IKI 30320: Sistem Cerdas
Kuliah 7: Constraint Satisfaction Problems

Ruli Manurung

Fakultas Ilmu Komputer
Universitas Indonesia

24 September 2007

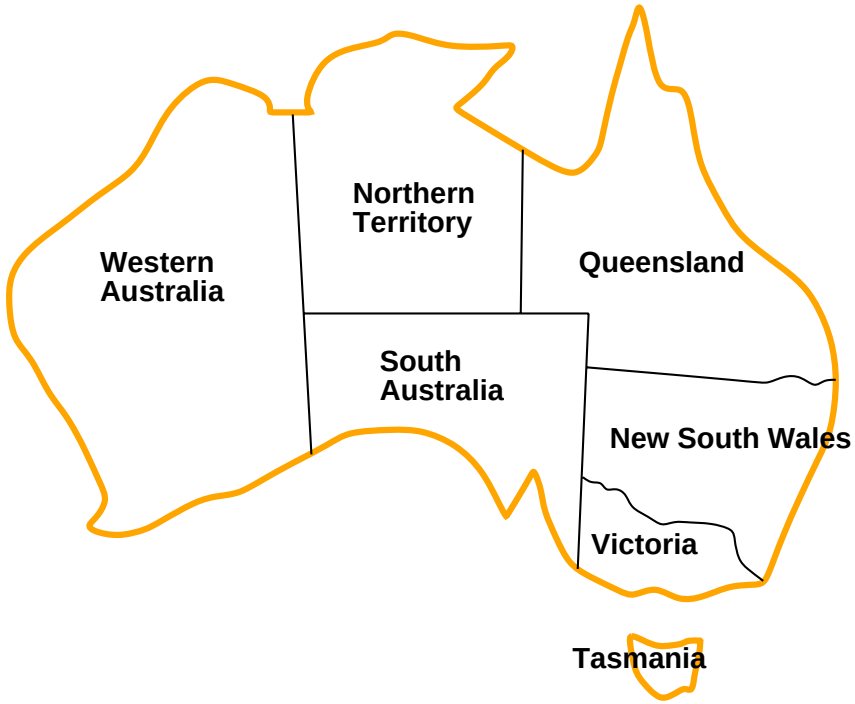
Outline

Outline

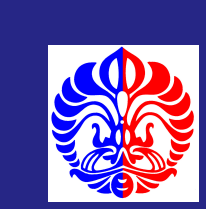
Mendefinisikan bentuk state

- State space search biasa: bentuk state terserah.
  $\rightarrow$ asal tersedia goal test dan successor function
- CSP: state terdiri dari sejumlah variable X_i yang nilainya dari domain/ranah D_i.
- Goal test adalah himpunan constraint/syarat yang harus dipenuhi berupa kombinasi nilai dari subset variable.
- Solusi adalah penetapan nilai (assignment) terhadap semua variable sehingga semua syarat dipenuhi.
- Definisi ini adalah contoh *bahasa representasi* yang formal.
- Tersedia beberapa algoritma CSP yang lebih canggih dari algoritma search biasa.

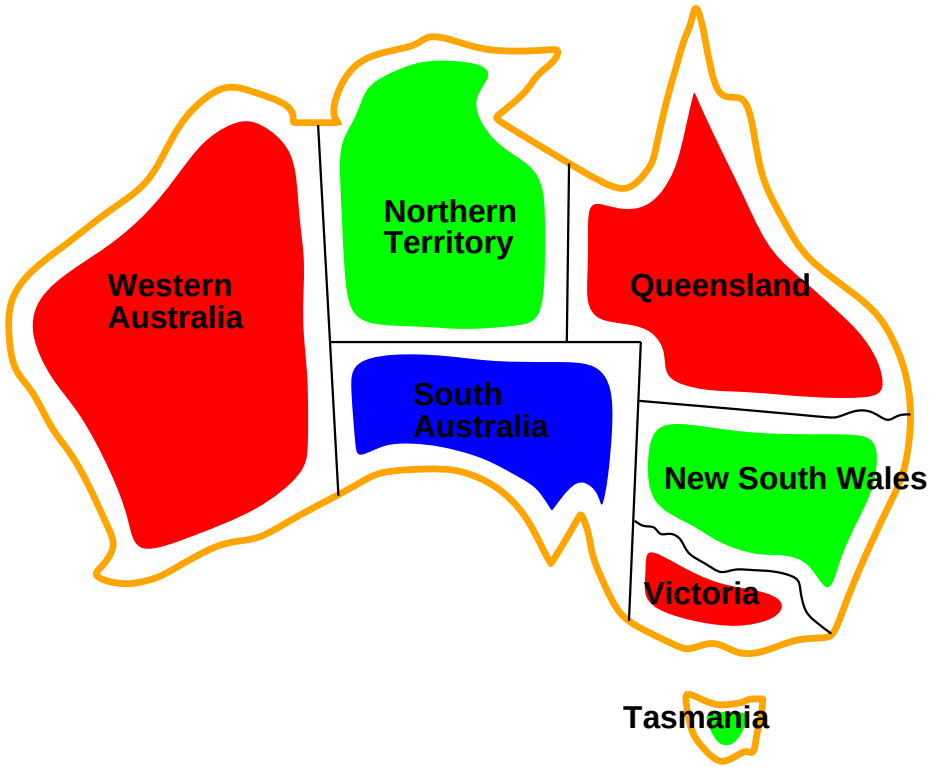
Contoh CSP: Mewarnai Peta



- Variabel: *WA, NT, Q, NSW, V, SA, T*
- Ranah: $D_i = \{red, green, blue\}$
- Syarat: 2 wilayah yang berbatasan harus berbeda warna:
  - $WA \neq NT, NT \neq SA, \ldots$
  - $(WA, NT) \in$
    $\{(red, green), (red, blue), (green, red), (green, blue), \ldots\}, \ldots$



Contoh Solusi CSP: Mewarnai Peta

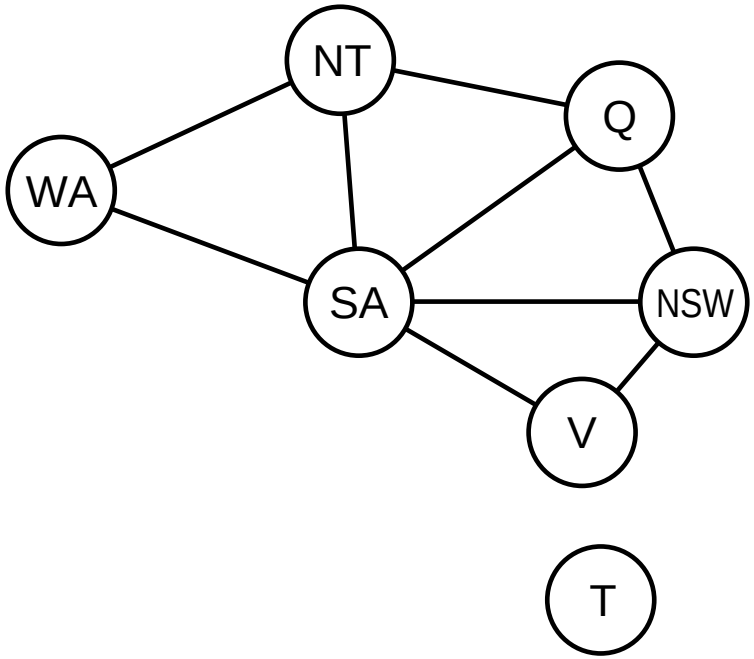


Solusi adalah pemberian nilai setiap variabel yang memenuhi syarat, mis:

$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

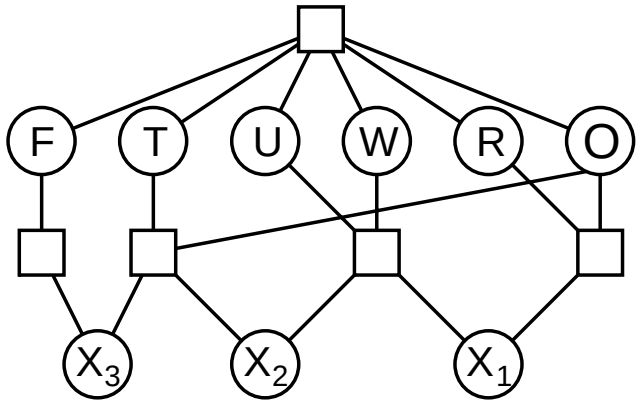
Constraint graph

- Binary CSP: sebuah constraint menyangkut hubungan maks. 2 variable.
- Constraint graph: representasi di mana node adalah variable, edge adalah constraint, mis:



Contoh lain: cryptarithmetic

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



- Variabel: $\{F, O, R, T, U, W, X_1, X_2, X_3\}$
- Ranah: $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Syarat:
  - $alldiff(F, O, R, T, U, W)$
  - $O + O = R + 10X_1$
  - $X_1 + W + W = U + 10X_2$
  - $X_2 + T + T = O + 10X_3$
  - $F = X_3$

Jenis CSP

- Variabel diskrit
  - Ranah berhingga: dengan ukuran d, ada $O(d^n)$ kemungkinan assignment yang lengkap.
  - Ranah tak hingga, mis: integer, string, dll.
    - Mis: penjadwalan (kuliah, bis, pekerjaan, dll.)
    - Perlu *constraint language*, mis: $StartJob_1 + 5 \leq StartJob_3$
    - Integer CSP dengan *linear constraint* dapat diselesaikan
- Variabel kontinyu: *linear programming*

Jenis constraint

- Unary constraint: menyatakan persyaratan sebuah variabel, mis: $SA \neq g$
- Binary constraint: menyatakan persyaratan sepasang variabel, mis: $SA \neq WA$
- n-ary constraint (*higher-order*): menyatakan persyaratan tiga atau lebih variabel, mis: cryptarithmetic
- Preference, atau *soft* constraint: syarat yang *sebaiknya* dipenuhi, tetapi tidak harus. Mis: *r* lebih baik dari *g*. Biasanya dinyatakan sebagai *cost* sebuah nilai variabel. $\rightarrow$ constrained optimization problem.

Masalah CSP betulan

- Penugasan kuliah, mis: siapa mengajar apa?
- Penjadwalan kuliah, mis: IKI30320 jam berapa? di ruangan mana?
- Konfigurasi hardware (constraint: budget, kebutuhan spesifikasi)
- Spreadsheet
- Penjadwalan angkutan umum
- Penjadwalan kerja di pabrik
- Penataan ruang

Outline

Menyelesaikan CSP dengan search biasa

Mari kita mulai dengan pendekatan sederhana...

Formulasi masalah CSP sebagai search

- Initial state: *assignment* kosong: {}
- Successor function: pilih nilai untuk sebuah variabel yang belum di-assign yang sah: tidak konflik dengan *assignment*. Jika tidak ada: gagal!
- Goal test: apakah *assignment* sudah lengkap

- Bisa menyelesaikan semua masalah CSP
- Solusi pasti di depth n untuk n variabel $\rightarrow$ depth first search
- Path tidak penting
- $b = (n - \ell)d$ pada depth $\ell \rightarrow$ ada $n!d^n$ leaf...

Backtracking search

- *Variable assignment* berlaku komutatif, dalam arti: [*WA=red* lalu *NT=green*] sama saja [*NT=green* lalu *WA=red*]
- Pada tiap level, hanya perlu meng-assign satu variabel $b = d$, ada d^n leaf...
- Depth first search pada CSP dengan assignment satu variabel tiap level disebut backtracking search.
- Algoritma uninformed standar untuk masalah CSP
- Bisa menyelesaikan n-queens problem untuk $n = 25$.

Backtracking search

function BACKTRACKINGSEARCH(*csp*) **returns** *solution/failure*

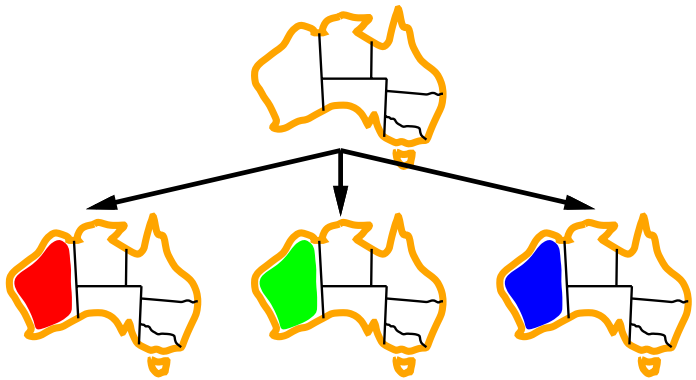
return RECURSIVEBACKTRACKING([],*csp*)

function RECURSIVEBACKTRACKING(*assignment*, *csp*) **returns** *solution/failure*

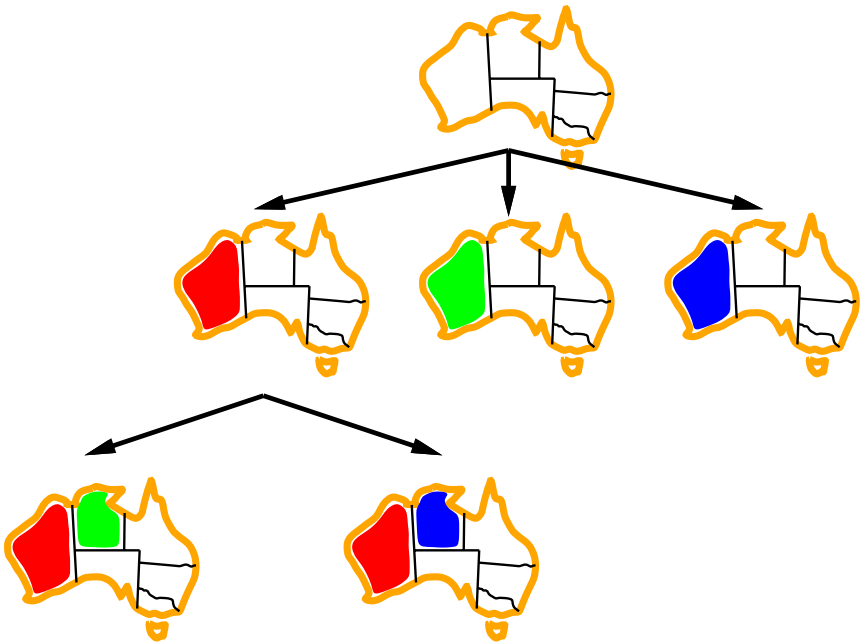
if *assigned* sudah lengkap **then return** *assigned*
var $\rightarrow$ SELECTUNASSIGNEDVARIABLE(*assigned*,*csp*)
for each *value* **in** ORDERDOMAINVALUES(*var*,*assigned*,*csp*) **do**
  if *value* tidak menimbulkan konflik dengan *assigned* dan *csp* **then**
    result $\rightarrow$ RECURSIVEBACKTRACKING([*var=value*|*assigned*,*csp*])
    if *result* $\neq$ *failure* **then return** *result*
end
return *failure*

Contoh backtracking

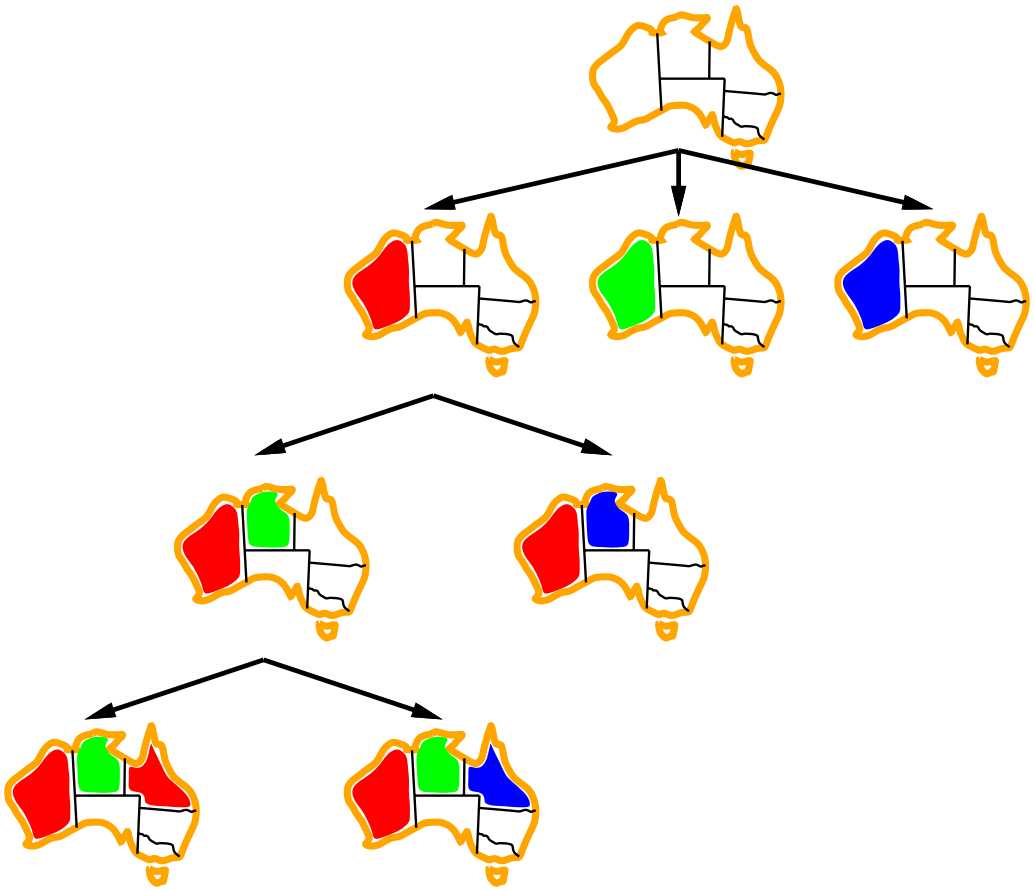
Contoh backtracking



Contoh backtracking



Contoh backtracking



Outline

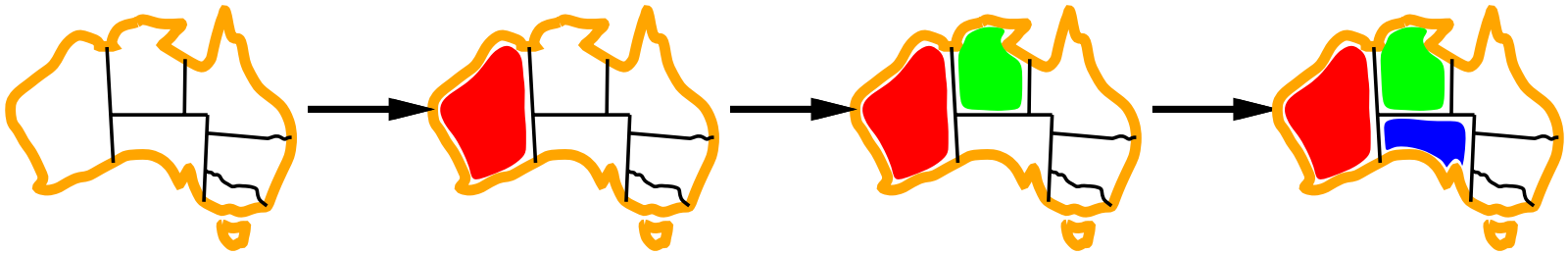
Memperbaiki kinerja backtracking

- Urutan pemilihan variable dan nilai mempengaruhi kinerja backtracking
- Terdapat beberapa strategi yang berlaku secara umum (*general-purpose*):
  1. Variable mana yang perlu di-assign terlebih dulu?
  2. Nilai apakah yang perlu dicoba terlebih dulu?
  3. Apakah kita bisa mendeteksi kepastian *failure* lebih awal?
  4. Apakah kita bisa memanfaatkan struktur masalah CSP? (Representasinya jelas!)

Prinsip 1: Most Constrained Variable

Variabel paling dibatasi

Pilih variable yang memiliki kemungkinan nilai sah paling sedikit

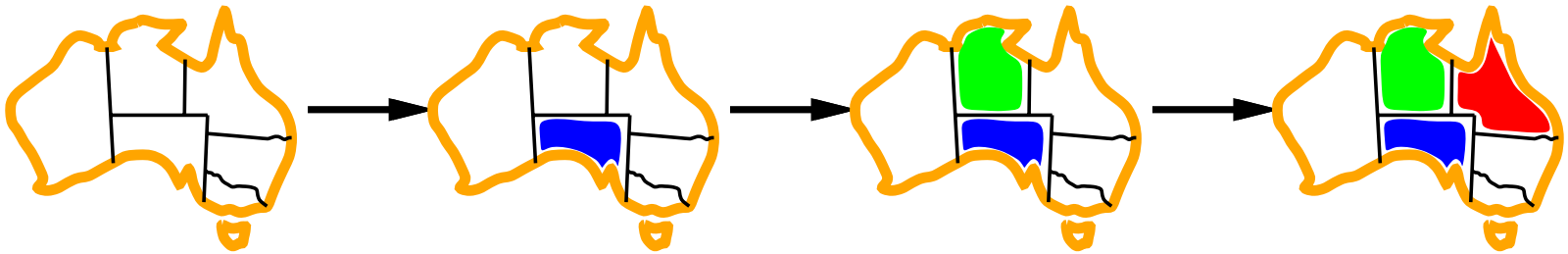


Prinsip 2: Most Constraining Variable

Variable paling membatasi

Pilih variable yang terlibat constraint dengan variable lain (yang belum di-*assign*) yang paling banyak.

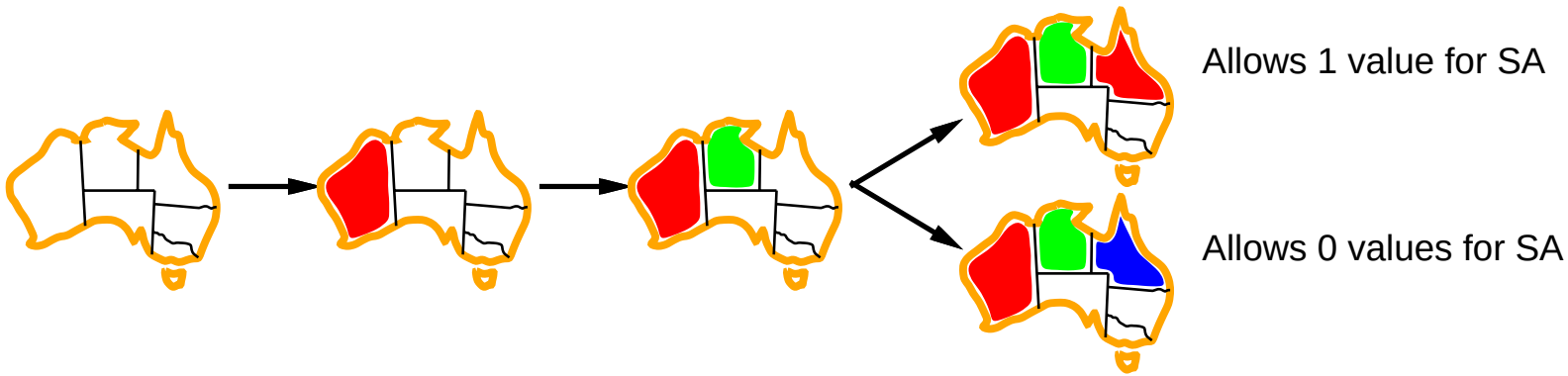
Tie – breaker: gunakan kalau ada 2 atau lebih variable yang sama bagusnya berdasarkan prinsip 1.



Prinsip 3: Least Constraining Value

Nilai paling membebasi

Pilih nilai yang menimbulkan batasan kemungkinan nilai variable lain (yang belum di-*assign*) yang paling sedikit.



Jika ketiga prinsip ini digunakan, masalah *n*-queens dengan *n*=1000 bisa diselesaikan!

Forward checking

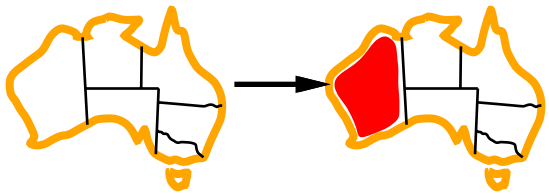
Forward checking

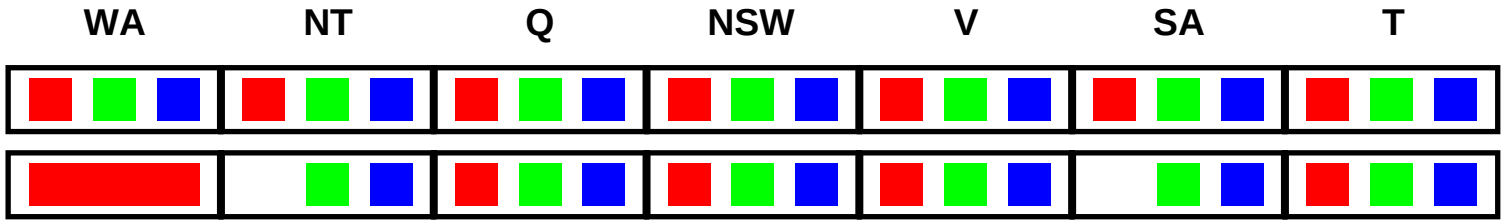
Catat kemungkinan nilai sah untuk semua variable yang belum di-assign. Jika ada sebuah variable yang tidak ada kemungkinan nilai sah, langsung *failure* (backtrack).

Forward checking

Forward checking

Catat kemungkinan nilai sah untuk semua variable yang belum di-assign. Jika ada sebuah variable yang tidak ada kemungkinan nilai sah, langsung *failure* (backtrack).

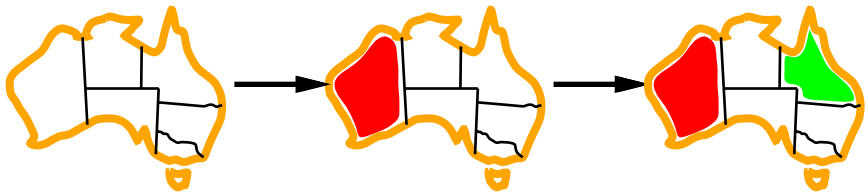


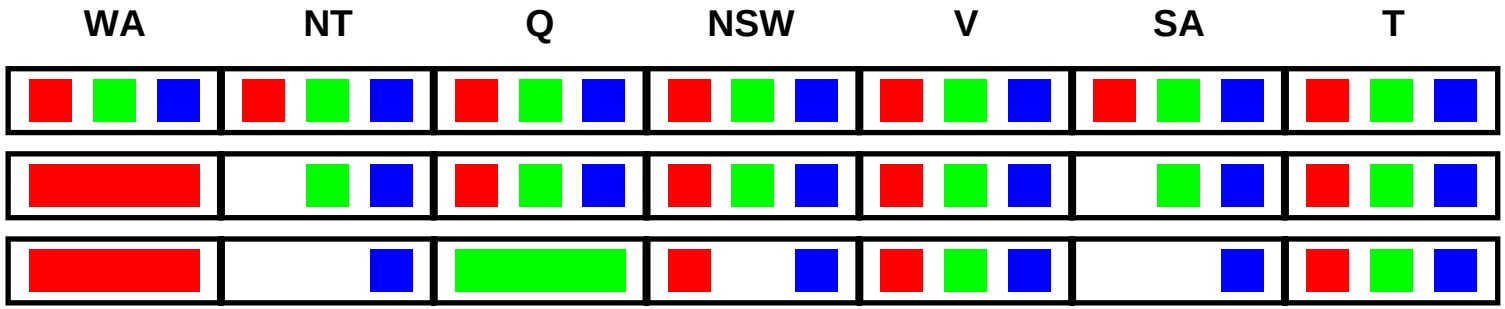


Forward checking

Forward checking

Catat kemungkinan nilai sah untuk semua variable yang belum di-assign. Jika ada sebuah variable yang tidak ada kemungkinan nilai sah, langsung *failure* (backtrack).

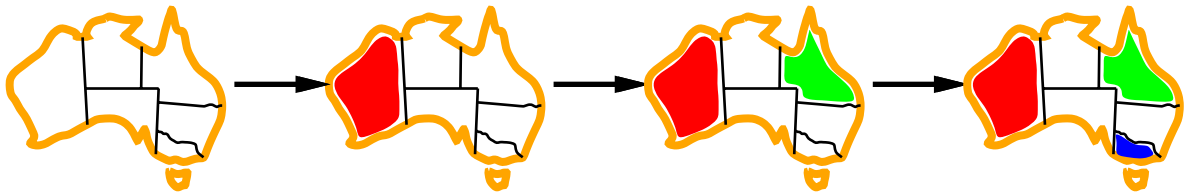


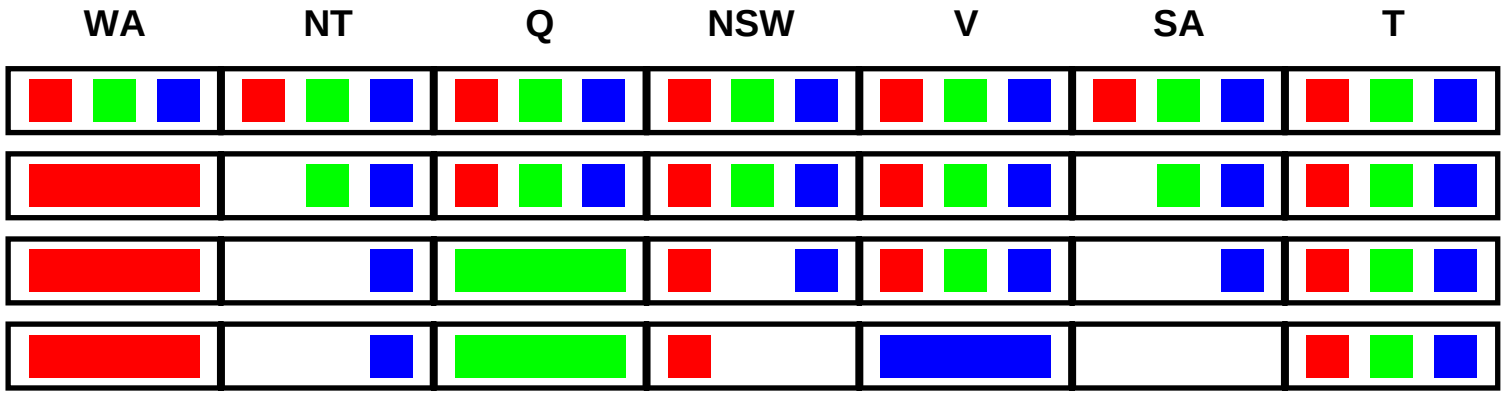


Forward checking

Forward checking

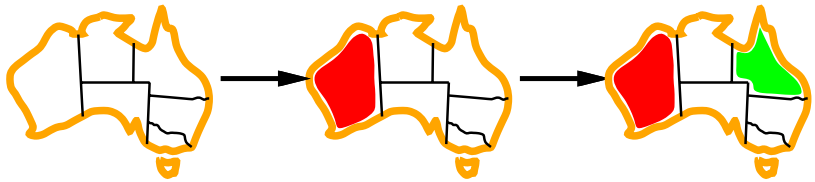
Catat kemungkinan nilai sah untuk semua variable yang belum di-assign. Jika ada sebuah variable yang tidak ada kemungkinan nilai sah, langsung *failure* (backtrack).





Constraint propagation

- Forward checking mem-propagasi (meneruskan) informasi dari variable yang sudah di-assign ke yang belum. Secara umum, ini disebut *constraint propagation*.
- Namun, tidak semua *failure* bisa di-deteksi secara dini:



NT dan *SA* tetanggaan → tidak boleh sama-sama biru!

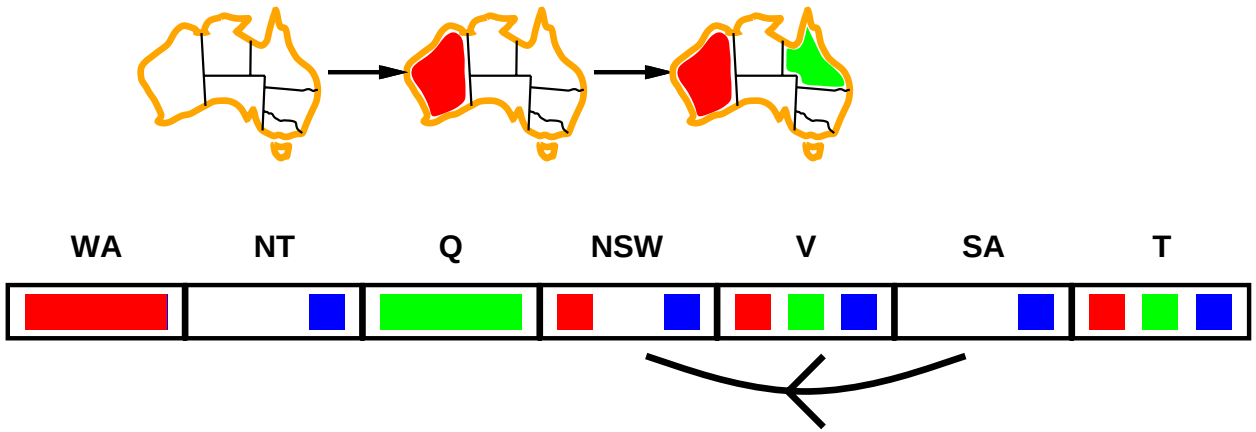
- Forward checking hanya mempertimbangkan setiap constraint secara terpisah.

Arc Consistency

- Arc consistency adalah metode *constraint propagation* yang lebih canggih. Konsistensi *antar* constraint dipertahankan.
- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan *konsisten* jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.

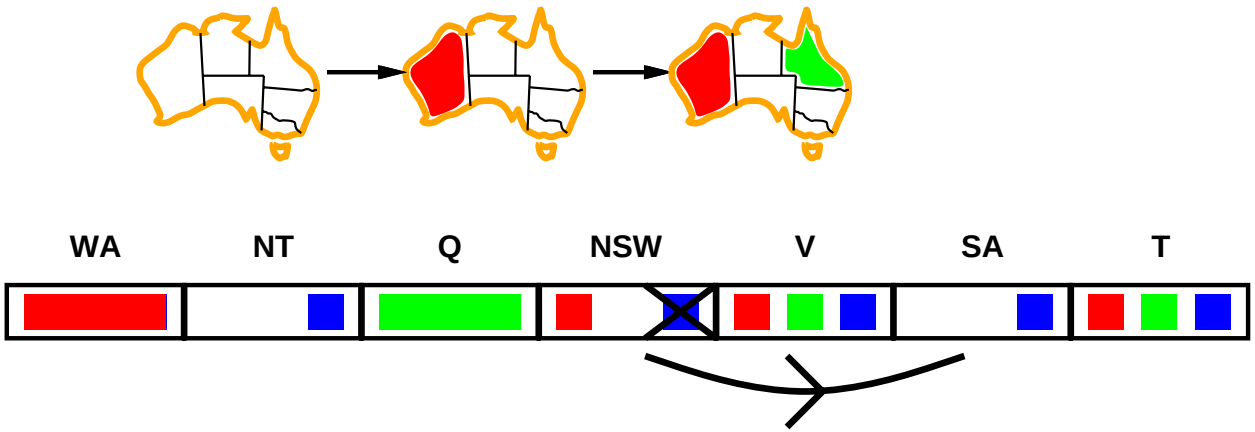


Arc Consistency

- Arc consistency adalah metode *constraint propagation* yang lebih canggih. Konsistensi *antar* constraint dipertahankan.
- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan *konsisten* jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.

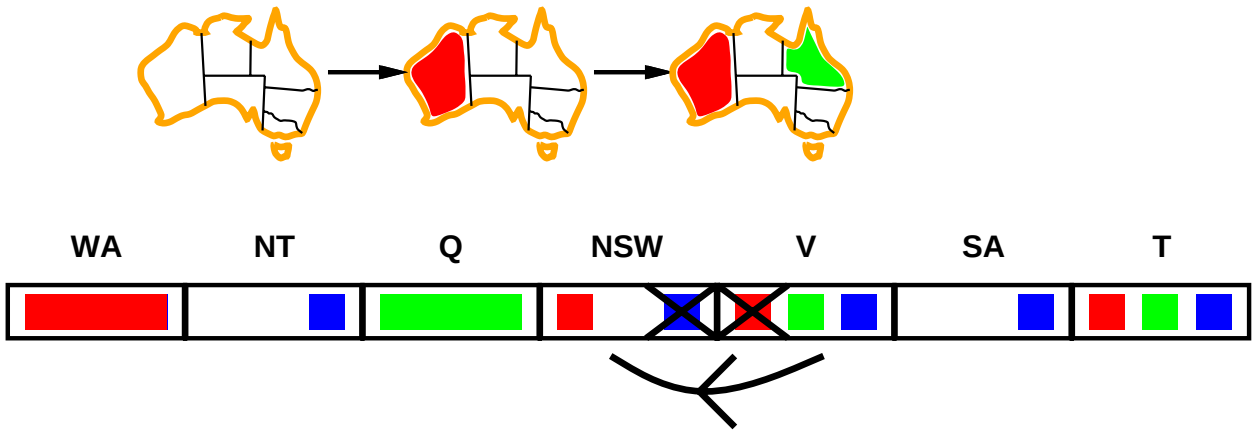


Arc Consistency

- Arc consistency adalah metode *constraint propagation* yang lebih canggih. Konsistensi *antar* constraint dipertahankan.
- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan *konsisten* jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.

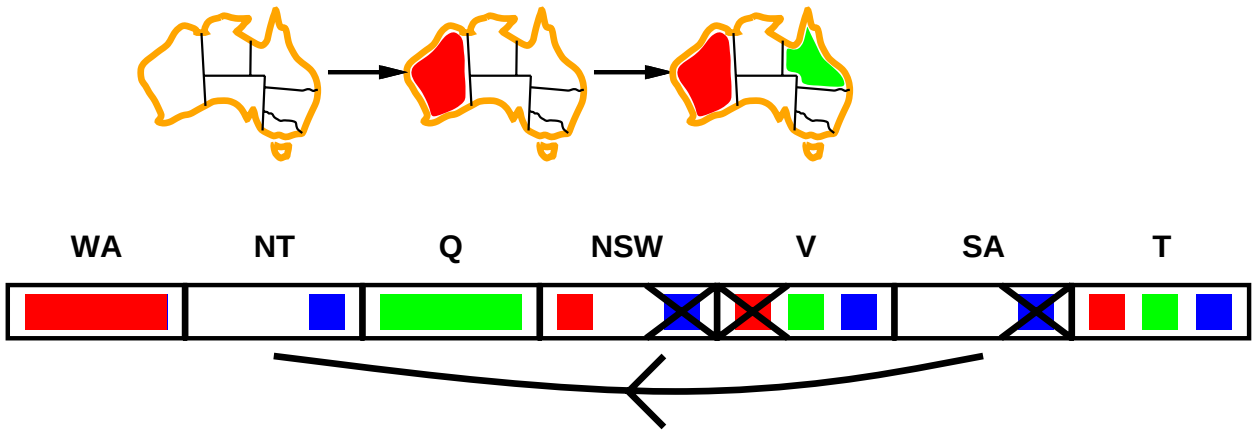


Arc Consistency

- Arc consistency adalah metode *constraint propagation* yang lebih canggih. Konsistensi *antar* constraint dipertahankan.
- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan *konsisten* jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.



Algoritma AC3

- Algoritma AC3 melakukan constraint propagation sbb:
  1. Periksa semua *arc* dalam constraint graph
  2. Jika untuk $X \rightarrow Y$ ada nilai sah x dari X sehingga tidak ada nilai sah y dari Y, buang x.
  3. Jika ada nilai x yang dibuang, periksa ulang semua "tetangga" X di dalam *constraint graph*.
- Algoritma AC3 ini bisa mendeteksi *failure* lebih cepat daripada *forward checking*.
- AC3 dapat dijalankan sebagai pra-proses sebelum melakukan *backtracking*, atau bisa dijalankan setelah setiap *assignment*.

Algoritma AC3

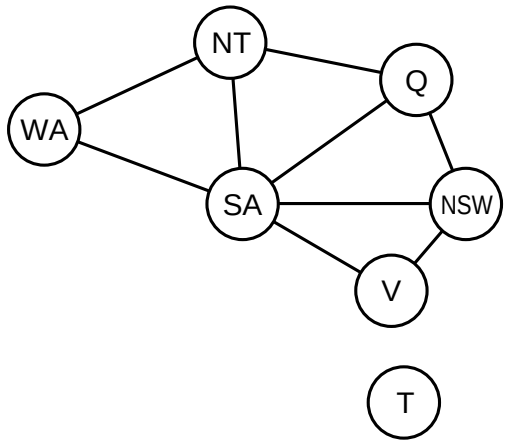
function AC3(*csp*) **returns** *csp* dengan *domain* disederhanakan

queue $\rightarrow$ semua *arc* dalam *csp*
loop while *queue* belum kosong **do**
  $(X_i, X_j) \rightarrow$ DEQUEUE(*queue*)
  if REMOVEINCONSISTENT(X_i, X_j) **then**
    for each $X_k \in$ NEIGHBOURS$[X_i]$ **do**
      tambahkan (X_k, X_i) ke *queue*

function REMOVEINCONSISTENT(X_i, X_j) **returns** *true* jhj nilai dibuang

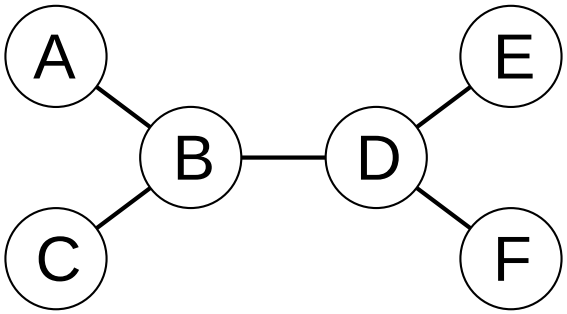
removed $\rightarrow$ *false*
loop for each $x \in$ DOMAIN$[X_i]$ **do**
  if tidak ada $y \in$ DOMAIN$[X_j]$ shg (x, y) memenuhi constraint (X_i, X_j)
    then buang x dari DOMAIN$[X_i]$; *removed* $\rightarrow$ *true*
return *removed*

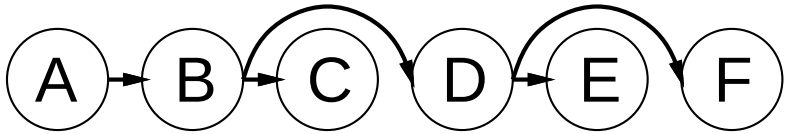
Submasalah independen



- *Assignment T* (Tasmania) adalah submasalah independen.
- Andaikan CSP dengan n variabel $\rightarrow$ submasalah masing2 c variabel:
  - Dari $O(d^n)$ menjadi $O(n/c \times d^c)$
  - Mis. boolean CSP $n = 80$, $d = 2$, $c = 20$, bisa memroses 10 juta node/detik
  - $2^{80} \approx 4$ miliar tahun
  - $4 \times 2^{20} \approx 0.4$ detik

CSP dengan constraint tree





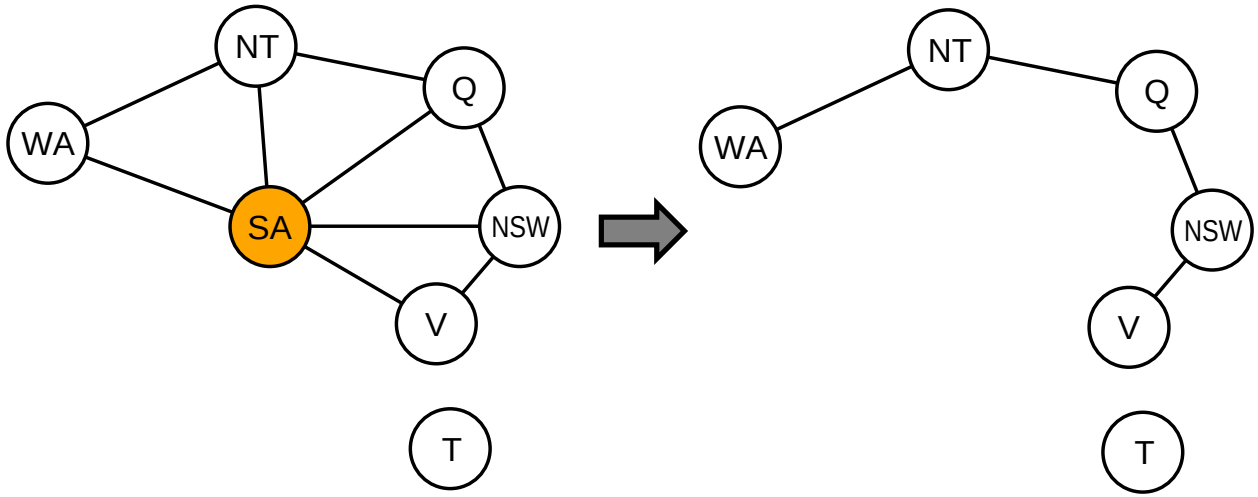
- 2 variable terhubung melalui maks. 1 path. (dkl: tidak ada loop)
- Bisa diselesaikan dalam $O(nd^2)$ (bandingkan dengan kasus umum: $O(d^n)$)

Algoritma:

1. Pilih sembarang variable sbg. root. Urutkan node shg. untuk setiap node, *parent*-nya di sebelah kiri.
2. For $i = n$ to 2, panggil REMOVEINCONSISTENT($Parent(X_i), X_i$).
3. For $i = 1$ to n, *assign* nilai X_i sehingga konsisten dengan $Parent(X_i)$.

- Ini contoh hubungan *trade-off* antara kompleksitas representation dan reasoning.

CSP dengan constraint graph hampir tree



Algoritma:

1. Pilih subset C dari CSP shg kalau C dibuang, "sisa"-nya, $R \rightarrow$ tree.
2. Untuk setiap solusi C (mis: backtracking), buang nilai tidak sah dari R.
3. Cari solusi R dengan algoritma untuk CSP berbentuk tree.

- Andaikan c ukuran C, running time $O(d^c \times (n-c)d^2)$
- Jika CSP "hampir" tree, c kecil $\rightarrow$ cepat!
- C disebut cycle cutset. Pencarian *cycle cutset* terkecil: NP-hard.

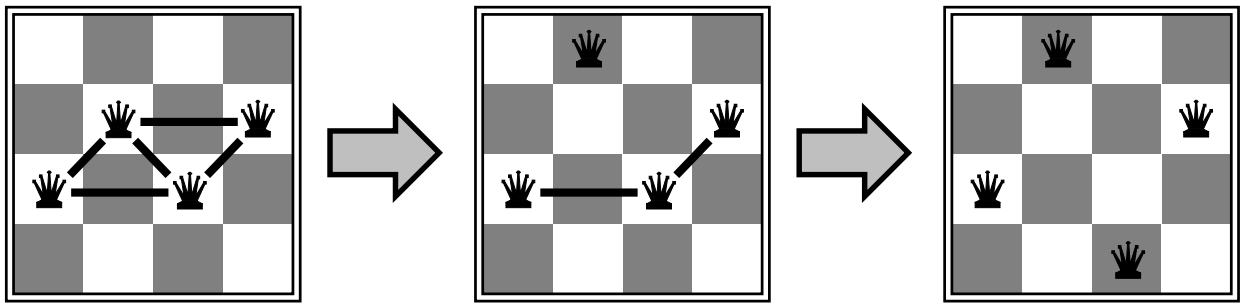
Outline

Pendekatan local search untuk solusi CSP

- Dalam praktek, local search cocok untuk CSP.
- State harus lengkap/complete (semua variable harus ter-*assign*) tapi boleh melanggar constraint.
- Operator/action-nya: *menukar* nilai variabel (*reassign*).
- Pemilihan variable: pilih secara acak variable yang melanggar sebuah constraint.
- Pemilihan nilai: gunakan *heuristic* minimum conflict: pilih nilai yang melanggar constraint paling sedikit.
- Lakukan local search (hillclimb, simulated annealing, genetic algorithm, dll.) meminimalkan $h(n) =$ jumlah pelanggaran constraint
- Perhatian: secara teoritis pendekatan ini tidak dijamin complete.

Local search untuk CSP 4-queens

- State: 4 menteri dalam 4 kolom ($4^4 = 256$ state)
- Operator/action: pindahkan menteri dalam kolom
- Goal test: tidak ada menteri saling makan
- Evaluation/fitness function: $h(n) =$ jumlah pasangan menteri saling makan
- Bisa menyelesaikan 1000000-queens problem!



Outline

Ringkasan

- CSP adalah masalah yang bentuknya spesifik:
  - State berupa assignment nilai terhadap himpunan variabel
  - Goal test berupa *constraint* terhadap nilai variabel.
- Backtracking: depth-first search mempertimbangkan satu variable di setiap node
- *Heuristic* urutan pemilihan variable dan nilai sangat mempengaruhi kinerja
- Forward checking dan arc consistency adalah bentuk constraint propagation yang mendeteksi kegagalan (*failure*) lebih dini.
- Representasi CSP yang spesifik memungkinkan analisis masalah $\rightarrow$ penyederhanaan: CSP berbentuk tree bisa diselesaikan dalam waktu linier.
- Metode local search untuk CSP (dengan heuristic min-conflicts) dalam praktek cukup efektif.