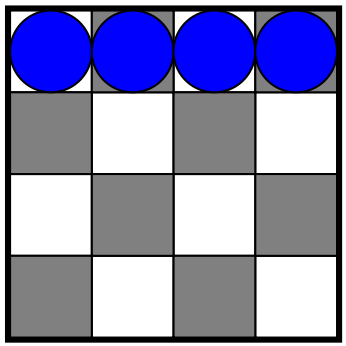
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$



inconsistente

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

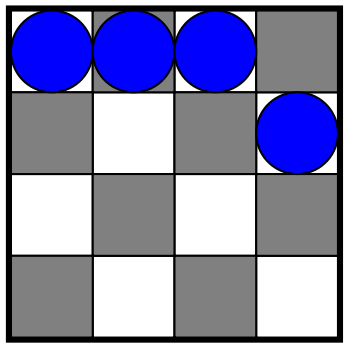
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$



inconsistente

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

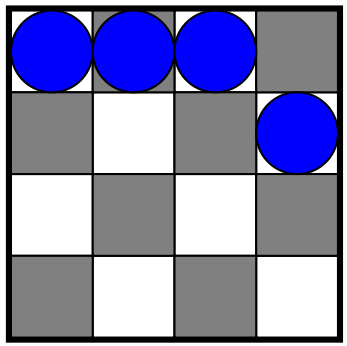
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$



inconsistente

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

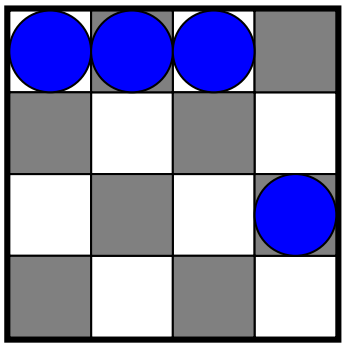
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$



inconsistente

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

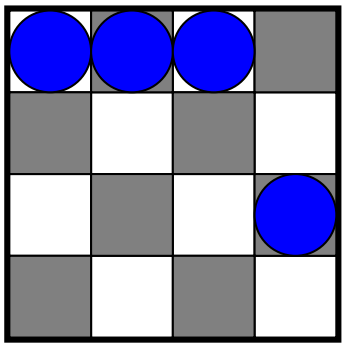
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$



inconsistente

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

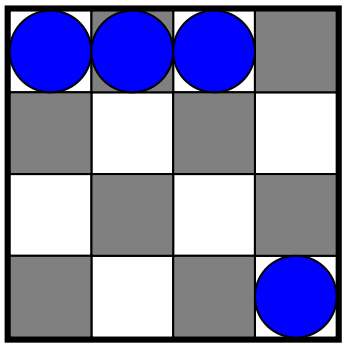
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$



inconsistente

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

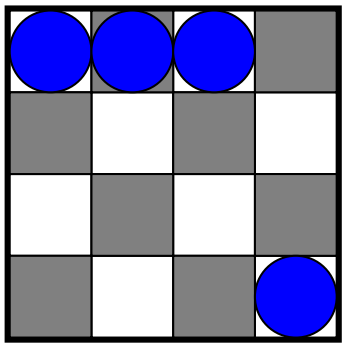
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$



inconsistente

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

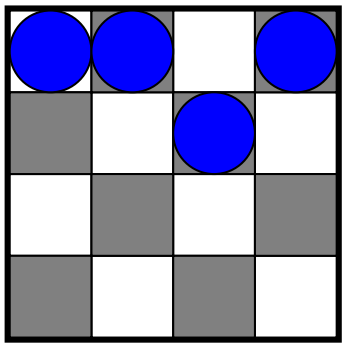
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$



inconsistente

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

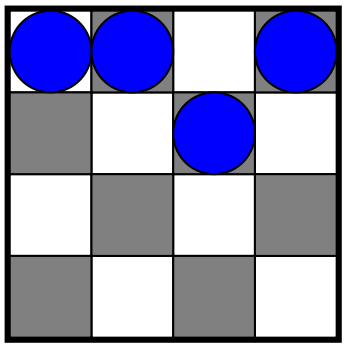
$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$



inconsistente

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

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

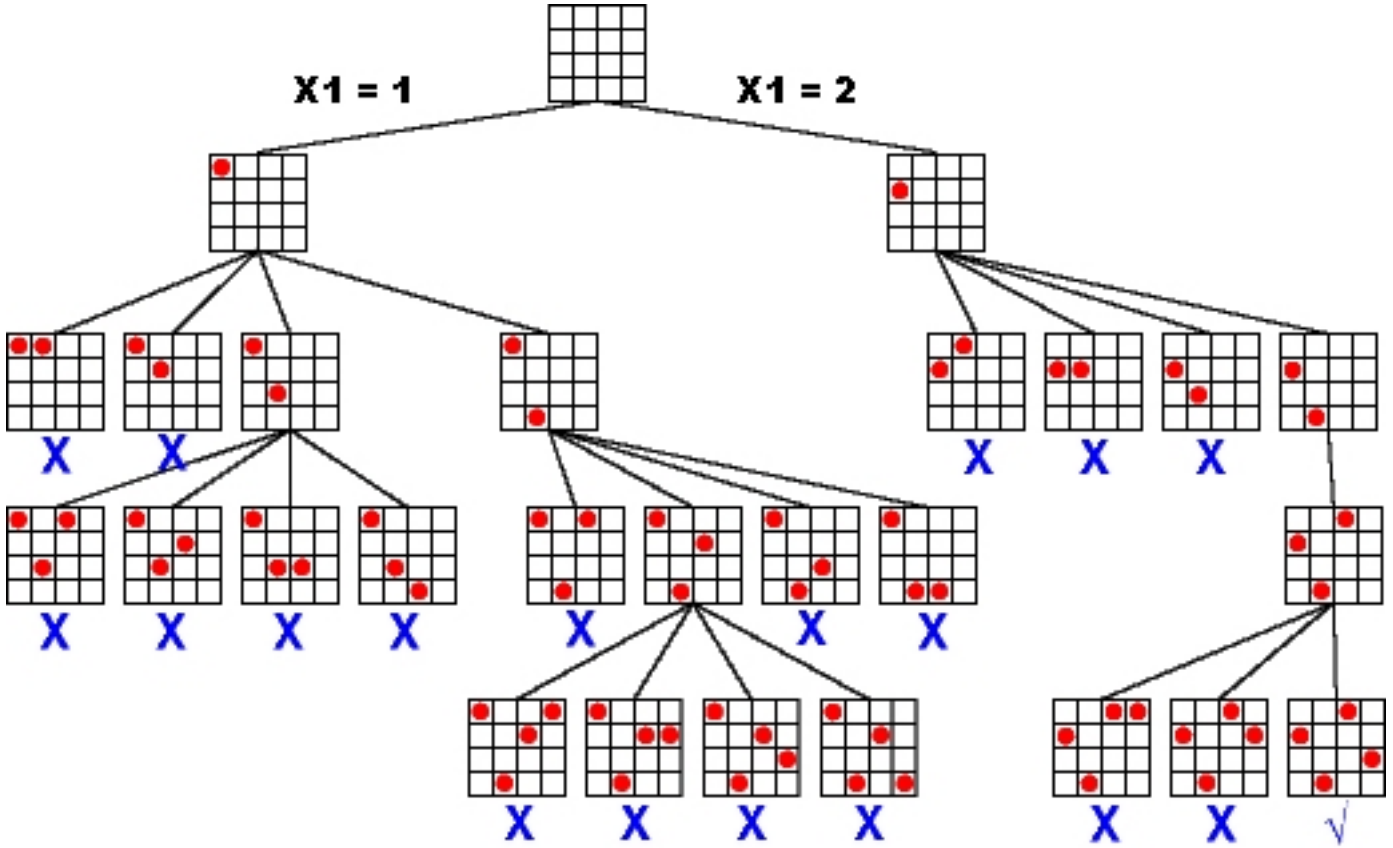


inconsistente . . . Si visita tutto l'albero.

Rischio $4^4 = 256$ tentativi

CSP/COP

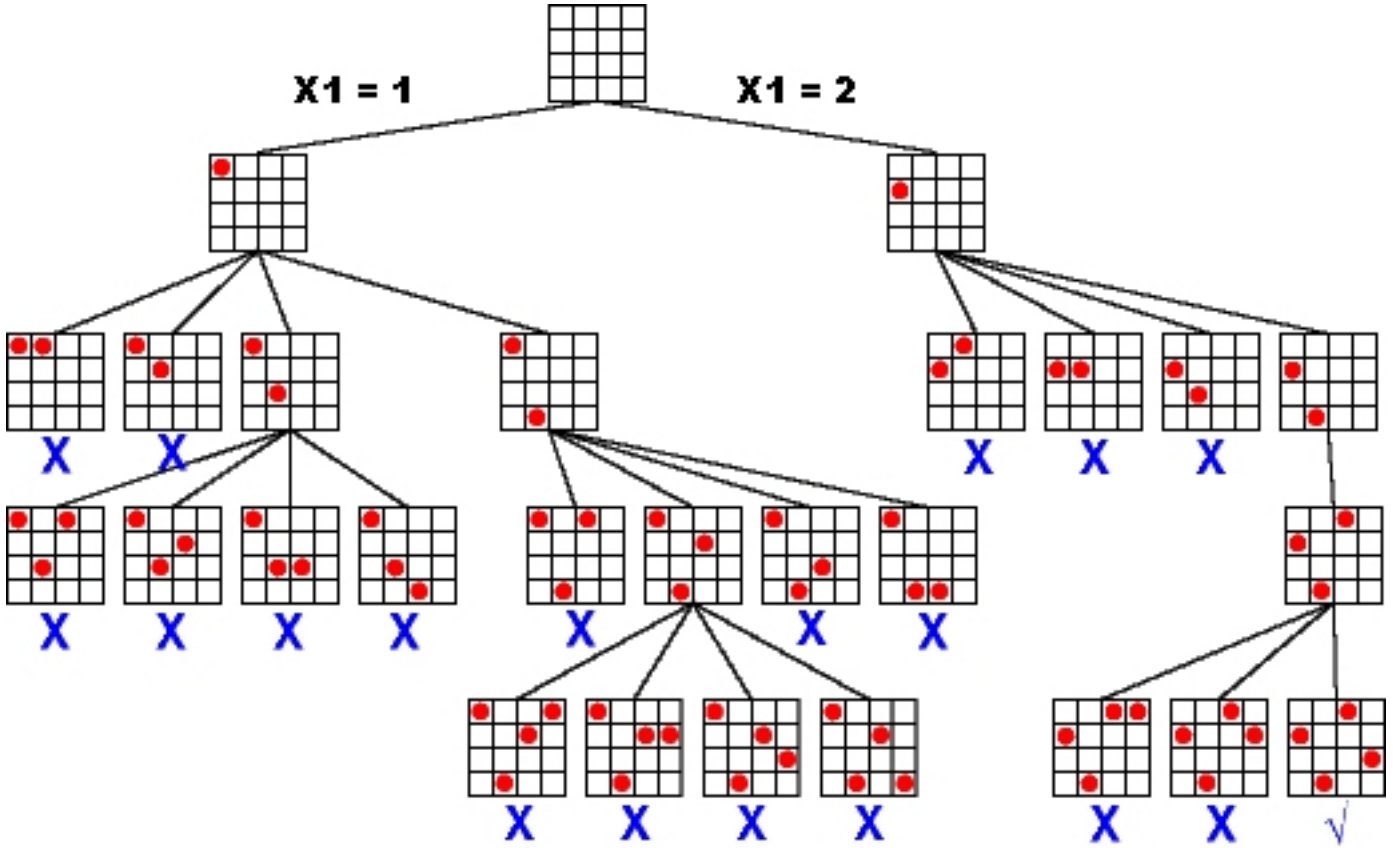
Ricerca soluzioni Naive: Assegnamenti parziali e backtracking



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

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: $\langle X_1 \in \mathcal{D}_1, \ldots, X_n \in \mathcal{D}_n; C, f(X_1, \ldots, X_n) \leq k \rangle$

CSP o COP?
CSP $\Rightarrow$ COP

- Supponiamo ora di saper risolvere il CSP $P' = \langle X_1 \in \mathcal{D}_1, \ldots, X_n \in \mathcal{D}_n; C, f(X_1, \ldots, X_n) \leq k \rangle$
- 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.

Richiami di complessità computazionale
Notazione O

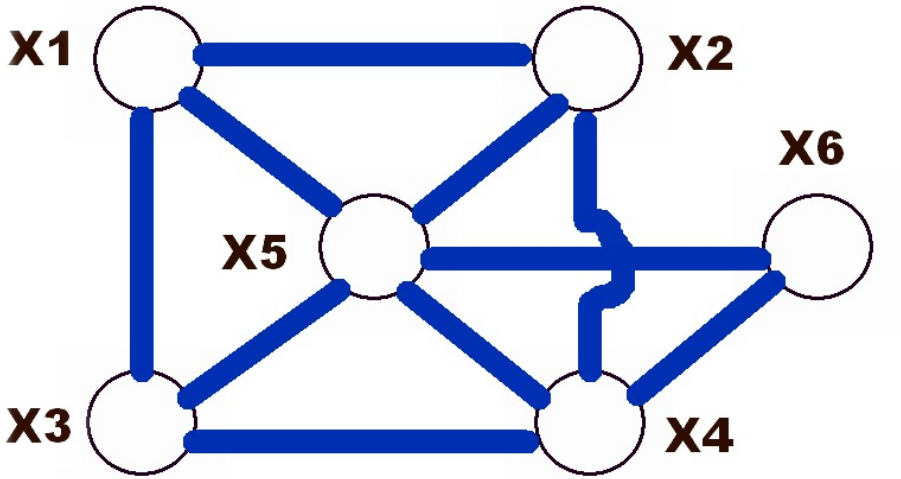
- I problemi possono essere classificati in base alla loro Complessità Computazionale, ovvero al numero di passi *necessario* ad un algoritmo per risolvere una dato problema.
- Di solito (abbiamo visto che non è una vera restrizione) si studiano i problemi decisionali
- Se si sviluppa un algoritmo che risolve un problema con un numero di passi dell'ordine di $f(n)$ dove n è una misura dell'input (p.es. $n =$ *lunghezza della struttura primaria della proteina*) allora il problema ha una complessità $O(f(n))$.
- Ma ciò non ci assicura che non esista un algoritmo migliore.
- È molto difficile dimostrare che non si può fare meglio di un dato algoritmo per risolvere un certo problema.

- Se esiste un algoritmo che risolve il problema con un numero di passi che è limitato da un polinomio in n (e.g., n, n^2, n^3, n^{10}) allora il problema appartiene a P (e.g., ordinare un vettore di n elementi può essere fatto in tempo $O(n \log n)$, ove $n \log n \leq n^2$)
- Problemi in P sono problemi *trattabili*.
  - Se n è la lunghezza della struttura primaria di una proteina, algoritmi n, n^2, n^3, n^4 sarebbero estremamante efficienti.
  - Tuttavia, per stringhe di DNA lunghe 10^9 basi, già algoritmi n^2 sarebbero inutili.
- EXP è la classe dei problemi chsi risolvono con algoritmi con numero di passi esponenziale $(2^n, 3^n, \ldots)$. Si sa che $P \subset EXP$.

- Problemi che stanno strettamente in EXP diventano non risolubili anche per n molto piccoli (20–30).
- Tra P ed EXP vi è la classe NP (non-deterministicamente polinomiali).
- Un problema sta in NP se si può verificare una sua soluzione in tempo polinomiale.
- P.es., il problema: esiste una conformazione di una proteina con energia libera $< k$?
- Data una conformazione di una proteina, verificare se la sua energia libera è $< k$? è facile

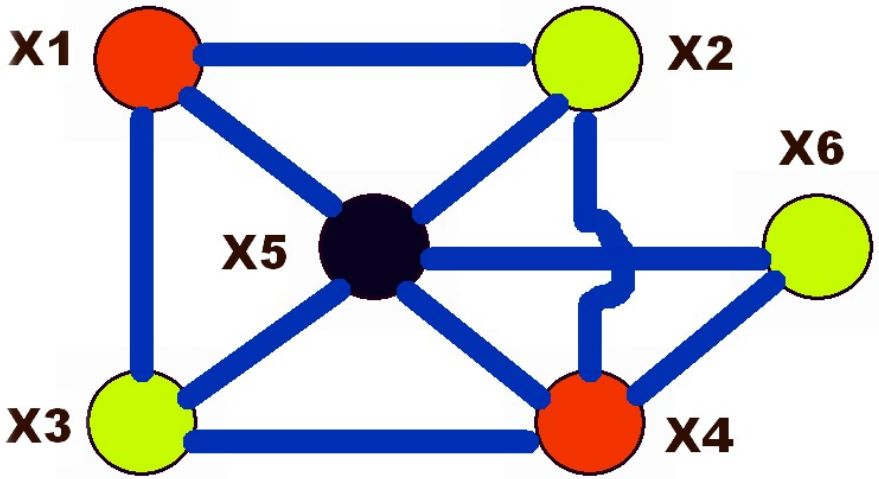
- Si sa che $P \subseteq NP \subseteq EXP$, si sa che $P \subset EXP$, ma non si sa nulla delle altre due inclusioni.
- In particolare non si sa se $P \subset NP$, però vi sono dei problemi in NP (NP-completi) per cui non si è mai trovato un algoritmo polinomiale (dai tempi di Gödel e Von Neumann).
- Un problema è NP-completo se sta in NP e la sua risoluzione in tempo polinomiale implicherebbe la risoluzione in tempo polinomiale di *OGNI* problema in NP.
- I problemi NP completi sono i più interessanti per un informatico: se si trovasse un algoritmo polinomiale per uno di loro si sarebbe a posto. Più umilmente ci si aspetta di fare il meglio possibile con diverse tecniche ed euristiche.

Problema (NP completo) del 3-coloring. Esiste una colorazione dei nodi di un grafo usando 3 colori in modo tale che nodi adiacenti abbiano colore diverso?



3 coloring si esprime (facilmente) come CSP:
$D_1 = \cdots = D_6 = \{\text{verde}, \text{rosso}, \text{nero}\}$,
$X_1 \neq X_2, X_1 \neq X_3, X_1 \neq X_5, \ldots, X_5 \neq X_6$

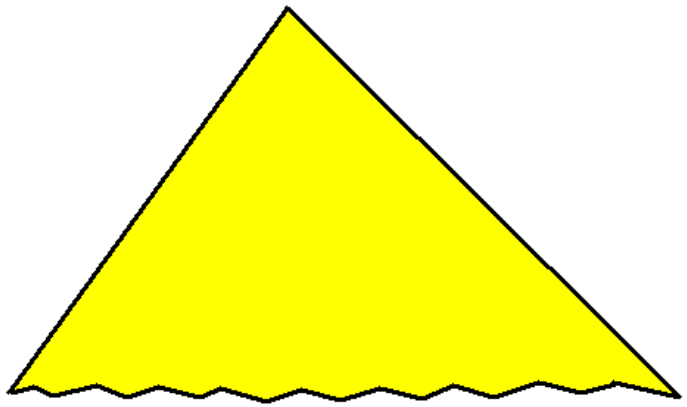
Problema (NP completo) del 3-coloring. Esiste una colorazione dei nodi di un grafo usando 3 colori in modo tale che nodi adiacenti abbiano colore diverso?



3 coloring si esprime (facilmente) come CSP:
$D_1 = \cdots = D_6 = \{\text{verde}, \text{rosso}, \text{nero}\}$,
$X_1 \neq X_2, X_1 \neq X_3, X_1 \neq X_5, \ldots, X_5 \neq X_6$

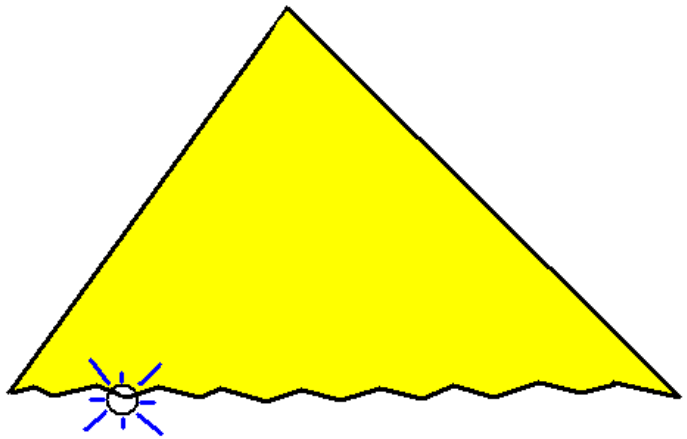
Local Search
Idea Generale per COP

Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



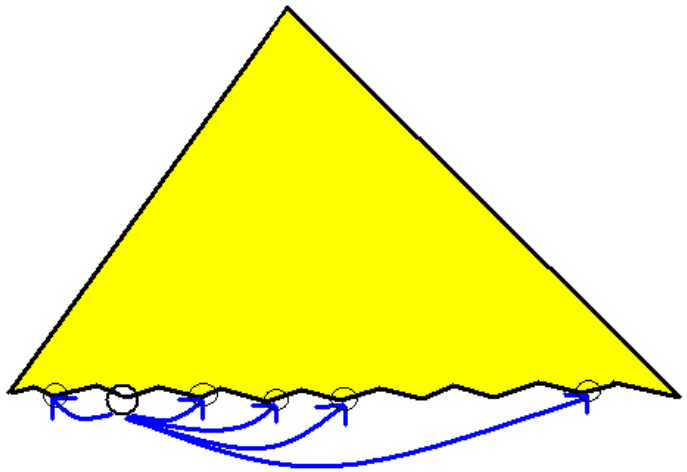
Local Search
Idea Generale per COP

Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



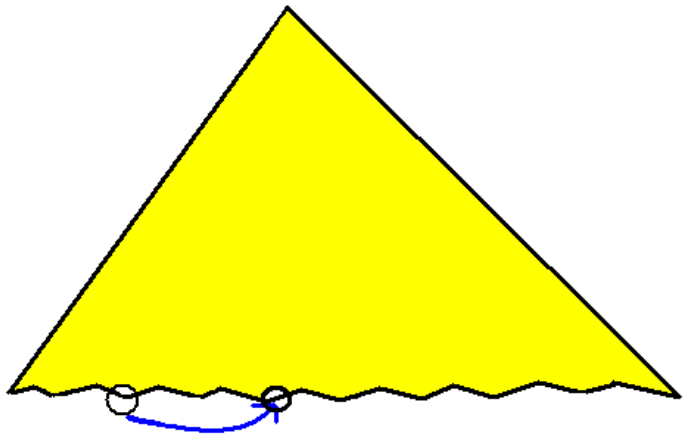
Local Search
Idea Generale per COP

Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



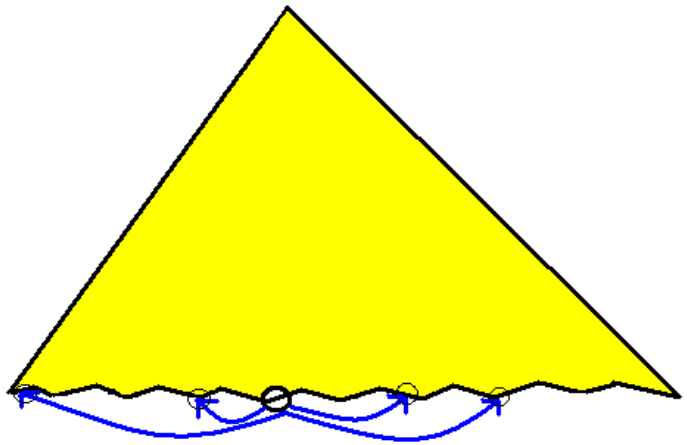
Local Search
Idea Generale per COP

Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



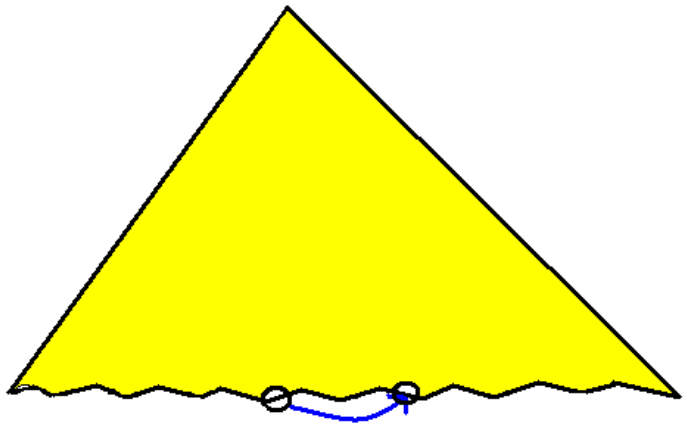
Local Search
Idea Generale per COP

Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



Local Search
Idea Generale per COP

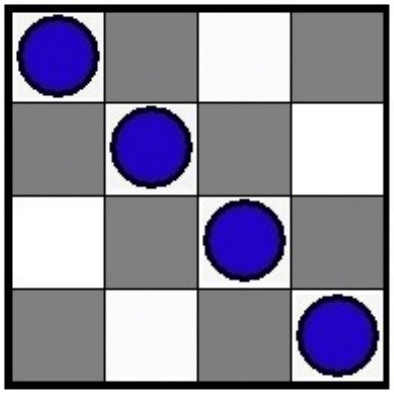
Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



Local Search
Idea Generale per CSP

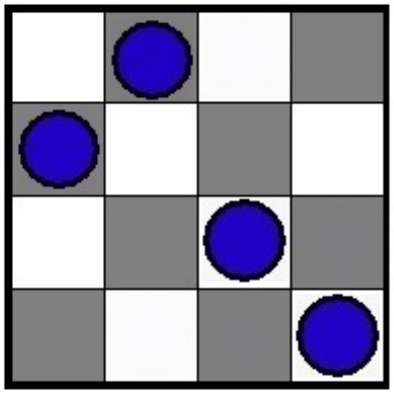
- Se si devono risolvere CSP, alcuni vincoli vengono indeboliti e spostati in una funzione da ottimizzare.
- Indebolendo i vincoli è facile trovare soluzioni al problema rilassato.
- Saranno soluzioni iniziali per il processo di local search
- Gli ottimi saranno sperabilmente soluzioni del CSP.
- Ad esempio nelle regine il vincolo diagonale scompare e la funzione da minimizzare è il numero di attacchi complessivo tra le regine.

Local Search
Esempio per le 4-regine

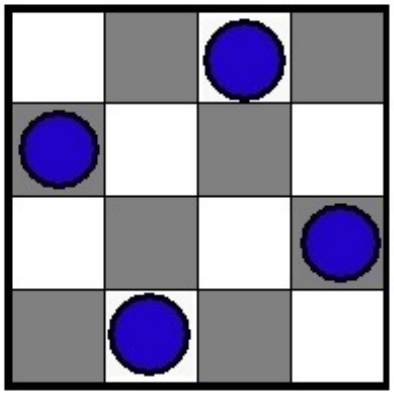


12 attacchi

$\Rightarrow$ swap(1,2)

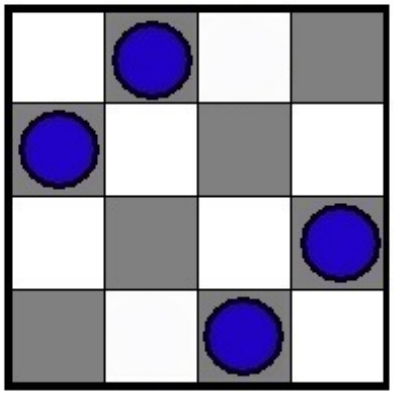


4 attacchi $\Downarrow$ swap(3,4)



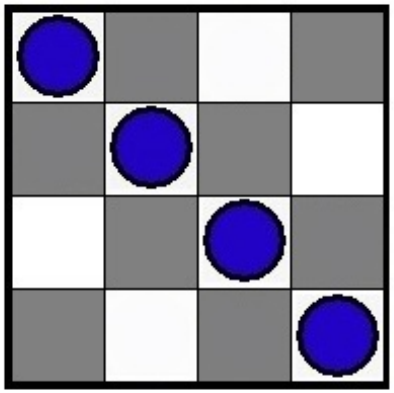
0 attacchi

$\Leftarrow$ swap(2,3)

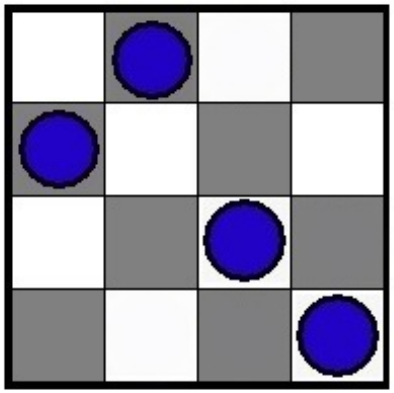


8 attacchi

Local Search
Esempio per le 4-regine

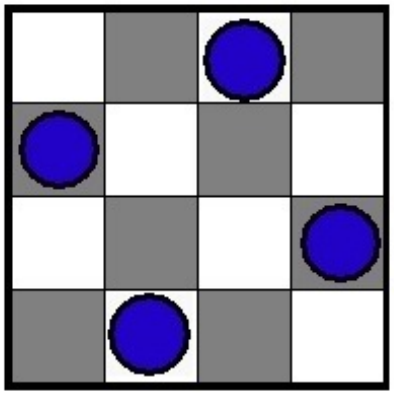


$\Rightarrow$ swap(1,2)

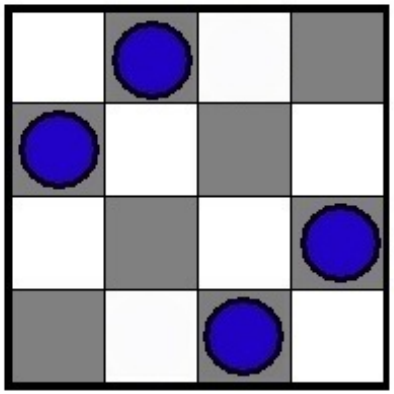


12 attacchi

4 attacchi $\Downarrow$ swap(3,4)



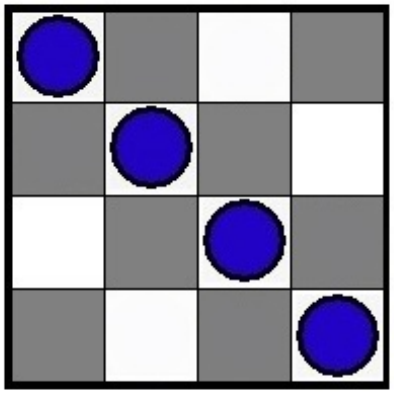
$\Leftarrow$ swap(2,3)



0 attacchi

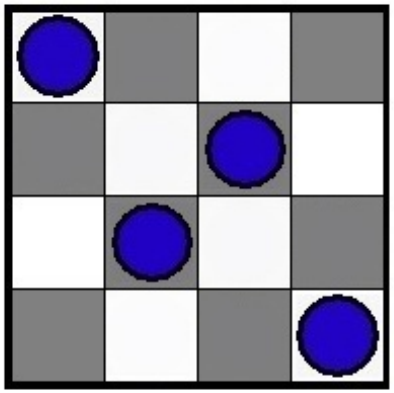
8 attacchi

Local Search
Altro esempio per le 4-regine

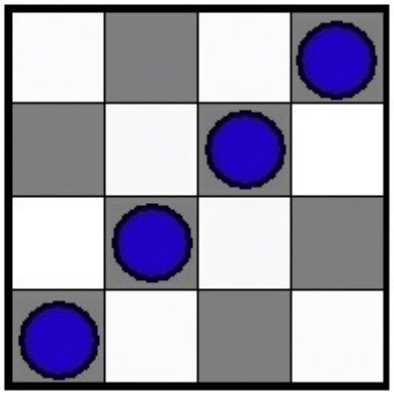


12 attacchi

$\Rightarrow$ swap(2,3)



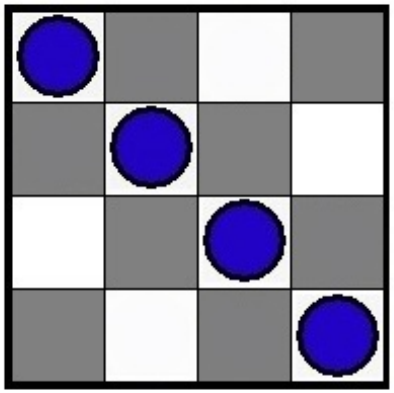
4 attacchi $\Downarrow$ swap(1,4)



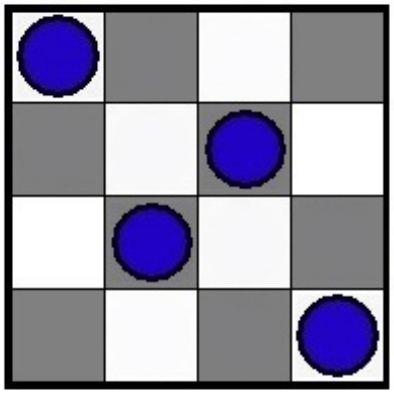
12 attacchi

Non tutte le scelte ci portano al minimo

Local Search
Altro esempio per le 4-regine

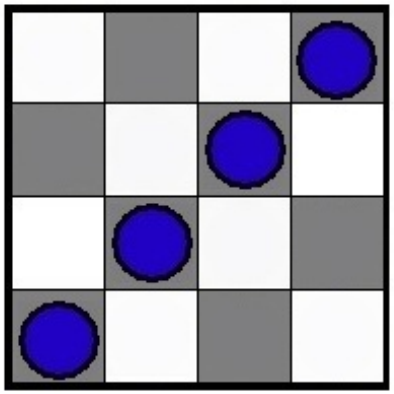


$\Rightarrow$ swap(2,3)



12 attacchi

4 attacchi $\Downarrow$ swap(1,4)



12 attacchi

Non tutte le scelte ci portano al minimo

- I vincoli vengono divisi in due famiglie:
  - Hard constraints: che devono essere verificati in tutte le soluzioni del CSP (nell'esempio erano i vincoli orizzontali)
  - Soft constraints: possono essere anche non soddisfatti, ma la loro violazione penalizza la funzione di costo (i vincoli diagonali)
- Lo spazio di ricerca (soluzioni degli hard constraints) dev'essere non vuoto.
- Va inoltre definita la funzione di costo $f(\sigma)$ che tenga conto (anche) dei soft constraints.

- Va detto come trovare la soluzione iniziale.
- Va definita una funzione di vicinato Neighborhood Relation $N(\sigma)$ che identifica un insieme di soluzioni raggiungibili da σ
- Se possibile, il vicinato viene caratterizzato mediante la nozione di *mossa* (p.es., lo scambio)
- Va inoltre detto:
  - Come scegliere la mossa
  - Come esplorare il vicinato
  - Come evitare soluzioni inadeguate (p. es., loop)
  - Quando fermarsi (di solito con un limite di tempo/numero di passi)

- Dato uno stato σ, si generi $N(\sigma)$.
  - Hill Climbing: scegli $\sigma' \in N(\sigma)$ che minimizza la funzione f, ovvero $f(\sigma') = \min\{f(\mu) : \mu \in N(\sigma)\}$
  - Simulated Annealing:
    - Sia T (temperatura) una var. che cala nel tempo.
    - Si scelga a caso $\sigma' \in N(\sigma)$.
    - Se $f(\sigma') < f(\sigma)$ ci si sposti su σ'.
    - Altrimenti, si generi un numero casuale $p \in [0, 1]$.
    - Sia F una funzione in $[0, 1]$ che cresce con il decrescere della temperatura e che tiene conto della distanza tra $f(\sigma)$ e $f(\sigma')$.
    - Se $p > F(T, f(\sigma), f(\sigma'))$ ci si sposta in σ' altrimenti si genera una nuova σ'.

- Monte Carlo: sta a monte di S.A. (anni 40 a Los Alamos). Tuttavia talvolta Monte Carlo sta per S.A. con T costante nel tempo e $F = 0.5$.
- Tabu Search: Come Simulated Annealing, con la differenza che ci si memorizza una lista degli ultimi stati visitati (per evitare loops).
- Altre ...

Local Search
Quando si usa

- Per problemi discreti dove sia naturale il concetto di scambio (p.es. problemi di turni di lavoro, orari lezioni, ferie etc.)
- Per problemi continui (simulazioni Montecarlo/Simulated Annealing)
- Relativamente facile da implementare. Consiglio tuttavia di usare tools ad hoc, quali ad esempio EasyLocal (Java o C++) che permettono di concentrarsi sul problema e sulle meta euristiche più adatte.

PROGRAMMAZIONE LINEARE INTERA

minimize $c_1 X_1 + c_2 X_2 + \cdots + c_n X_n$

subject to

$$a_{1,1} X_1 + a_{1,2} X_2 + \cdots + a_{1,n} X_n \; op \; b_1$$
$$a_{2,1} X_1 + a_{2,2} X_2 + \cdots + a_{2,n} X_n \; op \; b_2$$
$$\cdots$$
$$a_{m,1} X_1 + a_{m,2} X_2 + \cdots + a_{m,n} X_n \; op \; b_m$$
$$X_1, \ldots, X_n \in \mathbb{N}$$

op può essere $=, \leq, <, \geq, >$.

È un COP con funzione e vincoli *lineari*.

PROGRAMMAZIONE LINEARE INTERA

minimize $c_1 X_1 + c_2 X_2 + \cdots + c_n X_n$

subject to

$$a_{1,1} X_1 + a_{1,2} X_2 + \cdots + a_{1,n} X_n \; op \; b_1$$
$$a_{2,1} X_1 + a_{2,2} X_2 + \cdots + a_{2,n} X_n \; op \; b_2$$
$$\cdots$$
$$a_{m,1} X_1 + a_{m,2} X_2 + \cdots + a_{m,n} X_n \; op \; b_m$$
$$X_1, \ldots, X_n \in \mathbb{N}$$

op può essere $=, \leq, <, \geq, >$.

È un COP con funzione e vincoli *lineari*.

Programmazione Lineare
Quando si usa

- Un COP di PL ammette soluzione polinomiale (Elissoide, Karmarkar, Simplesso).
- Se il COP che dovete risolvere si può scrivere cosi', vanno assolutamente usati i tools per PL (p.es. CPLEX, DASH, MINTO, OSL, Qsopt, GLPK, . . .)

Programmazione Lineare Intera
Quando si usa

- Un COP di PLI può invece essere la codifica di un problema NP-completo (si pensi al 3-coloring)
- I tools suddetti richiamano iterativamente un problema rilassato (privo dei vincoli di interezza) per raggiungere l'ottimo sugli interi. In generale la computazione può richiedere tempo esponenziale.
- La codifica di un problema come PLI è un'arte. La codifica naturale raramente permette esecuzione efficiente. Quella buona viene raggiunta dopo fasi intermedie piuttosto onerose (generazione colonne).
- Se il problema da codificare non è ancora completamente chiaro (p.es. problemi biologici dove conoscenze e dati cambiano frequentemente) non è saggio usare PLI.

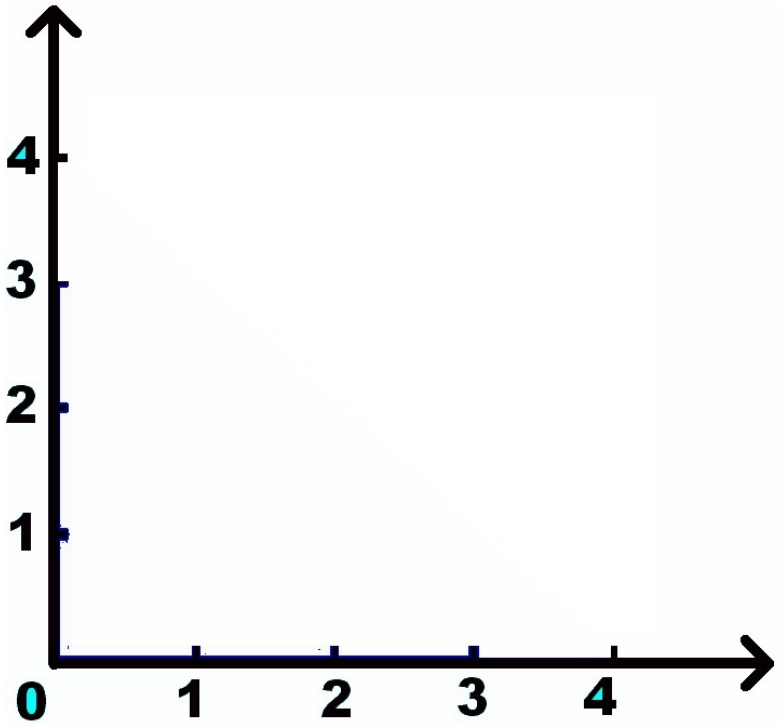
- Per risolvere il COP:
  minimize $C\vec{X}$, subject to $P \equiv A\vec{X} \leq B$, ove $\vec{X} \in \mathbb{N}$
- Scriviamo una funzione ricorsiva (all'inizio $K = +\infty$)

```
function IP(P, C, K)
    let (z, x⃗, flag) = LP(P,C)
%% z minimo, x⃗ soluzione, flag dice se esiste soluz.
    if not(flag) or z ≥ K return K
    elseif intera(x⃗) return z
    else pick a non-integer variable X_i from x
        K = IP(P ∧ X_i ≤ ⌊x_i⌋, C, K)
        K = IP(P ∧ X_i ≥ ⌈x_i⌉, C, K)
        return K
```

Programmazione Lineare Intera

Esempio: Branch and Bound

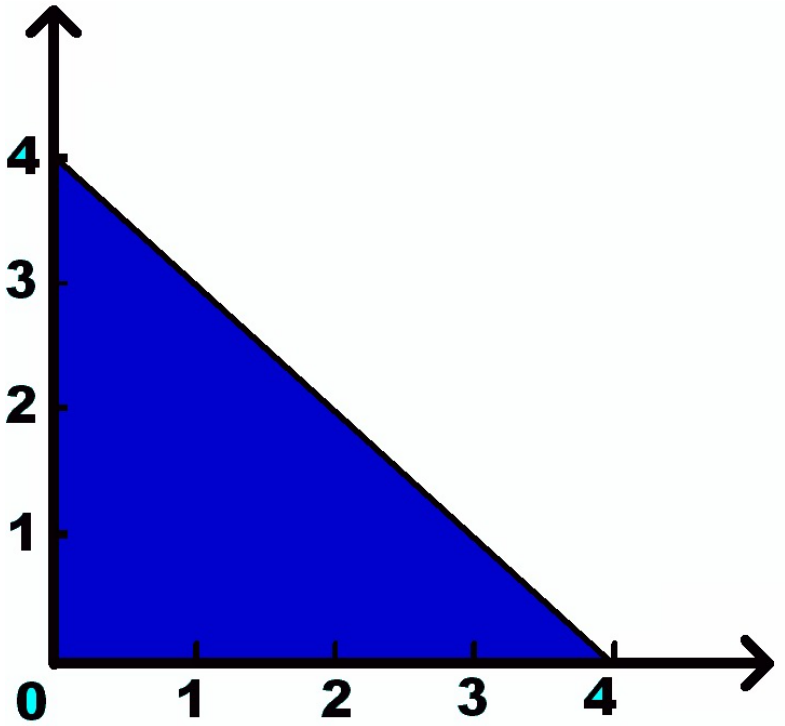
minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



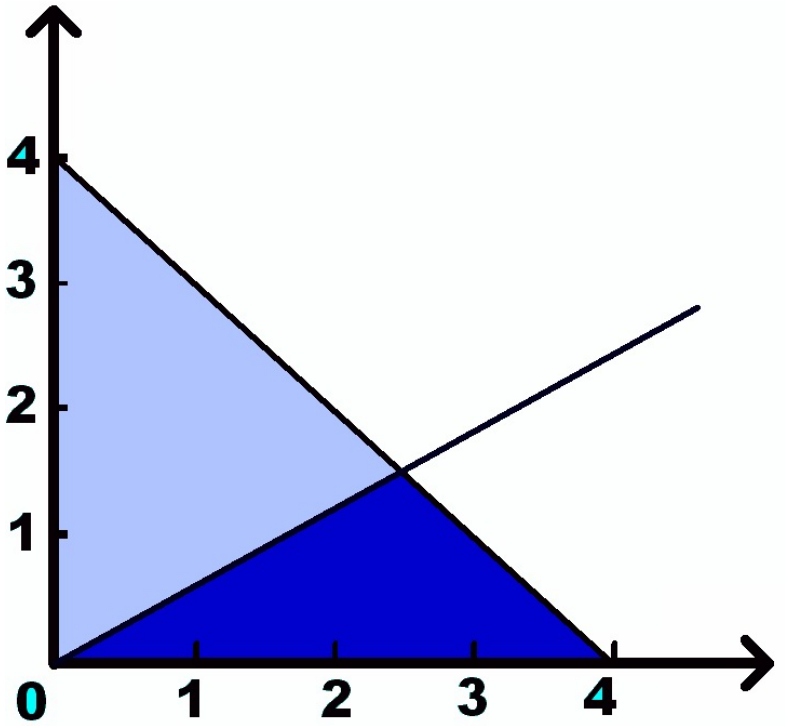
Programmazione Lineare Intera

Esempio: Branch and Bound

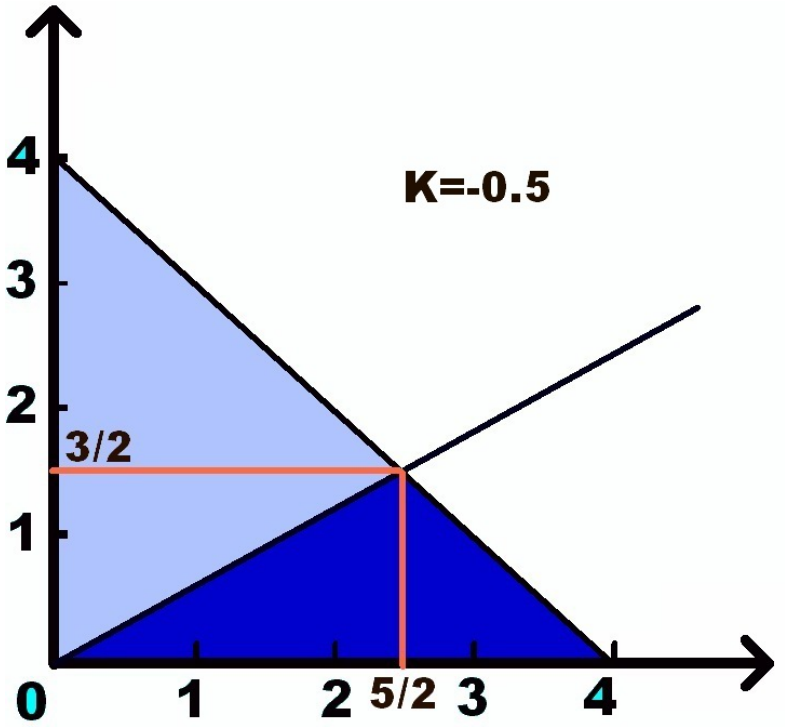
minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.

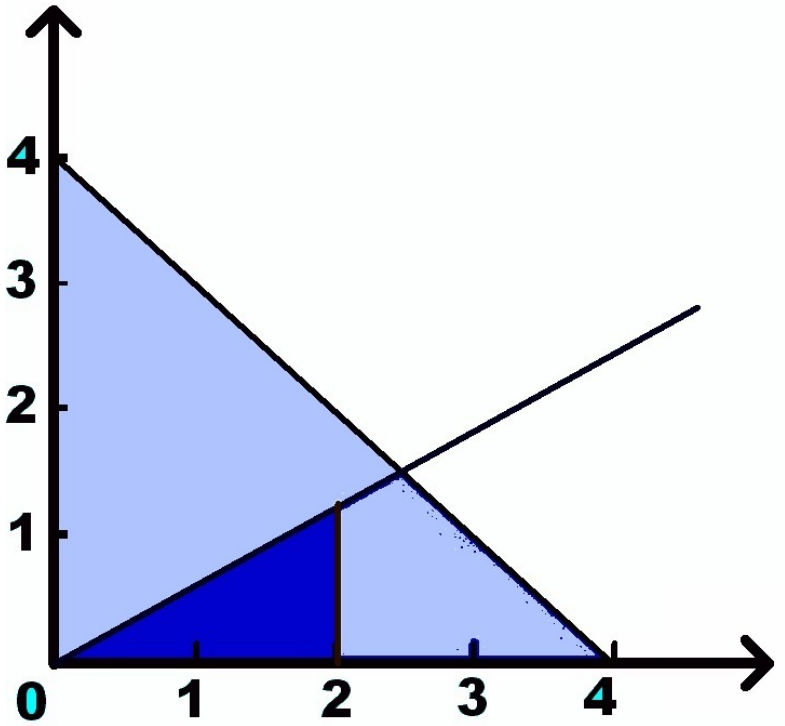


minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



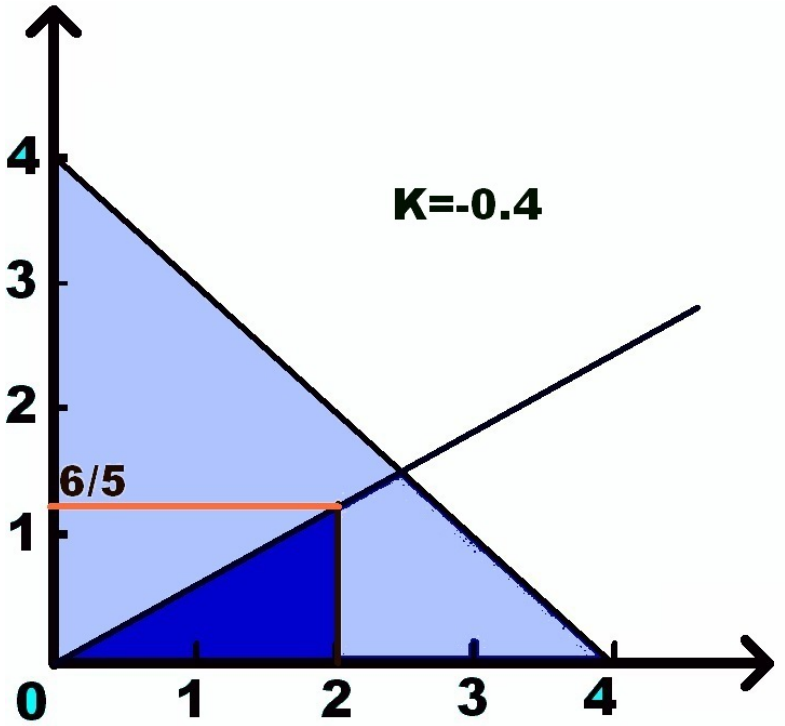
LP

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



$X \leq 2$

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.

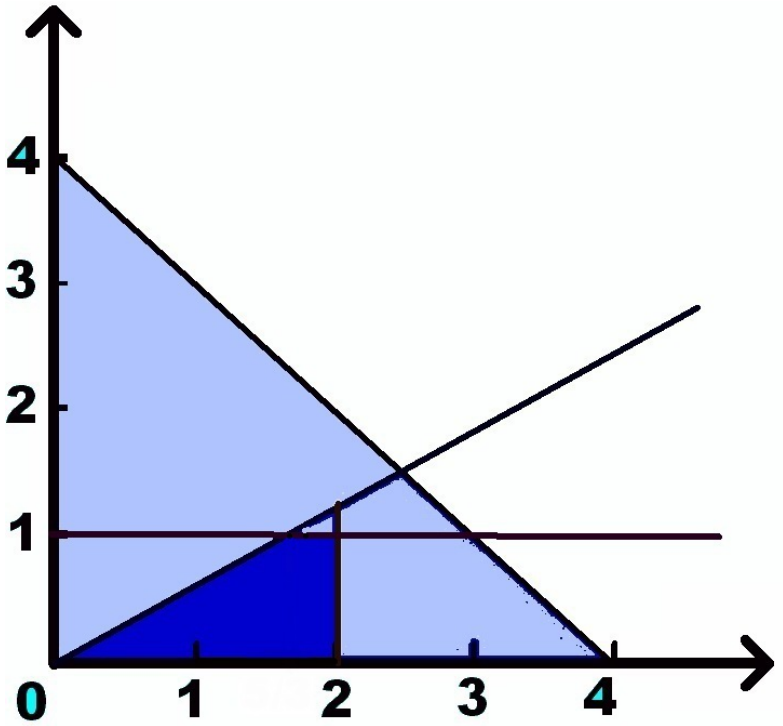


LP

Programmazione Lineare Intera

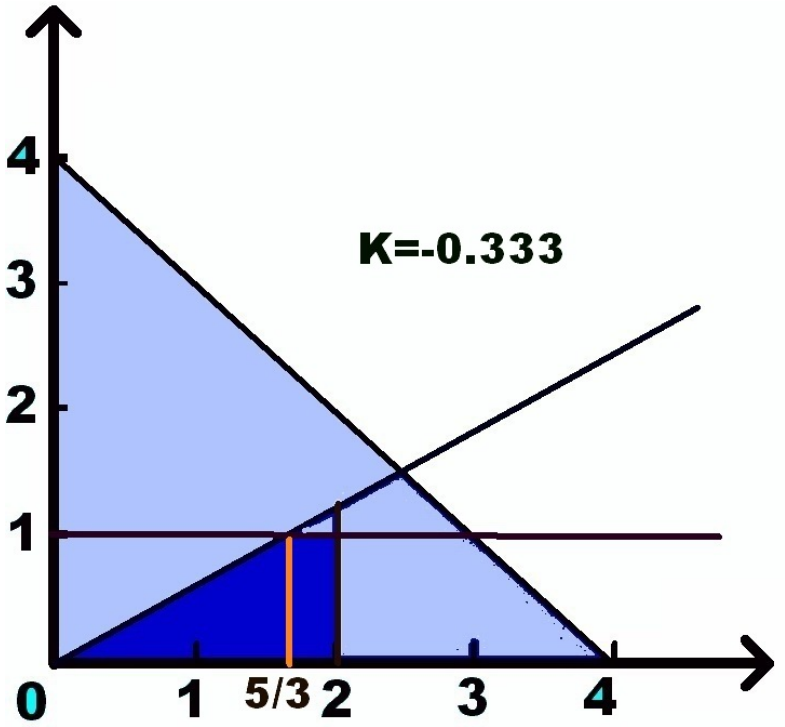
Esempio: Branch and Bound

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



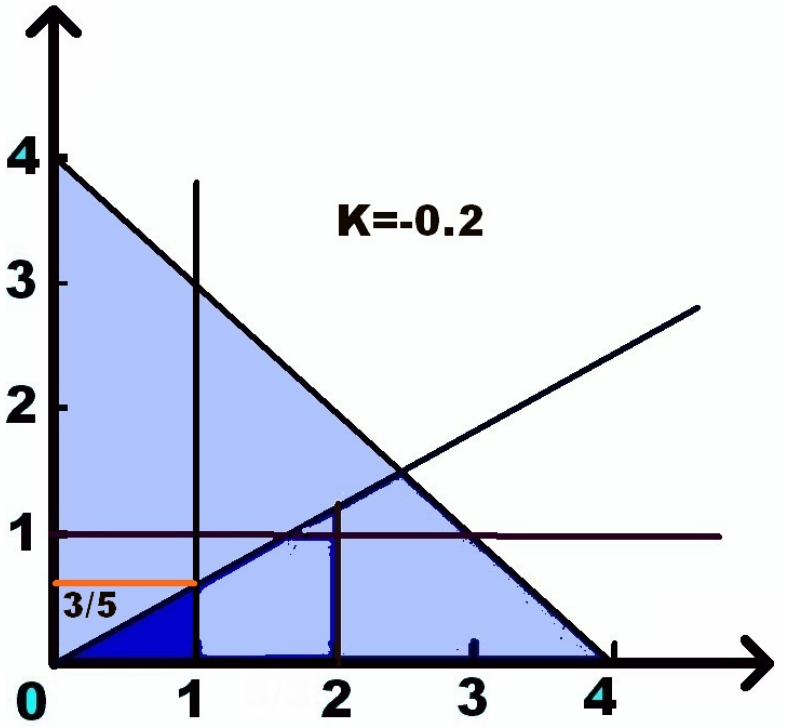
$$X \leq 2, Y \leq 1$$

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



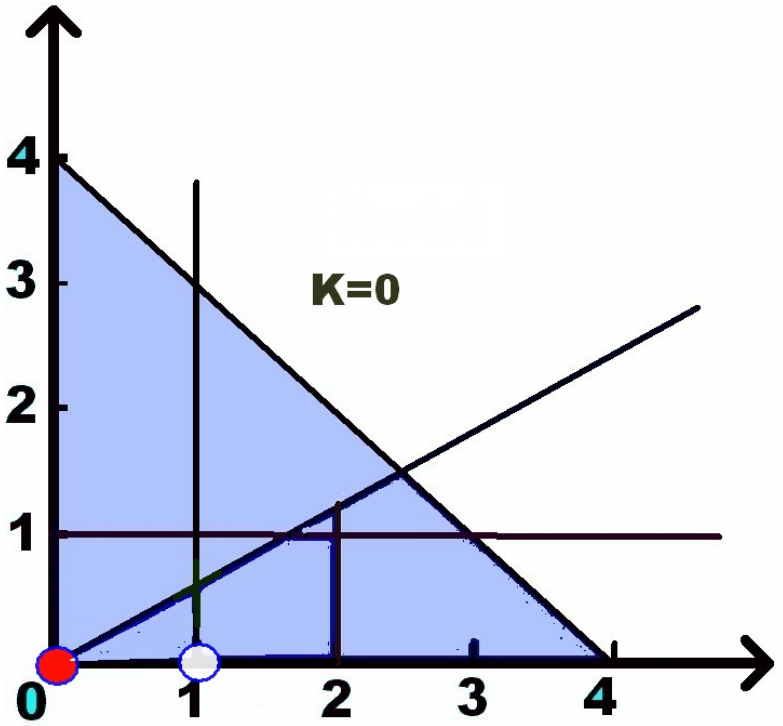
LP

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



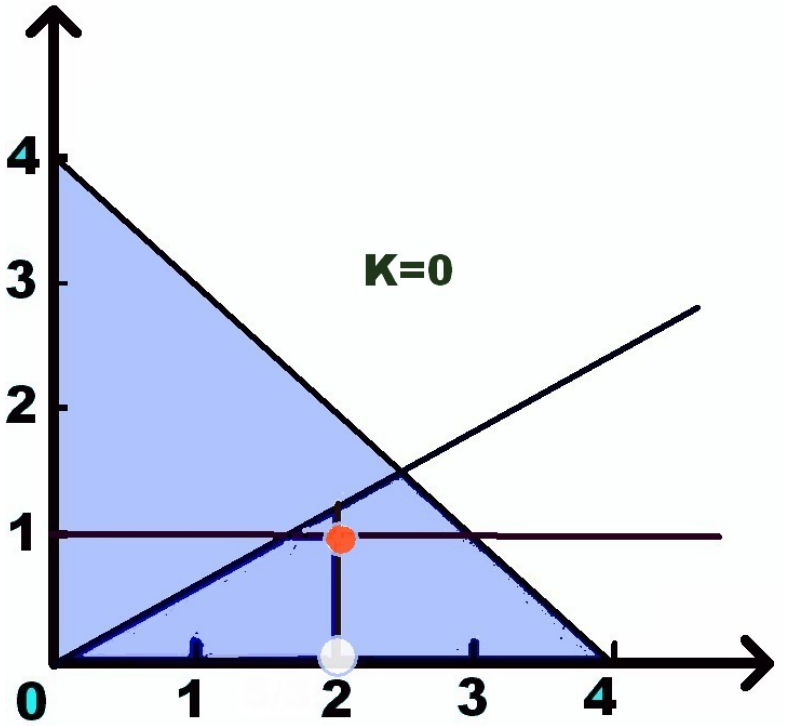
$X \leq 1, Y \leq 1$ –LP

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



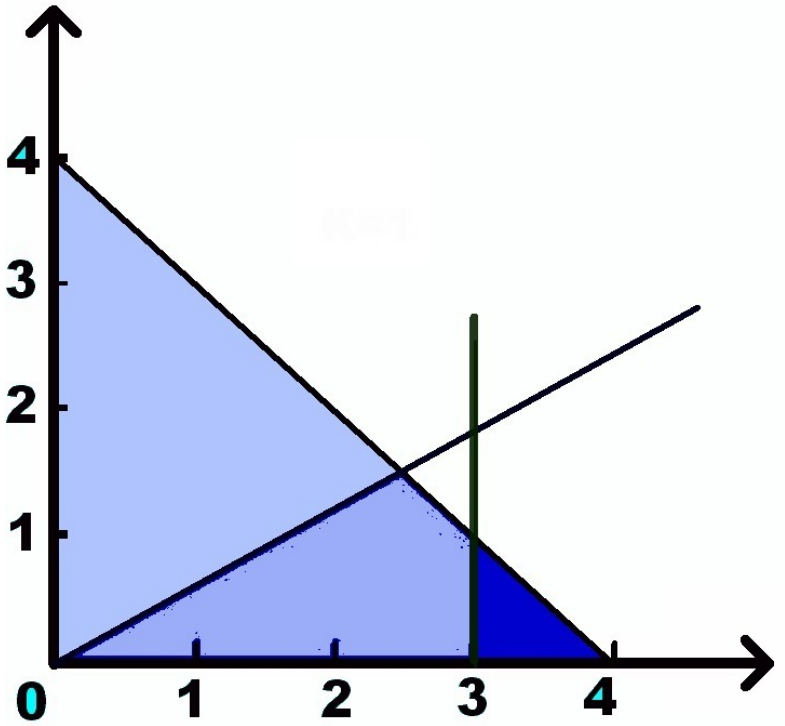
$X \leq 1, Y \leq 0$ –LP: feasible solution

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



$X \geq 2, X \leq 2, Y \leq 1$ — LP: feasible solution

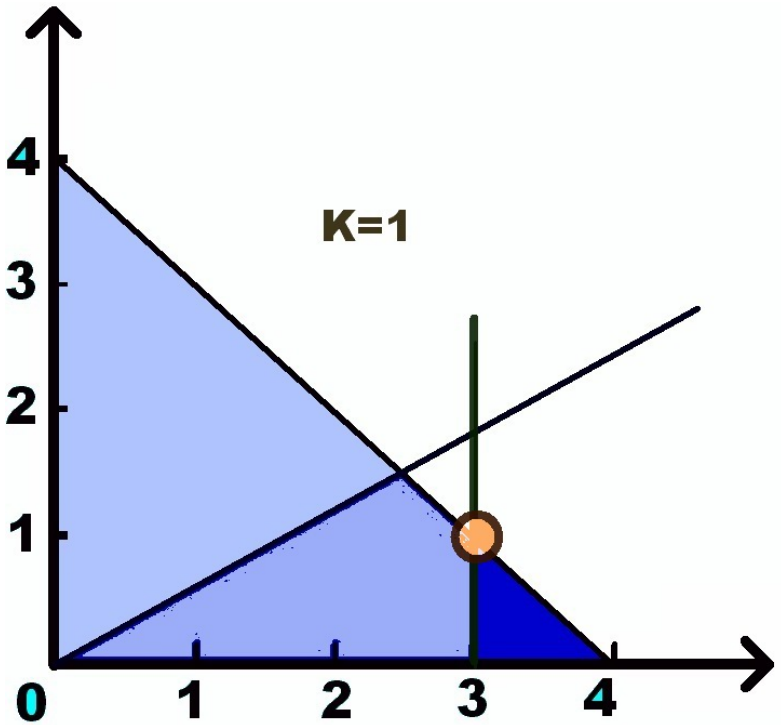
minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



$X \geq 3$

Programmazione Lineare Intera
Esempio: Branch and Bound

minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



LP: CUT
(non importa se (X, Y) siano interi o meno)

SAT
Definizione

- Dato un insieme di variabili $X_1, \ldots, X_n$ a valori Booleani e
- Una formula logica proposizionale ottenuta da esse, già in forma normale

$$(\ell_1^1 \vee \cdots \vee \ell_{n_1}^1) \wedge \cdots \wedge (\ell_1^k \vee \cdots \vee \ell_{n_k}^k)$$

  ove ogni ℓ_j^i è una variabile X_p o la sua negazione $\neg X_p$
- Stabilire (e trovare) un assegnamento delle variabili in {false,true} (o {0,1}) che renda vera la forumula.
- Ricordo che:
  - $0 \vee 0 = 0, 0 \vee 1 = 1 \vee 0 = 1 \vee 1 = 1$
  - $0 \wedge 0 = 0 \wedge 1 = 1 \wedge 0 = 0, 1 \wedge 1 = 1$
  - $\neg 0 = 1, \neg 1 = 0$

SAT

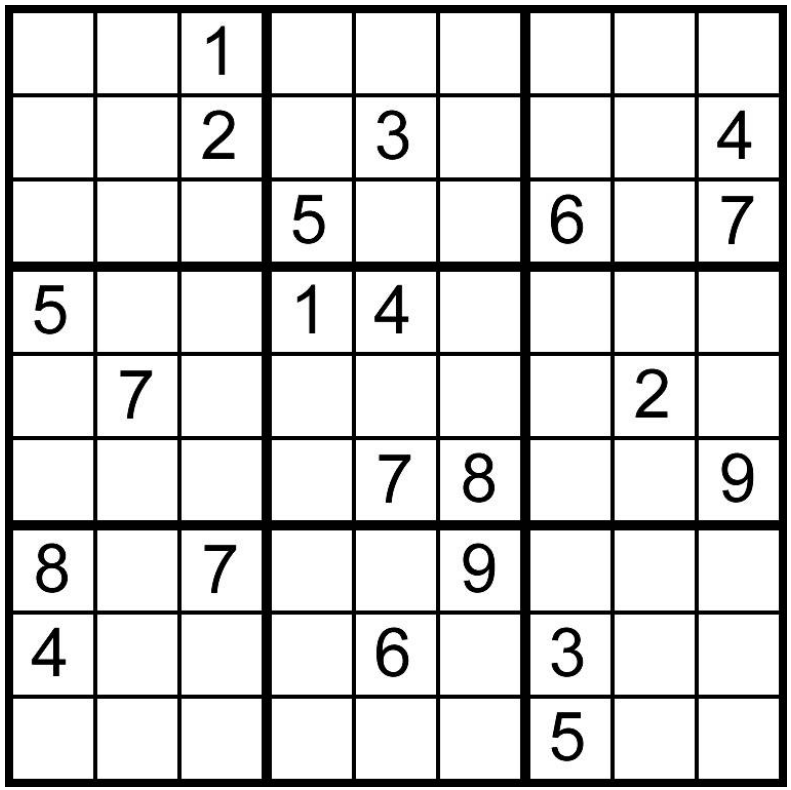
- SAT è stato il primo problema dimostrato essere NP-completo (Cook)
- Ogni problema in NP si può trasformare in un problema di SAT.
- Un CSP su domini finiti è tipicamente in NP (salvo uso di vincoli particolari che potrebbero farci “uscire” da NP)
- Pertanto potremmo studiare la *riduzione* del nostro problema a un’istanza di SAT.
- Esistono vari SAT solvers liberi e particolarmente efficienti (p.es. MiniSAT, PicoSAT, RelSAT, etc.—si veda la SAT competition) che possono poi essere usati indirettamente per risolvere il nostro problema.
- Le riduzioni non sono di solito immediate.

SAT
Esempio: Le 4 regine

- Dobbiamo dire che le variabili X_1, X_2, X_3, X_4 hanno valore da 1 a 4.
- Per ognuna di esse (poniamo X_i) introduciamo 4 variabili Booleane: $Z_1^i, Z_2^i, Z_3^i, Z_4^i$.
- Devo dire che una di esse è vera: $Z_1^i \vee Z_2^i \vee Z_3^i \vee Z_4^i$.
- Devo dire che due non sono vere: $\neg Z_1^i \vee \neg Z_2^i, \neg Z_1^i \vee \neg Z_3^i, \neg Z_1^i \vee \neg Z_4^i, \ldots, \neg Z_3^i \vee \neg Z_4^i$.
- Poi devo mettere i vincoli di attacco orizzontale: per $i \neq j$ devo dire: $\neg Z_1^i \vee \neg Z_1^j, \neg Z_2^i \vee \neg Z_2^j, \neg Z_3^i \vee \neg Z_3^j, \neg Z_4^i \vee \neg Z_4^j$.
- Infine devo mettere i vincoli di attacco diagonale: per $i \neq j$. Poniamo $i = 2, j = 4$, devo dire: $\neg Z_1^2 \vee \neg Z_3^4, \neg Z_2^2 \vee \neg Z_4^4, \neg Z_3^2 \vee \neg Z_1^4, \neg Z_4^2 \vee \neg Z_2^4$.
- Il tutto va poi codificato (parte facile) in formato .cnf

- Problemi non (troppo) numerici
- Comunque problemi di cui si conosce già bene la formalizzazione ad alto livello (ogni cambiamento ci potrebbe costringere a riformulare il tutto)

Un CSP divertente
Sudoku



$X_{1,3} = 1, X_{2,3} = 2, X_{2,5} = 3, X_{2,9} = 4, \ldots, X_{9,7} = 5$

$\text{alldifferent}(X_{1,1}, \ldots, X_{1,9}) \ldots \text{alldifferent}(X_{9,1}, \ldots, X_{9,9})$

$\text{alldifferent}(X_{1,1}, \ldots, X_{9,1}) \ldots \text{alldifferent}(X_{1,9}, \ldots, X_{9,9})$

$\text{alldifferent}(X_{1,1}, \ldots, X_{3,3}) \ldots \text{alldifferent}(X_{7,7}, \ldots, X_{9,9})$