Principles of Constraint Programming

Krzysztof R. Apt

Chapter 8
Ricerca

Obbiettivi

- Introdurre gli **alberi di ricerca**,
- Discutere vari tipi di **alberi di labeling**, in particolare alberi per
  - forward checking,
  - look ahead parziale,
  - mantenere la consistenza sugli archi (MAC).
- Discutere vari **algoritmi di ricerca** per alberi di labeling.
- Discutere **algoritmi di ricerca** per problemi di ottimizzazione di vincoli
- Introdurre varie **euristiche** per gli algoritmi di ricerca.

Un utile slogan

Algoritmo di ricerca =
Albero di ricerca + Algoritmo di visita.

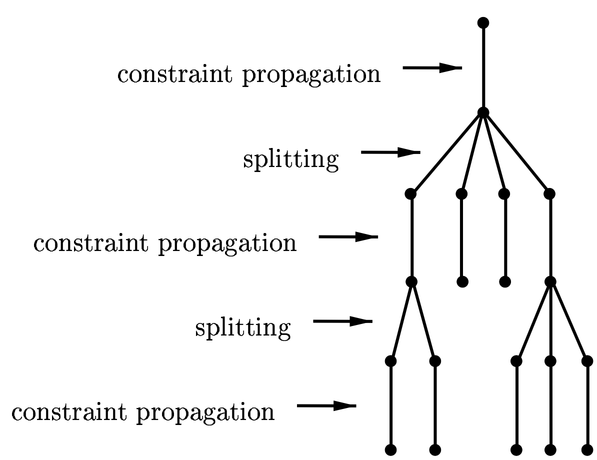
Alberi di ricerca

Consideriamo un CSP $\mathcal{P}$ con sequenza di variabili X.

Albero di ricerca per $\mathcal{P}$:
un albero finito tale che

- i suoi nodi sono CSP,
- la sua radice e' $\mathcal{P}$,
- i nodi ad un livello pari hanno esattamente un discendente diretto,
- se $\mathcal{P}_1, \ldots, \mathcal{P}_m$ sono discendenti diretti di $\mathcal{P}_0$, allora l'unione di $\mathcal{P}_1, \ldots, \mathcal{P}_m$ e' equivalente rispetto a X a $\mathcal{P}_0$.

Intuizione:



Alberi di Labeling

Alberi di ricerca specifici per CSP finiti.

- Lo splitting consiste di **labeling** del dominio di una variabile.
- La propagazione di vincoli consiste di un metodo di riduzione dei domini.

Alberi di Labeling Completi

Propagazione di vincoli **assente**.

Dati:
– un CSP $\mathcal{P}$ con domini non vuoti,
– $x_1, \ldots, x_n$ la sequenza di sue variabili ordinate linearmente da $\prec$.

Albero di labeling completo associato a $\mathcal{P}$ e $\prec$:

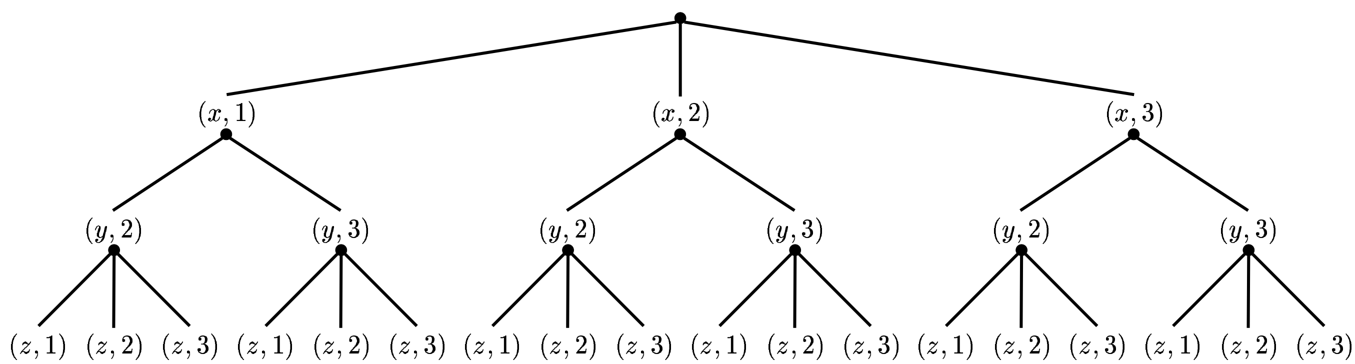
- i discendenti diretti della radice sono della forma (x_1, d),
- i dicendenti diretti di un nodo (x_j, d), dove $j \in [1..n-1]$, sono della forma (x_{j+1}, e),
- i suoi rami determinano tutte le istanziazioni con dominio $\{x_1, \ldots, x_n\}$.

Esempi

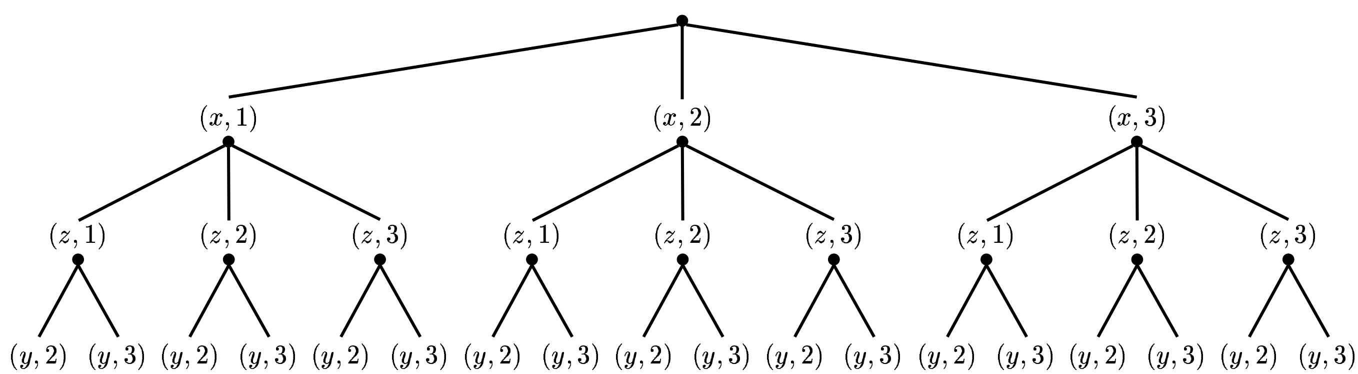
Consideriamo

$\langle x < y, y < z \,;\, x \in \{1,2,3\}, y \in \{2,3\}, z \in \{1,2,3\}\rangle$.

1. con l'ordinamento $x \prec y \prec z$.



2. con l'ordinamento $x \prec z \prec y$.



Grandezza degli alberi di labelling completi

Dati:
– un CSP con domini non vuoti,
– $x_1, \ldots, x_n$ la sequenza delle sue variabili ordinate linearmente da $\prec$.
– $D_1, \ldots, D_n$ domini delle variabili. Allora

- il numero di nodi nell'albero di labelling completo associato a $\prec$ e'

$$1 + \Sigma_{i=1}^{n}(\Pi_{j=1}^{i}|D_j|),$$

  $|A|$: cardinalita' dell'insieme A.

- L'albero di labeling completo ha il numero minimo di nodi se le variabili sono ordinate secondo la grandezza del loro dominio in ordine **crescente**.

Esempi

1.: Albero in **1.**

Cardinalita' dei domini: 3, 2, 3.

L'albero ha $1+3+3\cdot 2+3\cdot 2\cdot 3$, cioe' 28 nodi.

2.: Albero in **2.**

Cardinalita' dei domini: 3, 3, 2.

L'albero ha $1+3+3\cdot 3+3\cdot 3\cdot 2$, cioe' 31 nodi.

Entrambi gli alberi hanno lo stesso numero di foglie: 18.

Alberi di labeling ridotti

Un'istanziazione I e' **secondo l'ordinamento** $x_1, \ldots, x_n$ se il suo dominio e' $\{x_1, \ldots, x_j\}$ per qualche $j \in [1..n]$.

Dati:
– un CSP $\mathcal{P}$ con domini non vuoti,
– $x_1, \ldots, x_n$ la sequenza delle sue variabili ordinate linearmente da $\prec$.

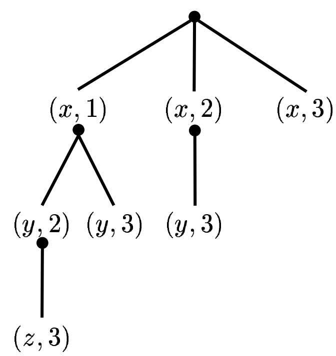
Albero di labelling ridotto associato a $\mathcal{P}$ e $\prec$:

- i discendenti diretti della radice sono della forma (x_1, d),
- i discendenti diretti di un nodo (x_j, d), dove $j \in [1..n-1]$, sono della forma (x_{j+1}, e),
- i suoi rami determinano tutte le istanziazioni consistenti secondo l'ordinamento $x_1, \ldots, x_n$.

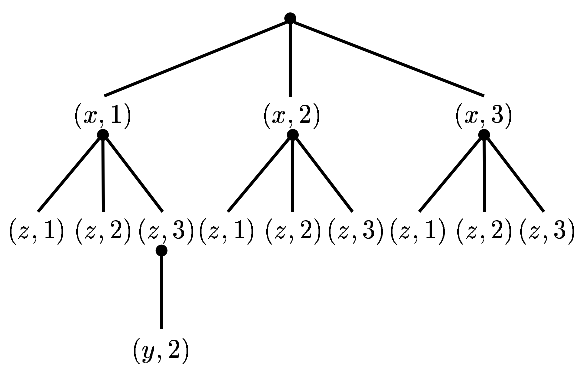
Esempi

Consideriamo
$\langle x < y, y < z \,;\, x \in \{1,2,3\}, y \in \{2,3\}, z \in \{1,2,3\}\rangle$.

1. con l'ordinamento $x \prec y \prec z$.



2. con l'ordinamento $x \prec z \prec y$.



Alberi di labelling ridotti possono avere un numero diverso di nodi e di foglie.

Alberi di Labeling con Propagazione di Vincoli

Dato:

$$\mathcal{P} := \langle \mathcal{C} \; ; \; x_1 \in D_1, \ldots, x_n \in D_n \rangle.$$

- Assumiamo la **propagazione di vincoli** *prop(i)* nella forma di una riduzioni di domini, dove $i \in [0..n-1]$.
- i determina la sequenza $x_{\mathbf{i+1}}, \ldots, x_n$ di variabili ai cui domini *prop(i)* e' applicata.
- Dati i domini della variabile corrente $E_1, \ldots, E_n$, la propagazione di vincoli *prop(i)* trasforma solo $E_{\mathbf{i+1}}, \ldots, E_n$.
- *prop(i)* dipende dai vincoli originali $\mathcal{C}$ di $\mathcal{P}$ e dai domini $E_1, \ldots, E_i$.

Alberi di labelling *prop*

Albero di labelling *prop* associato a $\mathcal{P}$:

- i suoi nodi sono sequenze di espressioni di dominio $x_1 \in E_1, \ldots, x_n \in E_n$,
- la sua radice e' $x_1 \in D_1, x_2 \in D_2, \ldots, x_n \in D_n$,
- nodo a livello pari $2i$ con $i \in [0..n]$:

  $x_1 \in \{d_1\}, \ldots, x_i \in \{d_i\}, x_{i+1} \in E_{i+1}, \ldots, x_n \in E_n$.

  Se $i = n$, questo nodo e' una foglia. Altrimenti, ha esattamente un discendente diretto, ottenuto usando *prop(i)*:

  $x_1 \in \{d_1\}, \ldots, x_i \in \{d_i\}, x_{i+1} \in E'_{i+1}, \ldots, x_n \in E'_n$

  dove $E'_j \subseteq E_j$ per $j \in [i+1..n]$
- nodo a livello dispari $2i+1$ con $i \in [0..n-1]$:

  $x_1 \in \{d_1\}, \ldots, x_i \in \{d_i\}, x_{i+1} \in E_{i+1}, \ldots, x_n \in E_n$.

  Se $E_j = \emptyset$ per qualche $j \in [i+1..n]$, questo nodo e' una foglia. Altrimenti ha discendenti diretti della forma

  $$x_1 \in \{d_1\}, \ldots, x_i \in \{d_i\}, x_{i+1} \in \{d\}, \qquad x_{i+2} \in E_{i+2}, \ldots, x_n \in E_n,$$

  per tutti i $d \in E_{i+1}$ tali che l'istanziazione $\{(x_1, d_1), \ldots, (x_i, d_i), (x_{i+1}, d)\}$ e' consistente.

Intuizione

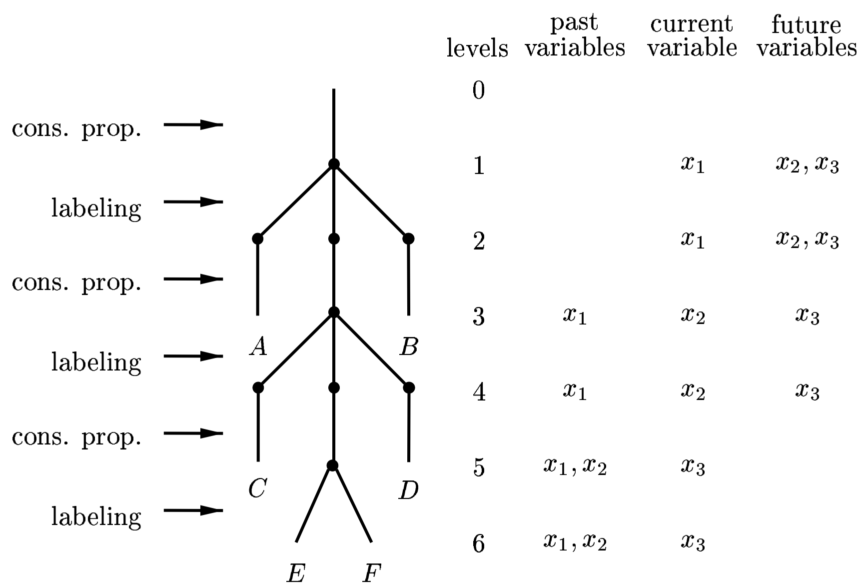
Dato: un nodo $x_1 \in E_1, \ldots, x_n \in E_n$ a livello $2i - 1$ or $2i$,

- se $i \in [2..n-1]$, chiamiamo $x_1, \ldots x_{i-1}$ le sue
  variabili passate,
- se $i \in [1..n]$, chiamiano x_i la sua **variabile corrente**, e
- se $i \in [0..n-1]$, chiamiamo $x_{i+1}, \ldots x_n$ le sue
  variabili future.

prop(i) modifica solo i domini delle variabili future.

Esempio di un albero di labelling *prop*

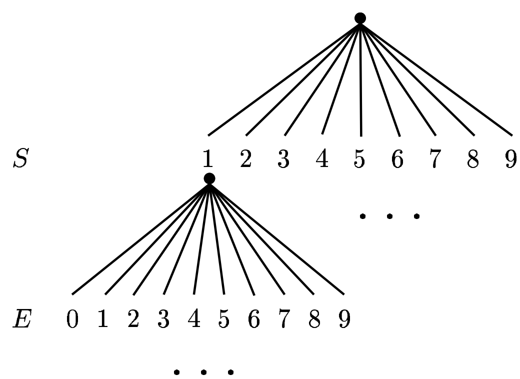
Consideriamo un CSP con tre variabili, x_1, x_2, x_3.



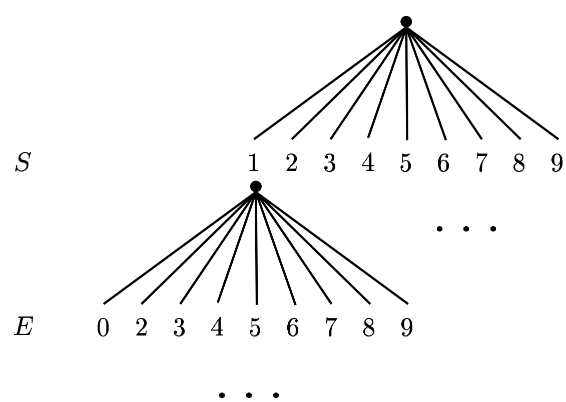
A, B, C e D sono nodi **falliti**.
E e F sono nodi con **successo**.

Esempio: $SEND + MORE = MONEY$

Albero di labelling completo:



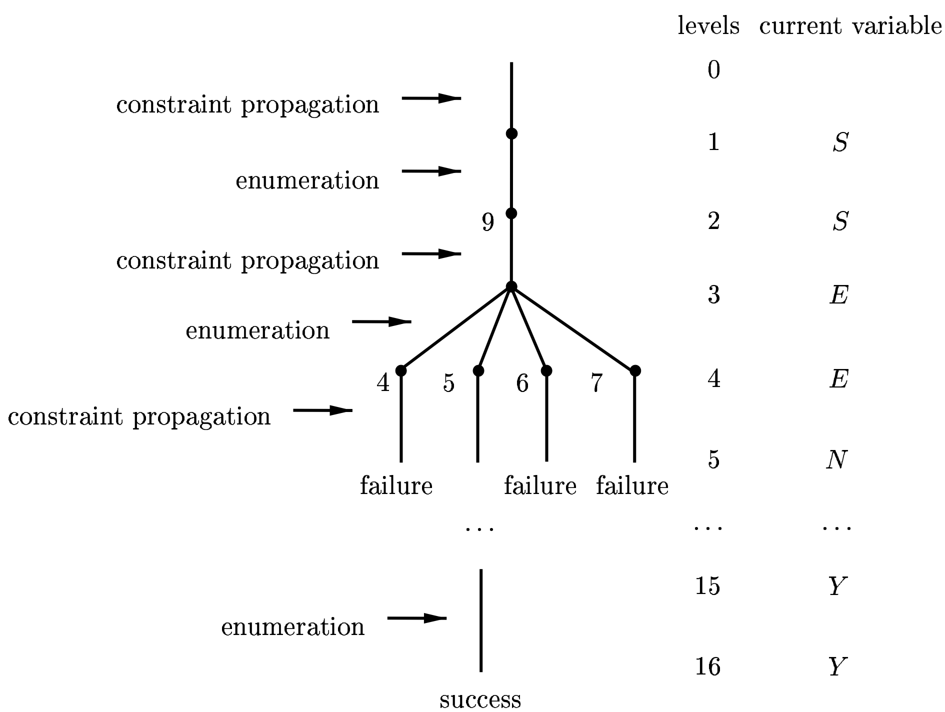
Albero di labelling ridotto:



SEND + MORE = MONEY

Usiamo come ***prop*** le regole per la riduzione dei domini per vincoli lineari del Capitolo 6.

Albero di labelling ***prop***:



Grazndezza degli alberi generati

Per $SEND + MORE = MONEY$:

- Albero completo.
  Numero totale di foglie: $9^2 \cdot 10^6 = 81000000$.
- Albero ridotto.
  Numero totale di foglie:
  $10 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4 - 2 \cdot (9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 \cdot 4) = 483840$.
  Guadagno: 99.4%
- Albero di labelling *prop*.
  Numero totale di foglie: 4.

Istanze degli alberi di labelling *prop*

- **forward checking**,
- **look ahead** parziale,
- **mantenere la consistenza sugli archi (MAC)**
  (detto anche "full look ahead").

Albero di ricerca per il forward checking

Ricordiamo la definizione di alberi di labelling *prop*:

- ogni nodo ad un livello **pari** $2i$ con $i \in [0..n]$ e' della forma

  $x_1 \in \{d_1\}, \ldots, x_i \in \{d_i\}, x_{i+1} \in E_{i+1}, \ldots, x_n \in E_n.$

  Se $i = n$, e' una foglia. Altrimenti, ha esattamente un discendente diretto, ottenuto usando *prop(i)*:

  $x_1 \in \{d_1\}, \ldots, x_i \in \{d_i\}, x_{i+1} \in E'_{i+1}, \ldots, x_n \in E'_n$

  dove $E'_j \subseteq E_j$ per $j \in [i+1..n]$.

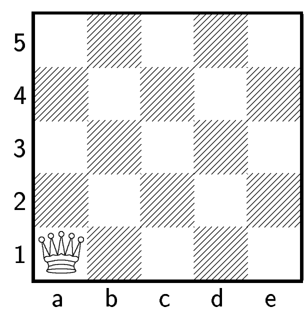
Definiamo

$$E'_j := \{e \in E_j \mid \{(x_1, d_1), \ldots, (x_i, d_i), (x_j, e)\} \text{ e' consistente}\}$$

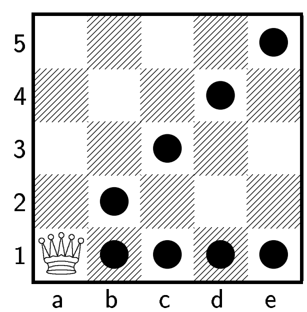
Esempio: problema delle 5 regine

Prendiamo il CSP **standardizzato** che corrisponde al problema delle 5 regine.
Interpretazione: le variabili x_1, x_2, x_3, x_4, x_5 corrispondono alle colonne `a,b,c,d,e`.

La prima regina e' piazzata in `a1`:



Effetto del **forward checking**:

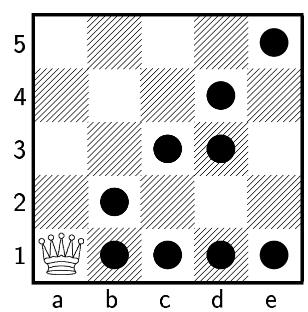


Albero di ricerca per il Look Ahead parziale

- Imponiamo il forward checking.
- Imponiamo la **consistenza direzionale sugli archi**, usando ad esempio l'algoritmo `DARC`.

Esempio: problema delle 5 regine.

Effetto del **look ahead parziale**:

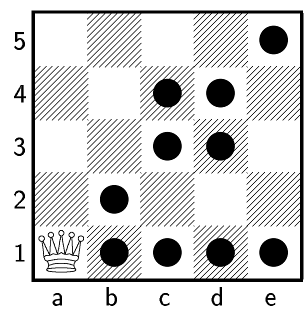


Albero di ricerca per MAC

- Imponiamo il forward checking.
- Imponiamo la **consistenza sugli archi**, ad esempio usando l'algoritmo `ARC`.

Esempio: problema delle 5 regine.

Effetto di **MAC**:



Algoritmi di ricerca per alberi di labelling

- Ricerca BACKTRACK-FREE
- Ricerca BACKTRACK-FREE con propagazione di vincoli,
- Ricerca con BACKTRACK,
- Ricerca con BACKTRACKcon propagazione di vincoli:
  - forward checking,
  - look ahead parziale, e
  - MAC.

Algoritmi di ricerca per problemi di ottimizzazione di vincoli:

- ricerca BRANCH AND BOUND,
- ricerca BRANCH AND BOUND CON PROPAGAZION DI VINCOLI.

Ricerca Backtrack-free

`cons(inst,j,d)` $\equiv$ "l'istanziazione $\{(x_1, \texttt{inst}[1]), \ldots, (x_{j-1}, \texttt{inst}[\texttt{j}-1]), (x_j, \texttt{d})\}$ e' consistente"

```
MODULE backtrack_free;
TYPE domains =  ARRAY [1..n] OF domain;
     instantiation = ARRAY [1..n] OF elements;
VAR inst: instantiation;
PROCEDURE backtrack_free(j: INTEGER; D: domains;
                         VAR success: BOOLEAN);
BEGIN
  WHILE D[j] <> {} AND NOT success DO
    choose d from D[j];
    D[j] := D[j] - {d};
    IF cons(inst,j,d) THEN
      inst[j] := d;
      success := (j=n);
      IF NOT success THEN
        j := j+1
      END
    END
  END
END backtrack_free;
BEGIN
  success := FALSE;
  backtrack_free(1,D,success)
END backtrack_free;
```

Ricerca Backtrack-free con Propagazione di Vincoli

```
MODULE backtrack_free_prop;
TYPE domains =  ARRAY [1..n] OF domain;
     instantiation = ARRAY [1..n] OF elements;
VAR inst: instantiation;
    failure: BOOLEAN;
PROCEDURE backtrack_free_prop(j: INTEGER; D: domains;
                              VAR success: BOOLEAN);
BEGIN
  WHILE D[j] <> {} AND NOT success DO
    choose d from D[j];
    D[j] := D[j] - {d};
    IF cons(inst,j,d) THEN
      inst[j] := d;
      success := (j=n);
      IF NOT success THEN
        prop(j,D,failure);
        IF NOT failure THEN
          j := j+1
        END
      END
    END
  END
END backtrack_free_prop;
BEGIN
  success := FALSE;
  prop(0,D,failure);
  IF NOT failure THEN backtrack_free_prop(1,D,success) END
END backtrack_free_prop;
```

Ricerca con Backtracking

```
MODULE backtrack;
TYPE domains =  ARRAY [1..n] OF domain;
     instantiation = ARRAY [1..n] OF elements;
VAR inst: instantiation;
PROCEDURE backtrack(j: INTEGER; D: domains;
                    VAR success: BOOLEAN);
BEGIN
  WHILE D[j] <> {} AND NOT success DO
    choose d from D[j];
    D[j] := D[j] - {d};
    IF cons(inst,j,d) THEN
      inst[j] := d;
      success := (j=n);
      IF NOT success THEN
        backtrack(j+1,D,success)
      END
    END
  END
END backtrack;
BEGIN
  success := FALSE;
  backtrack(1,D,success)
END backtrack;
```

Backtracking con Propagazione di Vincoli

```
MODULE backtrack_prop;
TYPE domains =  ARRAY [1..n] OF domain;
     instantiation = ARRAY [1..n] OF elements;
VAR inst: instantiation;
    failure: BOOLEAN;
PROCEDURE backtrack_prop(j: INTEGER; D: domains;
                         VAR success: BOOLEAN);
BEGIN
  WHILE D[j] <> {} AND NOT success DO
    choose d from D[j];
    D[j] := D[j] - {d};
    IF cons(inst,j,d) THEN
      inst[j] := d;
      success := (j=n);
      IF NOT success THEN
        prop(j,D,failure);
        IF NOT failure THEN
          backtrack_prop(j+1,D,success)
        END
      END
    END
  END
END backtrack_prop;
BEGIN
  success := FALSE;
  prop(0,D,failure);
  IF NOT failure THEN backtrack_prop(1,D,success) END
END backtrack_prop;
```

Forward Checking

```
PROCEDURE revise(j,k: INTEGER; VAR D: domains);
BEGIN
  D[k] := {d ∈ D[k] | {(x_1,inst[1]),...,(x_j,inst[j]),(x_k,d)}
                              e' una istanziazione consistente}
END revise;


PROCEDURE prop(j: INTEGER; VAR D: domains;
               VAR failure: BOOLEAN);
VAR k: INTEGER;
BEGIN
  failure := FALSE;
  k := j+1;
  WHILE k <> n+1 AND NOT failure DO
    revise(j,k,D);
    failure := (D[k] = {});
    k := k+1
  END
END prop;
```

Look Ahead parziale

```
PROCEDURE prop(j: INTEGER; VAR D: domains;
               VAR failure: BOOLEAN);
VAR k: INTEGER;
BEGIN
  failure := FALSE;
  k := j+1;
  WHILE k <> n+1 AND NOT failure DO
    revise(j,k,D);
    failure := (D[k] = {});
    k := k+1
  END;
  IF NOT failure THEN
    darc(j+1,D,failure)
  END
END prop;
```

MAC (Full Look Ahead)

```
PROCEDURE prop(j: INTEGER; VAR D: domains;
               VAR failure: BOOLEAN);
...
  IF NOT failure THEN
    arc(j+1,D,failure)
  END
END prop;
```

Cercare tutte le soluzioni

```
MODULE backtrack_all;
TYPE domains =  ARRAY [1..n] OF domain;
     instantiation = ARRAY [1..n] OF elements;
VAR inst: instantiation;
PROCEDURE backtrack_all(j: INTEGER; D: domains);
BEGIN
  WHILE D[j] <> {} DO
    choose d from D[j];
    D[j] := D[j] - {d};
    IF cons(inst,j,d) THEN
      inst[j] := d;
      IF j=n THEN
        PRINT(inst)
      ELSE
        backtrack_all(j+1,D)
      END
    END
  END
END backtrack_all;
BEGIN
  backtrack_all(1,D)
END backtrack_all;
```

Problemi di ottimizzazione di vincoli finiti

- $\mathcal{P} := \langle \mathcal{C} \ ; \ x_1 \in D_1, \ldots, x_n \in D_n \rangle$,
- $obj : Sol \to \mathcal{R}$
  dall'insieme Sol di tutte le soluzioni di $\mathcal{P}$ a $\mathcal{R}$.
- Funzione euristica

$$h : \mathcal{P}(D_1) \times \ldots \times \mathcal{P}(D_n) \to \mathcal{R}$$

Monotonicita' Se $\bar{E}_1 \subseteq \bar{E}_2$, allora $h(\bar{E}_1) \leq h(\bar{E}_2)$,
Limite $obj(d_1, \ldots, d_n) \leq h(\{d_1\}, \ldots, \{d_n\})$.

Nei programmi

```
PROCEDURE obj(inst: instantiation): REAL;

PROCEDURE h(inst: instantiation; j: INTEGER;
            D: domains): REAL;
```

`h(inst,j,D)` ritorna il valore di h su

$$(\{\texttt{inst}[1]\}, \ldots, \{\texttt{inst}[j]\}, \texttt{D}[\texttt{j}+1], \ldots, \texttt{D}[\texttt{n}]).$$

Branch and Bound

```
MODULE branch_and_bound;
TYPE domains =  ARRAY [1..n] OF domain;
     instantiation = ARRAY [1..n] OF elements;
VAR inst: instantiation;
PROCEDURE branch_and_bound(j: INTEGER; D: domains;
    VAR solution: instantiation; VAR bound: REAL);
BEGIN
  WHILE D[j] <> {} DO
    choose d from D[j];
    D[j] := D[j] - {d};
    IF cons(inst,j,d) THEN
      inst[j] := d;
      IF j=n THEN
        IF obj(inst) > bound THEN
          bound := obj(inst); solution := inst
        END
      ELSE
        IF h(inst,j,D) > bound THEN
          branch_and_bound(j+1,D,solution,bound)
        END
      END
    END
  END
END branch_and_bound;
BEGIN
  solution := NIL; bound := -infinity;
  branch_and_bound(1,D,solution,bound)
  END
END branch_and_bound;
```

Branch and Bound con Propagazione di Vincoli

```
MODULE branch_and_bound_prop;
TYPE domains =  ARRAY [1..n] OF domain;
     instantiation = ARRAY [1..n] OF elements;
VAR inst: instantiation;
    failure: BOOLEAN;
PROCEDURE branch_and_bound_prop(j: INTEGER; D: domains;
         VAR solution: instantiation; VAR bound: REAL);
BEGIN
  WHILE D[j] <> {} DO
    choose d from D[j];
    D[j] := D[j] - {d};
    IF cons(inst,j,d) THEN
      inst[j] := d;
      IF j=n THEN
        IF obj(inst) > bound THEN
          bound := obj(inst); solution := inst
        END
      ELSE
        prop(j,D,failure);
        IF NOT failure THEN
          IF h(inst,j,D) > bound THEN
            branch_and_bound_prop(j+1,D,solution,bound)
          END
        END
      END
    END
  END
END branch_and_bound_prop;
```

```
BEGIN
  solution := NIL;
  bound := -infinity;
  prop(0,D,failure);
  IF NOT failure THEN
    branch_and_bound_prop(1,D,solution,bound)
  END
END branch_and_bound_prop;
```

Euristiche per Algoritmi di Ricerca

Selezione di una variabile

- Selezionare una variabile con il dominio piu' piccolo.
- Selezionare una delle variabili *piu' vincolate*,
- (per domini numerici)

  Selezionare una variabile con la piu' piccola differenza tra i suoi valori massimo e minimo.

Selezione di un valore

- Seleziona un valore per cui la funzione euristica da' il risultato piu' alto.
- Seleziona il valore piu' piccolo,
- Seleziona il valore piu' grande,
- Seleziona il valore mediano.