Constraint Programming & Planning

Agostino Dovier

Università di Udine
Dipartimento di Matematica e Informatica

Udine, Marzo 2011

Sommario

Constraint Satisfaction Problem

Definizione rigorosa

- 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.

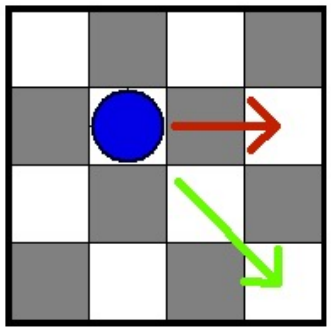
CSP/COP

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

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 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

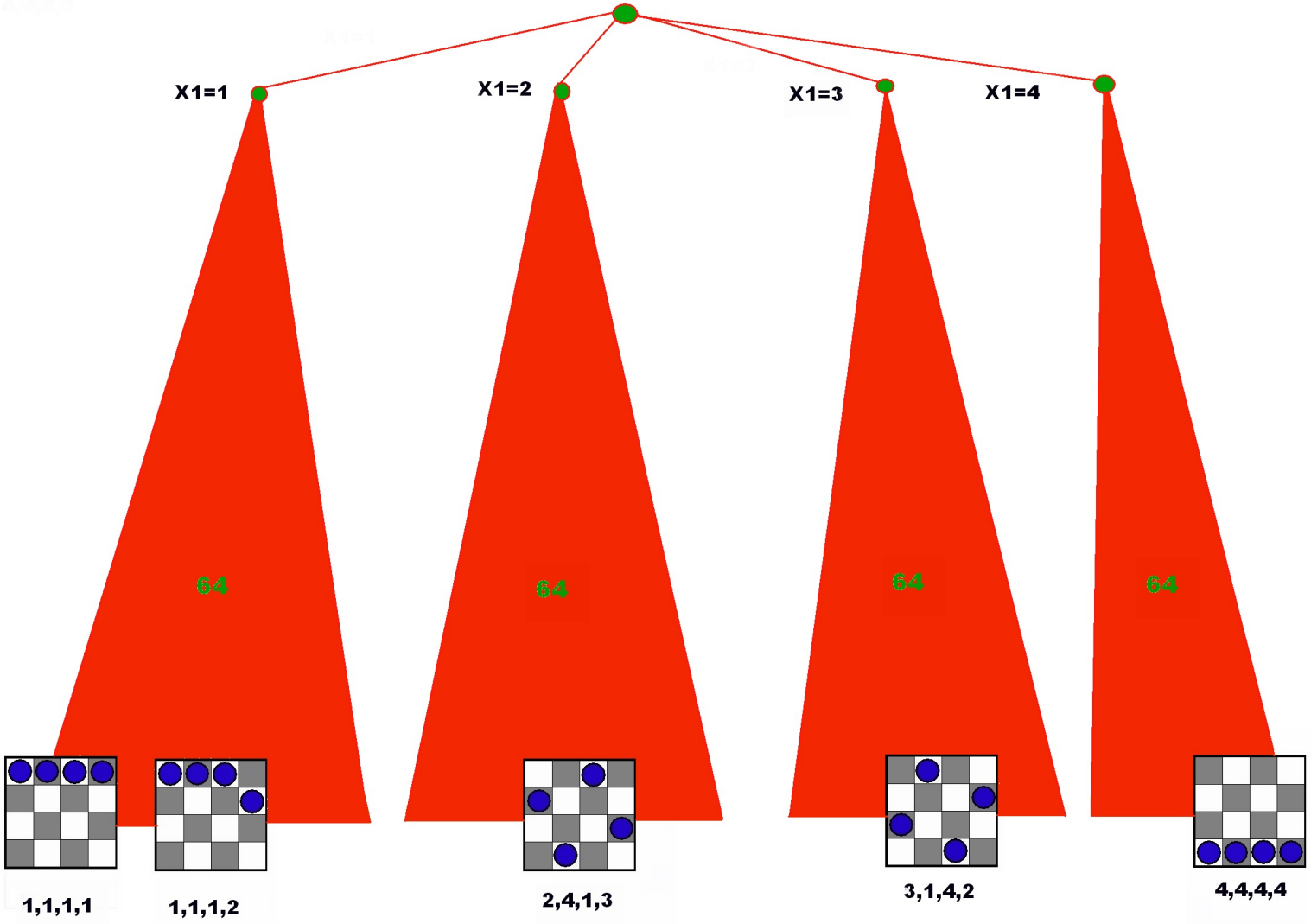
- Dato un CSP $\langle X_1 \in \mathcal{D}_1, \ldots, X_n \in \mathcal{D}_n; C \rangle$ (con C denotiamo l'insieme di tutti i vincoli)
- lo *spazio di ricerca* (search space) è l'insieme delle n-uple

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

  (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!

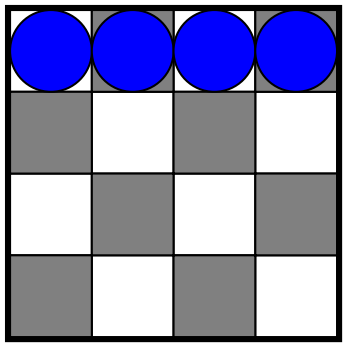
Spazio di ricerca

Esempio delle 4-regine



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

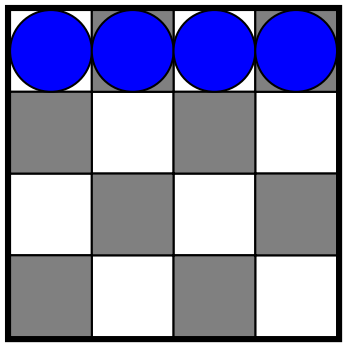
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$



inconsistente

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

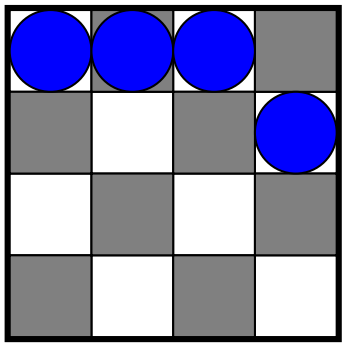
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 1$$



inconsistente

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

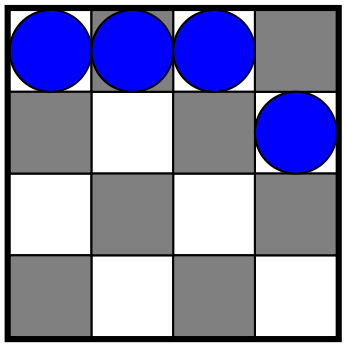
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$



inconsistente

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

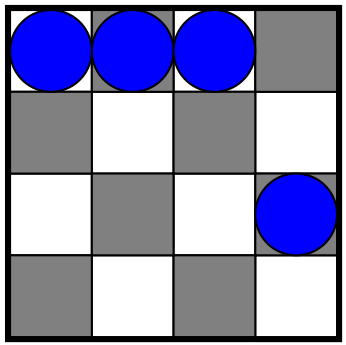
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 2$$



inconsistente

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

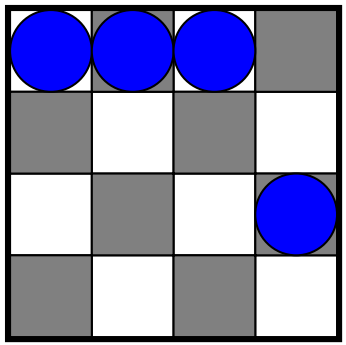
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$



inconsistente

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

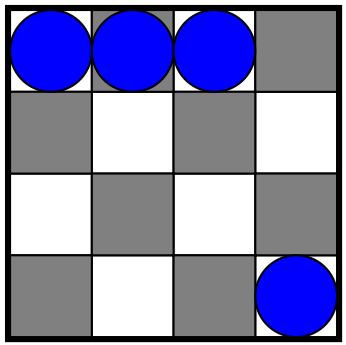
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 3$$



inconsistente

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

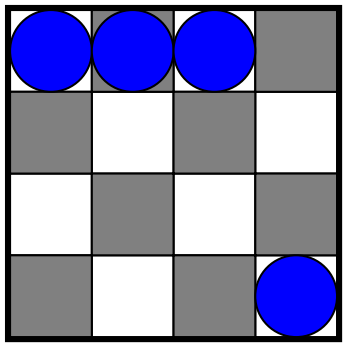
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$



inconsistente

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

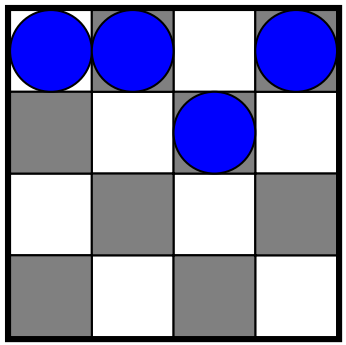
$$X_1 = 1, X_2 = 1, X_3 = 1, X_4 = 4$$



inconsistente

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

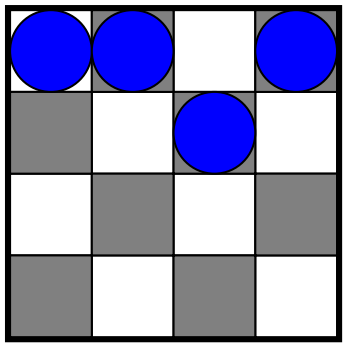
$$X_1 = 1, X_2 = 1, X_3 = 2, X_4 = 1$$



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

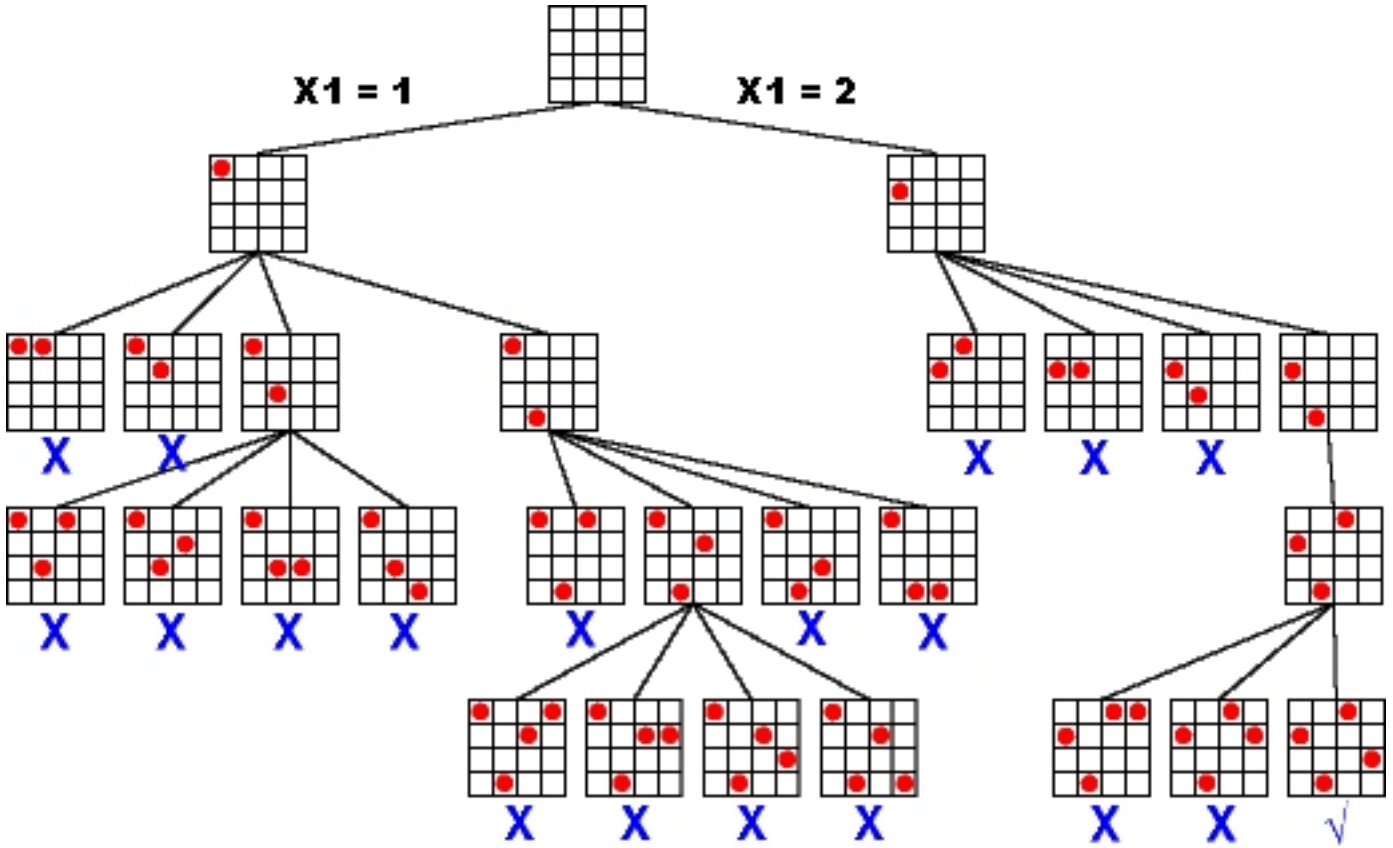
METODI PER RISOLVERE CSP/COP

CODIFICA DEL GENERATE & TEST

```
regine_get(N,Allocazione) :-
       genera(N,Allocazione),
       safe(Allocazione).
genera(N,Allocazione) :-
       length(Allocazione,N),
       dominio(N,ListaValori),
       funzione(Allocazione,ListaValori). %% c.f. permutation
dominio(0,[]).
dominio(N,[N|R]) :-
       N>0, M is N-1, dominio(M,R).
safe([]).
safe([Q|Qs]) :-
        safe(Qs),  \+ attacca(Q,Qs).
attacca(X,Xs) :- att(X,1,Xs).
att(X,Offset,[Y|_R]) :- X is Y+Offset.
att(X,_Offset,[X|_R]).
att(X,Offset,[Y|_R]) :- X is Y-Offset.
att(X,Offset,[_Y|R]) :- N1 is Offset+1,
        att(X,N1,R).
```

METODI PER RISOLVERE CSP/COP

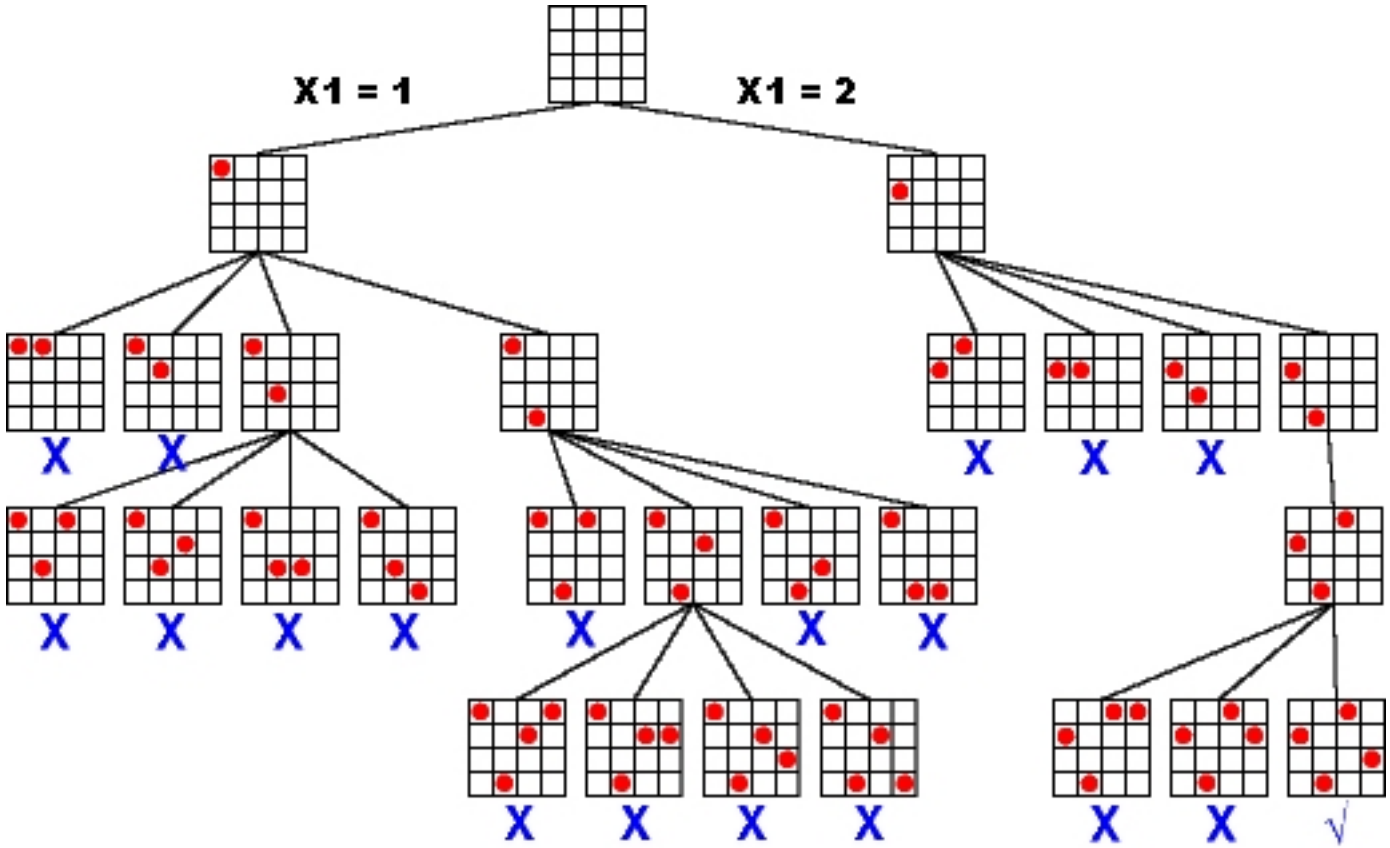
RICERCA SOLUZIONI NAIVE: ASSEGNAMENTI PARZIALI E BACKTRACKING



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

METODI PER RISOLVERE 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

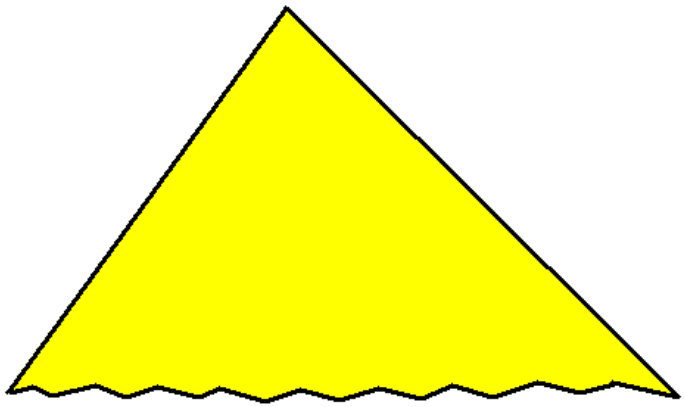
Codifica degli assegnamenti parziali


```
regine_opt(N,Allocazione) :-
   dominio(N,ListaValori),
   aggiungi_regina(ListaValori,[],Allocazione).
aggiungi_regina(NonMesse,Sicure,Allocazione) :-
   select(Q,NonMesse,Rimanenti),
   \+ attacca(Q,Sicure),
   aggiungi_regina(Rimanenti,[Q|Sicure],Allocazione).
aggiungi_regina([],Allocazione,Allocazione).
```

Local Search

Idea Generale per COP

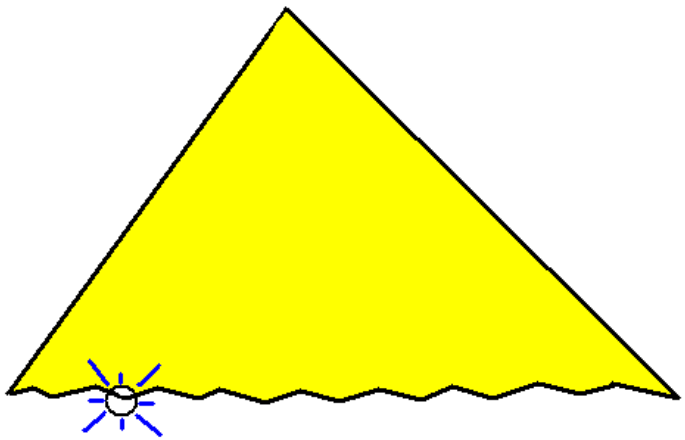
Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



Local Search

Idea Generale per COP

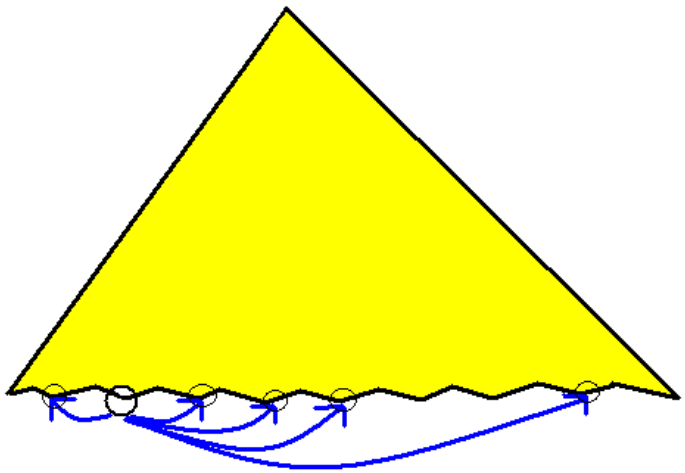
Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



Local Search

Idea Generale per COP

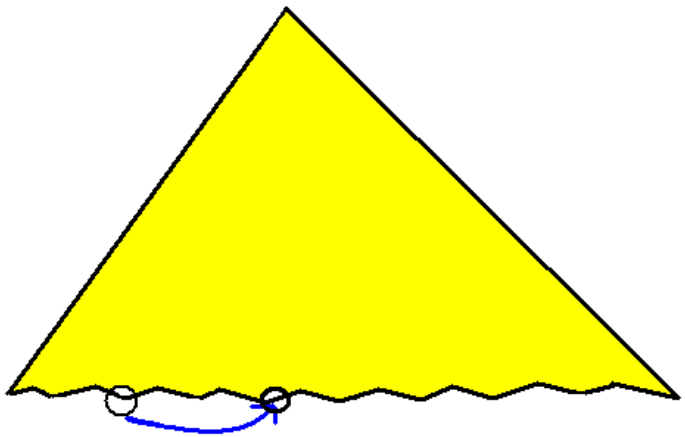
Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



Local Search

Idea Generale per COP

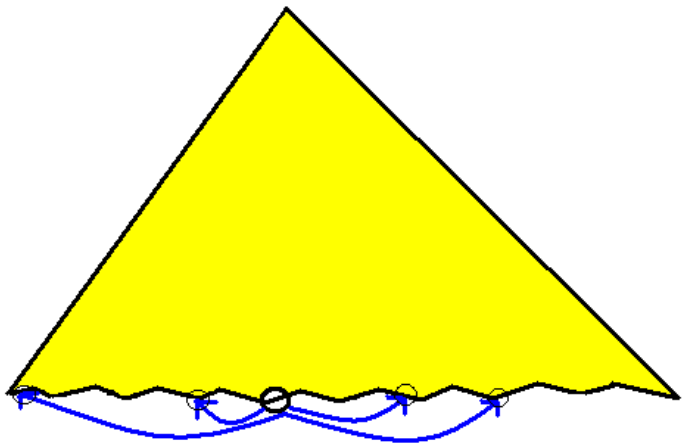
Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



Local Search

Idea Generale per COP

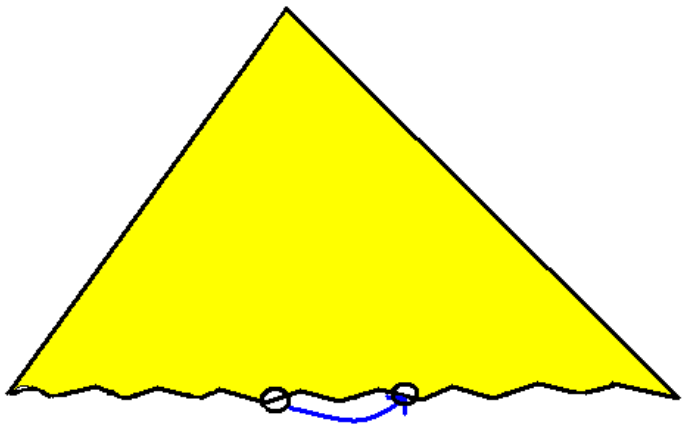
Ci si muove sulla frontiera dell'albero di ricerca per trovare soluzioni migliori. Ci sono varie tecniche per decidere come muoversi. Spesso sono stocastiche.



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.



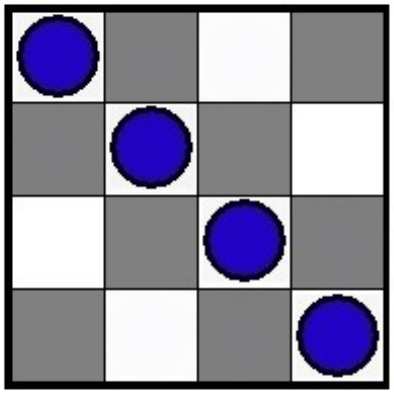
Local Search

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.

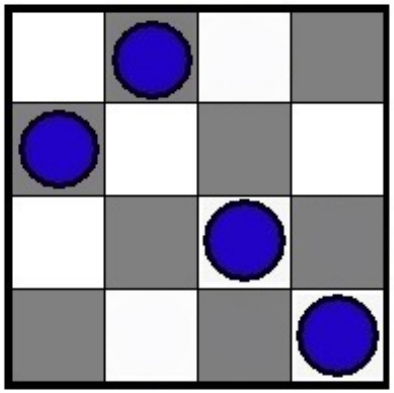
Local Search

Esempio per le 4-regine



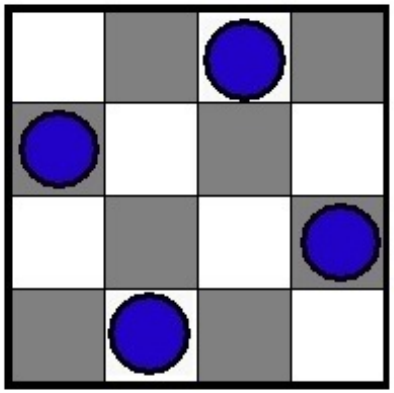
12 attacchi

$\Rightarrow$ swap(1,2)



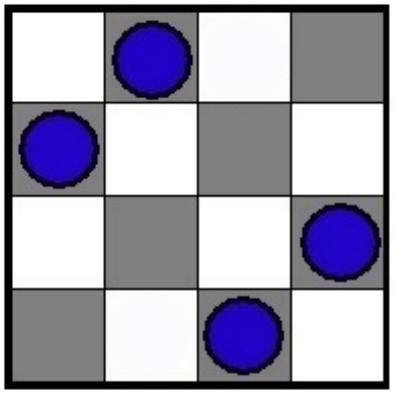
4 attacchi

$\Downarrow$ swap(3,4)



0 attacchi

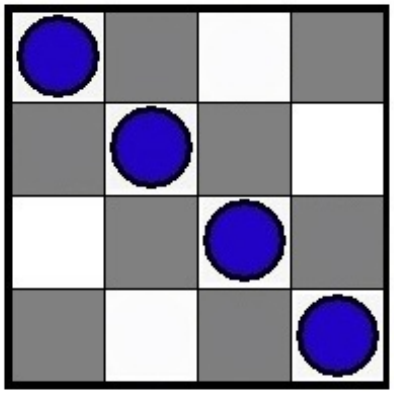
$\Leftarrow$ swap(2,3)



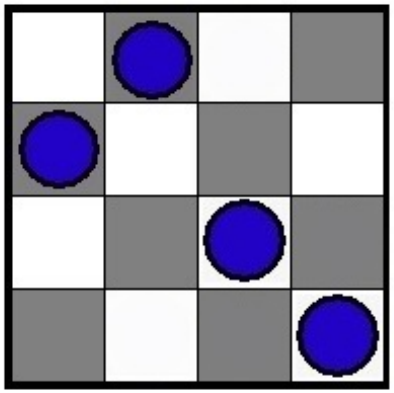
8 attacchi

Local Search

Esempio per le 4-regine

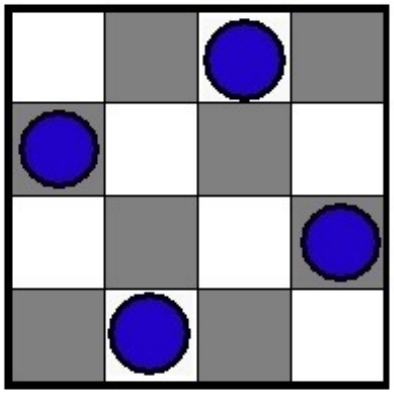


$\Rightarrow$ swap(1,2)

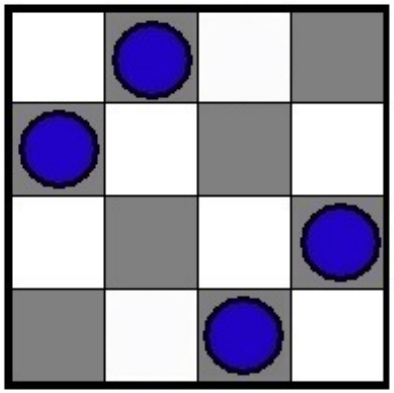


12 attacchi

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



$\Leftarrow$ swap(2,3)

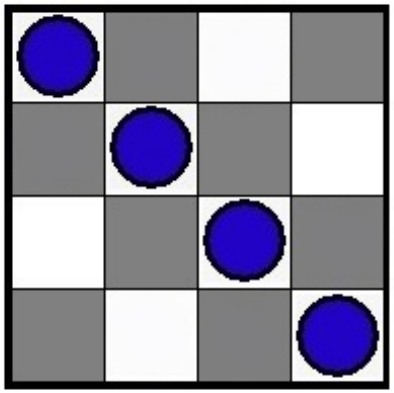


0 attacchi

8 attacchi

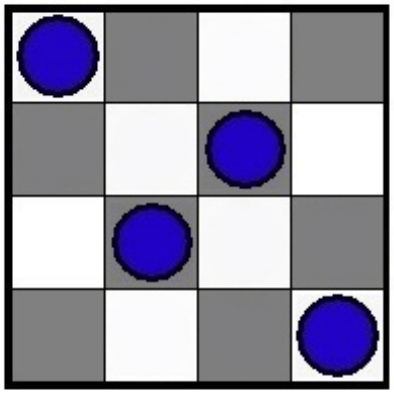
Local Search

Altro esempio per le 4-regine

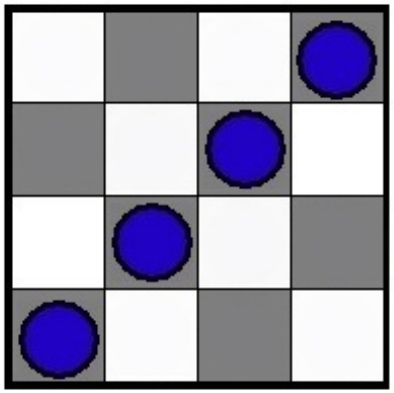


12 attacchi

$\Rightarrow$ swap(2,3)



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

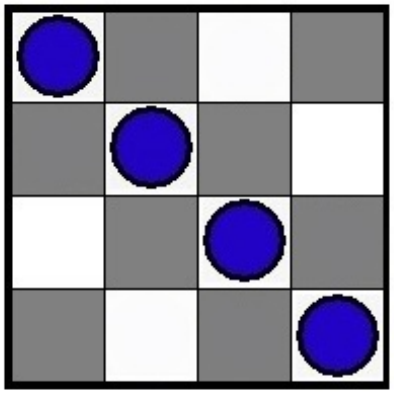


12 attacchi

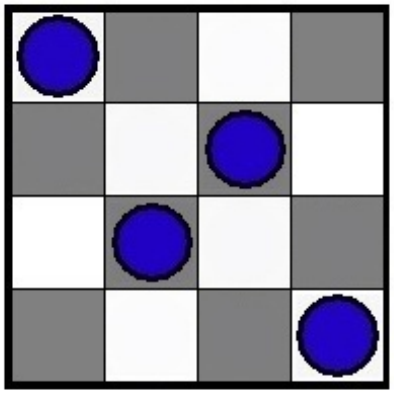
Non tutte le scelte ci portano al minimo

Local Search

Altro esempio per le 4-regine



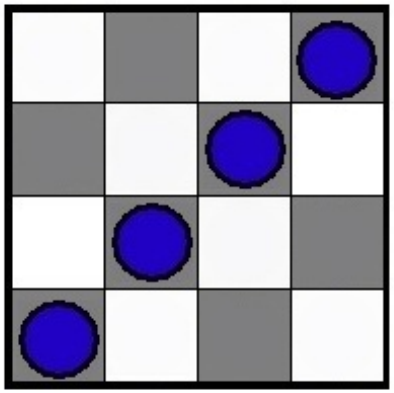
$\Rightarrow$ swap(2,3)



12 attacchi

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

Non tutte le scelte ci portano al minimo



12 attacchi

- 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)
- L'insieme delle soluzioni degli hard constraints dev'essere non vuoto.
- Va inoltre definita la funzione di costo $f(\sigma)$ che tenga conto (anche) dei soft constraints.

Local Search

Operativamente

- 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)

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 duddetti 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.

Programmazione Lineare Intera

Branch and Bound

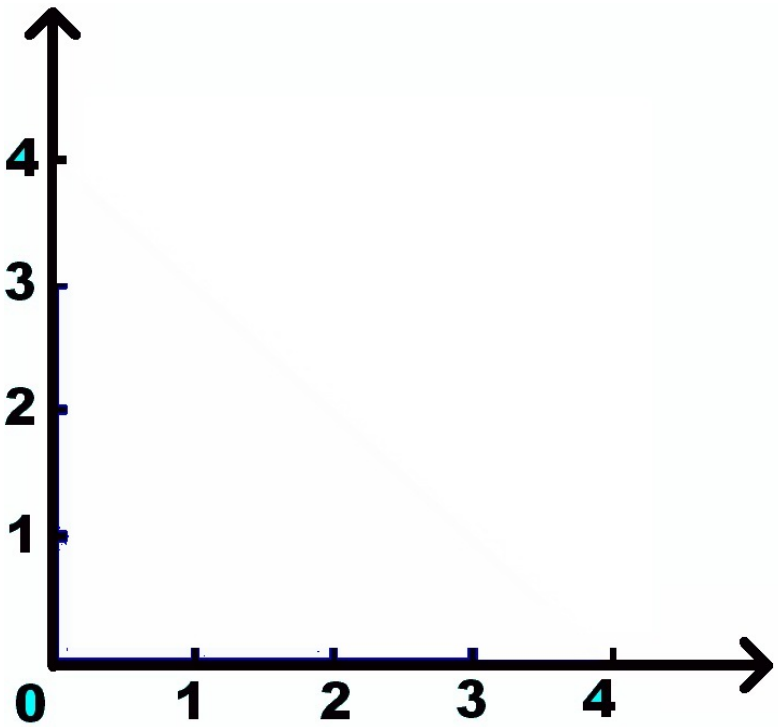
- Per risolvere il COP:
  minimize $C\vec{X}$, subject to $P \equiv A\vec{X} \leq B$, ove $\vec{X} \in \mathbb{N}$
- Scriviamo una funzione ricorsiva (all'inizio $K = +\infty$)

```
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

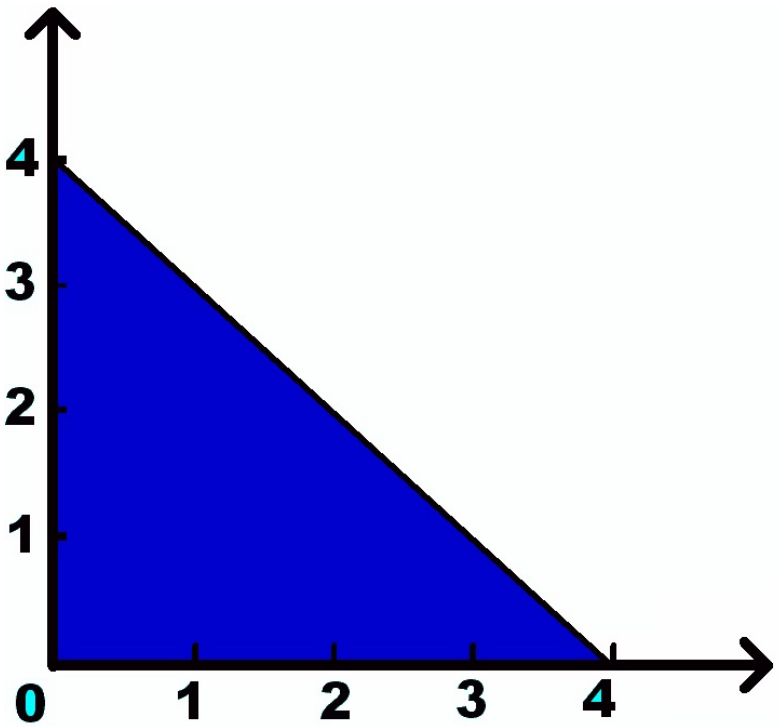
minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



Programmazione Lineare Intera

Esempio: Branch and Bound

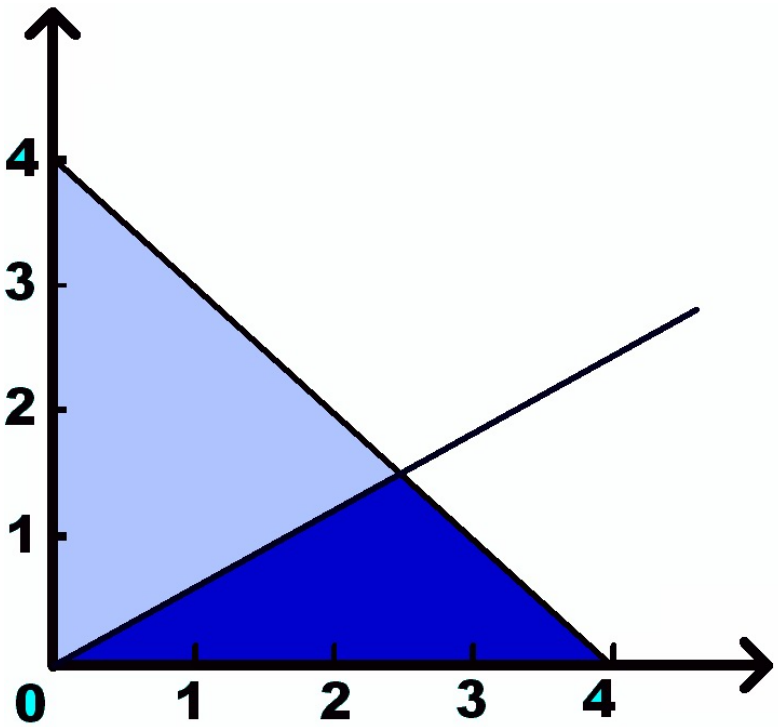
minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



Programmazione Lineare Intera

Esempio: Branch and Bound

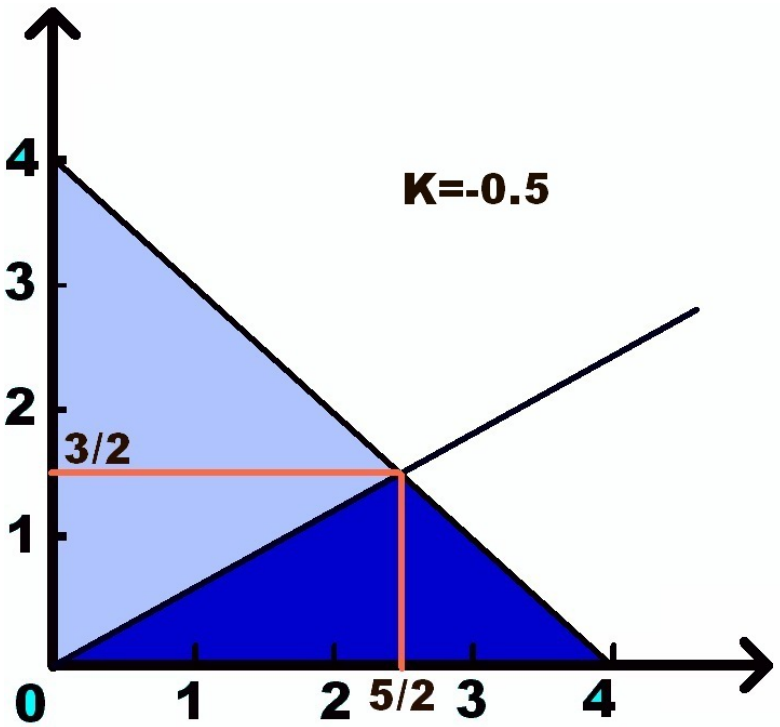
minimize $X - 2Y$ subject to
$X \geq 0, Y \geq 0, X + Y \leq 4, 3X - 5Y \geq 0, X, Y \in \mathbb{N}$.



Programmazione Lineare Intera

Esempio: Branch and Bound

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

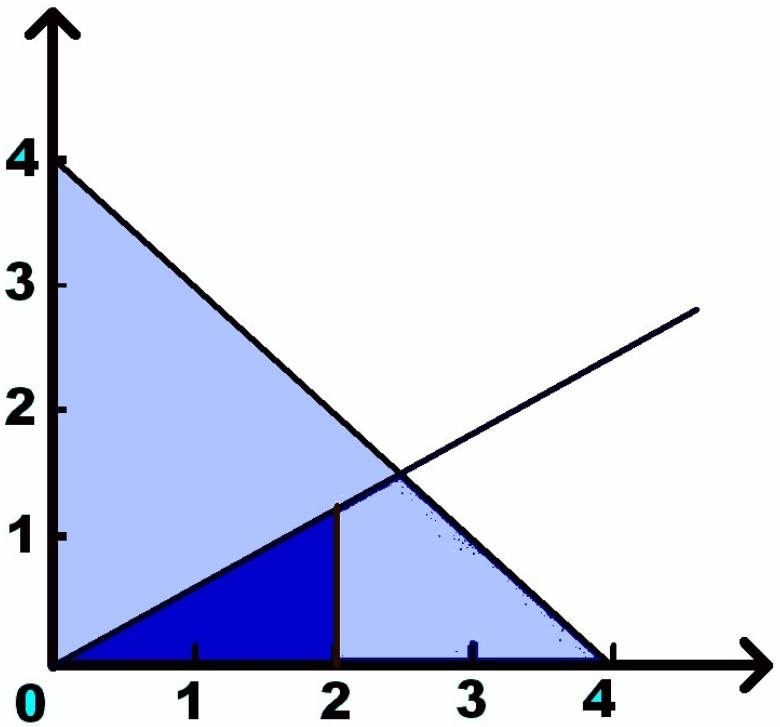


LP

Programmazione Lineare Intera

Esempio: Branch and Bound

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

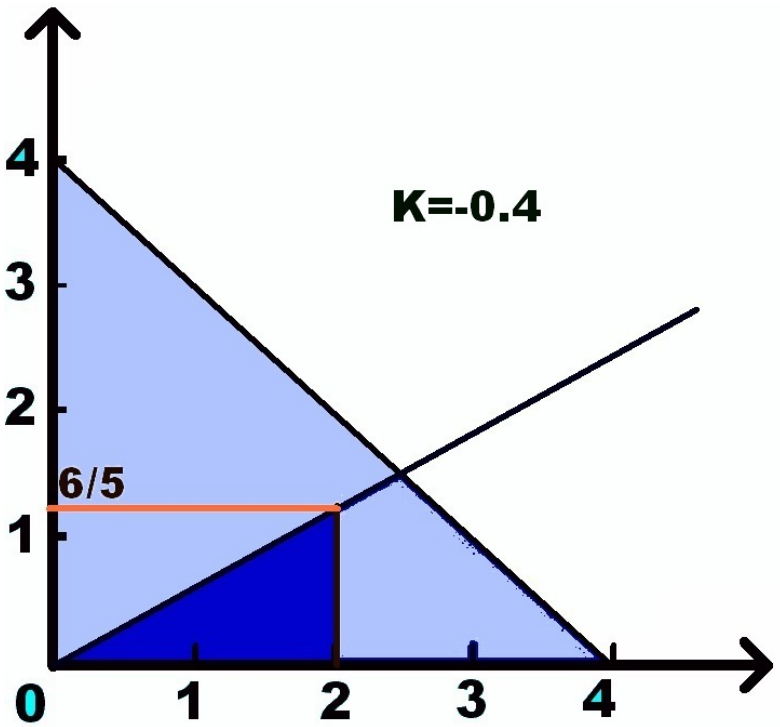


$X \leq 2$

Programmazione Lineare Intera

Esempio: Branch and Bound

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

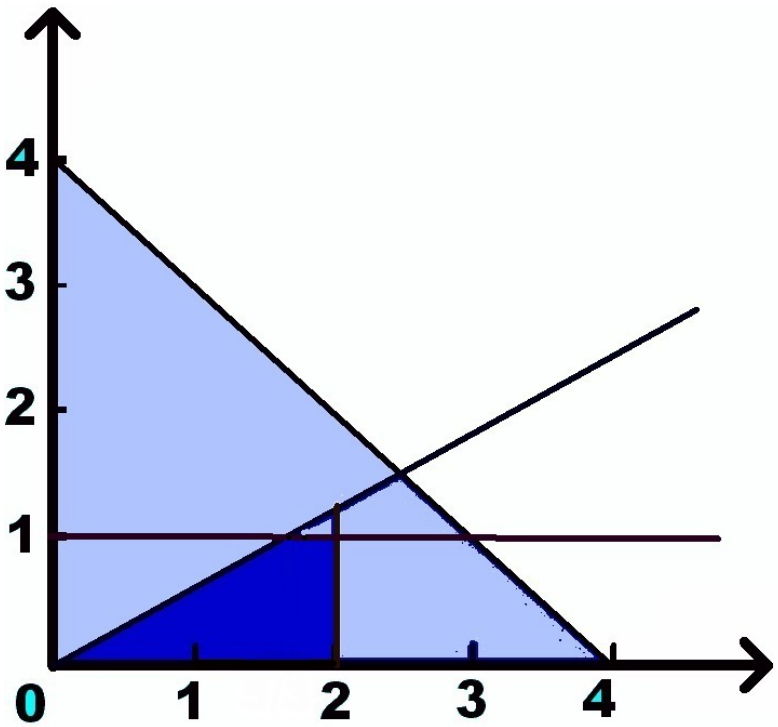


LP

Programmazione Lineare Intera

Esempio: Branch and Bound

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

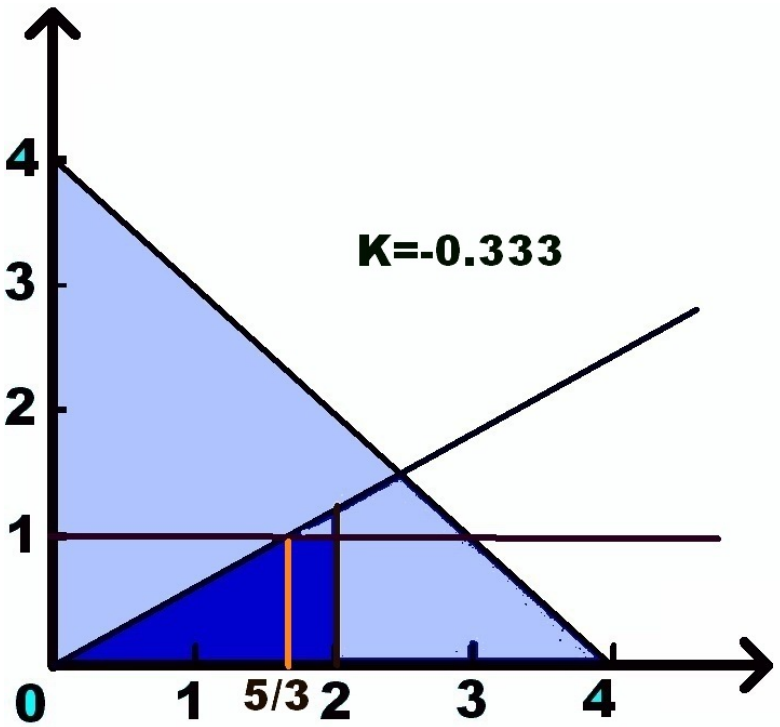


$X \leq 2, Y \leq 1$

Programmazione Lineare Intera

Esempio: Branch and Bound

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

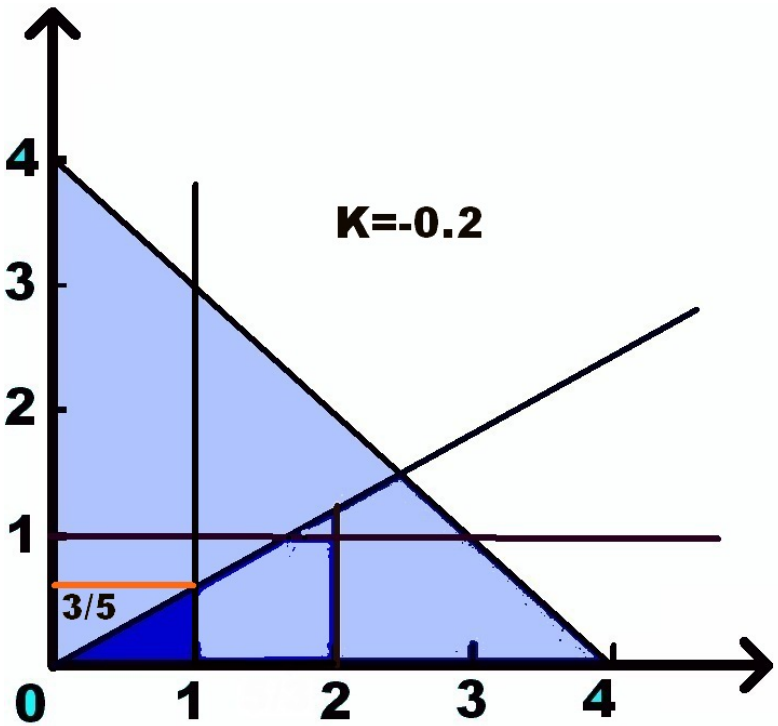


LP

Programmazione Lineare Intera

Esempio: Branch and Bound

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

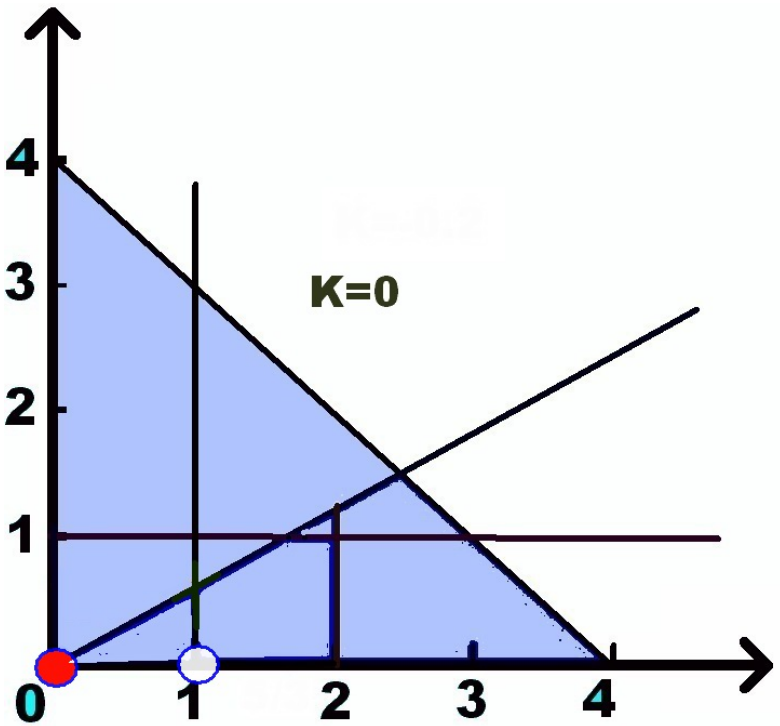


$X \leq 1, Y \leq 1$ –LP

Programmazione Lineare Intera

Esempio: Branch and Bound

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

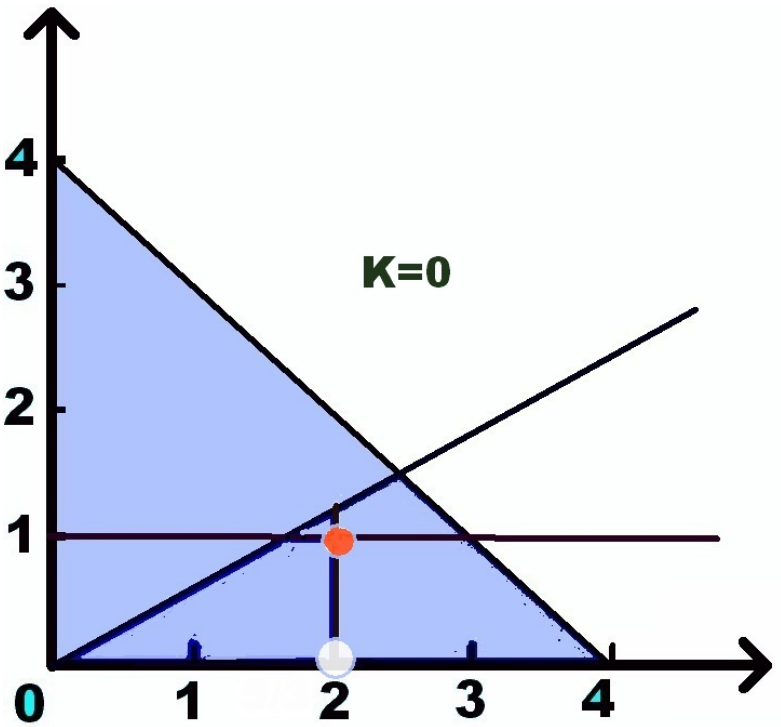


$X \leq 1, Y \leq 0$ –LP: feasible solution

Programmazione Lineare Intera

Esempio: Branch and Bound

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

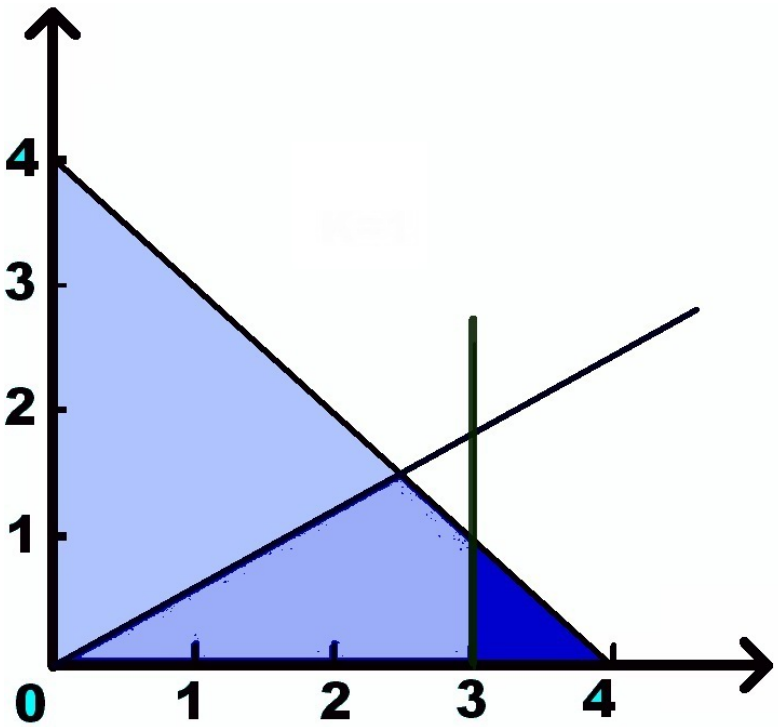


$X \geq 2, X \leq 2, Y \leq 1$ — LP: feasible solution

Programmazione Lineare Intera

Esempio: Branch and Bound

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

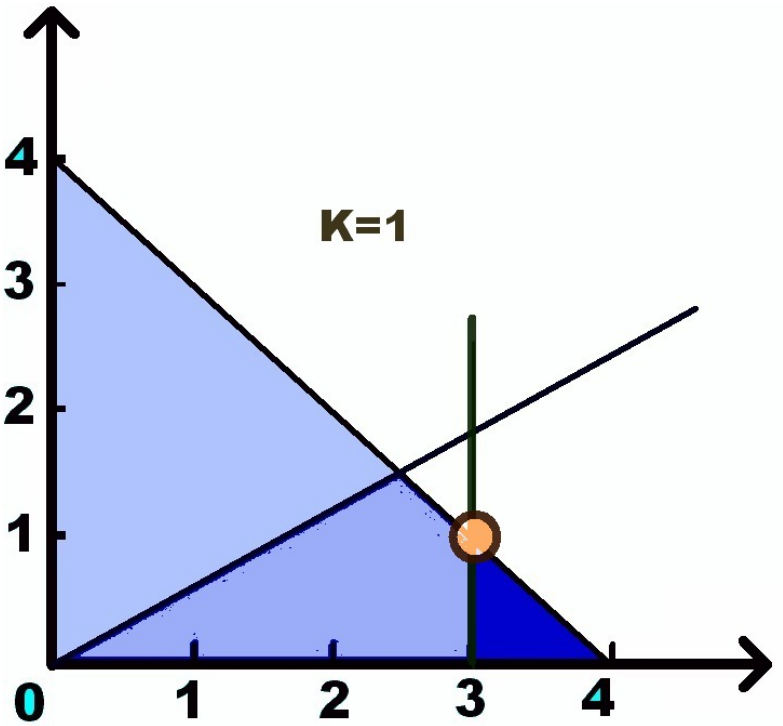


$X \geq 3$

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)