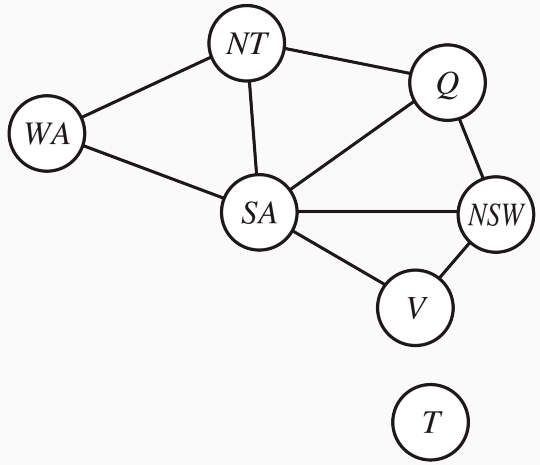
Backtracking++

Joó András, andrasjoo@ms.sapientia.ro

Sapientia–EMTE, Marosvásárhely

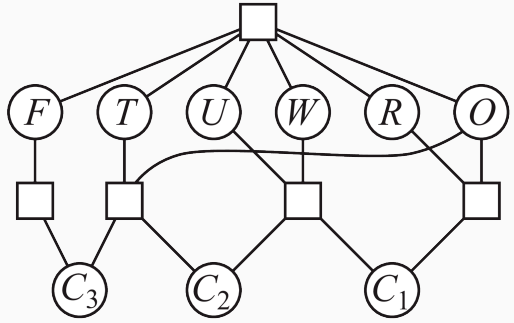
Térképszínezés



- "fessük ki a térképet három színnel, úgy, hogy két szomszédos tartomány ne legyen azonos színű"
- **változók**: $\{WA, NT, SA, Q, NSW, V, T\}$
- **doméniumok**: {red, green, blue}
- **bináris kényszerek**: $\{WA \neq NT, WA \neq SA, NT \neq SA, \ldots\}$
- **kényszer-gráf** (constraint graph)

Betűaritmetika

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



- "adjunk egymástól különböző, egyszámjegyű értékeket a változóknak, úgy, hogy az összeadás helyes legyen"
- változók: F, T, U, W, R, O
- doménium, minden változóra: {0, 1, ... 9}
- **n-áris** kényszerek:
  1. **AllDifferent**(F, T, U, W, R, O)
  2. $O + O = R + 10 \cdot C_1$
  3. $C_1 + W + W = U + 10 \cdot C_2$
  4. $C_2 + T + T = O + 10 \cdot C_3$
  5. $C_3 = F$
- **kényszer-hipergráf**

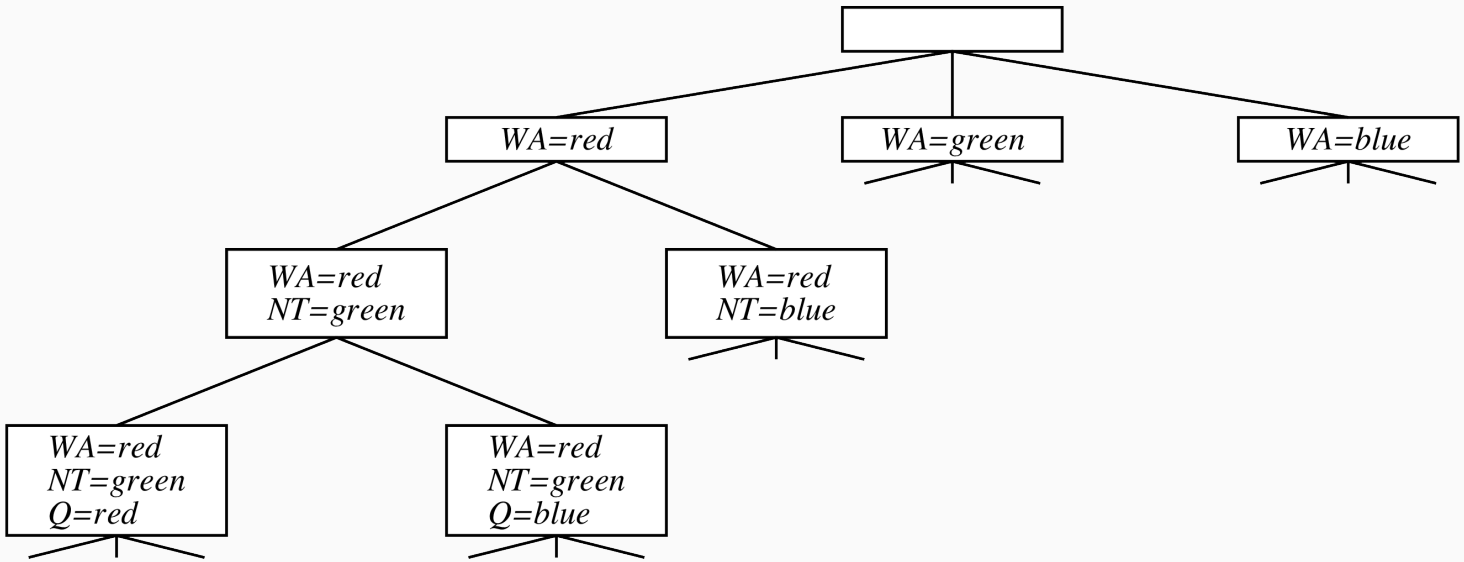
Kényszerkielégítési feladatok (constraint satisfaction problems)

- CSP $\equiv \left(\{X_i\}, \{D_j\}, \{C_k\}\right)$
- X_i változó
- D_j doménium: diszkrét/folytonos, véges/végtelen
- C_k kényszer: logikai/lineáris/nemlineáris, unáris/bináris/n-áris
- hozzárendelés ($\equiv$ állapot): konzisztens, teljes
- megoldás $\equiv$ konzisztens és teljes hozzárendelés
- opcionális: célfüggvény $\rightarrow$ min, preferenciális kényszerek
- feladat: megtalálni egy vagy az összes megoldást
- példák: n királynő, sudoku, órarend, ütemezés, logisztika

`job-shop`

Backtracking (1)

- olyan mélységi keresés, amely
- egyszerre egy változónak választ értéket és
- visszalép abban az esetben, ha nincs a kényszerek miatt hozzárendelhető értéke a változónak



- keresési fa, stack, komplexitás?

Backtracking (2)

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

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 $failure$

Változók és értékek kiválasztásának sorrendje

- milyen sorrendben vegyük a
  - változókat
  - a változók értékeit?
- … úgy, hogy:
  1. ✂ minél gyorsabban azonosítsuk, ha az aktuális részfa nem tartalmaz megoldást
  2. 🍀 a legígéretesebb részfába minél előbb menjünk bele

Változók kiválasztásának sorrendje (1)

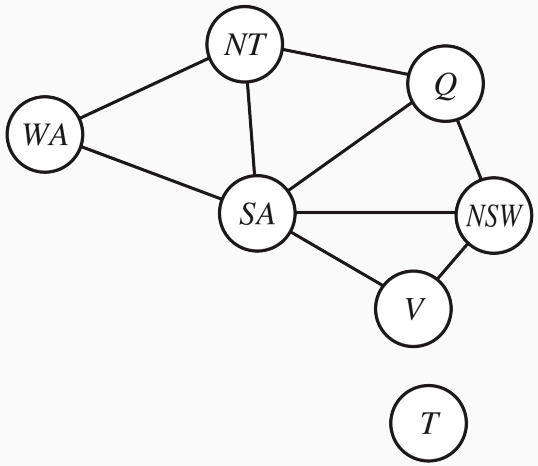
✂ Minimum remaining values (fail-first, MVR)

- azt a változót választjuk, amelynek a legkevesebb legális értéke van
- példa: $D_X = \{v_1, v_2, v_3\}$, $D_Y = \{v_1\}$, $D_Z = \varnothing$

Változók kiválasztásának sorrendje (2)

✂ A legelső változó kiválasztása

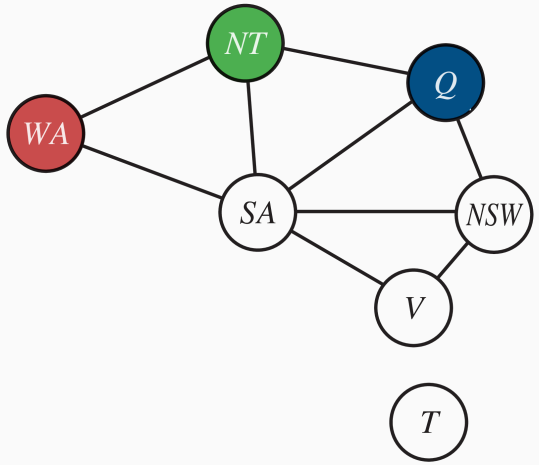
- azonos méretű doménium esetén a legerősebben kényszerített változót választjuk (degree heuristics)



Az értékek kiválasztásának sorrendje

Milyen sorrendben menjünk végig a változó értékein?

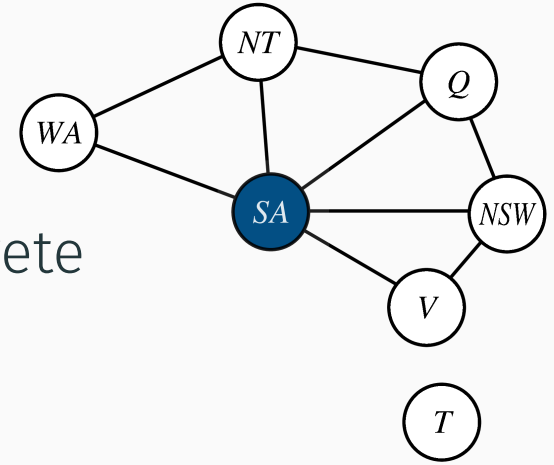
- 1. WA=red, 2. NT=green, 3. Q = blue 4. SA = ?



- 🍀 legkevésbé kényszerített változó heurisztika (least constraining value): azt értéket kezeli preferenciálisan, amely a szomszédos változók értékei közül a legkevesebbet szorítja ki
- NB: a sorrend közömbös, ha minden megoldást ki kell generálni

A kényszer kiterjesztése

A kényszer kiterjesztése (constraint propagation)

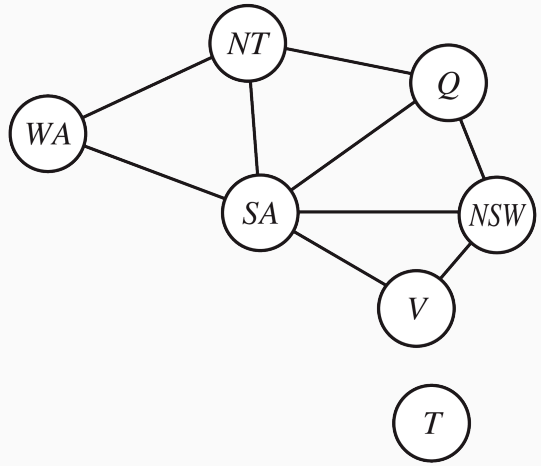


- $SA = blue$
- a (WA, NT, Q, NSW, V) keresési terének mérete
  - egyszerű kereséssel: $3^5 = 243$
  - a kényszer **kiterjesztésével**: $2^5 = 32$
- ha egy változó doméniumából értéket törlünk le, akkor az befolyásolja a vele egy kényszerben levő összes változó doméniumát ...
- ... tehát ezeket a doméniumokat is meg lehet nyesni, ami további doméniumokat egyszerűsítését teszi lehetővé
- **ha valamelyik változó doméniuma kiürül, akkor az illető részfa biztos nem tartalmaz megoldást**
- a kényszer kiterjesztését használhatjuk előfeldolgozóként, vagy keresés közben

Előretekintő ellenőrzés (forward checking)

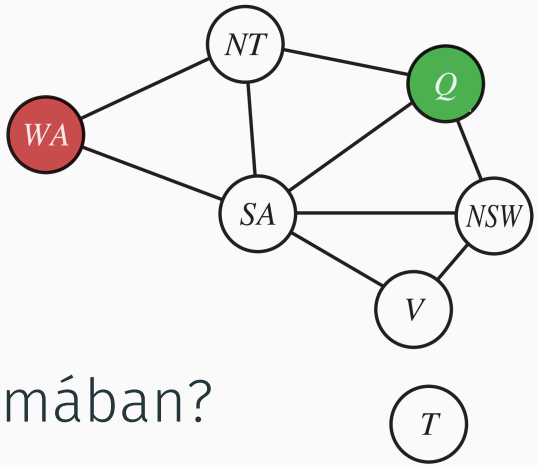
- ha egy *X* változó *x* értéket kap, az előretekintő ellenőrzés megnézi az összes olyan *Y* változót, amellyel *X* egy kényszer által össze van kötve és kitörli *Y* doméniumából azokat, amelyek nem konzisztensek az *x*-szel
- fontos: ne felejtsük el visszaépíteni a doméniumokat a rekurzióból való visszalépéskor
- fontos: a doméniumból való törlés / doméniumba való visszaszúrás **gyors** kell legyen

	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



Előretekintő ellenőrzés (2)

- az előretekintő ellenőrzés csak egy lépést lát előre, például:
  1. WA = red
  2. fwcheck(WA)
  3. Q = green
  4. fwcheck(Q)



- mi marad az NT és az SA doméniumában?

Él-konzisztencia (arc consistency)

- egy $X \to Y$ **irányított** él konzisztens, ha X bármely lehetséges x értékére létezik egy y érték az Y doméniumából, amely konzisztens x-el
- példa: legyen $D_{SA} = \{\text{red}\}$, $D_{NSW} = \{\text{red, blue}\}$, akkor
  - az $SA \to NSW$ él konzisztens
  - az $NSW \to SA$ él inkonzisztens
- egy CSP élkonzisztens, ha a kényszer-gráf minden $X \to Y$ éle konzisztens
- az élkonzisztenciát biztosító algoritmust (AC-3) használhatjuk előfeldolgozóként, vagy keresés közben

AC-3 algoritmus

Arc Consistency 3 (AC-3)

- ha egy X $\rightarrow$ Y él nem konzisztens, akkor töröljük le az X dóméniumából azokat az elemeket, amelyeknek nincs konzisztens párjuk az Y dóméniumában
- végezzük el az előbbi műveletet minden élre a kényszer-gráfban
- a törléssel újabb él-inkonzisztenciák jöhetnek létre
- ismételjük a fenti lépéseket mindaddig, amíg a kényszer-gráf él-konzisztenssé válik

AC-3 algoritmus (2)

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 $queue$

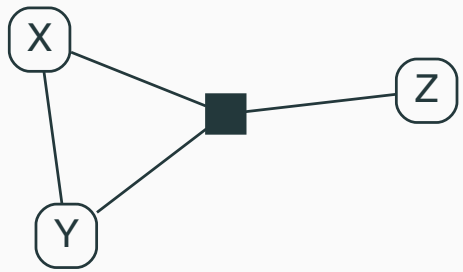
---

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 $removed$

N-áris kényszerek

N-áris kényszerek binárissá alakítása

$C_1 : X + Y = Z$, $C_2 : X < Y$, $D_X = \{1, 2\}$, $D_Y = \{3, 4\}$, $D_Z = \{5, 6\}$



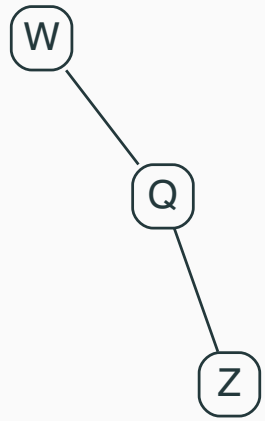
Duál gráf átalakítás:

1. legyen A a C_1 kényszernek megfelelő változó:
   $D_A =$
   $\{(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)\}$
   $\subseteq \mathbb{N}^3$
2. legyen B a C_2 kényszernek megfelelő változó:
   $D_B = \{(1, 3), (1, 4), (2, 3), (2, 4)\} \subseteq \mathbb{N}^2$
3. ⏲nyessük meg D_A-t C_1 és C_2 alapján: $D_A = \{(1, 4, 5), (2, 3, 5), (2, 4, 6)\}$
4. ⏲nyessük meg D_B-t C_1 és C_2 alapján: $D_B = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$
5. vezessünk be egy új kényszert: K_1: $A[1] = B[1]$
6. vezessünk be egy új kényszert: K_2: $A[2] = B[2]$

N-áris kényszerek binárissá alakítása (2)

Egyedi esetekben átírhatjuk a CSP-t úgy, hogy minden változó doméniuma továbbra is egydimenziós maradjon:

- $C_1 : X + Y = Z$, $C_2 : X < Y$, $D_X = \{1, 2\}$, $D_Y = \{3, 4\}$, $D_Z = \{5, 6\}$
- legyen $Q \equiv X + Y$ és $W \equiv X - Y$
- $K_1 : Q = Z$
- $K_2 : W < 0$
- $K_3 : (Q + W) \bmod 2 = 0$
- $D_Q = \{4, 5, 6\}, D_W = \{-3, -2, -1\}, D_Z = \{5, 6\}$



Összefoglaló

- CSP $\equiv \left(\{X_i\}, \{D_j\}, \{C_k\}\right)$
- backtrack: mélységi keresés, exponenciális komplexitás
- változók sorrendje: ✂ minimum remaining values, degree heuristics
- értékadás sorrendje: 🍀 least constraining value
- kényszer kiterjesztése: ✂ forward checking, AC-3
- duál gráf átalakítás

További technikák CSP-hez

- visszaugrás (backjumping)
- új kényszerek tanulása (constraint learning)
- lokális keresés (local search)
- **AllDifferent**() kényszer
- a kényszer-gráf struktúrájának kihasználása:
  - nem összefüggő komponensek a kényszer-gráfban
  - ciklus nélküli kényszer-gráfok

II. rész

Speciális kényszerek: AllDifferent

- $\text{AllDifferent}\left(\{X_i\}_{i\in\Theta}\right) \equiv \bigwedge_{\substack{j,k\in\Theta \\ j\neq k}} (X_j \neq X_k)$
- példa: sudoku
- legyen $D_{X_1} = \{a, b\}, D_{X_2} = \{c\}, D_{X_3} = \{a, c\}, D_{X_4} = \{b, c\}$
- kielégíthető az $\text{AllDifferent}(X_1, X_2, X_3, X_4)$?

AllDifferent (2)

Formálisan:

- ha m változónak összesen n különböző értéke van, és $m > n$, akkor az AllDifferent() kényszert nem lehet kielégíteni

Algoritmus prelúdium:

- $D_{X_1} = \{1, 2\}, D_{X_2} = \{3, 4\}, D_{X_3} = \{4\}, D_{X_4} = \{1, 3\}$

Mi az $\mathit{AllDifferent}(X_1, X_2, X_3, X_4)$ megoldása?

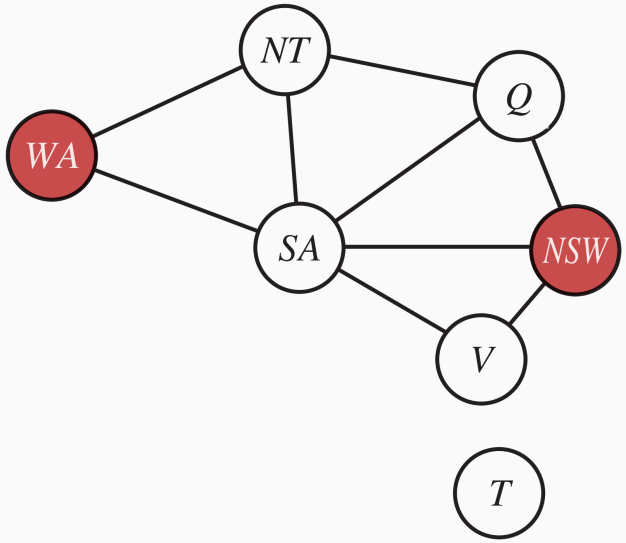
AllDifferent (3)

✂ **AllDifferent algoritmus**:

1. töröljünk ki a kényszerből egy olyan Y változót, amelynek egyelemű doméniuma van, $D_Y = \{y\}$
2. töröljük ki a többi változó doméniumából y-t
3. ismételjük az 1-2 lépést egész addig, amíg
   - nincs már olyan változó, aminek csak egyelemű doméniuma van, vagy...
   - valamelyik változó doméniuma kiürül, vagy...
   - több változó maradt, mint érték

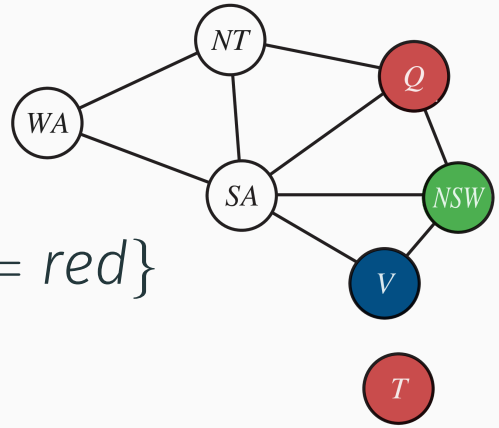
AllDifferent (4)

1. WA = red
2. fwcheck(WA)
3. NSW = red
4. fwcheck(NSW)
5. AllDifferent(SA, NT, Q) ✂



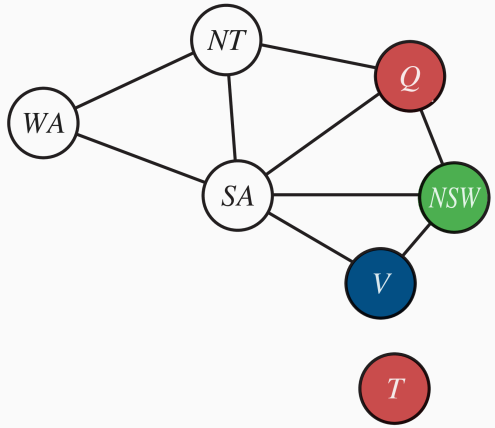
- a változók melyik részhalmazára hívjuk meg az AllDifferent()-et?
- a klikkek feltárása NP-teljes, lásd Clique problem

Intelligens visszalépés (backjumping)



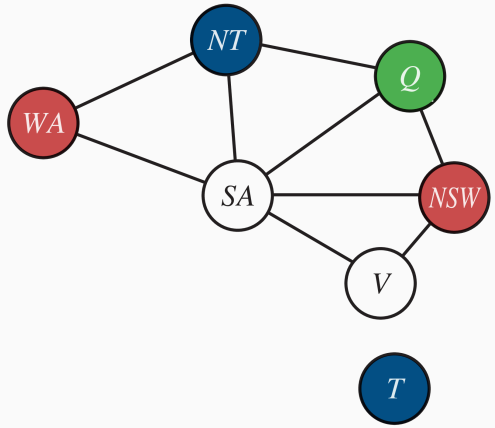
- változók sorrendje: $(Q, NSW, V, T, SA, WA, NT)$
- legyen: $\{Q = red, NSW = green, V = blue, T = red\}$
- $SA = ?$
- hová ugrik vissza a backtracking?
- hová kellene visszaugrani?
- X változó **konfliktushalmaza (conflict set)**: a korábban értéket kapott változók egy halmaza, amelyekhez X kényszerekkel van kötve
- $\text{conf}(SA) = \{Q = red, NSW = green, V = blue\}$
- **a backjumping a konfliktushalmazbeli legutolsó változóhoz lép vissza**
- milyen ismert technika tudná előállítani a konfliktushalmazt?

Backjumping (2)



- azokat az ágakat, amelyeket a backjumping levágna a forward checking (vagy AC-3) már korábban levágta
- az elv továbbra is jó: lépjünk vissza a hiba helyére

Backjumping (3)



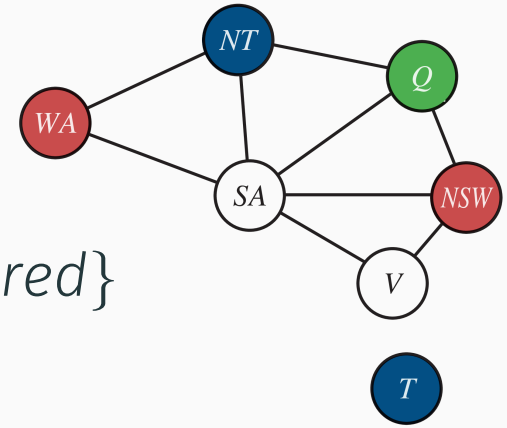
Egy részfa korábban levágásra lehet ítélve, semmint hogy egy változó doméniuma kiürülne:

- legyen: $S = \{WA = red, NSW = red, T = red\}$
- adjunk értéket (NT, Q, V, SA)-nak!

- kibontjuk a részfát... majd üres kézzel visszaérünk NT-be
- mi a D_{NT}? D_{NT} kompatibilis S-el?
- mi a conf(NT)? Megmagyarázza hogy NT miért bukott el?

- NT és az utána következő változók, i.e. (NT, Q, V, SA) **együtt buktak el** a $\{WA = red, NSW = red, \cancel{T = red}\}$ miatt

Backjumping (4)



- kezdeti értékek: $\{WA = red, NSW = red, T = red\}$
- változók sorrendje: (NT, Q, V, SA)

- újradefiniáljuk a konfliktushalmaz fogalmát:
- NT konfliktushalmaza az NT-et megelőző változók azon halmaza, **amely felelős NT és az összes NT-t követő változó kudarcáért**
- $\text{conf}(NT) = \{WA = red, NSW = red\}$

Conflict-based backjumping

Conflict-based backjumping: ha az aktuális X_k változó minden értéke meghiúsul, akkor

1. ugorjunk vissza a $\text{conf}(X_k)$-beli legutolsó X_j változóhoz, és
2. frissítsük X_j konfliktushalmazát
   $\text{conf}(X_j) \leftarrow \text{conf}(X_j) \bigcup \text{conf}(X_k) - \{X_j\}$

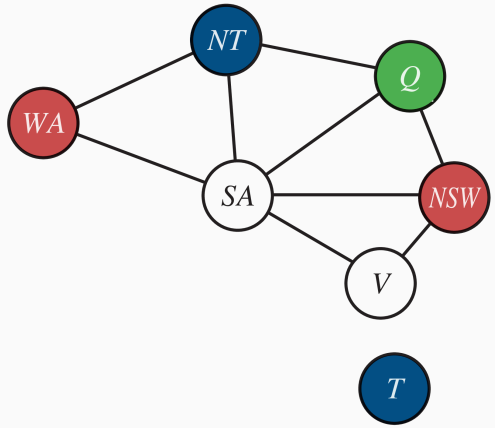
Példa. Legyen

- $\text{conf}(X_j) = \{X_1, X_2, X_3\}$
- $\text{conf}(X_k) = \{X_3, X_4, X_5, X_j\}$

Frissítés után:

- $\text{conf}(X_j) = \{X_1, X_2, X_3, X_4, X_5\}$

Conflict-based backjumping (2)



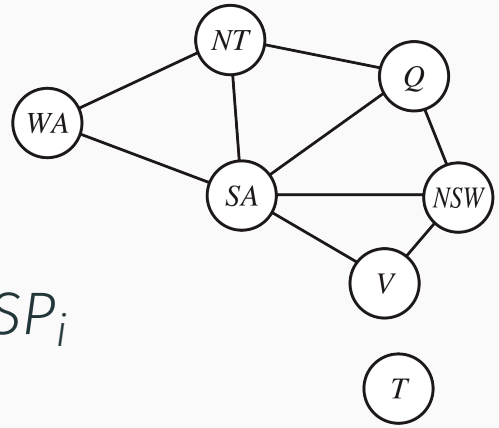
stack: $(WA = r, NSW = r, T = b, NT = b, Q = g, SA = \ldots)$
backjumping jelölés: $\rightsquigarrow$, dead-end jelölés: $\Omega(\cdot)!$

---

1. $\text{conf}(SA) = \{WA = r, NSW = r, NT = b, Q = g\}$
2. $\Omega(SA)! \rightsquigarrow Q$
3. $\text{conf}(Q) = \text{conf}(Q) \cup \text{conf}(SA) - \{Q\}$
   $= \{NSW = r, NT = b\} \cup \{WA = r, NSW = r, NT = b, Q = g\} - \{Q\}$
   $= \{WA = r, NSW = r, NT = b\}$
4. $\Omega(Q)! \rightsquigarrow NT$
5. $\text{conf}(NT) = \text{conf}(NT) \cup \text{conf}(Q) - \{NT\}$
   $= \{WA = r\} \cup \{WA = r, NSW = r, NT = b\} - \{NT\}$
   $= \{WA = r, NSW = r\}$
6. $NT = g$, kibontjuk a fát ... $\Omega(SA)!, \Omega(Q)!$
7. $\Omega(NT)! \rightsquigarrow NSW$
8. $NSW = g$, kibontjuk a fát ...

A kényszergráf struktúrájának kihasználása

Összefüggő komponensek

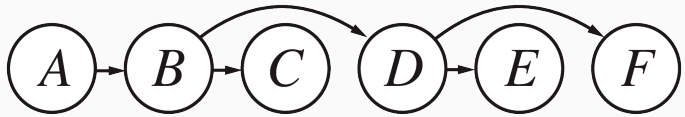


- a kényszergráf mindenik komponense egy CSP_i részproblémának felel meg
- ha az S_i hozzárendelés megoldása CSP_i-nek, akkor $\bigcup_i S_i$ megoldása $\bigcup_i CSP_i$-nek

- példa: legyen egy 80 logikai változót tartalmazó CSP, amit 4 egyenlő részre bontunk; mennyi a számítási idő felbontással illetve anélkül?

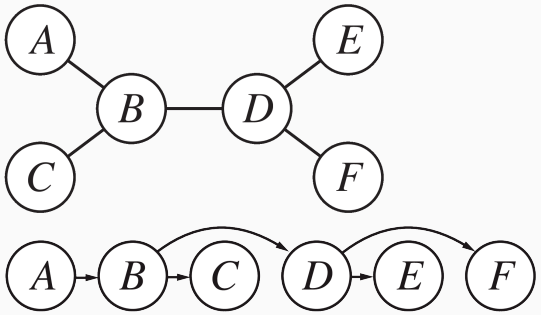
Fa alakú kényszergráf

1. hogyan tehető élkonzisztenssé a következő kényszergráf?



2. hogyan oldható meg miután élkonzisztenssé tettük?

Fa alakú kényszergráf (2)



Minden fa alakú kényszergráf megoldható **lineáris** időben.

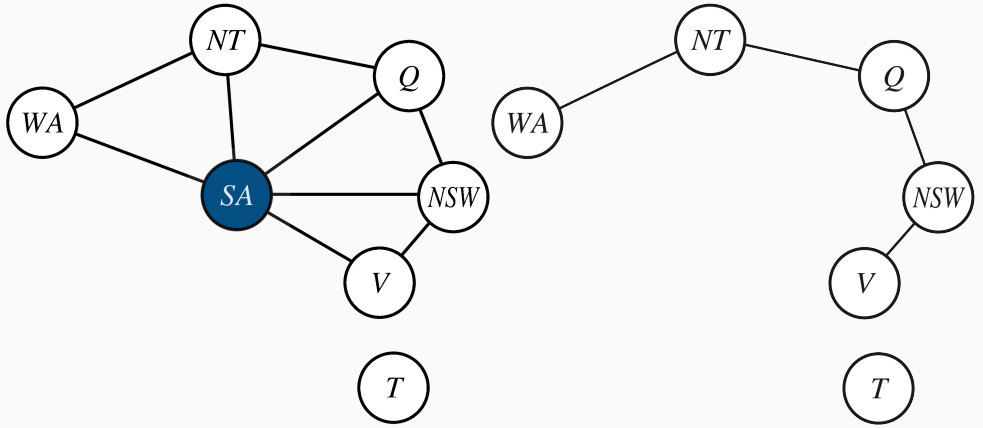
1. válasszunk ki egy tetszőleges gyökércsomópontot (X_1) és rendezzük **topologikusan** a gráfot; rendezés után legyen a változók sorrendje $X_1, X_2, \ldots, X_n$
2. minden $j = (n, n-1, \ldots, 2)$-re tegyük $(\mathrm{P}(X_j) \to X_j)$-t élkonzisztenssé, ahol $\mathrm{P}(X_j)$ X_j szülője
3. minden $j = 1, 2, \ldots, n$-re adjunk X_j-nek egy tetszőleges értéket X_j doméniumából

Fa alakú kényszergráf (3)

function TREE-CSP-SOLVER(csp) **returns** a solution, or failure
  inputs: csp, a CSP with components $X,\ D,\ C$

  $n \leftarrow$ number of variables in X
  $assignment \leftarrow$ an empty assignment
  $root \leftarrow$ any variable in X
  $X \leftarrow$ TOPOLOGICALSORT($X, root$)
  for $j = n$ **down to** 2 **do**
    MAKE-ARC-CONSISTENT(PARENT(X_j), X_j)
    if it cannot be made consistent **then return** $failure$
  for $i = 1$ **to** n **do**
    $assignment[X_i] \leftarrow$ any consistent value from D_i
    if there is no consistent value **then return** $failure$
  return $assignment$

A kényszergráf redukálása törléssel



1. $SA = blue$
2. fwcheck(SA)
3. tree-csp-solver(WA, NT, Q, NSW, V)

A kényszergráf redukálása törléssel (2)

- általánosan:
  1. válasszuk ki a változók egy S részhalmazát, úgy, hogy a kényszergráf S eltávolítása után fa legyen
     S egy **cycle cutset – ciklikusság vágóhalmaz**
  2. az S minden belső konzisztens hozzárendelésére oldjuk meg a maradék fát
- a minimális méretű vágóhalmaz megtalálása NP-teljes probléma

Összefoglaló

- AllDifferent
- Backjumping / Conflict-based bacjkumping
- összefüggő komponensek a kényszergráfban
- fa alakú kényszergráf
- a kényszergráf redukálása törléssel

Könyvészet, ábrák

- Russel & Norvig: Artificial Intelligence, a Modern Approach (2. és 3. kiadás)
- Rina Dechter: Constraint Processing