Προβλήματα Ικανοποίησης Περιορισμών

- Προβλήματα ικανοποίησης περιορισμών (constraint satisfaction problems - CSP)
- Αλγόριθμοι υπαναχώρησης
- Ευρετικοί αλγόριθμοι
- Αλγόριθμοι τοπικής αναζήτησης
- Δομή ενός προβλήματος και δυσκολία επίλυσης του

Προβλήματα Αναζήτησης

Ο φορμαλισμός των **προβλημάτων αναζήτησης** που μελετήσαμε στις προηγούμενες διαλέξεις είναι ένα πολύ ισχυρό εργαλείο για μοντελοποίηση προβλημάτων που εξαρτάται από την έννοια της **κατάστασης**.

Για ένα αλγόριθμο αναζήτησης, η κατάσταση είναι ένα **μαύρο κουτί** με άγνωστη εσωτερική δομή.

Μια κατάσταση μπορεί να αναπαρασταθεί από μια **δομή δεδομένων που εξαρτάται από το δοσμένο πρόβλημα** και μόνο εξειδικευμένες συναρτήσεις μπορούν να έχουν πρόσβαση σε αυτή: Action, Goal-Test κλπ.

Προβλήματα Ικανοποίησης Περιορισμών

Τα **προβλήματα ικανοποίησης περιορισμών (constraint satisfaction problems - CSP)** είναι προβλήματα αναζήτησης που έχουν **απλή δομή** και επιδέχονται μια **απλή τυπική αναπαράσταση.**

Για τα CSP, έχουμε αλγόριθμους αναζήτησης που εκμεταλλεύονται αυτή την απλή αναπαράσταση και χρησιμοποιούν **γενικούς ευρετικούς μηχανισμούς** (και όχι ειδικούς για το συγκεκριμένο πρόβλημα) για να επιτύχουν την επίλυση μεγάλων προβλημάτων.

Η απλή δομή των CSP μας επιτρέπει, επίσης, να ορίσουμε μεθόδους **αποσύνθεσης προβλημάτων (problem decomposition)** και μας προσφέρει μια βαθύτερη κατανόηση της σχέσης ανάμεσα στη δομή ενός προβλήματος και τη δυσκολία επίλυσής του.

Προβλήματα Ικανοποίησης Περιορισμών - Ορισμοί

Ένα **πρόβλημα ικανοποίησης περιορισμών (CSP)** ορίζεται από:

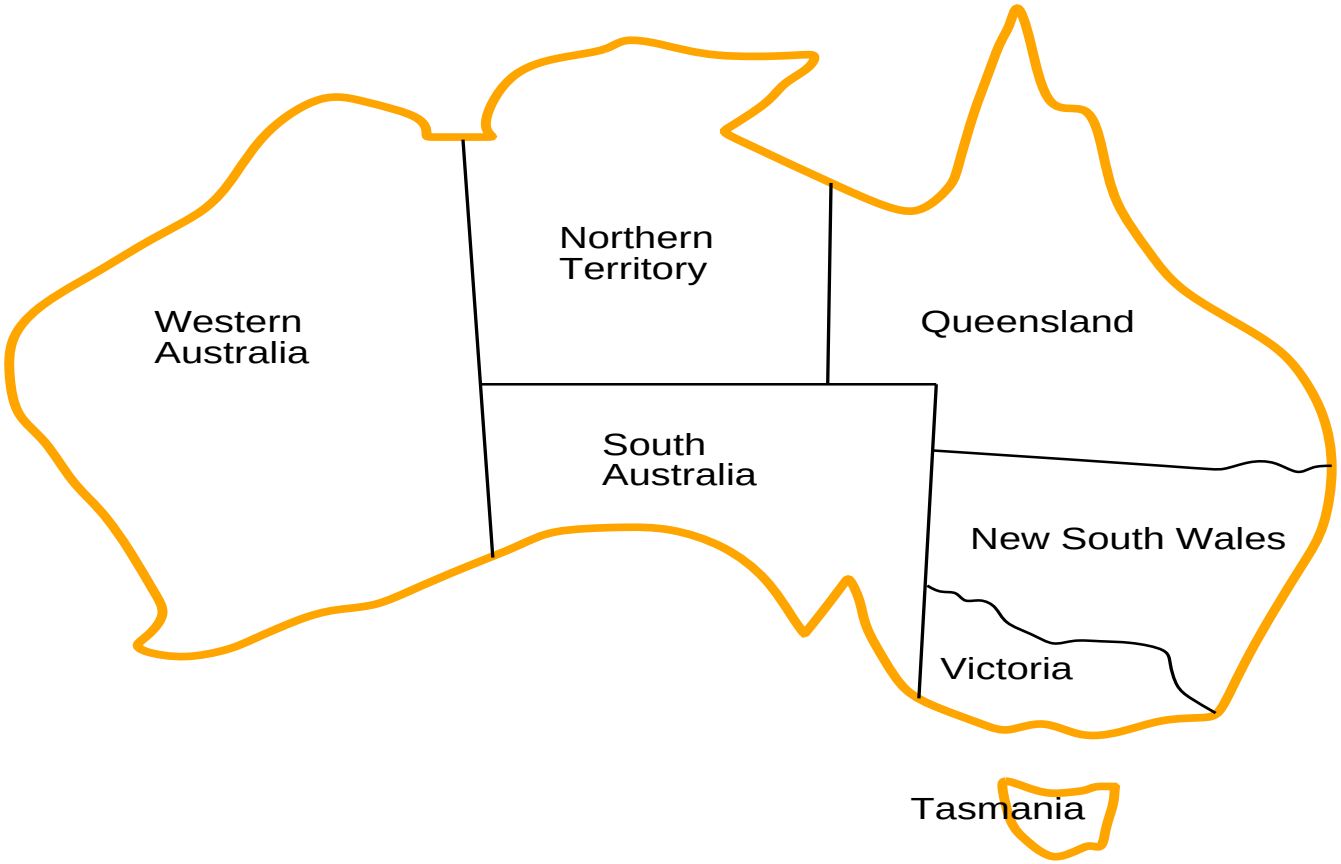
- Ένα σύνολο από **μεταβλητές** $X_1, \ldots, X_n$. Κάθε μεταβλητή X_i έχει ένα μη κενό **πεδίο (domain)** δυνατών τιμών D_i.
- Ένα σύνολο από **περιορισμούς** $C_1, \ldots, C_m$. Ένας περιορισμός καθορίζει τους επιτρεπτούς συνδυασμούς τιμών για ένα υποσύνολο του συνόλου των μεταβλητών.

  Τυπικά, ένας k-αδικός **περιορισμός** C με μεταβλητές $X_1, \ldots, X_k$ είναι ένα υποσύνολο του καρτεσιανού γινομένου $D_1 \times \cdots \times D_k$.

Λύση Ενός CSP

- Μια **λύση** ενός CSP είναι μια **ανάθεση** τιμών σε όλες τις μεταβλητές η οποία ικανοποιεί όλους τους περιορισμούς.
- Ένα CSP ονομάζεται **συνεπές (consistent)** αν έχει μια τουλάχιστον λύση, διαφορετικά ονομάζεται **ασυνεπές (inconsistent).**

Παράδειγμα: Χρωματισμός Χάρτη



Χρωματισμός Χάρτη - Τυπικός Ορισμός

- Μεταβλητές: WA, NT, SA, Q, NSW, V, T
- Πεδίο (ίδιο για όλες τις μεταβλητές): $\{ red,\ green,\ blue \}$
- Περιορισμοί:

$$C(WA, NT) = \{ (red, green),\ (red, blue),\ (green, red),$$
$$(blue, red),\ (blue, green) \}$$

Πιο απλά $WA \neq NT$. Ομοίως για τα υπόλοιπα ζεύγη μεταβλητών.

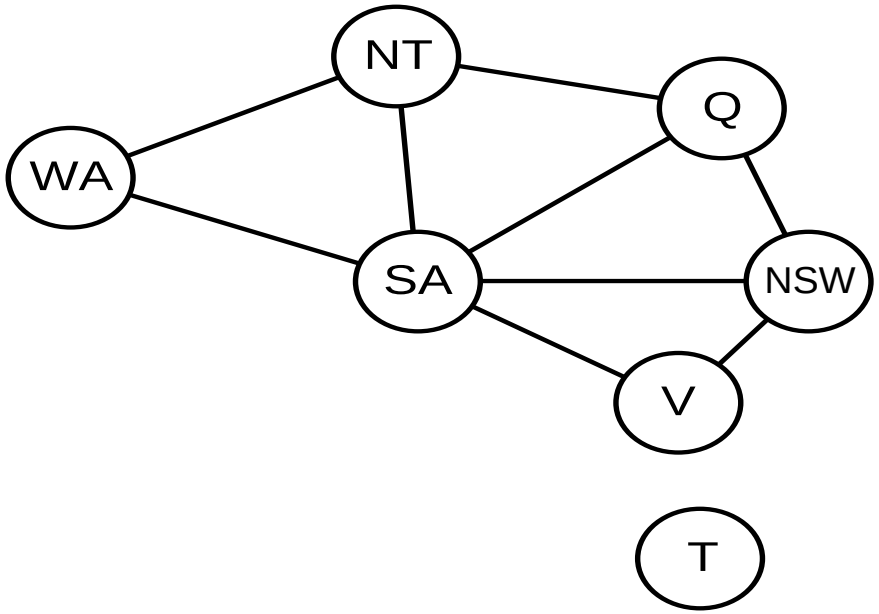
Χρωματισμός Χάρτη - Τυπικός Ορισμός

Το παραπάνω CSP είναι συνεπές. Μια λύση του είναι:

$$\{ WA = red,\ NT = blue,\ SA = green,$$

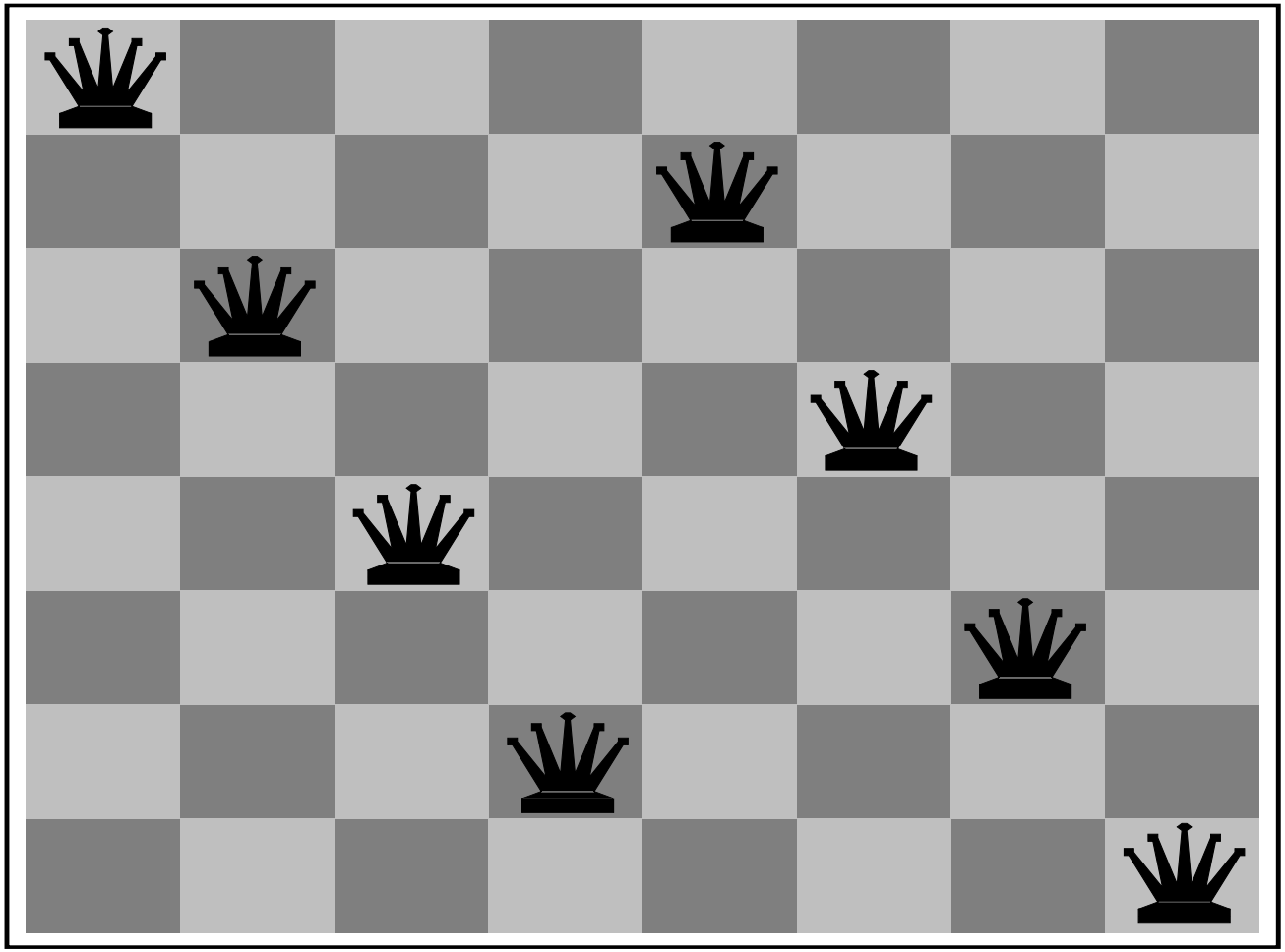
$$Q = red,\ NSW = blue,\ V = red,\ T = blue \ \}$$

Γράφοι Περιορισμών (Constraint Graphs)



Για κάθε CSP μπορούμε να κατασκευάσουμε ένα **γράφο περιορισμών.** Οι κόμβοι του γράφου περιορισμών είναι οι μεταβλητές του CSP και οι ακμές αντιστοιχούν στους περιορισμούς του.

Παράδειγμα: Το Πρόβλημα των 8 Βασιλισσών



8 Βασίλισσες - Τυπικός Ορισμός

- Μεταβλητές:

  Έστω ότι η μεταβλητή X_i $(i = 1, \ldots, 8)$ παριστάνει τη στήλη που καταλαμβάνει η i-οστή βασίλισσα στην i-οστή γραμμή.

  Αν οι στήλες παριστάνονται από τους αριθμούς $1, \ldots, 8$, τότε το πεδίο κάθε μεταβλητής X_i είναι

$$D_i = \{1, 2, \ldots, 8\}.$$

8 Βασίλισσες - Τυπικός Ορισμός

- Περιορισμοί:

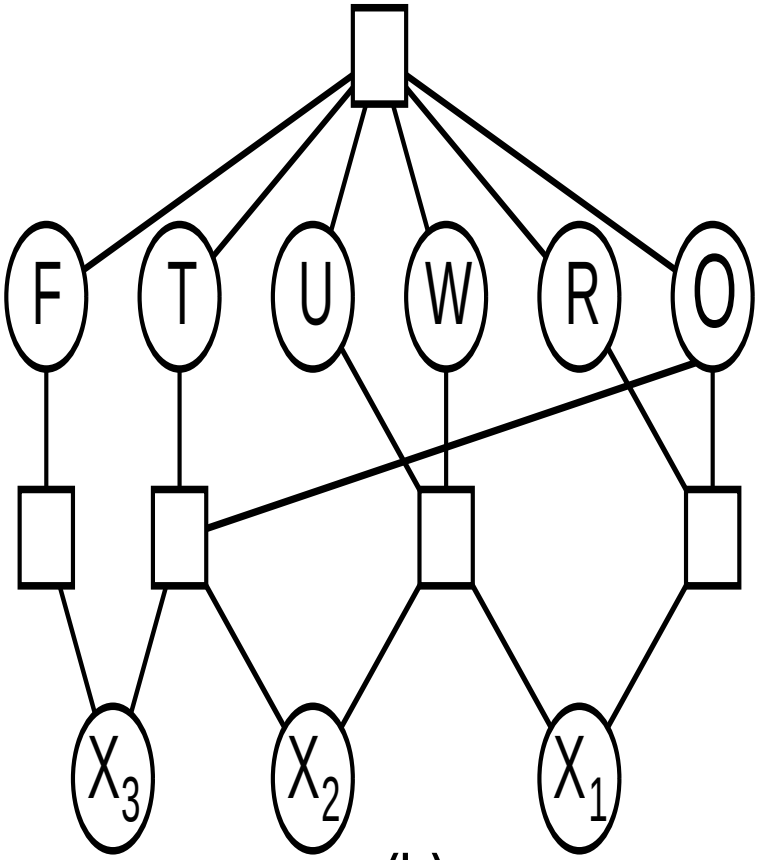
  Για κάθε ζεύγος μεταβλητών X_i και X_j, έχουμε δύο δυαδικούς περιορισμούς:

  - $X_i \neq X_j$ (δεν υπάρχει ζευγάρι βασιλισσών στην ίδια γραμμή)
  - $|i - j| \neq |X_i - X_j|$ (δεν υπάρχει ζευγάρι βασιλισσών στην ίδια διαγώνιο)

Παράδειγμα: Κρυπταριθμητική

```
  T W O
+ T W O
-------
F O U R
```

(a)



(b)

Κρυπταριθμητική - Τυπικός Ορισμός

- Μεταβλητές και πεδία:

$$F, T, U, W, R, O \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

$$X_1, X_2, X_3 \in \{0, 1\}$$

- Περιορισμοί:

$$alldiff(F, T, U, W, R, O)$$

$$O + O = R + 10X_1$$

$$X_1 + W + W = U + 10X_2$$

$$X_2 + T + T = O + 10X_3$$

$$X_3 = F$$

Είδη Περιορισμών σε Πραγματικές Εφαρμογές

Σε πολλά πρακτικά προβλήματα, εκτός από τους **απόλυτους περιορισμούς (hard constraints)**, υπάρχουν και **ελαστικοί περιορισμοί (soft constraints)**.

Οι ελαστικοί περιορισμοί κωδικοποιούν **προτιμήσεις (preferences)** που δεν είναι απαραίτητο να ικανοποιηθούν.

Παράδειγμα: Στην κατάστρωση του ωρολογίου προγράμματος για ένα Πανεπιστήμιο, ποιοι είναι οι απόλυτοι και ποιοι οι ελαστικοί περιορισμοί;

Άλλα Παραδείγματα CSP

- Χρονοπρογραμματισμός εργασιών
- Κατάστρωση προγράμματος (time tabling)
- Ικανοποιησιμότητα στην προτασιακή λογική
- Χρονική συλλογιστική
- Χωρική συλλογιστική
- Ακέραιος, γραμμικός και μη γραμμικός προγραμματισμός (επιχειρησιακή έρευνα)

Τεχνολογία CSP

Τα CSP είναι ένα **πολύ πετυχημένο παράδειγμα ιδεών της Τεχνητής Νοημοσύνης** με πολλές εφαρμογές. Υπάρχουν αρκετές εξειδικευμένες εταιρίες που αναπτύσσουν CSP τεχνολογίες και πολλές επιχειρήσεις που τις χρησιμοποιούν για να λύσουν συγκεκριμένα προβλήματα βελτιστοποίησης ή γενικότερα στη λήψη αποφάσεων:

- `http://www.ilog.com`
- `http://www.cosytec.com`
- ...

Δείτε το μπλόγκ `http://hsimonis.wordpress.com/` για εφαρμογές των CSPs.

Μια Ταξινόμηση των CSP

- **Διακριτές** ή **συνεχείς** μεταβλητές.
- **Πεπερασμένα** ή **μη πεπερασμένα** πεδία.
- **Απαρίθμηση** των συνδυασμών τιμών που επιτρέπει ένας περιορισμός ή **γλώσσες περιορισμών**.
- **Γραμμικοί** ή **μη γραμμικοί** περιορισμοί.
- **Μοναδιαίοι**, **δυαδικοί**, ... περιορισμοί.

Στη συνέχεια της παρουσίασης θα επικεντρώσουμε το ενδιαφέρον μας σε αλγορίθμους αναζήτησης για **δυαδικά CSP με πεπερασμένα πεδία μεταβλητών**.

Δυαδικοί - Μη Δυαδικοί Περιορισμοί

Κάθε μη δυαδικό CSP με πεπερασμένα πεδία μεταβλητών μπορεί να μετατραπεί σε ένα ισοδύναμο δυαδικό.

Δύο γενικές μέθοδοι γι΄ αυτό είναι: ο **δυϊκός μετασχηματισμός (dual transformation)** και η μετατροπή με **κρυφές μεταβλητές (hidden variables)**.

Δείτε τη δημοσίευση

> Fahiem Bacchus, Xinguang Chen, Peter van Beek, and Toby Walsh. Binary vs. non-binary constraints. Artificial Intelligence, 140:1-37, 2002.

για περισσότερες λεπτομέρειες.

Αλγόριθμοι Αναζήτησης για CSP

Ας εφαρμόσουμε το γενικό αλγόριθμο αναζήτησης σε ένα πρόβλημα CSP:

- **Αρχική κατάσταση:** σε καμία μεταβλητή δεν έχει ανατεθεί τιμή.
- **Ενέργειες:** Αναθέτουμε σε κάθε μεταβλητή X_i που δεν έχει τιμή, οποιαδήποτε τιμή από το D_i.
- **Έλεγχος στόχου:** Όλες οι μεταβλητές έχουν πάρει τιμές και όλοι οι περιορισμοί ικανοποιούνται.

Ο **παράγοντας διακλάδωσης** σ΄ αυτή την περίπτωση είναι $\sum_{i=1}^{n} |D_i|$ όπου $|D_i|$ είναι ο πληθικός αριθμός του πεδίου D_i. Το τελευταίο επίπεδο του δέντρου έχει $(\sum_{i=1}^{n} |D_i|)^n$ κόμβους.

Αλγόριθμοι Αναζήτησης για CSP

Καλύτερη προσέγγιση: Ταξινομούμε τις μεταβλητές! Τα CSP είναι **αντιμεταθετικά** προβλήματα αναζήτησης, δηλαδή η σειρά εφαρμογής των ενεργειών δεν έχει σημασία.

Χαρακτηριστικά:

- Το μέγεθος του χώρου αναζήτησης είναι **πεπερασμένο**. Εάν οι μεταβλητές είναι ταξινομημένες με τη σειρά $X_1, \ldots, X_n$, το πλήθος των κόμβων στο δέντρο αναζήτησης είναι $1 + \sum_{i=1}^{n} (|D_1| \cdots |D_i|)$.

  Πότε είναι το πλήθος των κόμβων στο δέντρο αναζήτησης μέγιστο; Ελάχιστο;
- Το βάθος του δέντρου αναζήτησης είναι **σταθερό**.
- Υπάρχουν **παρόμοια υποδέντρα**.

Αλγόριθμοι Αναζήτησης για CSP

Ποιος από τους αλγόριθμους αναζήτησης που περιγράψαμε ως τώρα είναι κατάλληλος για την επίλυση CSP:

- BFS;

  Όχι! Ο BFS δεν είναι αποδοτικός διότι οι καταστάσεις στόχου βρίσκονται στα φύλλα του δέντρου αναζήτησης.

- DFS;

  Καλύτερος από τον BFS. Ωστόσο, ο DFS μπορεί να σπαταλά χρόνο αναζητώντας λύση, ενώ κάποιοι περιορισμοί έχουν ήδη παραβιαστεί.

Αλγόριθμοι Υπαναχώρησης

Θα παρουσιάσουμε παραλλαγές του DFS για CSP. Αυτοί οι αλγόριθμοι βασίζονται στην ιδέα της **αναζήτησης με υπαναχώρηση (backtracking)**:

> Επιλέγουμε τιμές για μια μεταβλητή κάθε φορά, ελέγχουμε αν οι περιορισμοί ικανοποιούνται, και υπαναχωρούμε όταν δεν υπάρχουν άλλες επιτρεπτές τιμές για να επιλεγούν.

Αλγόριθμοι Υπαναχώρησης

Θα μελετήσουμε τους ακόλουθους **αλγόριθμους υπαναχώρησης (backtracking algorithms)** και τους δυνατούς συνδυασμούς τους:

- Απλή ή χρονολογική υπαναχώρηση (simple or chronological backtracking - BT)
- Πρώϊμος ελεγχος (forward checking - FC)
- Διατήρηση συνέπειας ακμής (maintaining arc consistency - MAC)
- Υπαναχώρηση με άλμα (backjumping - BJ)
- Υπαναχώρηση με άλμα κατευθυνόμενο από σύγκρουση (conflict-directed backjumping - CBJ)

Απλή ή Χρονολογική Υπαναχώρηση (BT)

Η βασική ιδέα σε κάθε αλγόριθμο υπαναχώρησης είναι να ξεκινήσουμε με μια **μερική λύση** και να την επεκτείνουμε μέχρι να καταλήξουμε σε μια **πλήρη λύση.**

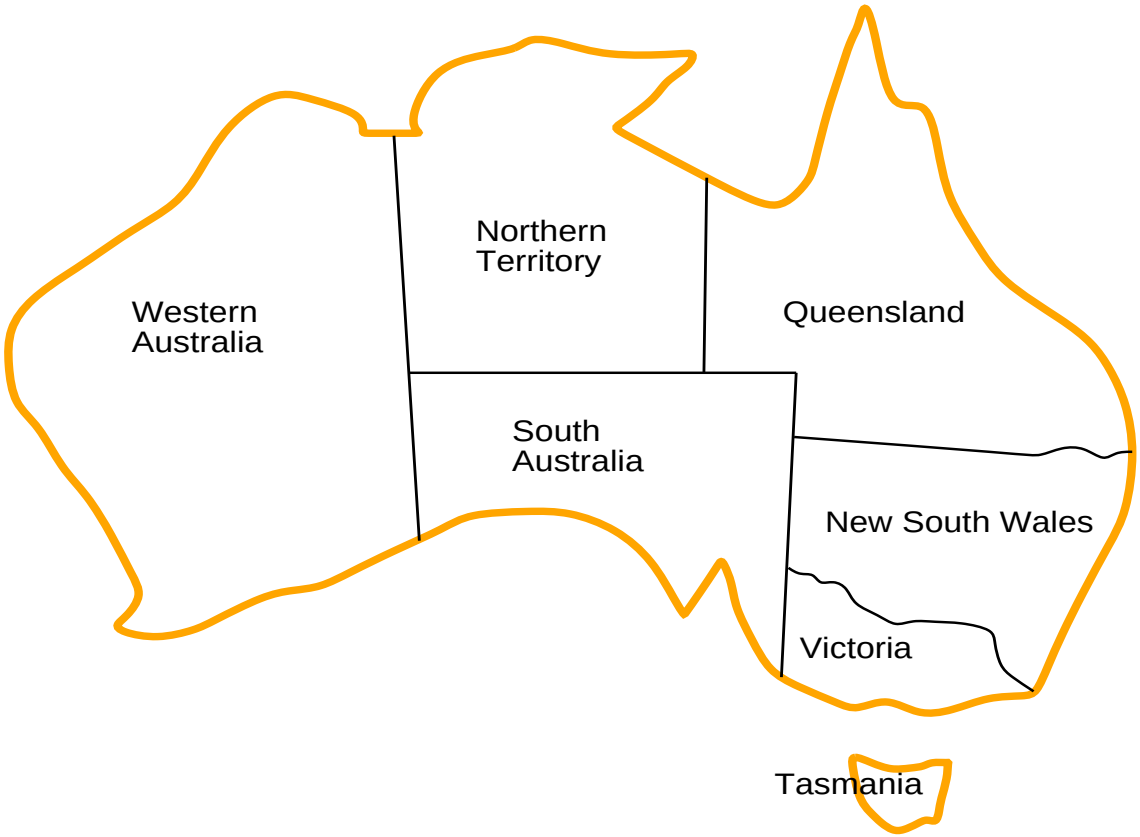
Ο αλγόριθμος BT ακολουθεί αυτή τη γενική μέθοδο. Επιπρόσθετα, όταν φτάνει σε αδιέξοδο, πάντοτε υπαναχωρεί στην τελευταία ληφθείσα απόφαση (εξ' ου και το όνομά του!).

function BACKTRACKING-SEARCH(*csp*)
returns a solution or failure
return RECURSIVE-BACKTRACKING($\{\}$, *csp*)

BT

function RECURSIVE-BACKTRACKING($assignment, csp$)
returns a solution or failure
if $assignment$ is complete **then return** $assignment$
$var \leftarrow$ SELECT-UNASSIGNED-VARIABLE(VARIABLES$[csp], assignment, csp$)
for each $value$ **in** ORDER-DOMAIN-VALUES($var, assignment, csp$) **do**
  if $value$ is consistent with $assignment$ according to CONSTRAINTS(csp)
  then
    add $\{var = value\}$ to $assignment$
    $result \leftarrow$ RECURSIVE-BACKTRACKING($assignment, csp$)
    if $result \neq failure$ **then return** $result$
    remove $\{var = value\}$ from $assignment$
return failure

Ο Αλγόριθμος BT στην Πράξη: Παράδειγμα



Αξιολόγηση του Αλγόριθμου BT

- **Πλήρης;** Ναι
- **Χρόνος:** $O(d^n e)$ όπου d είναι ο μέγιστος αριθμός στοιχείων ενός πεδίου, n είναι ο αριθμός των μεταβλητών, και e είναι ο αριθμός των περιορισμών.
- **Χώρος:** $O(nd)$. Είναι ο χώρος που χρειάζεται για την αποθήκευση των πεδίων των μεταβλητών.

Τα παραπάνω όρια χρονικής και χωρικής πολυπλοκότητας βασίζονται στην υπόθεση ότι **οι περιορισμοί μπορούν να αποθηκευτούν χρησιμοποιώντας σταθερό χώρο και οι έλεγχοι των περιορισμών μπορούν να γίνουν σε σταθερό χρόνο.**

Βελτιώσεις του Αλγόριθμου BT

Ο αλγόριθμος BT μπορεί να βελτιωθεί αν δώσουμε έξυπνες απαντήσεις στις ακόλουθες ερωτήσεις:

1. Ποια είναι η **επόμενη μεταβλητή** στην οποία θα αναθέσουμε τιμή και με ποια σειρά θα πρέπει να ελεγχθούν οι **τιμές** της;
2. Ποιες είναι οι **συνέπειες της τρέχουσας ανάθεσης** στις υπόλοιπες μεταβλητές που δεν έχουν τιμές;
3. Όταν μια διαδρομή αποτυγχάνει, μπορεί αυτή η αναζήτηση να **αποφύγει την ίδια αποτυχία** σε μελλοντικές διαδρομές;

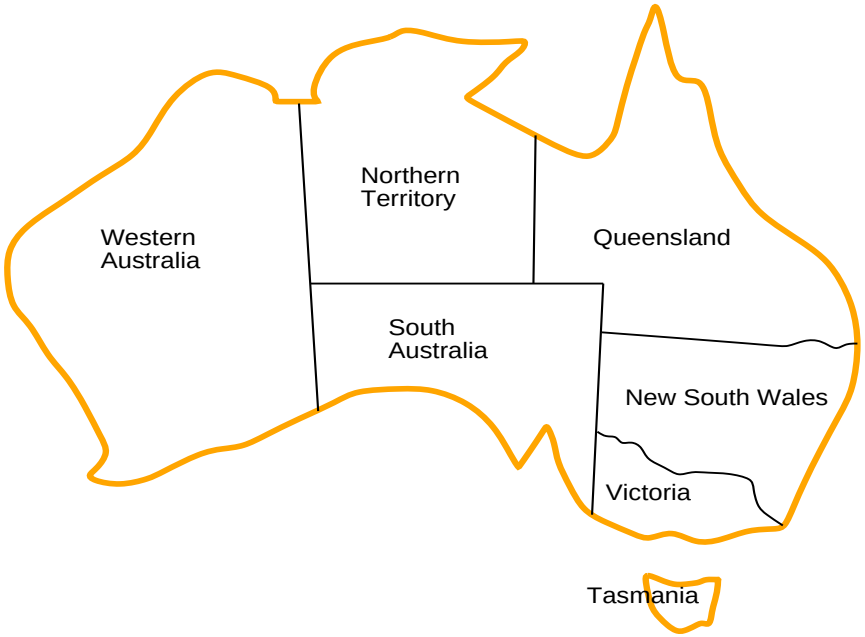
Ευρετικοί Μηχανισμοί Επιλογής Μεταβλητών

Η συνάρτηση SELECT-UNASSIGNED-VARIABLE επιλέγει την επόμενη μεταβλητή που δεν έχει τιμή με τη σειρά που προκύπτει από τη λίστα VARIABLES[CSP]. Αυτή η **στατική** επιλογή μεταβλητών σπάνια καταλήγει σε αποδοτική αναζήτηση.

Ο ευρετικός μηχανισμός των **ελάχιστων απομενουσών τιμών (minimum remaining values - MRV)** επιλέγει τη μεταβλητή με τις λιγότερες νόμιμες τιμές που απομένουν, δηλαδή ελαχιστοποιεί τον παράγοντα διακλάδωσης.

Με τον MRV έχουμε **δυναμική** επιλογή μεταβλητών.

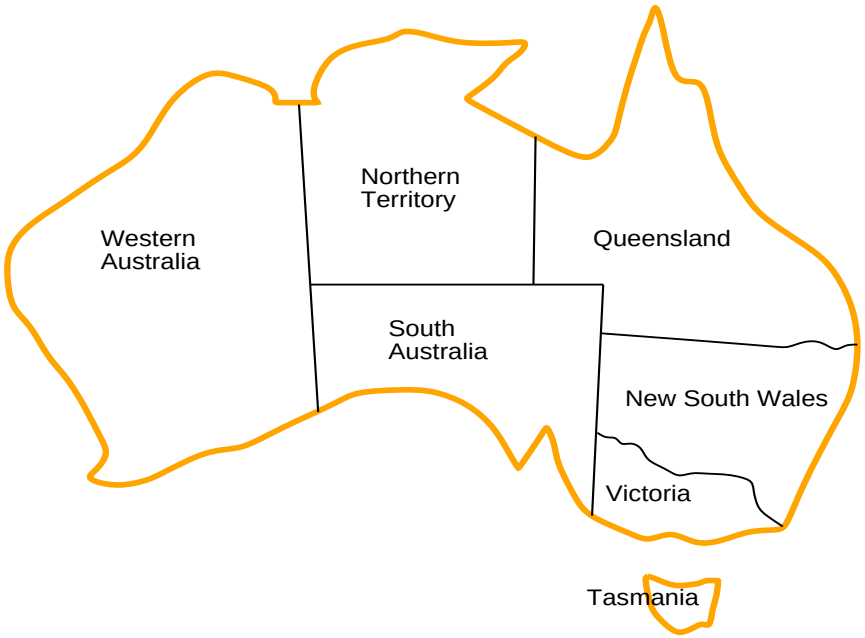
MRV - Παράδειγμα



Ερώτηση: Ποια είναι η μεταβλητή στην οποία θα πρέπει να ανατεθεί τιμή αφού γίνουν οι αναθέσεις

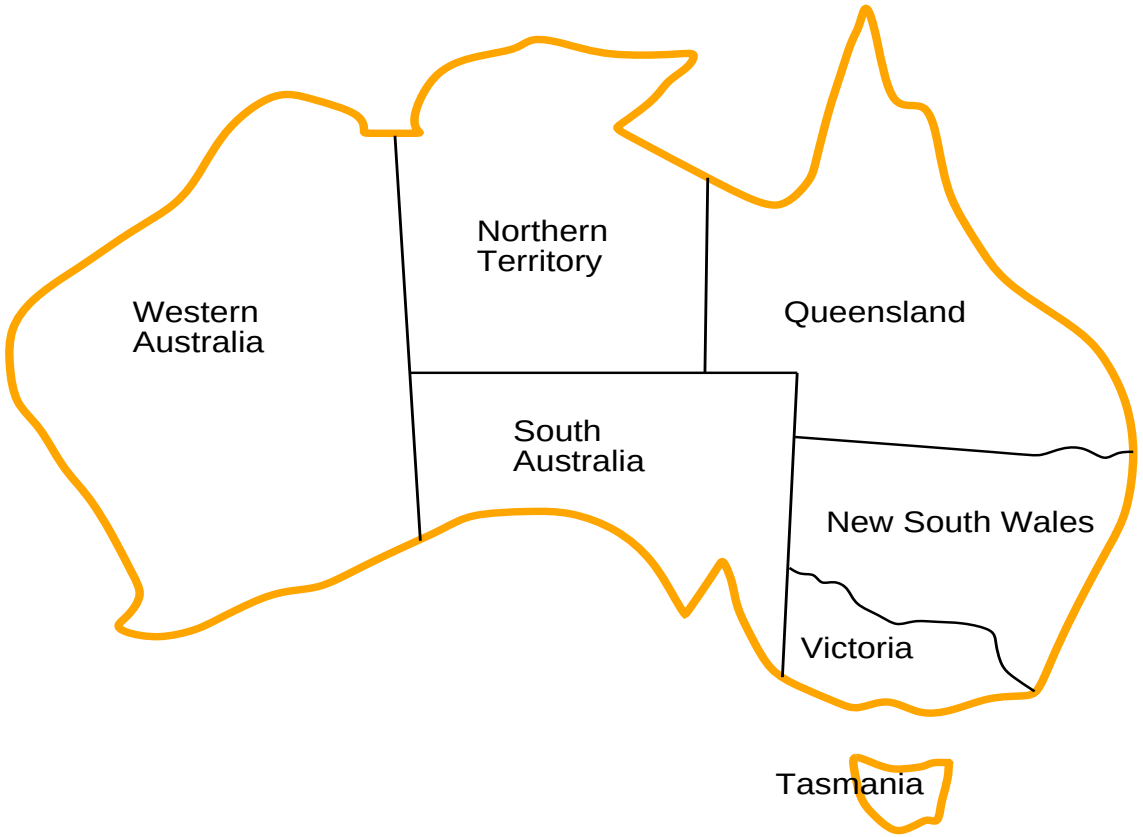
$$WA = red,\ NT = green?$$

MRV - Παράδειγμα



Απάντηση: Η SA, επειδή γι αυτήν μόνο η τιμή $blue$ είναι δυνατή.

Παράδειγμα: Χρωματισμός Χάρτη



Ερώτηση: Με ποια μεταβλητή ξεκινάμε την αναζήτηση για το χρωματισμό του χάρτη της Αυστραλίας;

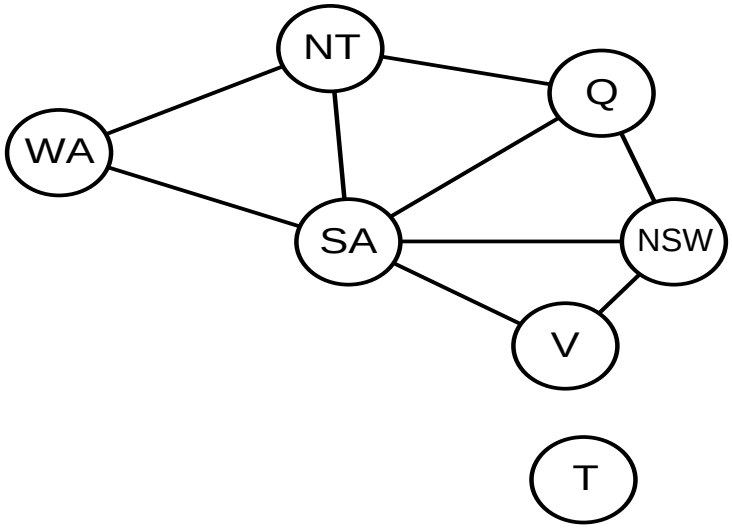
Ευρετικοί Μηχανισμοί Επιλογής Μεταβλητών

Ο ευρετικός μηχανισμός **βαθμού (degree heuristic)** είναι ο εξής: επιλέγουμε τη μεταβλητή που εμπλέκεται στο μεγαλύτερο πλήθος περιορισμών με άλλες μεταβλητές που δεν τους έχει ανατεθεί τιμή (δηλαδή τη μεταβλητή-κόμβο με το **μεγαλύτερο βαθμό** στον γράφο των περιορισμών). Αυτή η επιλογή είναι στατική.

Βαθμός κόμβου σ΄ ένα γράφο είναι το πλήθος των ακμών που πρόσκεινται στον κόμβο.

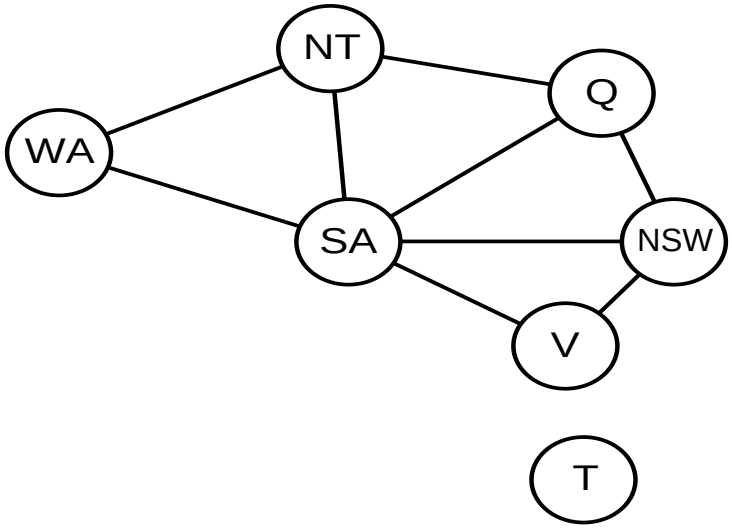
Διαίσθηση: Αυξάνουμε τη μελλοντική απαλειφή τιμών, προκειμένου να μειώσουμε τους μελλοντικούς παράγοντες διακλάδωσης.

Degree Heuristic - Παράδειγμα



Ερώτηση: Με ποια μεταβλητή ξεκινάμε την αναζήτηση για το χρωματισμό του χάρτη της Αυστραλίας;

Degree Heuristic - Παράδειγμα



Απάντηση: Με την SA που έχει βαθμό 5. Όλες οι άλλες μεταβλητές έχουν βαθμό 2 ή 3 εκτός από την T που έχει βαθμό 0.

Ευρετικοί Μηχανισμοί Επιλογής Μεταβλητών

Ο MRV είναι συνήθως πιο ισχυρός από τον degree heuristic. Μπορούν να χρησιμοποιηθούν μαζί: όταν ο MRV δεν μπορεί να ξεχωρίσει δύο μεταβλητές, χρησιμοποιείται ο degree heuristic.

Θα χρησιμοποιήσουμε τον όρο **MRV** για να αναφερθούμε σε αυτό το συνδυασμό ευρετικών μηχανισμών.

Αξιολόγηση

Problem	**BT**	**BT+MRV**	**FC**	**FC+MRV**	**Min-con**
USA	(>1000K)	(>1000K)	2K	60	64
n-Queens	(>40000K)	13500K	(>40000K)	817K	4K
Zebra	3859K	1K	35K	0.5K	2K

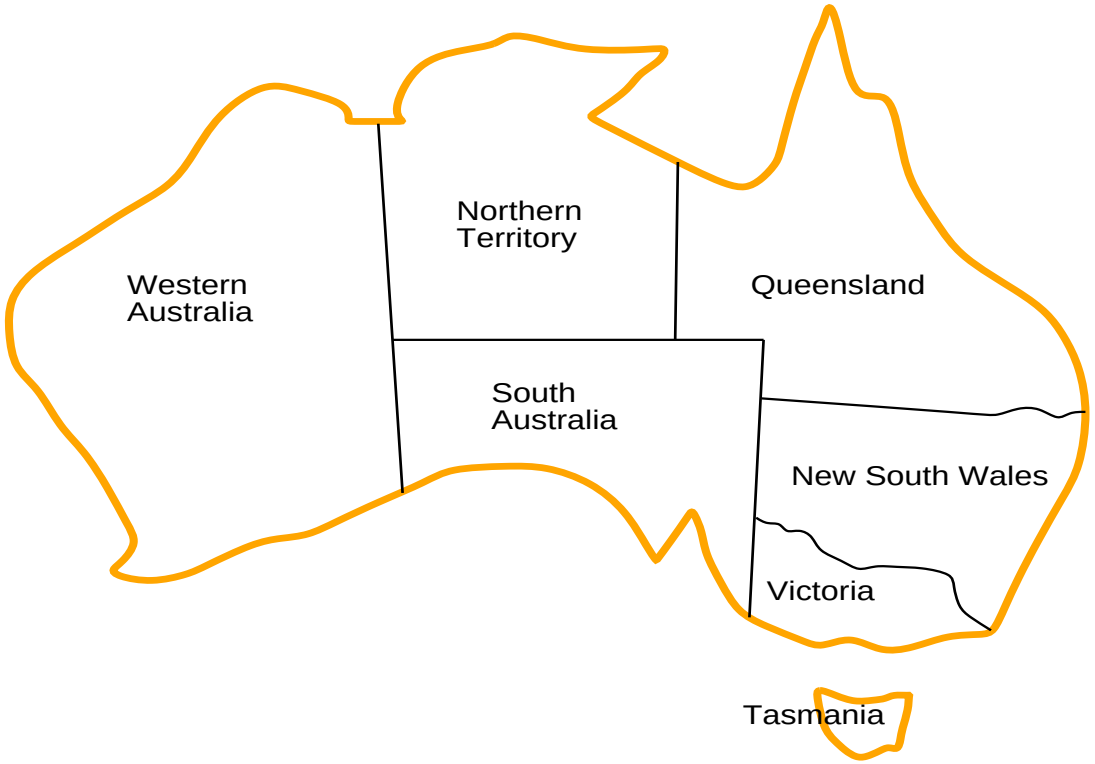
Ευρετικοί Μηχανισμοί Επιλογής Τιμών

Από τη στιγμή που μια μεταβλητή έχει επιλεγεί, ο αλγόριθμος πρέπει να αποφασίσει με ποια σειρά πρέπει να εξετάσει τις **τιμές** της.

Ο ευρετικός μηχανισμός της **λιγότερο δεσμεύουσας τιμής (least-constraining value - LCV)** προτιμά την τιμή που αποκλείει τις λιγότερες επιλογές τιμών από τις γειτονικές μεταβλητές του γράφου περιορισμών. Μ΄ άλλα λόγια, μας δίνει τη μέγιστη ευελιξία για τις επόμενες αναθέσεις μεταβλητών.

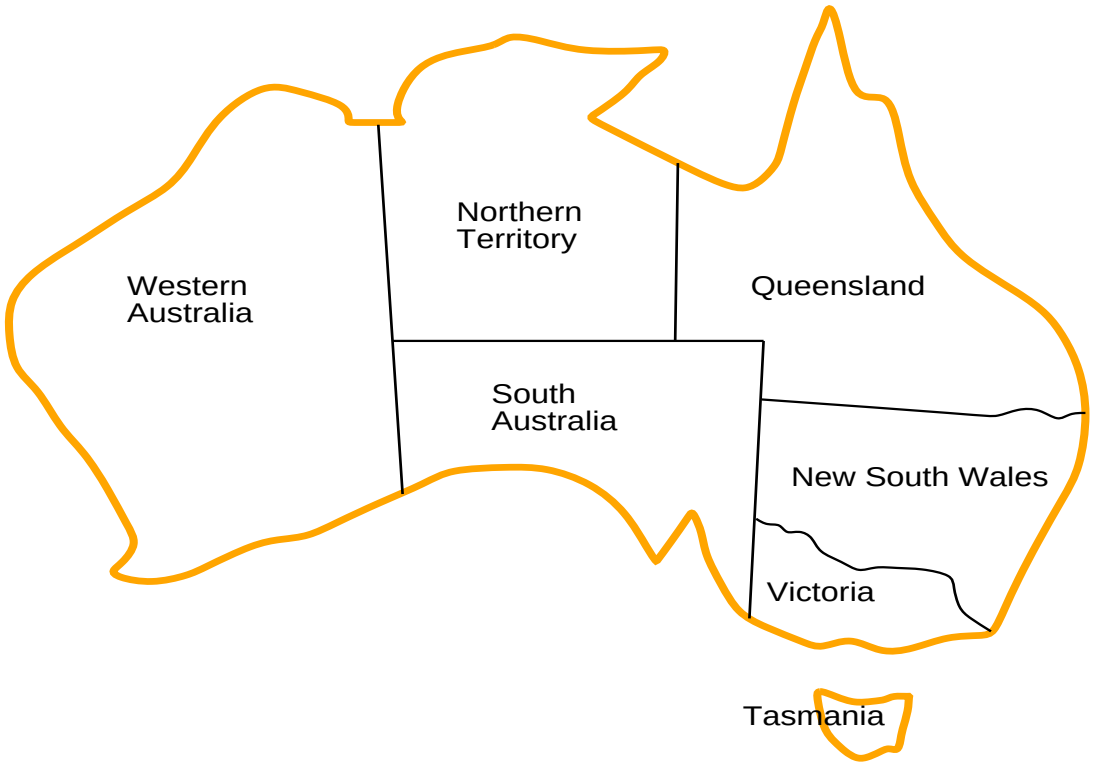
Ο LCV δεν είναι χρήσιμος αν ενδιαφερόμαστε για όλες τις λύσεις ή αν το πρόβλημα δεν έχει λύση.

LCV - Παράδειγμα



Ερώτηση: Ποια είναι η τιμή που θα ανατεθεί στη μεταβλητή Q μετά τις αναθέσεις $WA = red,\ NT = green$;

LCV - Παράδειγμα



Απάντηση: Μπορούμε να επιλέξουμε μόνο *blue* ή *red*. Η τιμή *blue* δεν είναι καλή επιλογή σύμφωνα με την LCV: αποκλείει την ίδια τιμή από δύο γειτονικές μεταβλητές (NSW και SA). Η *red* είναι καλύτερη διότι αποκλείει την ίδια τιμή μόνο από την NSW.

Διάδοση Περιορισμών

Η κεντρική ιδέα της **διάδοσης περιορισμών (constraint propagation)** είναι να λάβουμε υπόψη μας τους δοθέντες περιορισμούς όσο νωρίς μπορούμε κατά τη διάρκεια της αναζήτησης ή ακόμα και πριν αρχίσει η αναζήτηση!

Για παράδειγμα, σε κάθε βήμα της αναζήτησης μπορούμε να **κλαδεύουμε** τμήματα του χώρου αναζήτησης που απομένει να εξερευνηθεί, εξετάζοντας τις **συνέπειες** των μερικών αναθέσεων. Οι αλγόριθμοι που έχουν προταθεί γι΄ αυτό ονομάζονται συχνά αλγόριθμοι που βλέπουν μπροστά **(look-ahead)**.

Πρώιμος Έλεγχος (Forward Checking - FC)

Ο αλγόριθμος του **πρώιμου ελέγχου** στηρίζεται στην ικανοποίηση της ακόλουθης συνθήκης:

> Για κάθε μεταβλητή στην οποία δεν έχει ανατεθεί τιμή, υπάρχει τουλάχιστον μια τιμή στο πεδίο της που είναι συνεπής με τις τιμές που έχουν ανατεθεί σε άλλες μεταβλητές.

FC

Ο FC λειτουργεί ως εξής: κάθε φορά που μια τιμή v ανατίθεται σε μια μεταβλητή X, ο FC εξετάζει κάθε μεταβλητή Y στην οποία δεν έχουν ανατεθεί τιμές και **διαγράφει** όλες τις τιμές της Y που είναι ασυνεπείς με την v από το πεδίο της. Αν αυτή η ενέργεια έχει σαν αποτέλεσμα το πεδίο της Y να γίνει κενό, τότε η v απορρίπτεται.

Ο αλγόριθμος FC χρησιμοποιείται συνήθως **μαζί με τον ευρετικό μηχανισμό MRV**, μια που οι απαιτούμενες δομές για την υλοποίηση του MRV χρησιμοποιούνται και από τον FC.

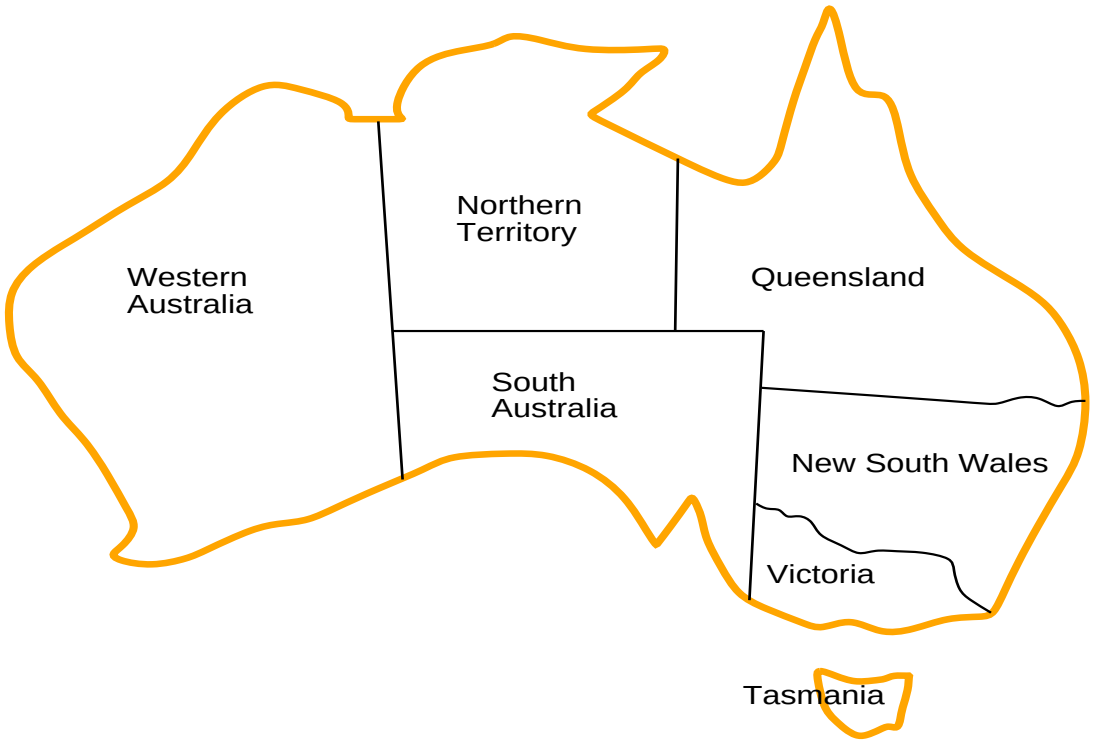
FC Χωρίς MRV - Παράδειγμα

	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After WA=red	(R)	G B	R G B	R G B	R G B	G B	R G B
After Q=green	(R)	B	(G)	R B	R G B	B	R G B
After V=blue	(R)	B	(G)	R	(B)		R G B

Αξιολόγηση

Problem	**BT**	**BT+MRV**	**FC**	**FC+MRV**	**Min-con**
USA	(>1000K)	(>1000K)	2K	60	64
n-Queens	(>40000K)	13500K	(>40000K)	817K	4K
Zebra	3859K	1K	35K	0.5K	2K

Παράδειγμα Δυσκολότερης Ασυνέπειας



Η ανάθεση $WA = red,\ Q = green$ συνεπάγεται $NT = blue,\ SA = blue$.
Αυτή η ασυνέπεια **δεν ανιχνεύεται** από τον FC αμέσως μετά που θα δοθούν τιμές στις μεταβλητές WA και Q.

Παράδειγμα

Η μερική ανάθεση

$$WA = red,\ Q = green$$

μαζί με τους περιορισμούς του προβλήματος

$$WA \neq NT,\ WA \neq SA, \quad Q \neq NT,\ Q \neq SA$$

$$WA, Q \in \{red, blue, green\}$$

συνεπάγεται

$$NT = blue,\ SA = blue.$$

Αυτοί οι **συνεπαγόμενοι (implied)** περιορισμοί μαζί με τον αρχικό περιορισμό $NT \neq SA$ είναι **ασυνεπείς (inconsistent)**.

Ο μηχανισμός του FC δεν περιλαμβάνει αυτό το **δεύτερο βήμα** διάδοσης περιορισμών.

Συνέπεια Ακμής (Arc Consistency)

Η έννοια της **συνέπειας ακμής** μας δίνει ένα ισχυρότερο εργαλείο διάδοσης περιορισμών.

Ορισμός. Έστω X, Y μεταβλητές ενός CSP P και (X, Y) μια κατευθυνόμενη ακμή στο γράφο περιορισμών του P. Η ακμή (X, Y) ονομάζεται **συνεπής** αν για κάθε τιμή x του X, υπάρχει μια τιμή y του Y έτσι ώστε η x να είναι συνεπής με την y.

Ορισμός. Ένα CSP ονομάζεται **συνεπές ως προς τις ακμές (arc consistent)** αν όλες οι ακμές του γράφου περιορισμών του είναι συνεπείς.

Το Παράδειγμα Επαναδιατυπωμένο

	WA	NT	Q	NSW	V	SA	T
Initial domains	R G B	R G B	R G B	R G B	R G B	R G B	R G B
After WA=red	(R)	G B	R G B	R G B	R G B	G B	R G B
After Q=green	(R)	B	(G)	R B	R G B	B	R G B
After V=blue	(R)	B	(G)	R	(B)		R G B

Παράδειγμα

Θεωρήστε την τρίτη γραμμή του παραπάνω πίνακα για το πρόβλημα του χρωματισμού του χάρτη της Αυστραλίας με χρήση του FC (τη γραμμή στην οποία κάνουμε ανάθεση τιμής στην Q).

Σε αυτό το στάδιο του αλγόριθμου, ας ελέγξουμε τη συνέπεια μερικών ακμών του γράφου περιορισμών:

- Ακμή (SA, NSW): Τα τρέχοντα πεδία των κόμβων SA και NSW είναι $\{ blue \}$ και $\{ red, blue \}$, γι′ αυτό η (SA, NSW) είναι **συνεπής.**
- Ακμή (NSW, SA): Αυτή είναι **ασυνεπής**, επειδή η ανάθεση $NSW = blue$ δεν έχει συνεπή ανάθεση για το SA. Σε αυτή την περίπτωση, θα πρέπει να **διαγράψουμε την τιμή** $blue$ από το πεδίο της NSW για να κάνουμε την ακμή συνεπή.

Παράδειγμα

- Ακμή (SA, NT): Αυτή η ακμή είναι **ασυνεπής** επειδή το τρέχον πεδίο για τις SA και NT είναι $\{blue\}$. Αν προσπαθήσουμε να κάνουμε αυτή την ακμή συνεπή, θα διαγράψουμε την τιμή $blue$ από το πεδίο της SA, αφήνοντας αυτό το πεδίο **κενό**.

Έτσι, η εφαρμογή της συνέπειας ακμής οδήγησε **νωρίτερα στον εντοπισμό μιας ασυνέπειας** κατά τη διάρκεια της αναζήτησης.

Συνέπεια Ακμής

Στον αλγόριθμο BT, η συνέπεια ακμής μπορεί να εφαρμοστεί ως:

- **Βήμα προεπεξεργασίας (preprocessing step)** πριν αρχίσει η αναζήτηση.
- **Βήμα διάδοσης περιορισμών (constraint propagation step)** μετά από κάθε ανάθεση (επαναληπτικά, μέχρι να μην υπάρχουν άλλες ασυνέπειες ακμής). Ο νέος αλγόριθμος που προκύπτει είναι γνωστός ως **MAC (maintaining arc consistency)**.

Παράδειγμα: Αν χρησιμοποιήσουμε τον MAC μετά την ανάθεση

$$WA = red,\ Q = green$$

στο παράδειγμα χρωματισμού χάρτη, ανακαλύπτουμε αμέσως την αδυναμία επέκτασης αυτής της ανάθεσης επειδή η ακμή (SA, NT) είναι ασυνεπής.

Ο Αλγόριθμος AC-3

function AC-3(*csp*) **returns** the CSP, possibly with reduced domains

inputs: *csp*, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$

local variables: *queue*, a queue of arcs, initially all the arcs in *csp*

while *queue* is not empty **do**

$(X_i, X_j) \leftarrow$ REMOVE-FIRST(*queue*)

if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**

for each X_k **in** NEIGHBORS$[X_i]$ **do**

add (X_k, X_i) to *queue*

Ο Αλγόριθμος AC-3

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff we remove a value

removed$\leftarrow$ *false*
for each x **in** DOMAIN[X_i] **do**
if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint between X_i and X_j **then**
delete x from DOMAIN[X_i]; *removed*$\leftarrow$ *true*
return *removed*

AC-3

Ο AC-3 εκτελείται σε $O(n^2 d^3)$ χρόνο, όπου n είναι το πλήθος των μεταβλητών και d είναι το μέγιστο πλήθος τιμών ενός πεδίου.

Απόδειξη:

- Ένα δυαδικό CSP έχει το πολύ $O(n^2)$ ακμές
- Μια ακμή μπορεί να εισέλθει στην ουρά το πολύ d φορές.
- Ο έλεγχος συνέπειας μια ακμής μπορεί να γίνει σε χρόνο $O(d^2)$.

Υπάρχουν **ταχύτεροι** αλγόριθμοι συνέπειας ακμής.

Άλλες Έννοιες Συνέπειας

- **1-consistency ή συνέπεια κόμβου:** Ένα CSP είναι 1-consistent αν και μόνο αν για κάθε μοναδιαίο περιορισμό $c(X)$ ισχύει το εξής: για κάθε τιμή $v \in domain(X)$, v είναι μια λύση του c.

  Για να γίνει ένα CSP 1-consistent, αφαιρούμε από το πεδίο κάθε μεταβλητής X που έμφανίζεται σ' ένα μοναδιαίο περιορισμό $c(X)$ τις τιμές v που παραβιάζουν την παραπάνω συνθήκη.

- **2-consistency ή συνέπεια ακμής**.

Ισχυρότερες Έννοιες Συνέπειας

- **3-consistency ή path consistency**
- **4-consistency**
- ...
- **k-consistency**
- ...
- **global consistency** (k-consistency για όλα τα $1 \leq k \leq n$ σε ένα CSP με n μεταβλητές).

k-συνέπεια

Ορισμός. Ένα CSP είναι k-**consistent** εάν κάθε ανάθεση τιμών σε $k-1$ μεταβλητές $\{X_1 = v_1, \ldots, X_{k-1} = v_{k-1}\}$ που ικανοποιεί όλους τους περιορισμούς που εμπλέκουν τις μεταβλητές $X_1, \ldots, X_{k-1}$, μπορεί να επεκταθεί σε μια ανάθεση $\{X_1 = v_1, \ldots, X_{k-1} = v_{k-1}, X_k = v_k\}$ που ικανοποιεί όλους τους περιορισμούς που εμπλέκουν τις μεταβλητές $X_1, \ldots, X_{k-1}, X_k$.

Για να γίνει ένα CSP k-consistent, εισάγουμε στο CSP **νέους περιορισμούς** με $k-1$ μεταβλητές που αποκλείουν τις αναθέσεις σε $k-1$ μεταβλητές που παραβιάζουν την παραπάνω συνθήκη. Αυτοί οι περιορισμοί έπονται λογικά από τους υπάρχοντες περιορισμούς, αλλά η εφαρμογή της k-συνέπειας τους φέρνει στο φως.

Παράδειγμα

Το CSP

$$X_1 - X_2 \leq 2,\ X_2 - X_3 \leq 5,\ X_1 - X_3 \leq 10,\ X_1, X_2, X_3 \in [0, 100]$$

είναι 1-consistent και 2-consistent αλλά όχι 3-consistent. Για παράδειγμα, η ανάθεση μεταβλητών

$$\{X_1 = 10, X_3 = 0\}$$

που ικανοποιεί τον περιορισμό $X_1 - X_3 \leq 10$ δεν μπορεί να επεκταθεί σε μια ανάθεση τιμών στις μεταβλητές X_1, X_3, X_2 που ικανοποιεί και τους τρεις περιορισμούς.

Παράδειγμα

Το CSP

$$X_1 - X_2 \leq 2,\ X_2 - X_3 \leq 5,\ X_1 - X_3 \leq 7,\ X_1, X_2, X_3 \in [0, 100]$$

που έχει έναν αυστηρότερο περιορισμό πάνω στα X_1, X_3 είναι 3-consistent.

Ο περιορισμός $X_1 - X_3 \leq 7$ **συνεπάγεται** από τους δοσμένους περιορισμούς και μπορεί να εξαχθεί με **διάδοση περιορισμών**.

Άλλες Έννοιες Συνέπειας

Οι έννοιες συνέπειας που είναι ισχυρότερες από τη συνέπεια ακμής μπορούν να χρησιμοποιηθούν από τους αλγόριθμους υπαναχώρησης με παρόμοιο τρόπο.

Σε κάθε βήμα της διαδικασίας αναζήτησης, οι έλεγχοι συνέπειας **φέρνουν στο φως τις συνέπειες των περιορισμών** εξετάζοντας 3 μεταβλητές, 4 μεταβλητές, ..., n μεταβλητές κάθε φορά.

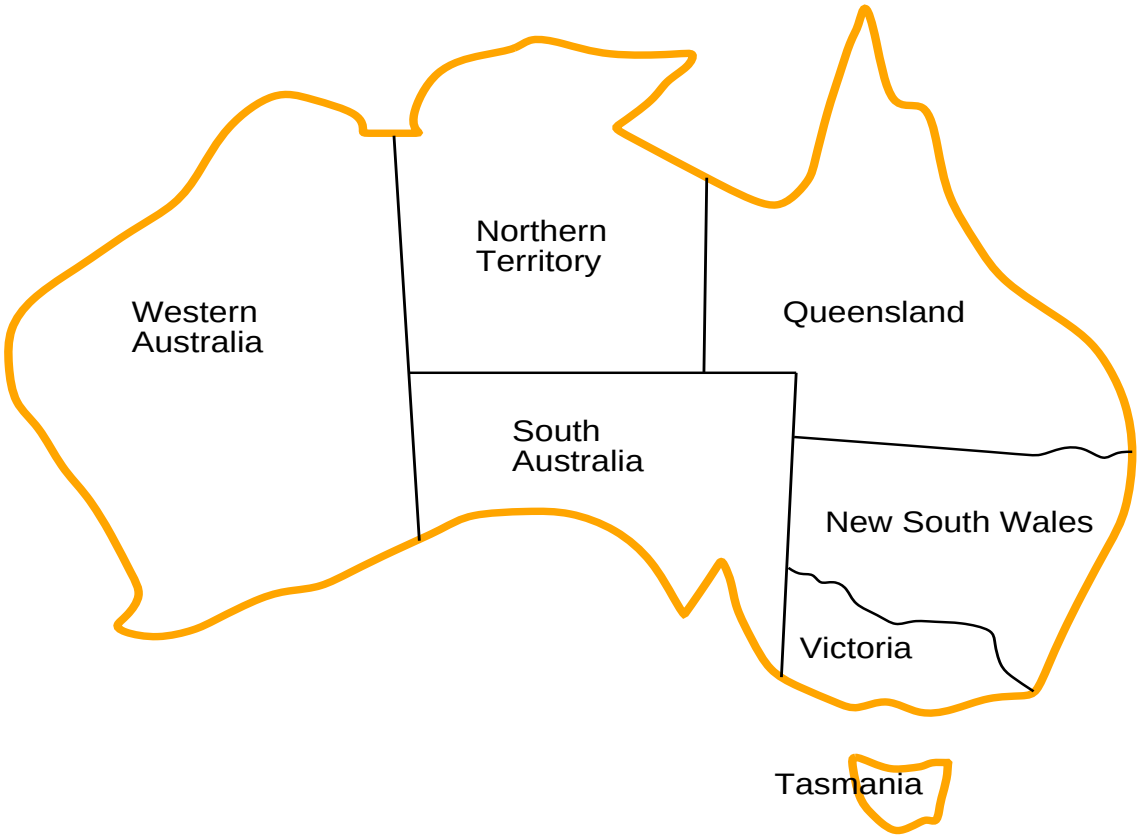
Προσοχή: Το **κόστος** των βημάτων διάδοσης περιορισμών πρέπει προσεκτικά να εξισορροπηθεί με τα πλεονεκτήματά τους. Ένας τρόπος για να το καθορίσουμε αυτό είναι να κάνουμε μια λεπτομερή πειραματική μελέτη.

Ευφυής Υπαναχώρηση (Intelligent Backtracking)

Οι αλγόριθμοι υπαναχώρησης μπορούν να γίνουν πιο αποδοτικοί αν υπαναχωρούν με πιο έξυπνο τρόπο! Οι τεχνικές που έχουν προταθεί για να επιτευχθεί αυτό, συχνά αναφέρονται ως τεχνικές **look-back**.

Η ιδέα αυτών των τεχνικών είναι να μην υπαναχωρούμε χρονολογικά όπως ο BT, αλλά με κάποιο ευφυή τρόπο, προς την **πραγματική αιτία** της αποτυχίας.

Χρονολογική Υπαναχώρηση



Χρονολογική Υπαναχώρηση

Ας εφαρμόσουμε τον ΒΤ στο πρόβλημα του χρωματισμού χάρτη με την ακόλουθη στατική σειρά επιλογής μεταβλητών:

$$Q, NSW, V, T, SA, WA, NT.$$

Έστω ότι έχουμε παράγει την ακόλουθη μερική ανάθεση:

$$Q = red,\ NSW = green,\ V = blue,\ T = red$$

Όταν επιλέξουμε την επόμενη μεταβλητή (SA), βλέπουμε ότι κάθε τιμή της παραβιάζει ένα περιορισμό. Τώρα ο ΒΤ μας επιβάλλει να υπαναχωρήσουμε και να δοκιμάσουμε ένα νέο χρώμα για την Τασμανία. Αυτό δεν είναι καθόλου καλή ιδέα!

Υπαναχώρηση με Άλμα (Backjumping)

Ο αλγόριθμος υπαναχώρησης με άλμα (**backjumping - BJ**) είναι ένας αλγόριθμος ευφυούς υπαναχώρησης.

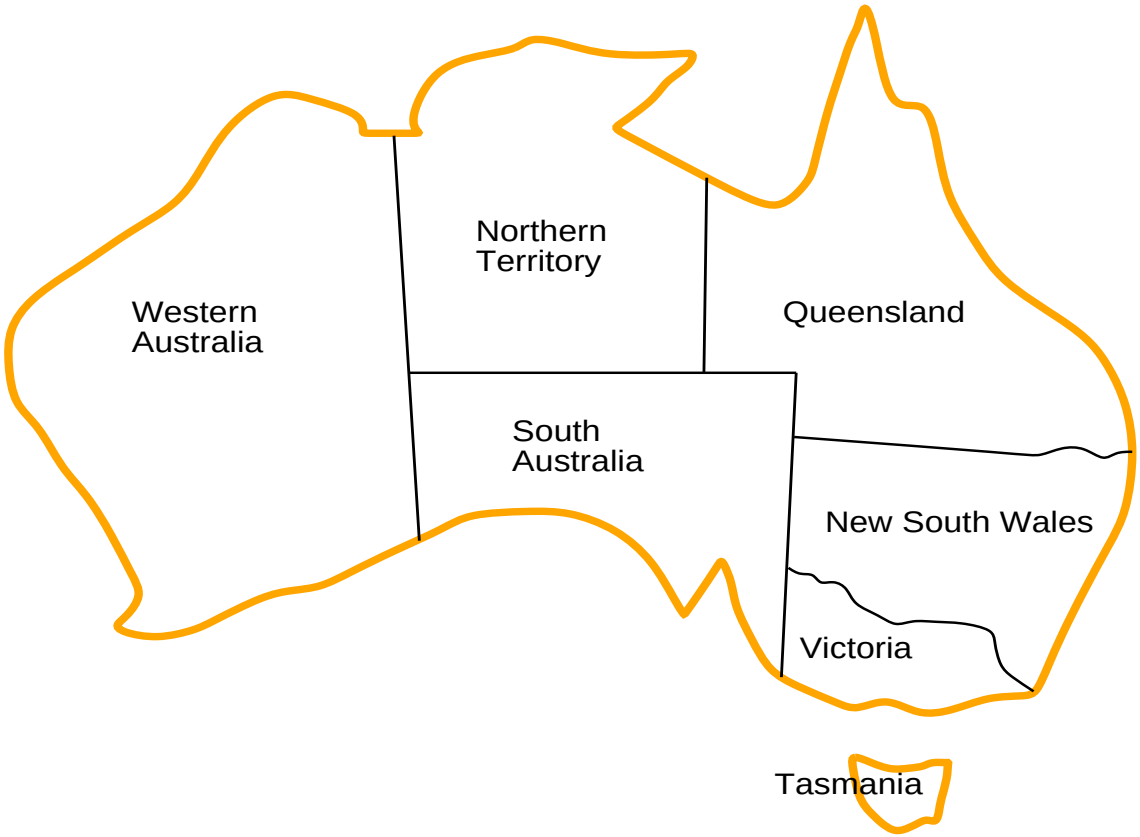
Όταν φτάνουμε σε **αδιέξοδο (dead-end)** για μια μεταβλητή X, ο BJ δεν υπαναχωρεί στην προηγούμενη μεταβλητή όπως ο BT, αλλά στη μεταβλητή που βρίσκεται **βαθύτερα** στο δέντρο αναζήτησης (δηλαδή στην **πιο πρόσφατη** μεταβλητή) που είχε σαν συνέπεια να απαλειφθεί μια τιμή από το πεδίο της X.

Αδιέξοδο - Σύνολο Συγκρούσεων

Είμαστε σε **αδιέξοδο** για μια μεταβλητή X, όταν κάθε δυνατή τιμή της X μαζί με την μέχρι τώρα μερική ανάθεση παραβιάζει κάποιο περιορισμό (συνεπώς απαλείφεται από το πεδίο της X).

Όταν φτάσουμε σε αδιέξοδο για μια μεταβλητή X, το σύνολο των μεταβλητών της μερικής ανάθεσης που προκάλεσαν την απαλειφή όλων των τιμών από το πεδίο της X ονομάζεται **σύνολο συγκρούσεων (conflict set)** της X.

BJ - Παράδειγμα



BJ

Ας εφαρμόσουμε τώρα τον BJ στο πρόβλημα του χρωματισμού χάρτη με την ίδια στατική σειρά επιλογής μεταβλητών που εφαρμόσαμε νωρίτερα για τον BT:

$$Q, NSW, V, T, SA, WA, NT$$

Έστω ότι έχουμε παράγει την ακόλουθη μερική ανάθεση:

$$Q = red,\ NSW = green,\ V = blue,\ T = red$$

Όταν επιλέξουμε την επόμενη μεταβλητή (SA) βλέπουμε ότι κάθε τιμή της παραβιάζει ένα περιορισμό.

Οι προηγούμενες μεταβλητές που προκάλεσαν την απόρριψη όλων των πιθανών τιμών για την SA (δηλαδή τα μέλη του συνόλου συγκρούσεων της SA) είναι οι Q, NSW, V. Τώρα ο BJ **επιβάλλει να υπαναχωρήσουμε στην μεταβλητή V.**

BJ

Ο BJ υπαναχωρεί με άλμα **μόνο όταν βρεθούμε σε αδιέξοδο** που προκύπτει από τις προηγούμενες αναθέσεις.

Για να υλοποιήσουμε τον BJ, φτάνει να **θυμόμαστε τη μεταβλητή που βρίσκεται βαθύτερα στο δένδρο αναζήτησης και προκάλεσε την απόριψη κάποιας τιμής της τρέχουσας μεταβλητής** (δηλ. τη μεταβλητή του συνόλου συγκρούσεων που βρίσκεται βαθύτερα).

Σύγκριση BJ και FC

Όταν ο BJ πραγματοποιεί ένα άλμα προς τα πίσω, όλες οι τιμές ενός πεδίου βρίσκονται σε σύγκρουση με την τρέχουσα ανάθεση. Αυτό το γεγονός θα είχε ανακαλυφθεί **νωρίτερα** από τον FC!

Πρόταση. Κάθε κλαδί του δέντρου αναζήτησης που κλαδεύεται από τον BJ κλαδεύεται και από τον FC.

Συνεπώς, ο BJ είναι περιττός σε μια αναζήτηση που χρησιμοποιεί τον FC ή έναν ισχυρότερο αλγόριθμο διάδοσης περιορισμών όπως τον MAC.

Από τον BJ στον CBJ

Ας θεωρήσουμε και πάλι τον BJ στο πρόβλημα του χρωματισμού χάρτη με την ακόλουθη στατική σειρά επιλογής μεταβλητών:

$$WA, NSW, T, NT, Q, V, SA$$

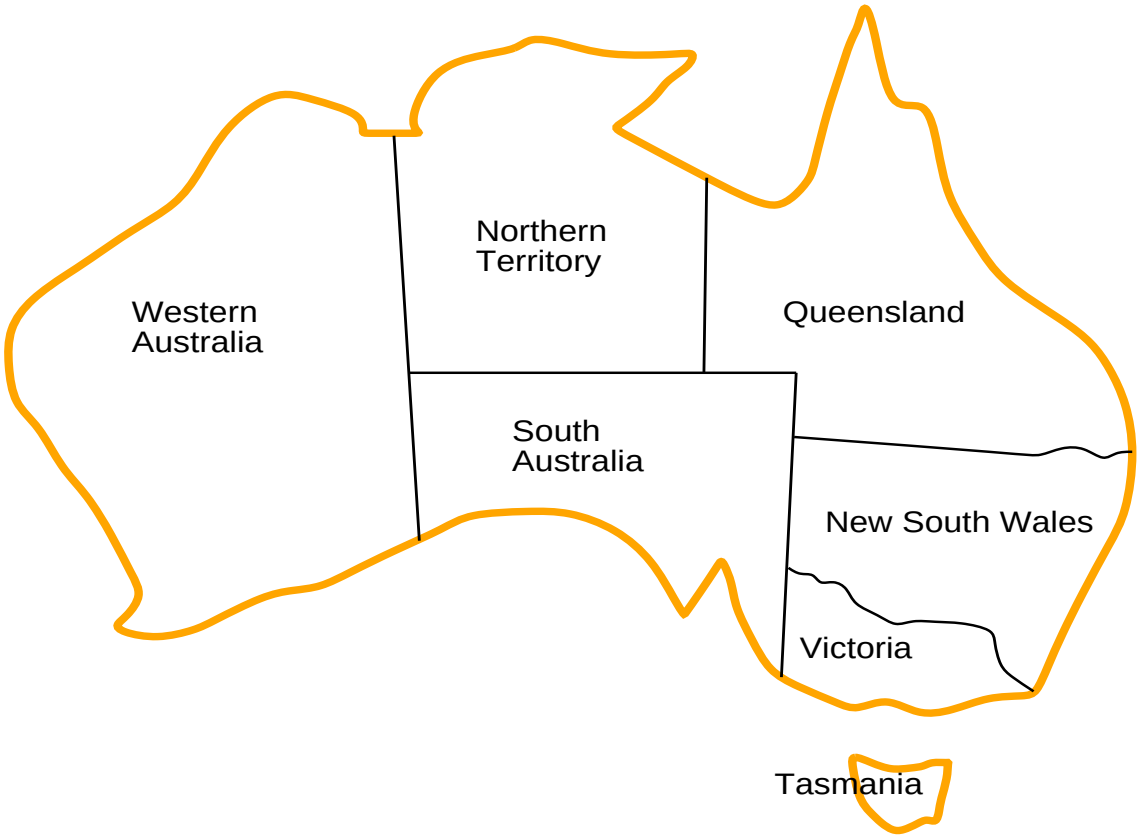
Έστω ότι έχουμε παράγει την εξής μερική ανάθεση:

$$WA = red,\ NSW = red$$

Αυτή η ανάθεση δεν μπορεί να μας οδηγήσει σε λύση.

Αλλά ας κάνουμε την ανάθεση $T = red$, και ας συνεχίσουμε με τις μεταβλητές NT, Q, V, SA. Αυτό δεν θα πετύχει και κατά συνέπεια θα εξαντληθούν οι δυνατές τιμές για την NT. Σε ποιά μεταβλητή θα πρέπει να υπαναχωρήσουμε;

Παράδειγμα


Northern
Territory
Western
Australia
Queensland
South
Australia
New South Wales
Victoria
Tasmania

Από τον BJ στον CBJ

Ο BJ δεν μπορεί να μας βοηθήσει σ΄ αυτή την περίπτωση επειδή **το σύνολο συγκρούσεων της NT είναι κενό**.

Αυτό συμβαίνει γιατί η NT έχει τιμές που είναι συνεπείς με όλες τις προηγούμενες μεταβλητές (αν και καμία από αυτές τις τιμές δεν μπορεί να μας οδηγήσει σε λύση), άρα **δεν είμαστε σε αδιέξοδο.**

Προσέξτε ότι σ΄ αυτή την περίπτωση, οι μεταβλητές NT, Q, V, SA **σαν σύνολο** έρχονται σε σύγκρουση με τις προηγούμενες μεταβλητές.

Από τον BJ στον CBJ

Αυτό οδηγεί σε μια βαθύτερη έννοια του **συνόλου συγκρούσεων** μιας μεταβλητής X: είναι το σύνολο των **προηγούμενων μεταβλητών** που οδήγησε τη μεταβλητή X, **μαζί με οποιεσδήποτε επόμενες μεταβλητές**, σε αποτυχία.

Με το νέο ορισμό, το σύνολο συγκρούσεων για την NT είναι $\{WA, NSW\}$ και θα πρέπει να υπαναχωρήσουμε στην NSW.

Αυτή είναι η βασική ιδέα του αλγόριθμου **υπαναχώρησης με άλμα κατευθυνόμενο από σύγκρουση (conflict-directed backjumping - CBJ).**

CBJ

Για την υλοποίηση του αλγόριθμου CBJ, κάθε μεταβλητή έχει ένα **σύνολο συγκρούσεων** που υπολογίζεται ως εξής:

- Όταν η ανάθεση τιμής v_i στην τρέχουσα μεταβλητή X_i είναι ασυνεπής με την τιμή v_k που έχει ανατεθεί σε μια προηγούμενη μεταβλητή X_k, τότε η X_k προστίθεται στο **σύνολο συγκρούσεων** της X_i.
- Όταν δεν υπάρχουν άλλες τιμές προς δοκιμή για την X_i, ο CBJ υπαναχωρεί προς τη μεταβλητή X_h του συνόλου συγκρούσεων της X_i που βρίσκεται **βαθύτερα** στο δέντρο αναζήτησης. Ταυτόχρονα, οι μεταβλητές στο σύνολο συγκρούσεων της X_i (εκτός από την X_h) **προστίθενται στο σύνολο συγκρούσεων** της X_h, έτσι ώστε να μη χαθεί πληροφορία για συγκρούσεις. Αυτή είναι η σημαντικότερη διαφορά του CBJ από τον BJ.

Εκτέλεση του CBJ στο Παράδειγμά μας

Σειρά Επιλογής Μεταβλητών: WA, NSW, T, NT, Q, V, SA

Σειρά Επιλογής Τιμών: $red,\ blue,\ green$

Ο CBJ εκτελείται ως εξής (παρουσιάζουμε τις τιμές που ανατίθενται στις μεταβλητές και τα σύνολα συγκρούσεων):

- $WA = red$
- $NSW = red$
- $T = red$
- $NT = blue,\ CS(NT) = \{WA\}$
- $Q = green,\ CS(Q) = \{NSW, NT\}$
- $V = blue,\ CS(V) = \{NSW\}$
- Η SA δεν μπορεί να πάρει κάποια τιμή και $CS(SA) = \{NSW, V, Q\}$.

  Υπαναχωρούμε στην V (τη μεταβλητή του συνόλου συγκρούσεων που είναι βαθύτερα στο δέντρο αναζήτησης). Το $CS(V)$ γίνεται $\{NSW, Q\}$.

Εκτέλεση του CBJ στο Παράδειγμά μας

Σειρά Επιλογής Μεταβλητών: WA, NSW, T, NT, Q, V, SA

Σειρά Επιλογής Τιμών: $red,\ blue,\ green$

Ο CBJ συνεχίζει ως εξής:

- $V = green,\ CS(V) = \{NSW, Q\}$
- Η SA δεν μπορεί να πάρει καμια τιμή και $CS(SA) = \{NSW, NT, V\}$. Υπαναχωρούμε στην V. Το $CS(V)$ γίνεται $\{NSW, NT, Q\}$.
- Η V δεν έχει καμιά άλλη διαθέσιμη τιμή και $CS(V) = \{NSW, NT, Q\}$. Υπαναχωρούμε στην Q. Το $CS(Q)$ γίνεται $\{NSW, NT\}$.

Εκτέλεση του CBJ για το Παράδειγμά μας

Σειρά Επιλογής Μεταβλητών: WA, NSW, T, NT, Q, V, SA

Σειρά Επιλογής Τιμών: $red,\ blue,\ green$

Ο CBJ συνεχίζει ως εξής:

- Δεν υπάρχουν άλλες τιμές προς δοκιμή για την Q και $CS(Q) = \{NSW, NT\}$.

  Υπαναχωρούμε στην NT. Το $CS(NT)$ γίνεται $\{NSW, WA\}$.

- $NT = green,\ CS(NT) = \{WA, NSW\}$.

  Τώρα οι μεταβλητές Q, V, SA θα δοκιμαστούν με παρόμοιο τρόπο και δεν θα βρεθεί λύση.

  Όταν ξαναγυρίσουμε πίσω στην NT για την οποία δεν υπάρχουν άλλες δυνατές τιμές, **υπαναχωρούμε στην** NSW (στη μεταβλητή του συνόλου συγκρούσεων της NT που βρίσκεται βαθύτερα στο δένδρο αναζήτησης) και όχι στην T.

Υβριδικοί Αλγόριθμοι

Οι αλγόριθμοι υπαναχώρησης που παρουσιάσαμε μέχρι τώρα ανήκουν σε δύο κατηγορίες:

- Αλγόριθμοι διάδοσης περιορισμών (FC, MAC)
- Αλγόριθμοι ευφυούς υπαναχώρησης (BJ, CBJ)

Οι τεχνικές των δύο κατηγοριών μπορούν να συνδυαστούν για να αναπτύξουμε **υβριδικούς αλγόριθμους υπαναχώρησης**, για παράδειγμα, τον FC-CBJ ή τον MAC-CBJ.

Οι ευρετικοί μηχανισμοί που παρουσιάσαμε μπορούν να συνδυαστούν και μ΄ αυτούς τους υβριδικούς αλγόριθμους.

Μάθηση Περιορισμών

Ο αλγόριθμος CBJ μας ξαναγυρίζει στο σωστό σημείο του δένδρου αναζήτησης που είναι υπεύθυνο για την αποτυχία εύρεσης λύσης. Όμως δεν μας εμποδίζει να κάνουμε τα **ίδια λάθη σε άλλα κλαδιά** του δέντρου αναζήτησης.

Η μέθοδος της **μάθησης περιορισμών (constraint learning)** τροποποιεί το δοσμένο CSP προσθέτοντας περιορισμούς που προκύπτουν από (όσο το δυνατόν πιο μικρά) σύνολα συγκρούσεων με την ελπίδα ότι αυτοί οι περιορισμοί θα αποβούν χρήσιμοι στη συνέχεια.

Δεν θα παρουσιάσουμε τις λεπτομέρειες των σχετικών αλγορίθμων. Αν το θέμα σας ενδιαφέρει, δείτε το Κεφάλαιο 6 του βιβλίου

Rina Dechter. *Constraint Processing.* Morgan Kaufmann. 2003

Αξιολόγηση Αλγορίθμων Υπαναχώρησης

Κριτήρια:

1. Χρονική/χωρική πολυπλοκότητα χειρίστης περίπτωσης
2. Χρόνος εκτέλεσης
3. Πλήθος κόμβων που επισκεπτόμαστε στο δέντρο αναζήτησης
4. Πλήθος ελέγχων συνέπειας που πραγματοποιούνται

Αξιολόγηση Αλγόριθμων Υπαναχώρησης

Αποτελέσματα:

- Οι BT, FC, BJ, CBJ, MAC και οι παραλλαγές τους έχουν **εκθετική χρονική πολυπλοκότητα** στη χειρότερη περίπτωση.
- Ο χρόνος εκτέλεσης μπορεί να ποικίλει λόγω των λεπτομερειών κάθε υλοποίησης.
- Πλήθος κόμβων που επισκεπτόμαστε:

$$\text{FC-CBJ} \leq \text{FC} \leq \text{BJ} \leq \text{BT}$$

$$\text{CBJ} \leq \text{BJ}$$

$$\text{MAC-CBJ} \leq \text{MAC} \leq \text{FC}$$

Αξιολόγηση Αλγόριθμων Υπαναχώρησης

- Πλήθος ελέγχων συνέπειας που πραγματοποιούνται:

$$\text{CBJ} \leq \text{BJ} \leq \text{BT}$$

$$\text{FC-CBJ} \leq \text{FC}$$

  Ο FC μπορεί να πραγματοποιήσει περισσότερους ή λιγότερους ελέγχους από τον BJ και τον BT ανάλογα με το πρόβλημα.

- Πειραματικά αποτελέσματα έχουν δείξει ότι στις περισσότερες περιπτώσεις ένας καλός αλγόριθμος διάδοσης περιορισμών (όπως ο MAC ή ο FC) σε συνδυασμό με ένα καλό σύνολο ευρετικών μηχανισμών (όπως ο MRV και ο LCV) μπορεί να λύσει δύσκολα CSP με επιτυχία.

Σχετικά Δημοσιεύματα και Λογισμικό

- Grzegorz Kondrak and Peter van Beek. A theoretical evaluation of selected backtracking algorithms. Artificial Intelligence, 89:365-387, 1997.
- Xinguang Chen and Peter van Beek. Conflict-directed backjumping revisited. Journal of Artificial Intelligence Research, 14:53-81, 2001.
- `csplib`: Μια βιβλιοθήκη συναρτήσεων της C για την επίλυση δυαδικών CSP.

Τα παραπάνω είναι διαθέσιμα στην ιστοσελίδα του Peter van Beek:
`http://ai.uwaterloo.ca/~vanbeek`

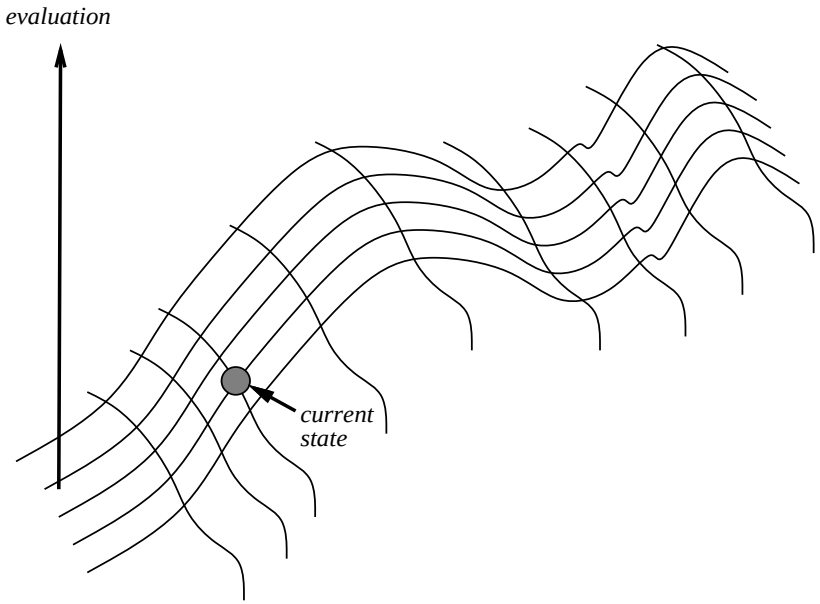
Αλγόριθμοι Τοπικής Αναζήτησης για CSP

Όλοι οι αλγόριθμοι υπαναχώρησης που παρουσιάσαμε ως τώρα είναι **συστηματικοί**: εξερευνούν το χώρο αναζήτησης προσεκτικά κρατώντας πληροφορίες για κάθε διαδρομή που εξερεύνησαν μέχρι να βρουν μια λύση.

Έχουμε ήδη δει πως επιλύουμε προβλήματα αναζήτησης (π.χ., το πρόβλημα των n-βασιλισσών) χρησιμοποιώντας τοπική αναζήτηση. Μπορούμε να λύσουμε **οποιοδήποτε CSP** με αλγόριθμους τοπικής αναζήτησης;

Αλγόριθμοι Τοπικής Αναζήτησης

Ιδέα: Ξεκινάμε με μια 'λύση' και εκτελούμε μετατροπές μέχρι να φτάσουμε σε μια λύση. Γραφικά:



Αλγόριθμοι Τοπικής Αναζήτησης για CSP

Η τοπική αναζήτηση είναι ιδιαίτερα χρήσιμη για επίλυση CSP. Ένας αλγόριθμος τοπικής αναζήτησης ξεκινά με μια τυχαία ανάθεση τιμών σε μεταβλητές και στη συνέχεια μεταβάλλει (**επιδιορθώνει**) αυτή την ανάθεση μέχρι να γίνει λύση.

Αυτοί οι αλγόριθμοι αναφέρονται και ως αλγόριθμοι **ευρετικής επιδιόρθωσης (heuristic repair)** στη βιβλιογραφία.

Έχουμε ήδη δει την εφαρμογή του αλγόριθμου αναρρίχησης λόφων στον πρόβλημα των 8 βασιλισσών.

Ο Ευρετικός Μηχανισμός των Ελάχιστων Συγκρούσεων

Όταν επιλέγουμε μια νέα τιμή για μια μεταβλητή, ένας χρήσιμος ευρετικός μηχανισμός είναι να **επιλέγουμε αυτή που θα προκαλέσει τον ελάχιστο αριθμό συγκρούσεων με την τρέχουσα ανάθεση στις άλλες μεταβλητές** δηλαδή αυτή που θα δώσει **τον ελάχιστο αριθμό μεταβλητών που θα χρειαστεί να 'επιδιορθωθούν".**

Ο παραπάνω μηχανισμός ονομάζεται **ευρετικός μηχανισμός των ελάχιστων συγκρούσεων (min-conflicts heuristic)** και είναι εκπληκτικά ισχυρός για πολλά CSP.

Min-Conflicts

function MIN-CONFLICTS(*csp*, *max-steps*) **returns** a solution or failure

inputs: *csp*, a constraint satisfaction problem
max-steps, the number of steps allowed before giving up

local variables: *current*, a complete assignment
var, a variable
value, a value for a variable

current ← an initial complete assignment for *csp*
for $i = 1$ to *max-steps* **do**
  if *current* is a solution for *csp* **then return** *current*
  var ← a randomly chosen, conflicted variable from VARIABLES[*csp*]
  value ← the value *v* for *var* that minimizes CONFLICTS(*var*, *v*, *current*, *csp*)
  set *var*=*value* in *current*
return *failure*

Min-Conflicts

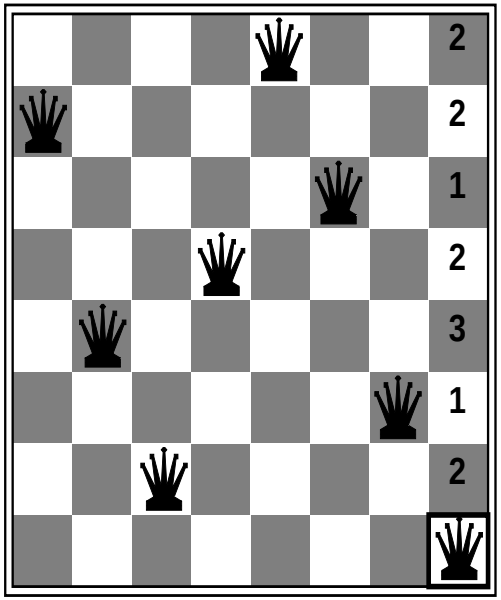
Ορισμός: Δύο μεταβλητές **βρίσκονται σε σύγκρουση** αν οι τιμές τους παραβιάζουν ένα περιορισμό.

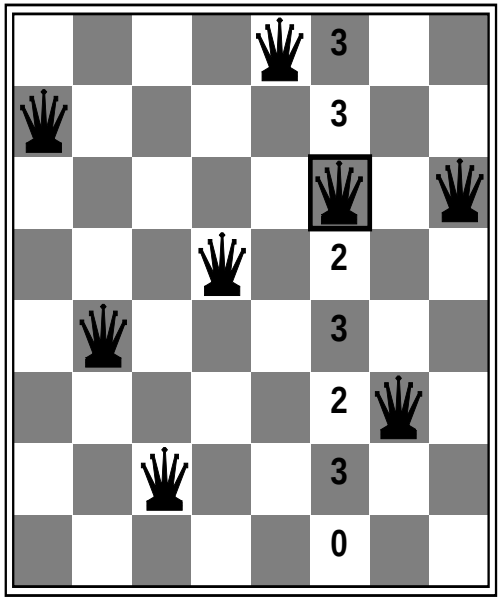
Η συνάρτηση CONFLICTS$(var, v, current, csp)$ υπολογίζει τον **αριθμό των συγκρούσεων** από την ανάθεση της τιμής v στη μεταβλητή var, με δεδομένη την υπόλοιπη τρέχουσα ανάθεση τιμών.

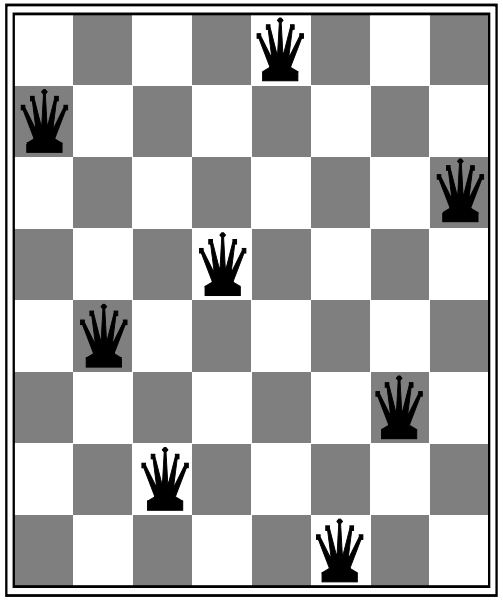
Για δυαδικά CSP, αυτός ο αριθμός είναι **ο αριθμός των μεταβλητών που βρίσκονται σε σύγκρουση με την** var **όταν** $var = v$.

Για γενικά CSP, η μέθοδος μέτρησης των μεταβλητών που θα χρειαστεί να επιδιορθωθούν εξαρτάται από τη μορφή των περιορισμών.

Παράδειγμα







Εφαρμογή στο Πρόβλημα των N Βασιλισσών

Ξεκινώντας από μια καλή αρχική κατάσταση, ο ευρετικός μηχανισμός των ελάχιστων συγκρούσεων μπορεί να λύσει προβλήματα με **εκατομμύρια βασίλισσες** σε περίπου 50 βήματα (στην πραγματικότητα, ο χρόνος εκτέλεσής του είναι σχεδόν **ανεξάρτητος από το μέγεθος του προβλήματος**).

Μια καλή **αρχική κατάσταση** υπολογίζεται διατρέχοντας τις στήλες της σκακιέρας (μεταβλητές) και τοποθετώντας κάθε βασίλισσα στη γραμμή (τιμή) όπου συγκρούεται με τον ελάχιστο αριθμό ήδη τοποθετημένων βασιλισσών.

Η επιδιόρθωση μπορεί να πραγματοποιηθεί σε χρόνο $O(n)$ διατηρώντας μια λίστα από όλες τις βασίλισσες που βρίσκονται σε σύγκρουση (δηλαδή, απειλούνται από άλλες) μαζί με μετρητές που δείχνουν το πλήθος των επιτιθέμενων βασιλισσών για κάθε εναλλακτική θέση αυτών των βασιλισσών.

Εφαρμογή στο Πρόβλημα των N Βασιλισσών

Παρατήρηση: Το πρόβλημα των N βασιλισσών είναι **εύκολο για την τοπική αναζήτηση** επειδή οι λύσεις είναι πυκνά κατανεμημένες στο χώρο καταστάσεων.

Όμως, ο ευρετικός μηχανισμός των ελάχιστων συγκρούσεων μπορεί να έχει πολύ καλά αποτελέσματα και για δύσκολα CSP.

Αξιολόγηση

Problem	**BT**	**BT+MRV**	**FC**	**FC+MRV**	**Min-con**
USA	(>1000K)	(>1000K)	2K	60	64
n-Queens	(>40000K)	13500K	(>40000K)	817K	4K
Zebra	3859K	1K	35K	0.5K	2K

Προβλήματα Χρονοπρογραμματισμού

Ο ευρετικός μηχανισμός των ελάχιστων συγκρούσεων έχει χρησιμοποιηθεί σε αλγόριθμους χρονοπρογραμματισμού παρατηρήσεων του τηλεσκόπιου Hubble (`http://hubblesite.org/`) της NASA ελαχιστοποιώντας το χρόνο χρονοπρογραμματισμού από 3 εβδομάδες σε 10 λεπτά!

Για λεπτομέρειες δείτε τη δημοσίευση

> Steven Minton, Mark D. Johnston, Andrew B. Philips and Philip Laird. *Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems.* Artificial Intelligence 58(1-3), pages 161-205 (1992).

On-line Περιβάλλοντα

Οι αλγόριθμοι τοπικής αναζήτησης είναι σημαντικοί για προβλήματα CSP επειδή μπορούν να χρησιμοποιηθούν σε on-line περιβάλλοντα όπου το πρόβλημα αλλάζει.

Παράδειγμα: Χρονοπρογραμματισμός και επανα-χρονοπρογραμματισμός.

Οι τεχνικές τοπικής αναζήτησης τις οποίες έχουμε μελετήσει (αναρρίχηση λόφων, προσομοιωμένη ανόπτηση, κ.λ.π.) μπορούν επίσης να εφαρμοστούν και σε **προβλήματα βελτιστοποίησης περιορισμών (constraint optimization problems)**.

Η Δομή ενός CSP

Υπάρχουν τρόποι να εκμεταλλευτούμε τη **δομή ενός CSP** για να βρούμε λύσεις γρήγορα:

- Αποσύνθεση σε **ανεξάρτητα υποπροβλήματα**
- Έλεγχος αν το CSP είναι **δενδροειδές** και χρήση της συνέπειας κατευθυνόμενης ακμής
- **Αναγωγή γενικών γράφων περιορισμών σε δένδρα**:
  - Ανάθεση τιμών σε κάποιες μεταβλητές έτσι ώστε οι υπόλοιπες μεταβλητές να σχηματίζουν ένα δενδροειδές CSP.
  - Κατασκευή μιας αποσύνθεσης δένδρου του γράφου περιορισμών και μετατροπή του σε σύνολο συνδεδεμένων υποπροβλημάτων.

Αποσύνθεση σε Ανεξάρτητα Υποπροβλήματα

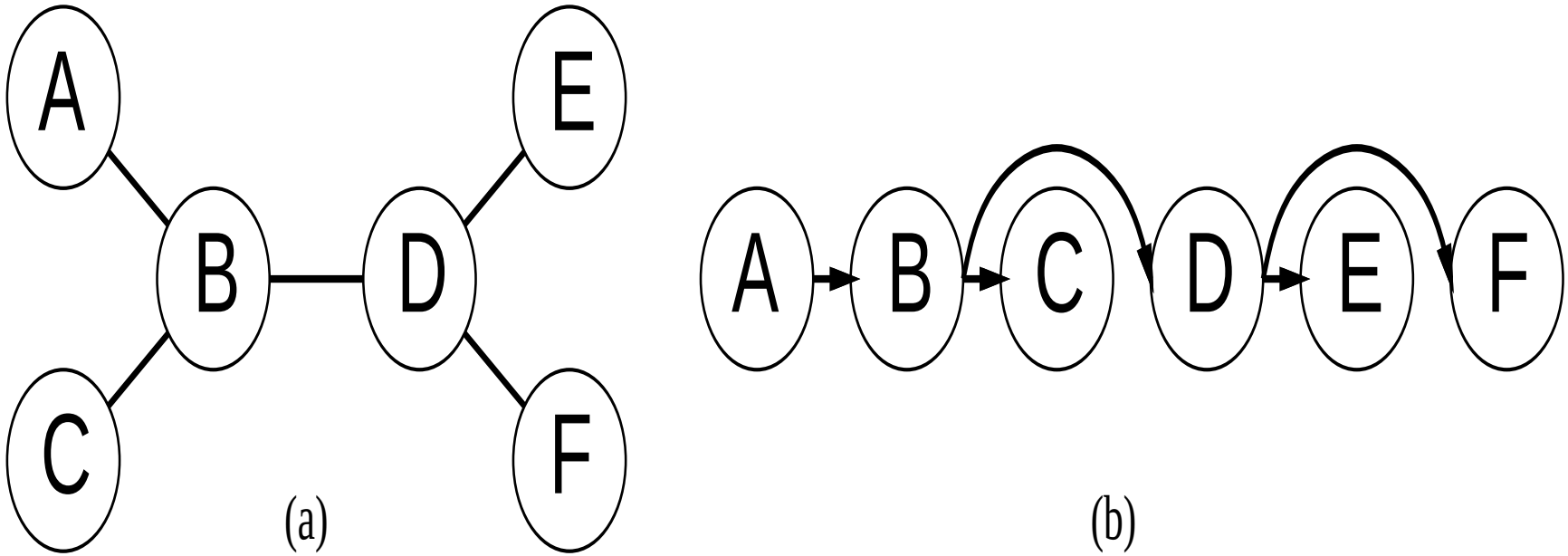
Αυτή η μέθοδος λειτουργεί ως εξής:

- Αναζητούμε **συνδεδεμένες συνιστώσες (connected components** στο γράφο περιορισμών. Κάθε τέτοια συνιστώσα μας καθορίζει ένα υποπρόβλημα CSP_i.

  Παράδειγμα: Ο χρωματισμός της Τασμανίας είναι ένα υποπρόβλημα του προβλήματος χρωματισμού του χάρτη της Αυστραλίας που είναι ανεξάρτητο από το πρόβλημα χρωματισμού της υπόλοιπης Αυστραλίας.

- Αν κάθε CSP_i έχει c μεταβλητές, τότε η επίλυση κάθε υποπροβλήματος ξεχωριστά και στη συνέχεια ο συνδυασμός των λύσεων απαιτεί χρόνο $O(d^c \cdot n/c)$ (γραμμικό στο n), ενώ η λύση του αρχικού προβλήματος απαιτεί χρόνο $O(d^n)$ (εκθετικό στο n).

Δενδροειδή CSP



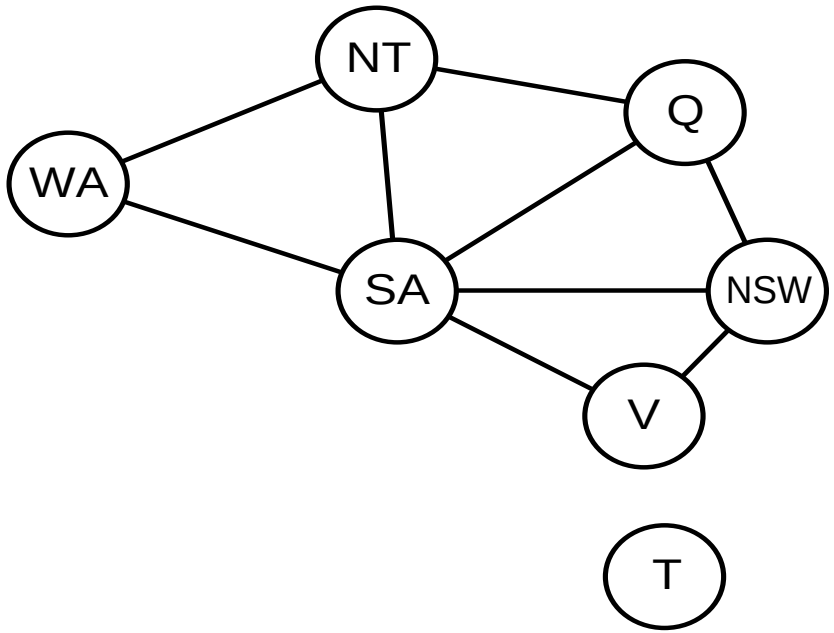
Στα δενδροειδή CSP ο γράφος περιορισμών έχει τη μορφή δένδρου (δηλαδή, δύο οποιεσδήποτε μεταβλητές συνδέονται με το πολύ ένα μονοπάτι).

Δενδροειδή CSP και Συνέπεια Κατευθυνόμενης Ακμής

Ένα δενδροειδές CSP μπορεί να λυθεί σε χρόνο $O(nd^2)$ (**γραμμικό** ως προς το πλήθος των μεταβλητών) ως εξής:

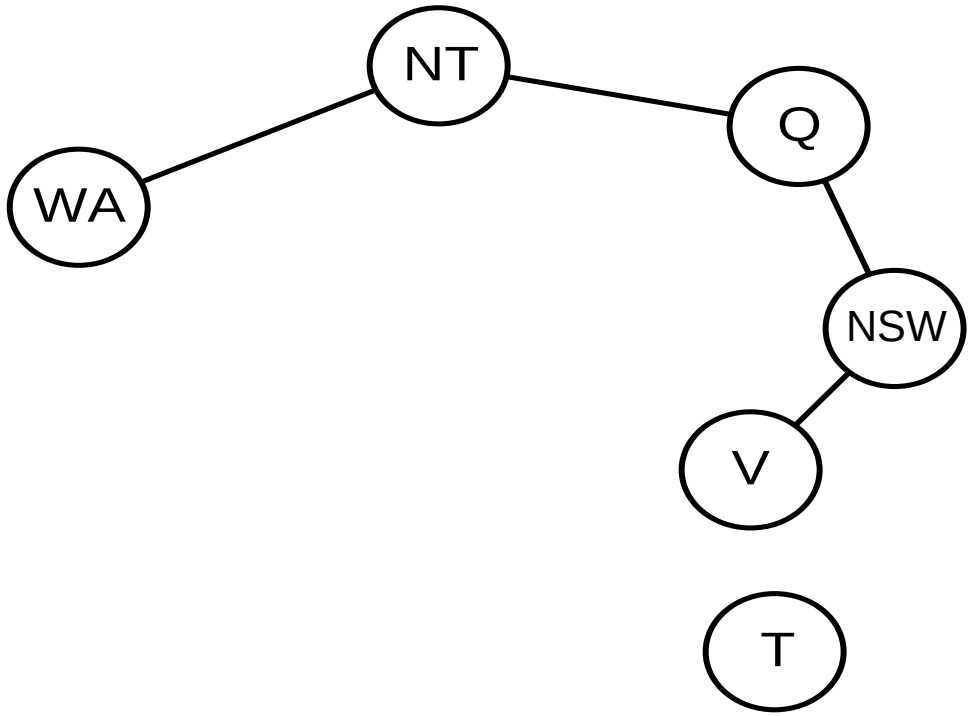
- Επιλέγουμε οποιαδήποτε μεταβλητή ως ρίζα του δέντρου και διατάσσουμε τις μεταβλητές με τη σειρά $X_1, \ldots, X_n$ από τη ρίζα προς τα φύλλα, με τέτοιο τρόπο ώστε ο γονέας κάθε κόμβου στο δέντρο να βρίσκεται πριν από αυτόν στη διάταξη.
- Για $j = n$ downto 2, εφαρμόζουμε συνέπεια ακμής στην ακμή (X_i, X_j), όπου X_i είναι ο γονέας του X_j, απαλείφοντας τιμές από το πεδίο της X_i όταν είναι απαραίτητο. Αυτή η διαδικασία ονομάζεται **συνέπεια κατευθυνόμενης ακμής.**
- Για $j = 1$ to n, αναθέτουμε στο X_j οποιαδήποτε τιμή που είναι συνεπής με την τιμή που ανατέθηκε στο X_i, όπου το X_i είναι ο γονέας του X_j.

Αναγωγή Γράφων Περιορισμών σε Δέντρα



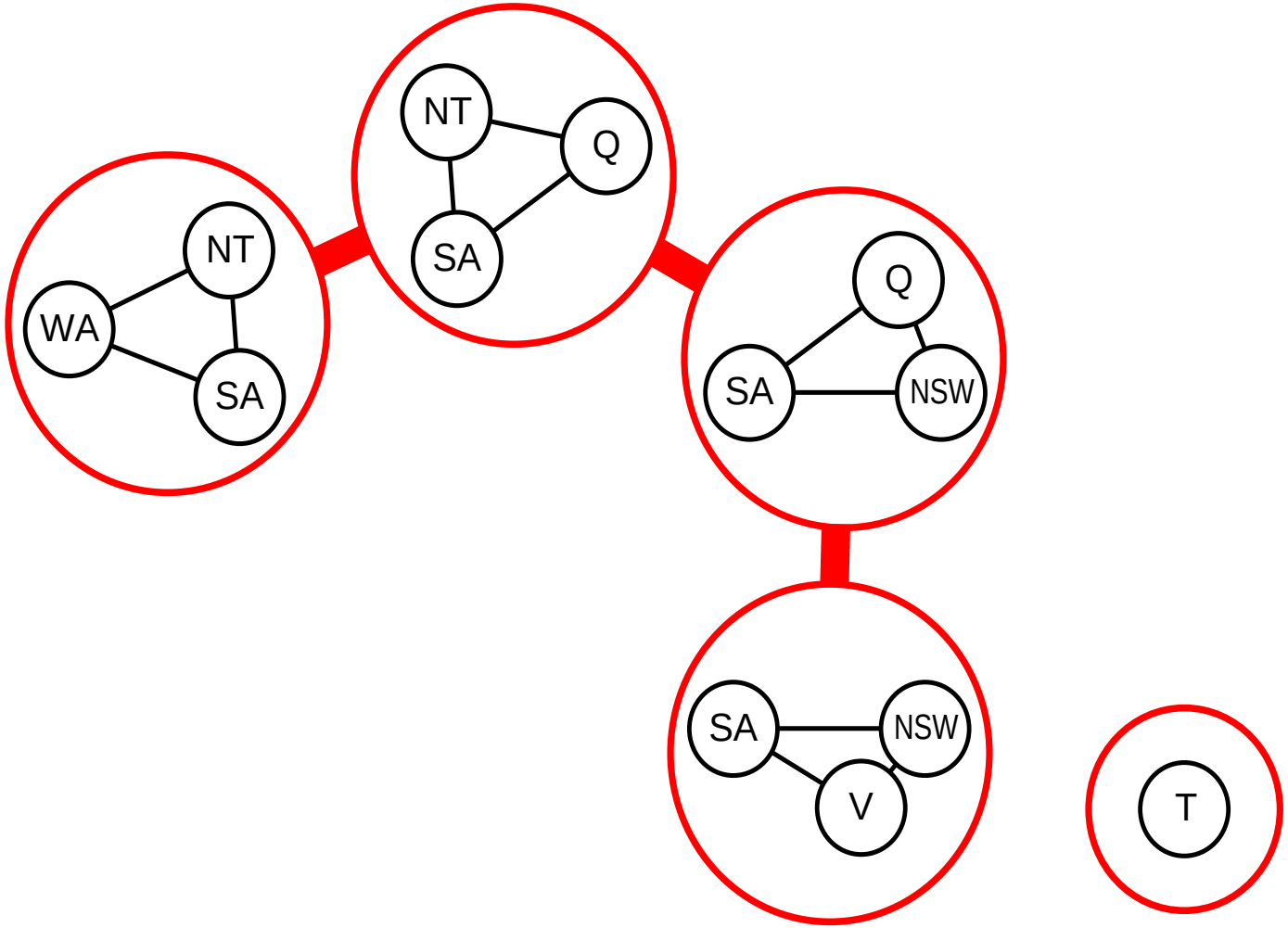
Βήμα 1: Αναθέτουμε μια τιμή στη μεταβλητή SA, αφαιρούμε τις ασυνεπείς τιμές από τις άλλες μεταβλητές και στη συνέχεια αφαιρούμε τον κόμβο SA από το γράφο.

Αναγωγή Γράφων Περιορισμών σε Δέντρα



Βήμα 2: Λύνουμε το υποπρόβλημα που μένει με χρήση αλγορίθμων για δενδροειδή CSP.

Αποσύνθεση Δένδρου (Tree Decomposition)



Μελέτη

Κεφάλαιο 5 του AIMA.