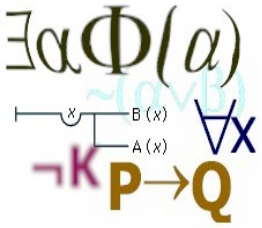
Πανεπιστήμιο Μακεδονίας
Τμ. Εφαρμοσμένης Πληροφορικής

ECLiPSe Constraints II



Δομή

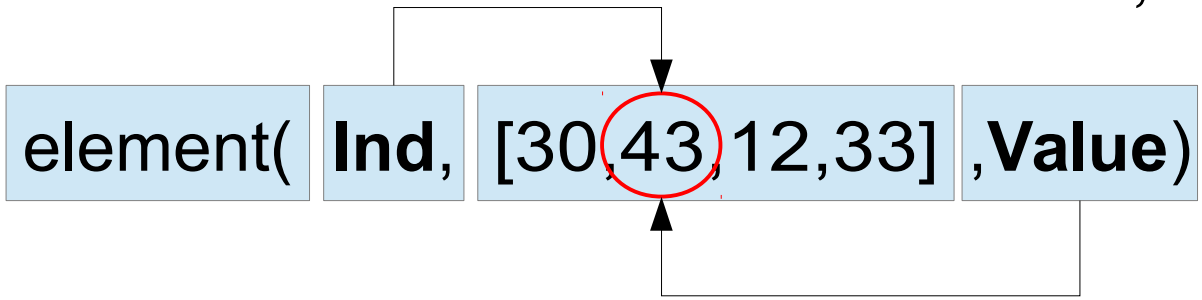
- Μοντελοποίηση με element/3 περιορισμό.
- Αναζήτηση με τον αλγόριθμο Branch and Bound.

Element/3

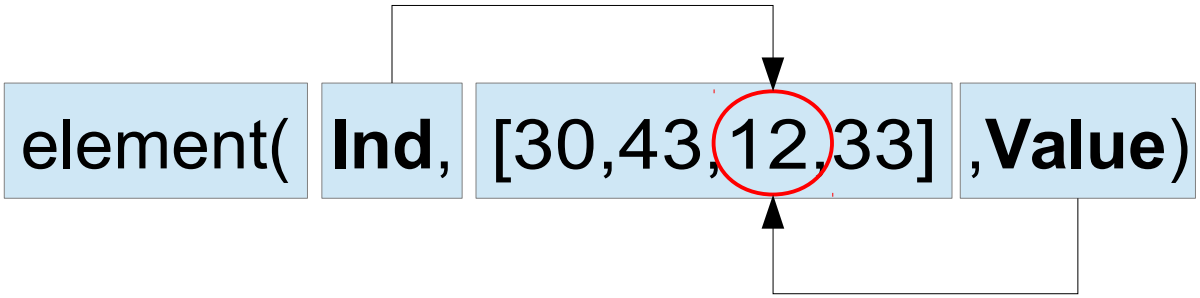
- **element(Index,List,Value)**
  - Η τιμή Value είναι η Index-th τιμή της λίστας List.
- Μοντελοποίηση περιορισμών πινάκων (tabular constraints).

Γραφική Απεικόνιση

Ind = 2, Value = 43



Ind = 3, Value = 12



Παράδειγμα: Ανάθεση εργασιών

- Σε ένα έργο πληροφορικής υπάρχουν 10 **εργασίες** και 10 **ομάδες**.
- Κάθε ομάδα *είναι ικανή* να εκτελέσει **ένα υποσύνολο** από τις διαθέσιμες εργασίες.
- Σε κάθε ομάδα πρέπει να ανατεθεί **μια** εργασία.
- Οι ομάδες εκτελούν τις εργασίες με διαφορετικό κόστος.
  - Δηλαδή αν η ομάδα 1 εκτελέσει την εργασία 5 τότε το κόστος είναι 10, ενώ αν η ομάδα 2 εκτελέσει το εργασία 5 το κόστος είναι 60.
- *Ποια είναι η ανάθεση εργασιών στις ομάδες και ποιο είναι το κόστος;*

Πίνακας με τις εργασίες και το κόστος των εργασιών

- Για παράδειγμα:

Εργασία / Ομάδα	1	2	3	4	5	6	7	8	9	10
1	10	-	30	40	20	10	-	-	-	-
2	90	-	20	-	70	140	-	-	-	-
3	-	-	-	80	10	-	20	60	-	30
4										
5										

Μοντελοποίηση

- Κάθε ομάδα αναπαρίσταται από μια μεταβλητή.
  - Πεδίο κάθε μεταβλητής: οι εργασίες που μπορεί να εκτελέσει.
  - Άρα για παράδειγμα, W1 :: [4,1,3,5,6]
- Για κάθε ομάδα υπάρχει ένα κόστος που συνδέεται με την εργασία που θα αναλάβει κάθε ομάδα (μεταβλητή C1).
  - Πεδίο της μεταβλητής τα διάφορα κόστη τα οποία μπορεί να πάρει η μεταβλητή C1.
- **Πώς συνδέεται η εργασία που θα επιλεγεί με το κόστος της?**

Μοντελοποίηση με element/3

- Περιορισμός element
  - element(Ind,List,Value)
- Έστω ότι η ομάδα 1 μπορεί να εκτελέσει τις εργασίες **[4,1,3,5,6]** με αντίστοιχα κόστη **[40,10,30,20,10]**:
- Το παραπάνω μοντελοποιείται ως:

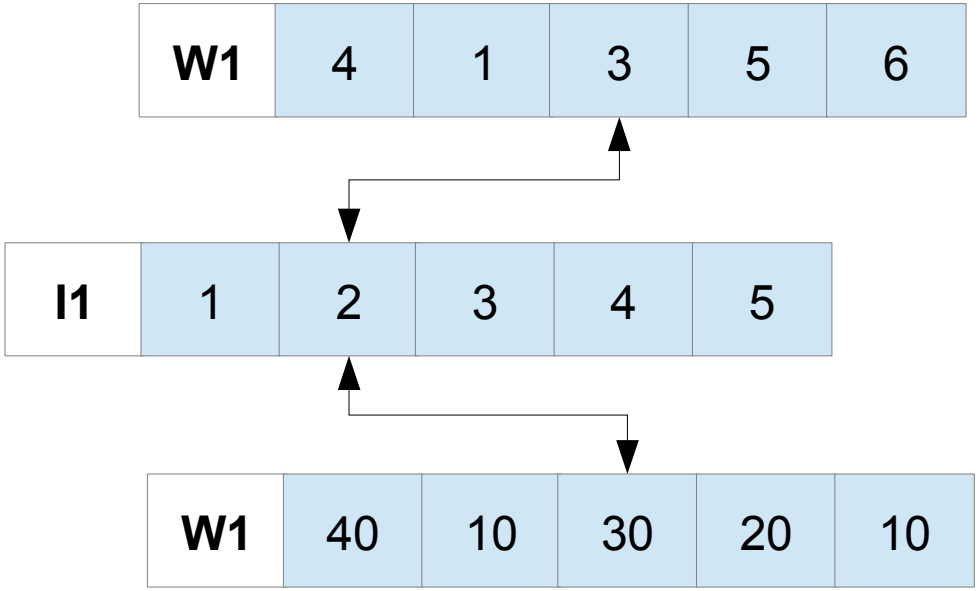
```
element(I1,[4,1,3,5,6],W1),
element(I1,[40,10,30,20,10],C1),
```

- “Κάθε ομάδα μπορεί να εκτελέσει μόνο μια εργασία”

```
alldifferent([W1,W2,...,W10])
```

Γραφική Απεικόνιση

element(I1,[4,1,3,5,6],W1),
element(I1,[40,10,30,20,10],C1),



Λύση

```
:-use_module(library(ic)).

solve(Workers,Cost):-
        Workers = [W1,W2,W3,W4,W5,W6,W7,W8,W9,W10],
        element(I1,[4,1,3,5,6],W1),  element(I1,[40,10,30,20,10],C1),
        element(I2,[6,3,5,2,4],W2),  element(I2,[140,20,70,10,90],C2),
        ...
        element(I10,[1,2,7,9,3],W10), element(I10,[20,20,30,40,50],C10),
        alldifferent(Workers),
        Cost #= C1 + C2 + C3 + C4 + C5 + C6 + C7 + C8 + C9 + C10,
        labeling(Workers).
```

Για το πρόβλημα της ανάθεσης εργασιών

- Εναλλακτικά για κάθε εργασία και ομάδα έχουμε

  W1 :: [1,5,7,9], C1::[30,60,20,10],

  (W1 #=1)=> (C1#=30),

  (C1#=30) => (W1#=1),

  ...

  (W1 #=9)=> (C1#=10),

  (C1#=10) => (W1#=9),

  ...

Πρόβλημα

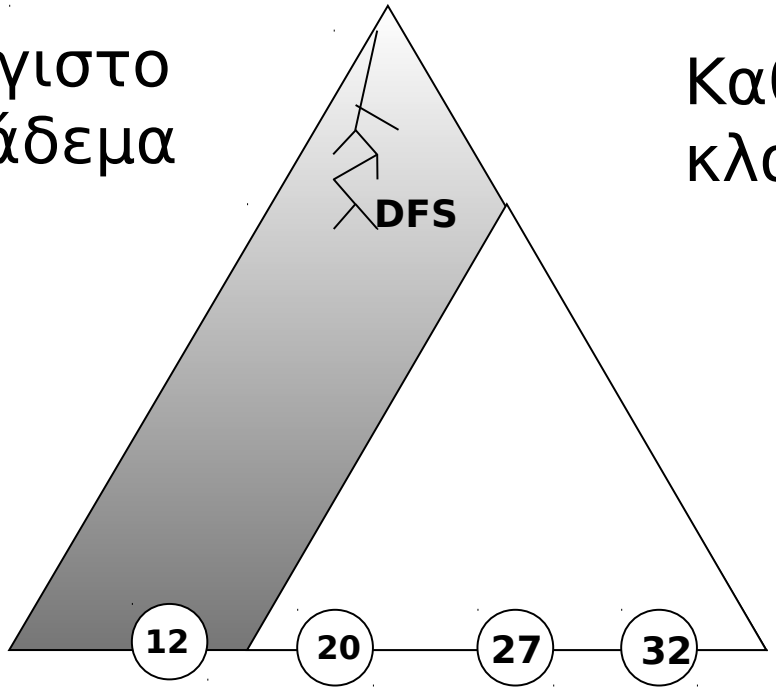
- Είναι σχετικά απλό να βρεθεί μια οποιαδήποτε ανάθεση.
- Απαιτείται εκείνη που θα δώσει την καλύτερη λύση.
  - Λύση με χαμηλότερο κόστος.
- Βελτιστοποίηση
  - Κλασική απαίτηση σε πλήθος προβλημάτων
  - Αλγόριθμος branch and bound

Αλγόριθμος BB

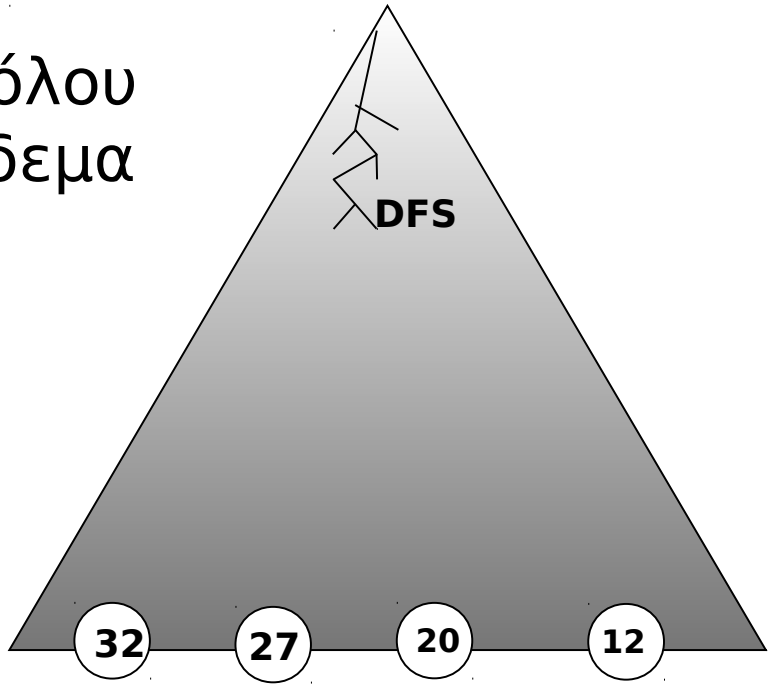
- Εξαντλητική αναζήτηση στο χώρο των λύσεων, ώστε να βρεθεί η καλύτερη λύση.
- Διαδικασία:
  - Βρες μια λύση χρησιμοποιώντας ένα τυφλό αλγόριθμο αναζήτησης. Αποθήκευσε το κόστος της (καλύτερη λύση).
  - Συνέχισε την αναζήτηση λύσης, "**κλαδεύοντας**" κάθε **μερική λύση** που το κόστος της ξεπερνά το κόστος της καλύτερης.
  - Αν βρεθεί πλήρης λύση με κόστος **μικρότερο** εκείνου της **μέχρι τώρα καλύτερης**, τότε αποθήκευσε την σαν την καλύτερη.
  - Συνέχισε μέχρι να εξαντληθεί ο χώρος αναζήτησης.

Καλύτερη και χειρότερη περίπτωση για τον BB

Μέγιστο κλάδεμα



Καθόλου κλαδεμα



Βιβλιοθήκη branch_and_bound

- Χρήση της βιβλιοθήκης
  - :-use_module(library(branch_and_bound)).
- Σημαντικότερο Κατηγόρημα
  - bb_min(+Goal, ?Cost, ?Options)
  - Βρίσκει βέλτιστη λύση χρησιμοποιώντας τον αλγόριθμο branch and bound.
  - **Goal**: το κατηγόρημα αναζήτησης
  - **Cost**: η τιμή που πρέπει να ελαχιστοποιηθεί.
  - **Options**: διάφορες επιλογές σχετικές με τον αλγόριθμο αναζήτησης

BB Options

- bb_options{***options***}
  - strategy: continue, restart/step, dichotomic
  - from: κάτω όριο και to: άνω όριο
  - delta:
  - timeout:
  - …
- Εξ' ορισμού τιμές
  bb_options{strategy:dichotomic, timeout:60}

BB Βιβλιοθήκη

- **bb_min/6:**
  - Παρόμοιο κατηγόρημα με το bb_min/3
- **minimize(+Goal, ?Cost)**
  - κατηγόρημα που ελαχιστοποιεί την τιμή της μεταβλητής Cost, εκτελώντας το Goal.

Προσέγγιση με βελτιστοποίηση

```
:-use_module(library(ic)).
:-use_module(library(branch_and_bound)).
solve(Workers,Cost):-
    Workers = [W1,W2,W3,W4,W5,W6,W7,W8,W9,W10],
        element(I1,[4,1,3,5,6],W1),  element(I1,[40,10,30,20,10],C1),
        ...
        element(I10,[1,2,7,9,3],W10), element(I10,
[20,20,30,40,50],C10),
        alldifferent(Workers),
        Cost #= C1 + C2 + C3 + C4 + C5 + C6 + C7 + C8 + C9 + C10,
        bb_min(labeling(Workers),Cost,_).
```

Κώδικας

```
:-use_module(library(ic)).
:-use_module(library(branch_and_bound)).
solve(Workers,Cost):-
    Workers = [W1,W2,W3,W4,W5,W6,W7,W8,W9,W10],
        element(I1,[4,1,3,5,6],W1),  element(I1,[40,10,30,20,10],C1),
        element(I2,[6,3,5,2,4],W2),  element(I2,[140,20,70,10,90],C2),
        element(I3,[8,4,5,7,10],W3), element(I3,[60,80,10,20,30],C3),
        element(I4,[3,7,8,9,1],W4), element(I4,[30,40,10,70,10],C4),
        element(I5,[7,1,5,6,4],W5), element(I5,[40,10,30,20,10],C5),
        element(I6,[8,4,7,9,5],W6),element(I6,[20,100,130,220,50],C6),
        element(I7,[5,6,7,4,10],W7), element(I7,[30,30,30,20,10],C7),
        element(I8,[2,6,10,8,3],W8), element(I8,[50,40,20,10,60],C8),
        element(I9,[1,3,10,9,6],W9), element(I9,[50,40,10,20,30],C9),
        element(I10,[1,2,7,9,3],W10), element(I10,[20,20,30,40,50],C10),
        alldifferent(Workers),
        Cost #= C1 + C2 + C3 + C4 + C5 + C6 + C7 + C8 + C9 + C10,
        bb_min(labeling(Workers),Cost,_).
```

Άλλες Βιβλιοθήκες

- :-lib(ic_global).
- Πλήθος περιορισμών.
- Καλύτερη υλοποίηση **alldifferent/1**.
- **alldifferent(Vars, Capacity)**
  - Κάθε τιμή μπορεί να εμφανιστεί Capacity φορες.
- **minlist(List,Min)**
- **maxlist(List,Max)**
  - Μέγιστο και ελάχιστο στοιχείο μιας λίστας.
- **sumlist(List,Sum)**
  - Άθροισμα της λίστας.

ic_global (συν.)

- **ordered(Relation,List)**
  - Επιβάλει μια σχέση διάταξης στις μεταβλητές της λίστας.
  - [X,Y] #:: [1..10], ordered(<,[X,Y]).
- **ordered_sum(List,Sum)**
  - Επιβάλει την σχέση “=<” και θέτει το άθροισμα στη μεταβλητή Sum

ic_global (συν.)

- **occurrences(Value,List,N)**
  - Η τιμή της μεταβλητής value, εμφανίζεται στη λίστα ακριβώς N φορές.
- **atmost(N,List,Value)**
  - το πολύ N μεταβλητές από την λίστα List, έχουν τιμή Value.

Άλλες Βιβλιοθήκες

- Υπάρχει πλήθος βιβλιοθηκών και άλλων περιορισμών.