Mesterséges intelligencia előadássorozat

Az előadás diái az AIMA könyvre épülve (http://aima.cs.berkeley.edu) készültek a University of California, Berkeley mesterséges intelligencia kurzusának anyagainak felhasználásával (http://ai.berkeley.edu).

These slides are based on the AIMA book (http://aima.cs.berkeley.edu) and were adapted from the AI course material of University of California, Berkeley (http://ai.berkeley.edu).

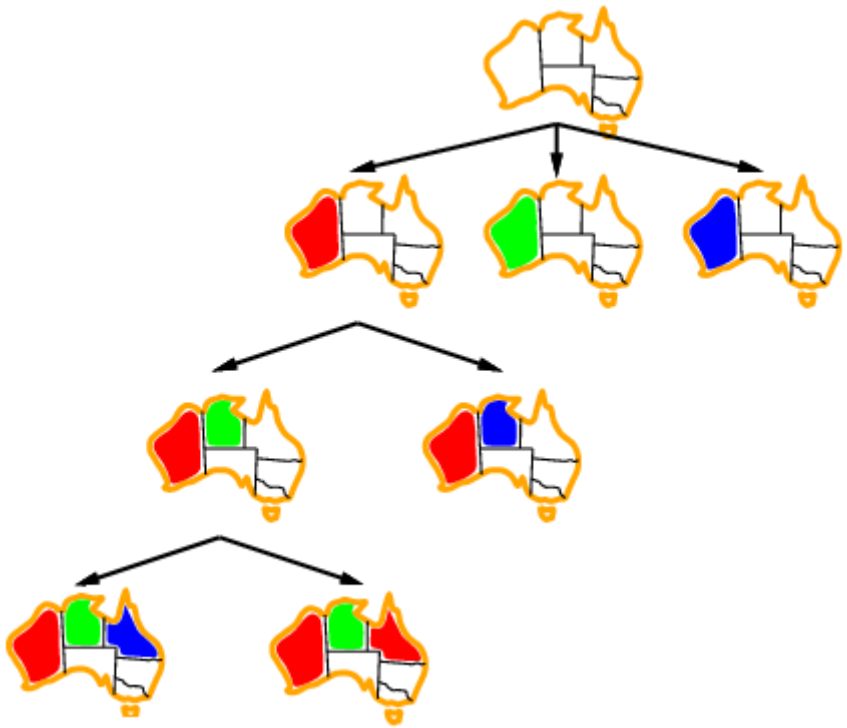
Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Mesterséges Intelligencia és Rendszertervezés Tanszék

Mesterséges intelligencia

Kényszerkielégítési problémák II.

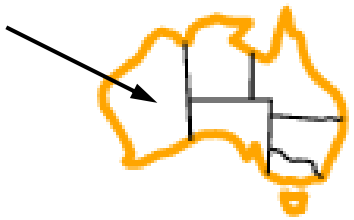
Előadó: Dr. Hullám Gábor
Előadás anyaga: Dr. Gézsi András, Dr. Hullám Gábor

Visszalépéses keresés



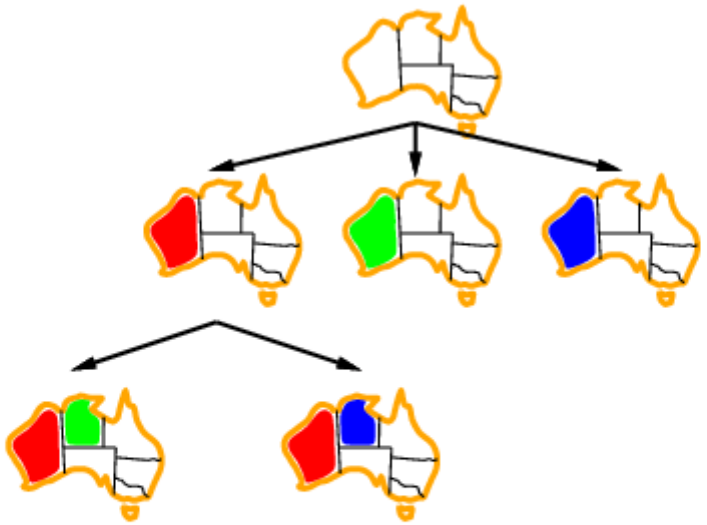
- A **visszalépéses keresés** az alapvető, nem-informált algoritmus kényszerkielégítési problémák megoldására
- **1. ötlet: Egy lépésben egy változóhoz rendeljünk értéket**
  - A változó-hozzárendelés kommutatív, rögzítsük a változók sorrendjét
    Pl: [WA = vörös, majd NT = zöld] ua. mint
    [NT = zöld, majd WA = vörös]
  - Minden egyes lépésben a soron következő változóhoz rendeljünk értéket
- **2. ötlet: Ellenőrizzük a kényszereket a változó-hozzárendelések során**
  - Olyan értéket válasszunk, amely a korábbi hozzárendelésekkel nincs konfliktusban
  - A kényszerek ellenőrzése további számítással is járhat
  - „Fokozatos célállapotteszt”
- Mélységi keresés + 1. ötlet + 2. ötlet = **visszalépéses keresés** (backtracking search)
- N-királynő probléma megoldható: N ≈ 25

Visszalépéses keresés

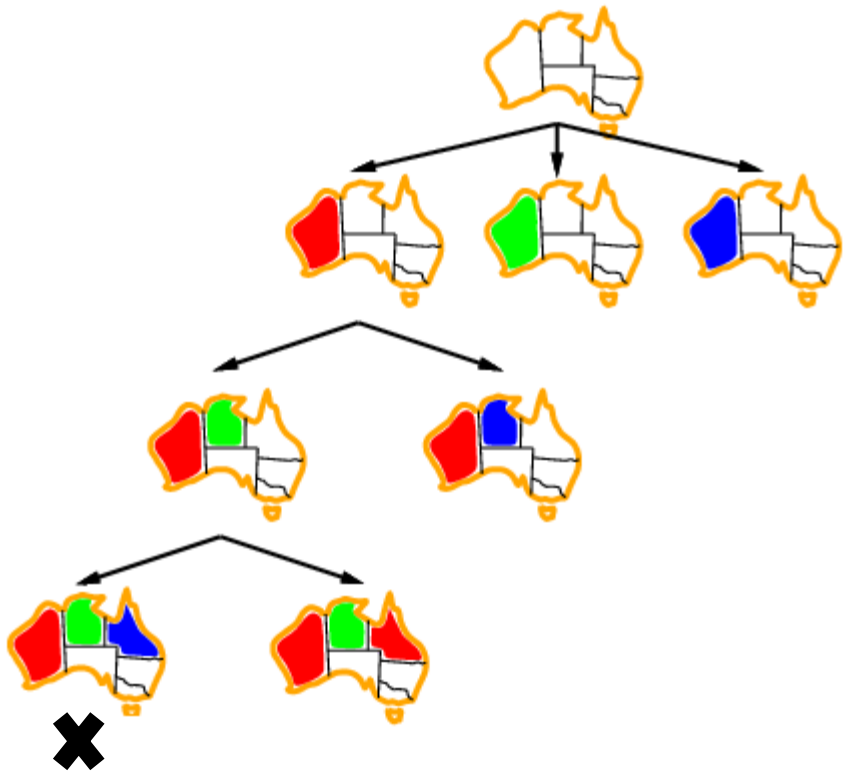


Visszalépéses keresés

Visszalépéses keresés



Visszalépéses keresés



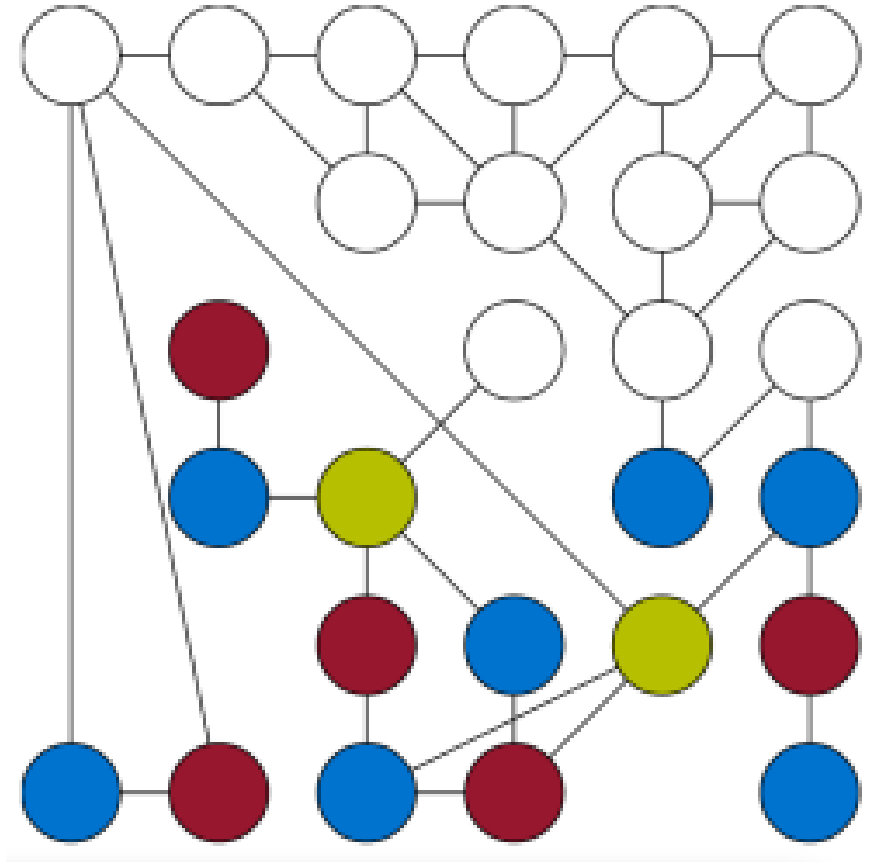
Visszalépéses keresés

```
function VISSZALÉPÉSÉS-KERESÉS(csp) returns egy megoldást, vagy meghiúsul
    return REKURZÍV-VISSZALÉPÉSES({},csp)

function REKURZÍV-VISSZALÉPÉSES(hozzárendelések, csp) returns egy megoldást, vagy meghiúsul
    if hozzárendelések teljes then return hozzárendelések
    var ← HOZZÁRENDELETLEN-VÁLTOZÓ-KIVÁLASZTÁSA(VÁLTOZÓK[csp],hozzárendelések,csp)
    for each érték in TARTOMÁNY-ÉRTÉKEK-SORRENDEZÉSE(var, hozzárendelések, csp) do
        if érték konzisztens a hozzárendelések-kel KÉNYSZEROK(csp) szerint then
            hozzáad {var = érték } hozzárendelések-hez
            eredmény ← REKURZÍV-VISSZALÉPÉSES(hozzárendelések, csp)
            if eredmény ≠ meghiúsulás then return eredmény
```

Néhány gyengeség

- Tekintsük az itt látható részleges hozzárendelést
- A csomópontokat lentről felfelé, balról jobbra járjuk be
- Vajon a jelenlegi hozzárendelés kudarcra van ítélve?
- A naív visszalépéses keresés túl későn veszi észre a problémát



A visszalépéses keresés hatékonyságának növelése

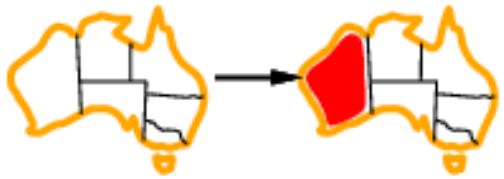
- Általános, tárgyterület-független heurisztikákkal növelhetjük a hatékonyságot
- Szűrés:
  - Ki tudjuk korán szűrni a kudarcra ítélt megoldásokat?
- Sorrendezés:
  - Változók sorrendezése: Melyik változóhoz rendeljünk értéket a következő lépésben?
  - Értékek sorrendezése: Melyik értéket rendeljük hozzá először a változóhoz?
- Struktúra:
  - Ki tudjuk használni a probléma struktúráját?

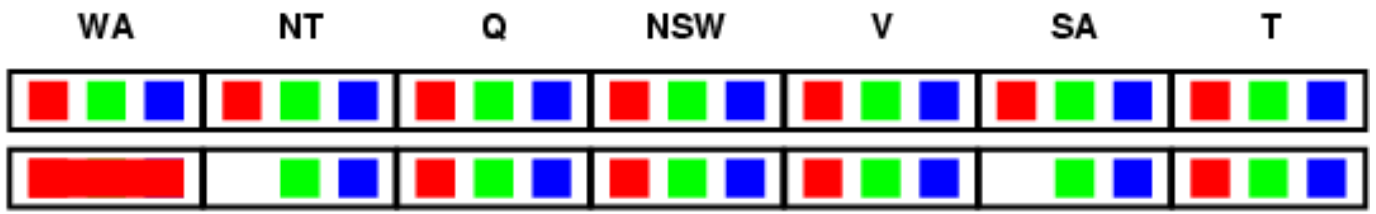
Szűrés: Előretekintő ellenőrzés

Az **előretekintő ellenőrzés** minden egyes alkalommal, amikor egy X változó értéket kap, minden, az *X-hez kényszerrel kapcsolt, lekötetlen* Y-t megvizsgál, és Y tartományából *törli* az X számára választott értékkel inkonzisztens értékeket.

Szűrés: Előretekintő ellenőrzés

Az **előretekintő ellenőrzés** minden egyes alkalommal, amikor egy X változó értéket kap, minden, az *X-hez kényszerrel kapcsolt, lekötetlen* Y-t megvizsgál, és Y tartományából *törli* az X számára választott értékkel inkonzisztens értékeket.



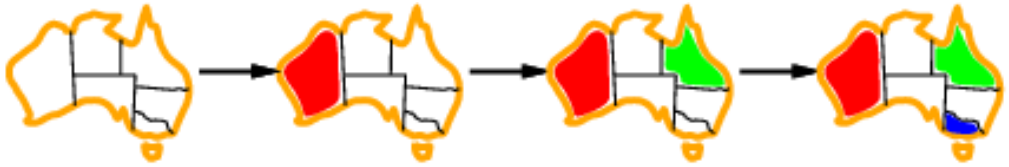


Szűrés: Előretekintő ellenőrzés

Az **előretekintő ellenőrzés** minden egyes alkalommal, amikor egy X változó értéket kap, minden, az *X-hez kényszerrel kapcsolt, lekötetlen* Y-t megvizsgál, és Y tartományából *törli* az X számára választott értékkel inkonzisztens értékeket.

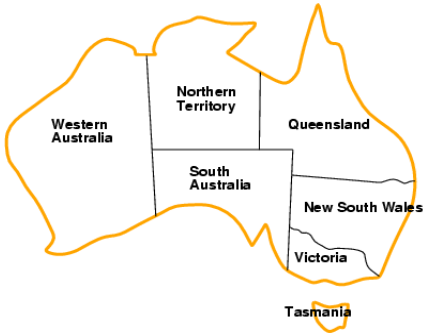
Szűrés: Előretekintő ellenőrzés

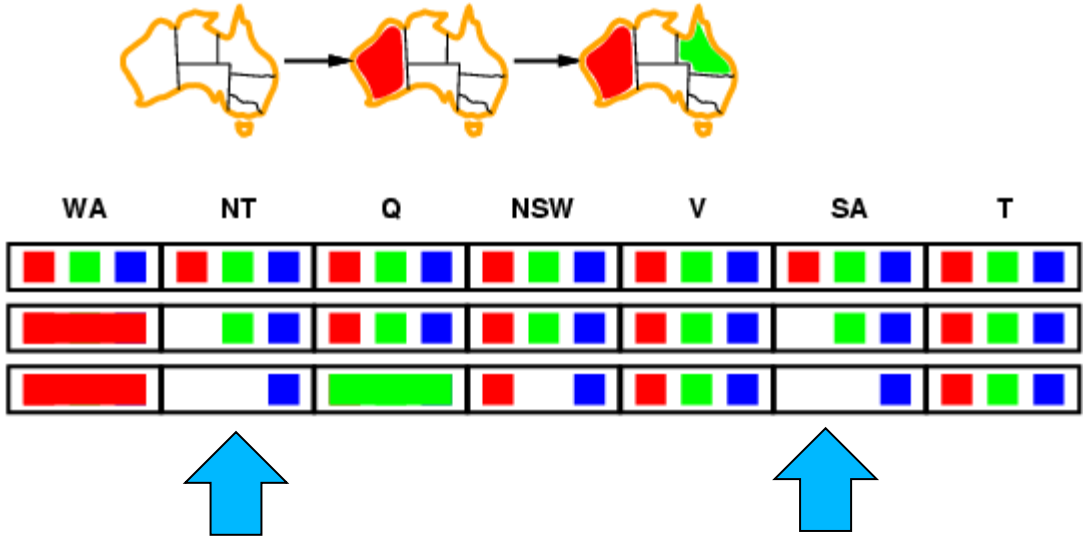
Az **előretekintő ellenőrzés** minden egyes alkalommal, amikor egy X változó értéket kap, minden, az *X-hez kényszerrel kapcsolt, lekötetlen* Y-t megvizsgál, és Y tartományából *törli* az X számára választott értékkel inkonzisztens értékeket.



Szűrés: Előretekintő ellenőrzés (korlátai)

- Az előretekintő ellenőrzés ugyan sok inkonzisztenciát észrevesz, de nem mindent.
- Ráadásul nem látja jól előre a kudarcokat.



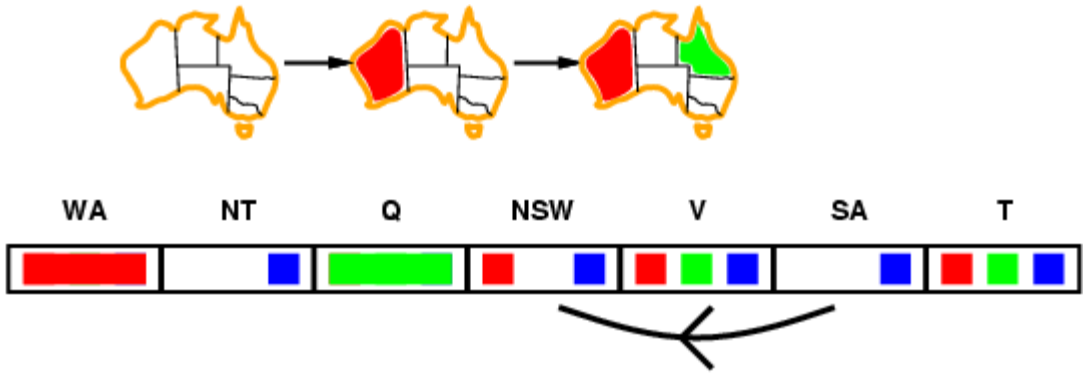


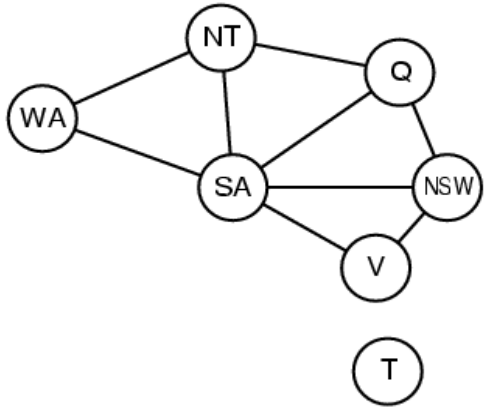
NT és SA egyszerre nem lehet kék, mivel szomszédosak.

Élkonzisztencia

X → *Y* él **konzisztens** akkor és csak akkor, ha

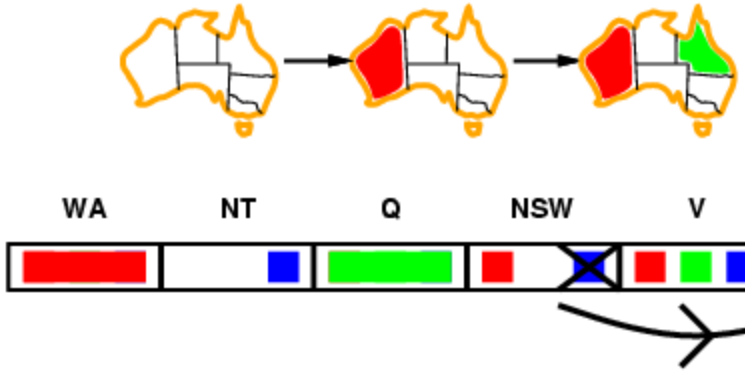
X minden x értékére létezik Y-nak valamilyen megengedett y értéke

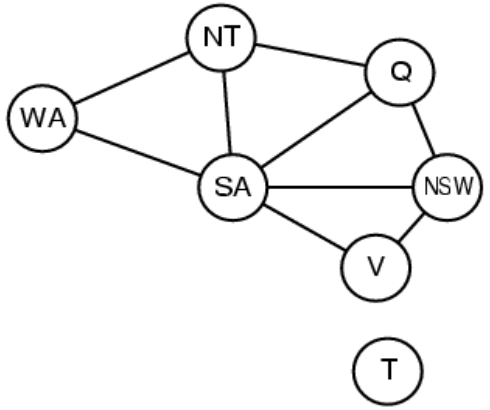




Szűrés: Élkonzisztencia terjesztése

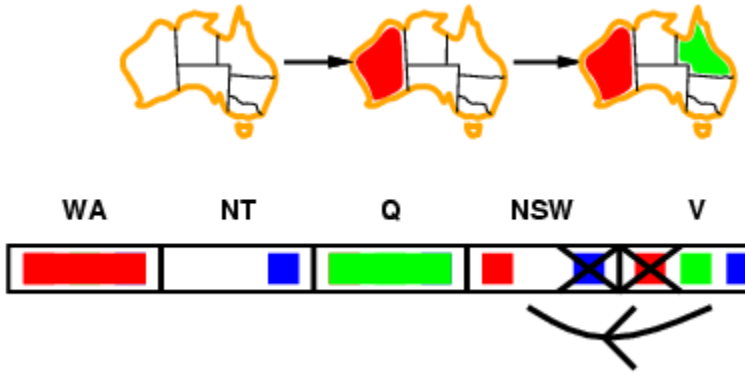
- **Ötlet:** terjesszük el az élkonzisztenciát a kényszergráf összes élére, biztosítsuk az **összes** él konzisztenciáját!
- Ha X tartományából törlünk egy értéket, X szomszédait újra ellenőrizzük.

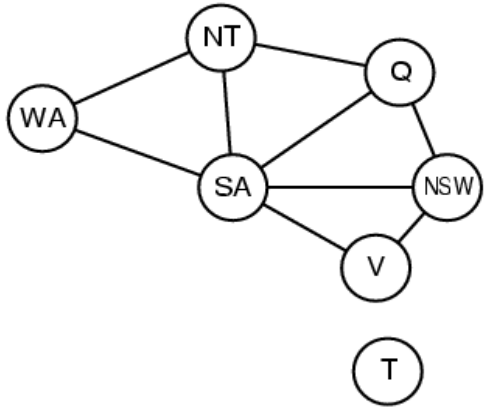




Szűrés: Élkonzisztencia terjesztése

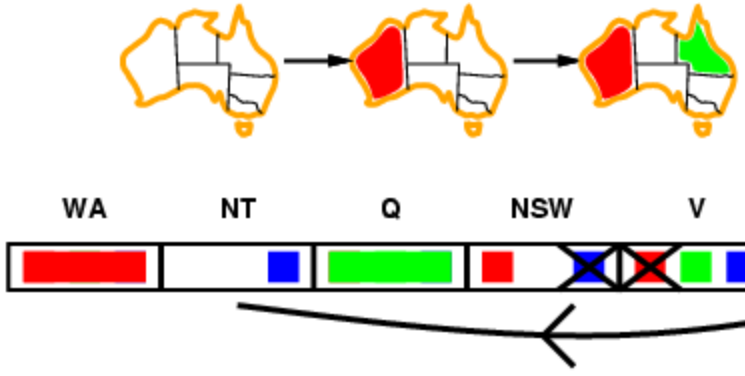
- **Ötlet:** terjesszük el az élkonzisztenciát a kényszergráf összes élére, biztosítsuk az **összes** él konzisztenciáját!
- Ha X tartományából törlünk egy értéket, X szomszédait újra ellenőrizzük.

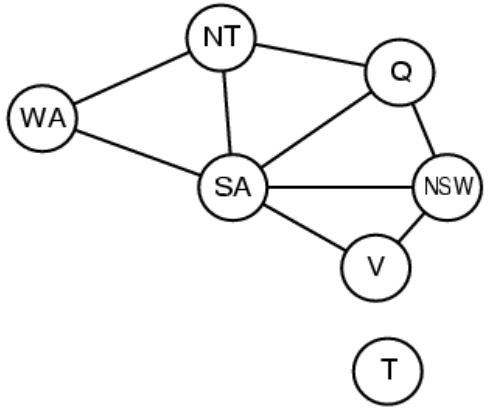




Szűrés: Élkonzisztencia terjesztése

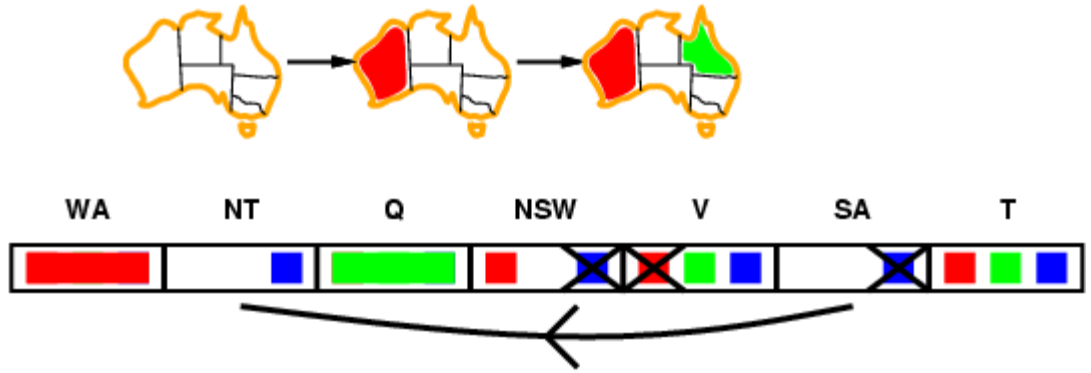
- **Ötlet:** terjesszük el az élkonzisztenciát a kényszergráf összes élére, biztosítsuk az **összes** él konzisztenciáját!
- Ha X tartományából törlünk egy értéket, X szomszédait újra ellenőrizzük.





Élkonzisztencia terjesztése

- Az élkonzisztencia-ellenőrzés alkalmazható:
  - **előfeldolgozó lépésként** a keresés megkezdése előtt, vagy a
  - keresési folyamat minden egyes hozzárendelését követő **terjesztési lépésként**
- **Előbb tárja fel a problémát, mint az előretekintő ellenőrzés**
- Fontos: Ha X tartományából törlünk egy értéket, X szomszédait újra ellenőrizzük.
- Algoritmus neve: **AC-3**



A visszalépéses keresés hatékonyságának növelése

- Általános, tárgyterület-független heurisztikákkal növelhetjük a hatékonyságot
- Szűrés:
  - Ki tudjuk korán szűrni a kudarcra ítélt megoldásokat?
- Sorrendezés:
  - Változók sorrendezése: Melyik változóhoz rendeljünk értéket a következő lépésben?
  - Értékek sorrendezése: Melyik értéket rendeljük hozzá először a változóhoz?
- Struktúra:
  - Ki tudjuk használni a probléma struktúráját?

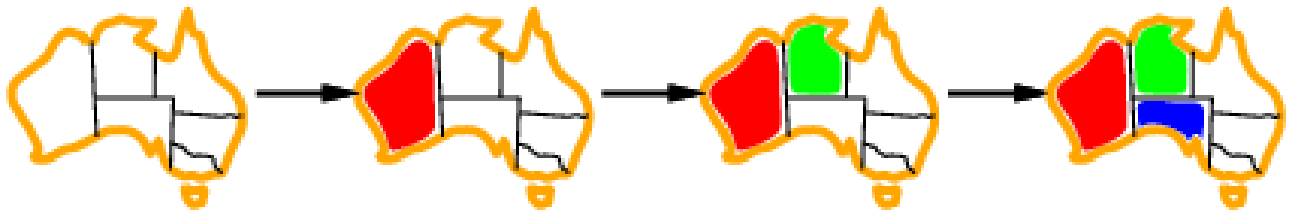
1. *A legkevesebb fennmaradó érték* ötlete (melyik változót válasszuk?)

A leginkább korlátozott változó:

- a legkisebb számú megengedett értékkel rendelkező változóval kezdjünk, ill. folytassunk (lokálisan kicsi az elágazási tényező!)

= **legkevesebb fennmaradó érték** heurisztika

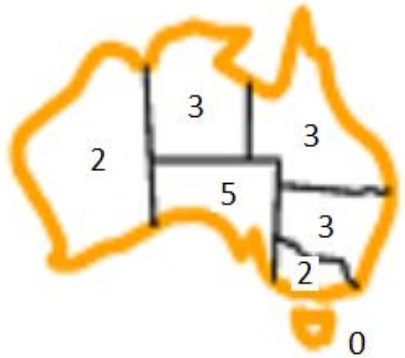
(minimum remaining values, MRV)

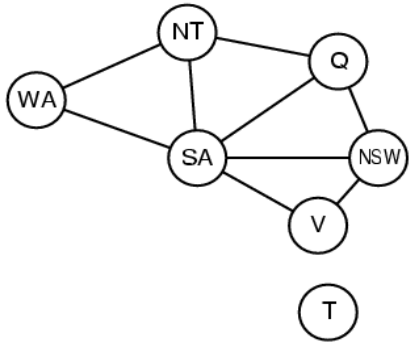


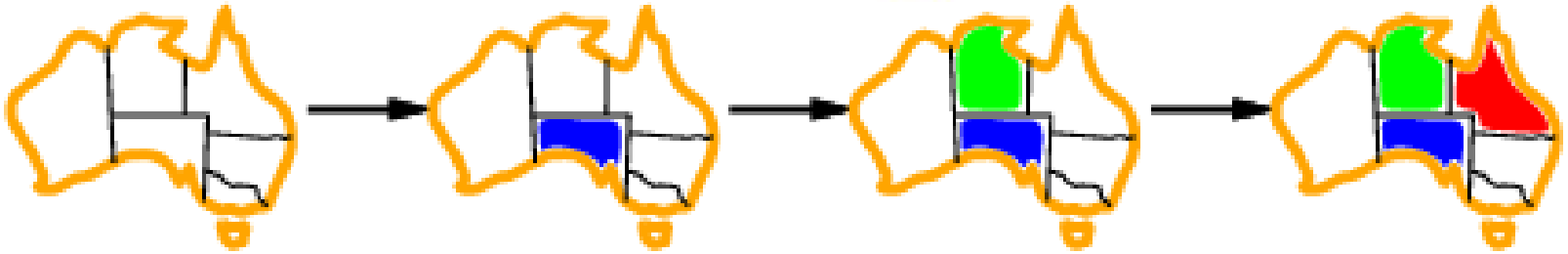
(NT ill. SA csak 2 megengedett érték (piros már nem lehet), minden más 3)

2. *Fokszám heurisztika* ötlete (melyik változót válasszuk?)

- Az MRV-heurisztika semmit sem segít abban, hogy melyik régiót válasszuk ki elsőként Ausztrália kiszínezésekor, mert a kiinduláskor mindegyik régiónak három megengedett színe van.
- A *későbbi választások* elágazási tényezőjét csökkentheti, ha azt a változót választjuk ki, amely a legtöbbször szerepel a még hozzárendeletlen változókra vonatkozó kényszerekben.







3. *A legkevésbé korlátozó érték* ötlete

Előnyben részesítjük azt az értéket, amely a legkevesebb választást zárja ki a kényszergráfban a szomszédos változóknál.

1 értéket enged SA-nak

0 értéket enged SA-nak

Ezekkel az ötletekkel az N-királynő probléma N ≈ 1000-re is megoldható!

A visszalépéses keresés hatékonyságának növelése

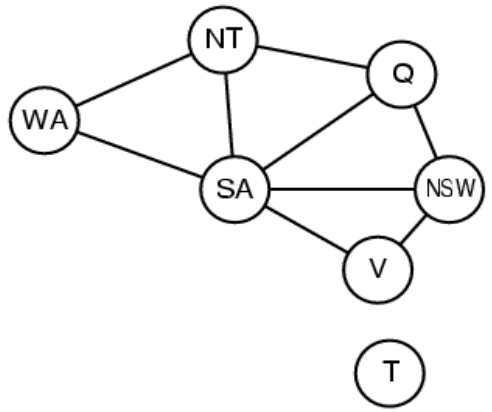
- Általános, tárgyterület-független heurisztikákkal növelhetjük a hatékonyságot
- Szűrés:
  - Ki tudjuk korán szűrni a kudarcra ítélt megoldásokat?
- Sorrendezés:
  - Változók sorrendezése: Melyik változóhoz rendeljünk értéket a következő lépésben?
  - Értékek sorrendezése: Melyik értéket rendeljük hozzá először a változóhoz?
- Struktúra:
  - Ki tudjuk használni a probléma struktúráját?

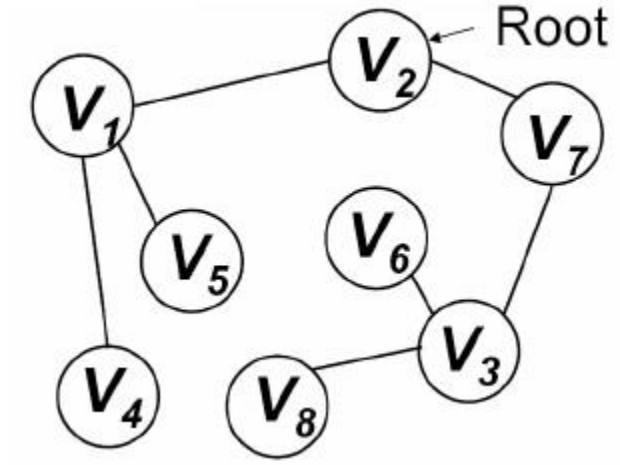
CSP struktúrája – miben segíthet?

CSP gráfja: független komponensek

CSP fa gráf: megoldás könnyű, változószámban lineáris

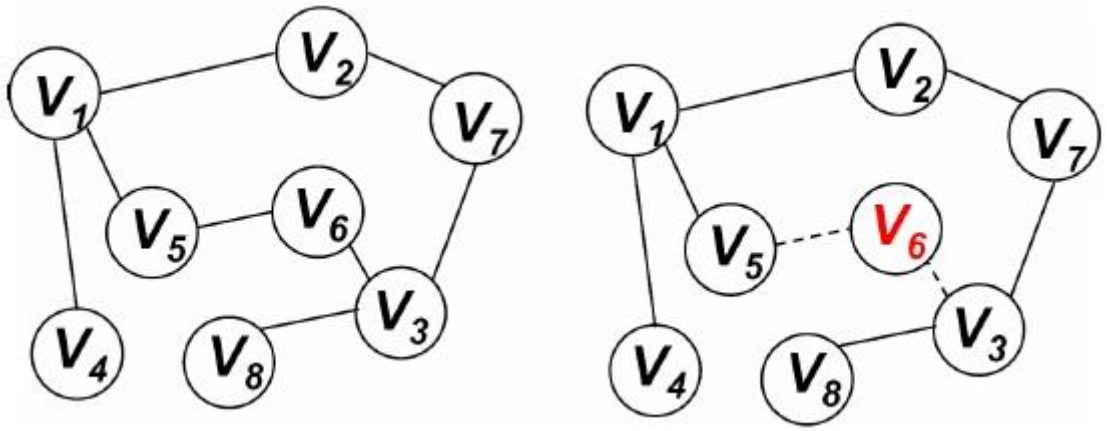
1. válasszunk egy levelet gyökérnek,
   majd sorrendezzük a változókat (élek irányítása)
2. élkonzisztencia-nyesés gyerekektől a szülőik felé
3. értékek hozzárendelése a gyökér csomóponttól kezdve





CSP struktúrája – miben segíthet?

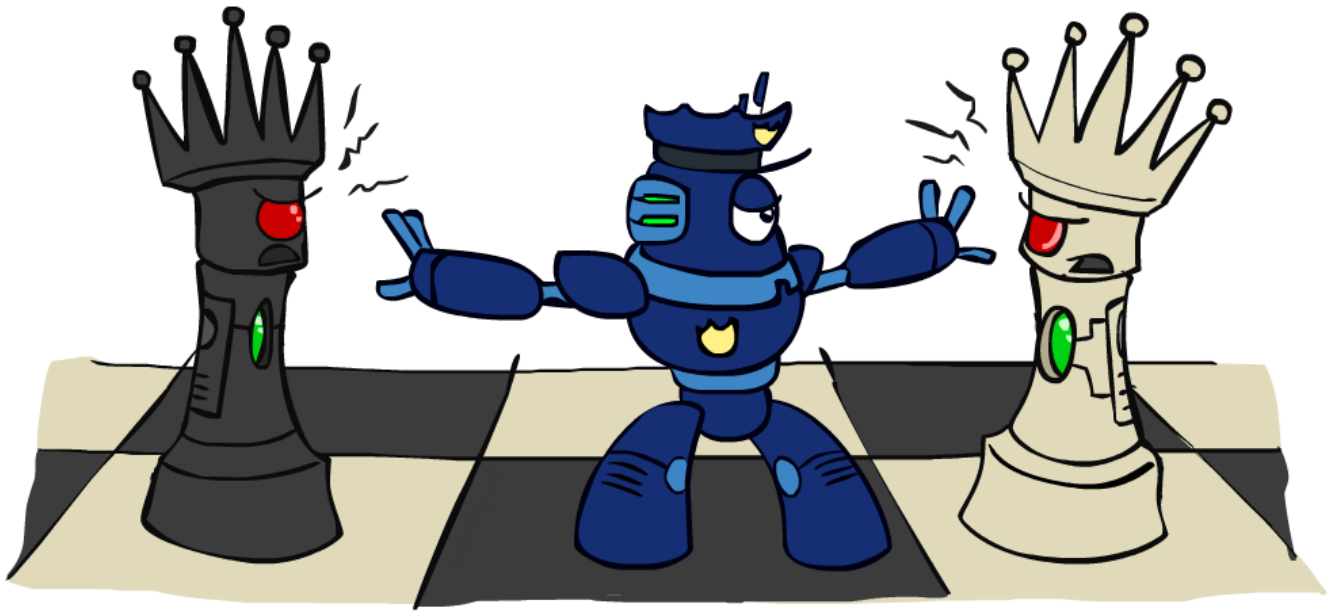
CSP hurkos gráf: megoldás általánosságban NP-nehéz



Gráf CSP konvertálása fába:
- vágóhalmaz
- fa-**dekompozíció**

Megoldáskeresés
V_6 minden értékére

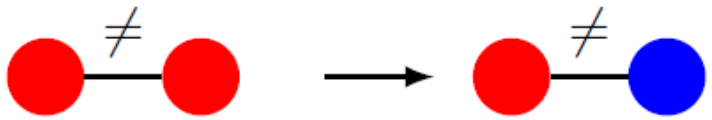
Kényszerkielégítési problémák megoldása lokális kereséssel



CSP lokális kereséssel

Kiindulás: teljes állapotleírás = minden változónak van értéke (esetleg megsértett, nem teljesült kényszerekkel)

Operátorok: megváltoztatják a változók hozzárendelését, hogy csökkenjen a sérült kényszerek száma



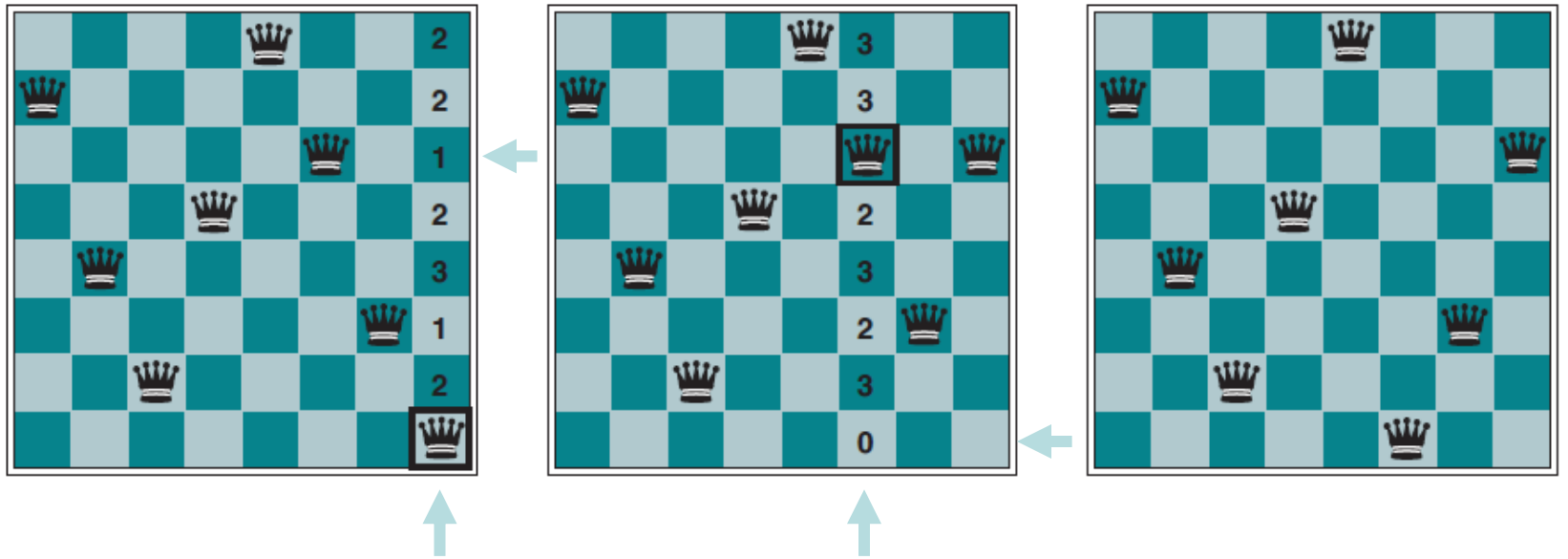
Algoritmus: Amíg nincs megoldás,

- **Változó kiválasztása**: véletlen módon, bármely konfliktusban lévő (valamelyik kényszer sérül) változót választhatjuk
- **Új érték választása**: min-konfliktus heurisztika

CSP lokális kereséssel

Min-konfliktus heurisztika:

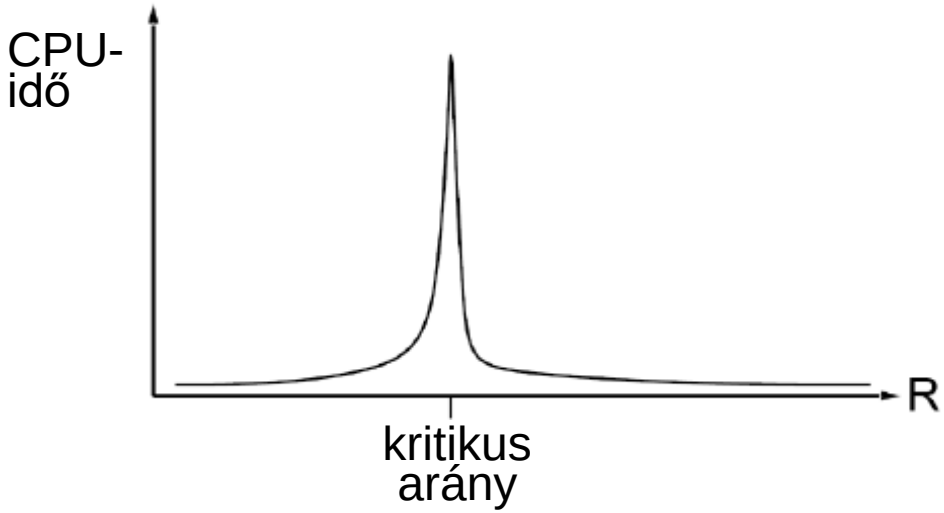
azt az értéket állítjuk be, amely a legkevesebb számú korlátot sérti



CSP lokális kereséssel

- A min-konfliktus heurisztika nagyon hatékony, véletlen kezdőállapotból kiindulva meg tudja oldani az N-királynő problémát gyakorlatilag a *probléma méretétől függetlenül*
  (pl. 1millió-királynő esetén átlagosan 50 lépésben!)
- Ugyanez igaz véletlenszerűen generált CSP-kre is, kivéve egy szűk tartományt

- $R = \frac{kényszerek\ száma}{változók\ száma}$



Összefoglalás

- A kényszerkielégítési problémák speciális keresési feladatok:
  - Az állapotok részleges hozzárendelések
  - A célállapottesztet kényszerek definiálják
- Alapvető megoldás: visszalépéses keresés
- Gyorsítás:
  - Sorrendezés
  - Szűrés
  - Struktúra
- Iteratív min-konfliktus (lokális keresés) a gyakorlatban gyakran hatékony megoldást ad

Illusztráció

https://inst.eecs.berkeley.edu/~cs188/fa19/assets/demos/csp/csp_demos.html