CSCM603130: Sistem Cerdas
Constraint Satisfaction Problems

Fariz Darari, Aruni Yasmin Azizah

Fakultas Ilmu Komputer
Universitas Indonesia

2019/2020• Semester Ganjil

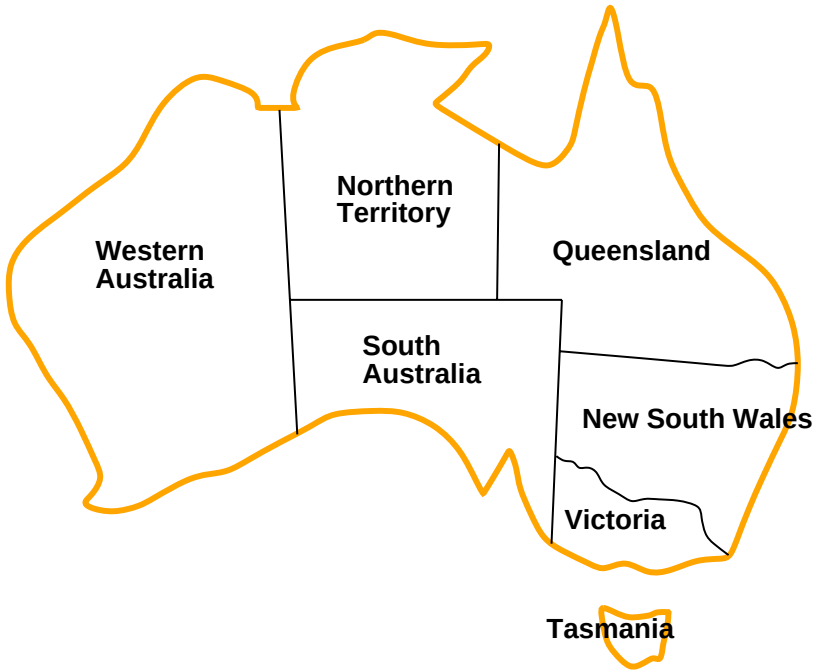
Outline

Outline

Mendefinisikan CSP

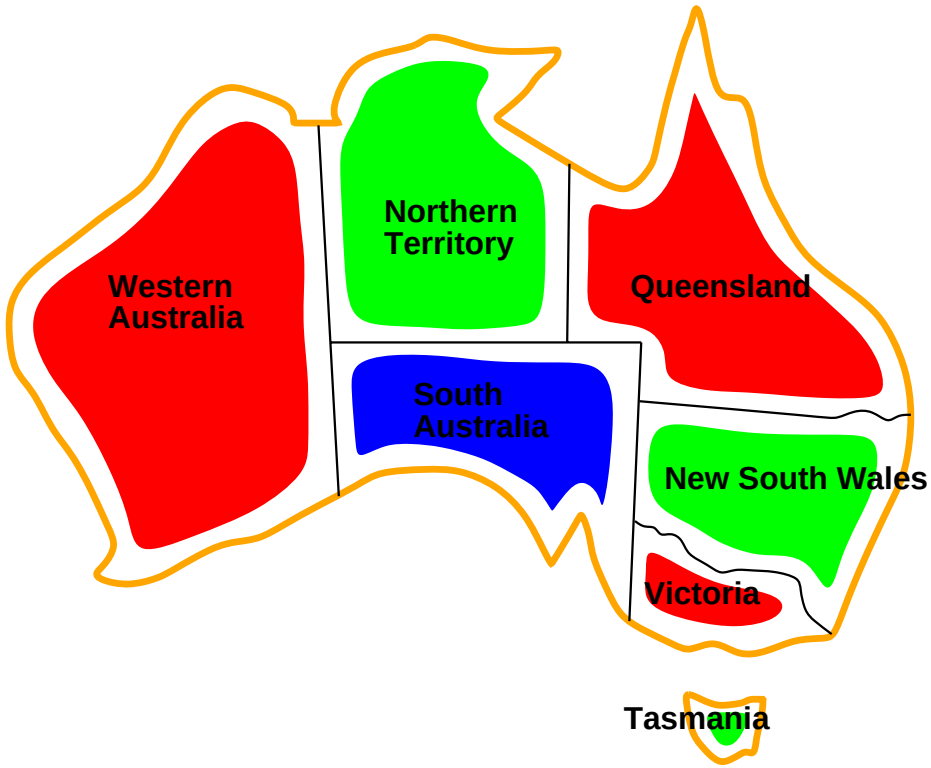
- Bentuk state pada teknik search sebelumnya: atomik (black-box).
- CSP: state memiliki factored representation.
  Terdiri dari himpunan X dari sejumlah variable X_i dengan nilai X_i dari domain D_i dan himpunan C dari sejumlah constraint/syarat C_j.
- Solusi adalah penetapan nilai (assignment) terhadap semua variable (complete) sehingga semua syarat dipenuhi (consistent).

Contoh CSP: Mewarnai Peta



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

Contoh Solusi CSP: Mewarnai Peta



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

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

Contoh lain: cryptarithmetic

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

- Variabel: $\{F, O, R, T, U, W, X_1, X_2, X_3\}$
- Domain: $D_i = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Syarat:
  - Nilai dari F, O, R, T, U, W harus berbeda satu sama lain
  - $O + O = R + 10X_1$
  - $X_1 + W + W = U + 10X_2$
  - $X_2 + T + T = O + 10X_3$
  - $F = X_3$

Variasi CSP (1) - Variable & Domain

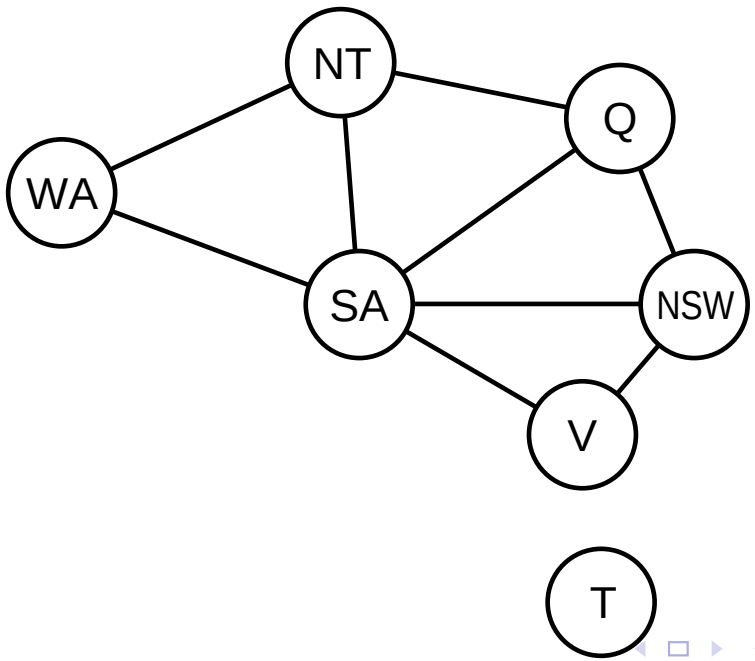
- Variabel dengan domain diskrit
  - Domain berhingga, mis: warna variable pada map-coloring.
  - Domain tak hingga, mis: integer, dll.
    - Contoh: job-shop scheduling (no deadline).
    - Perlu *constraint language*, mis: $StartJob_1 + 5 \leq StartJob_3$
    - CSP dengan variabel integer dan *linear constraint* dapat diselesaikan dengan pendekatan khusus.
- Variabel dengan domain kontinyu
  - Contoh: *linear programming*

Variasi CSP (2) - Constraint

- Unary constraint: menyatakan persyaratan sebuah variabel, mis: $SA \neq g$
- Binary constraint: menyatakan persyaratan sepasang variabel, mis: $SA \neq WA$
- Global constraint: menyatakan persyaratan sembarang banyaknya variabel, mis: constraints pada cryptarithmetic
- Preference, atau *soft* constraint: syarat yang *sebaiknya* dipenuhi, tetapi tidak harus. Biasanya dinyatakan sebagai *cost* sebuah nilai variabel $\rightarrow$ constraint optimization problem.

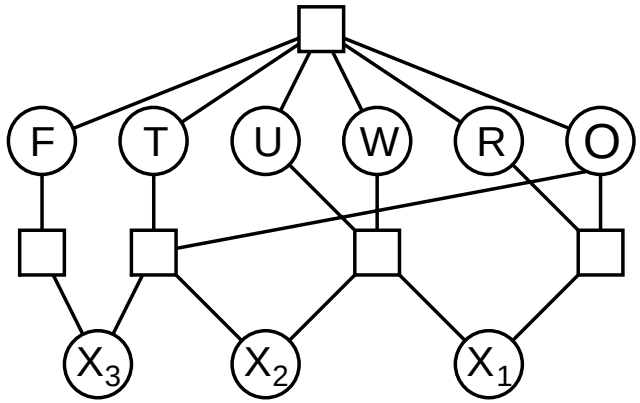
Constraint graph

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



Contoh lain: cryptarithmetic (constraint hypergraph)

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



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

Outline

Outline

Constraint propagation

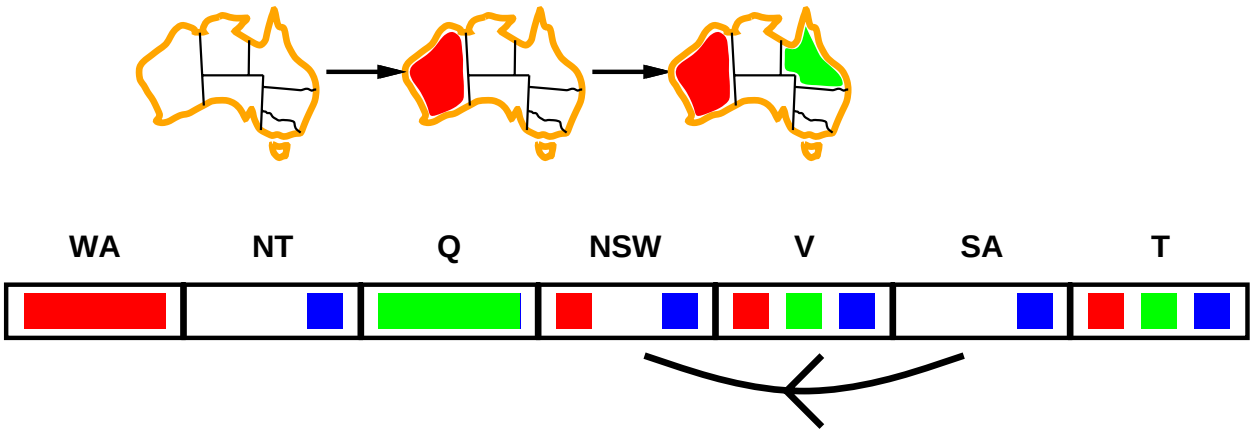
- Menyelesaikan CSP tidak hanya melalui search, tetapi melibatkan juga inferensi: constraint propagation.
- Constraint Propagation: mereduksi banyaknya nilai yang diijinkan untuk suatu variabel yang berdampak pada reduksi nilai variabel yang lain, dst.
- Dapat dilakukan bersama-sama dengan *search*, dapat juga sebagai *preprocessing*.
  - Terkadang constraint propagation sebagai preprocessing cukup untuk menyelesaikan problem (tanpa memerlukan search).
- Ide utama: local consistency pada constraint graph.
  - Dengan local consistency, nilai-nilai variabel yang inkonsisten pada graph akan dieliminasi secara sistematis.
  - Tipe-tipe: node consistency, arc consistency, path consistency, K-consistency, ...

Arc Consistency

- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan arc-consistent jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.

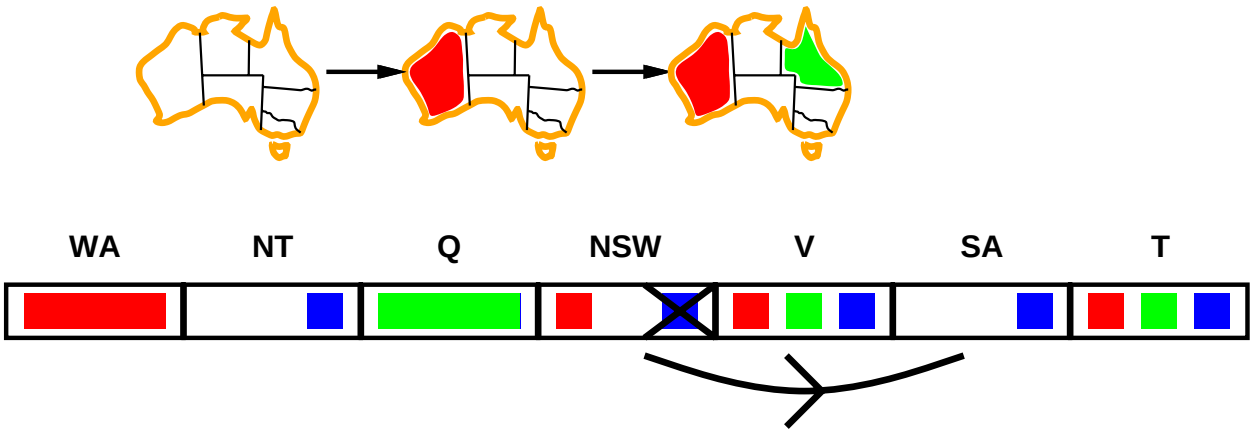


Arc Consistency

- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan arc-consistent jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.

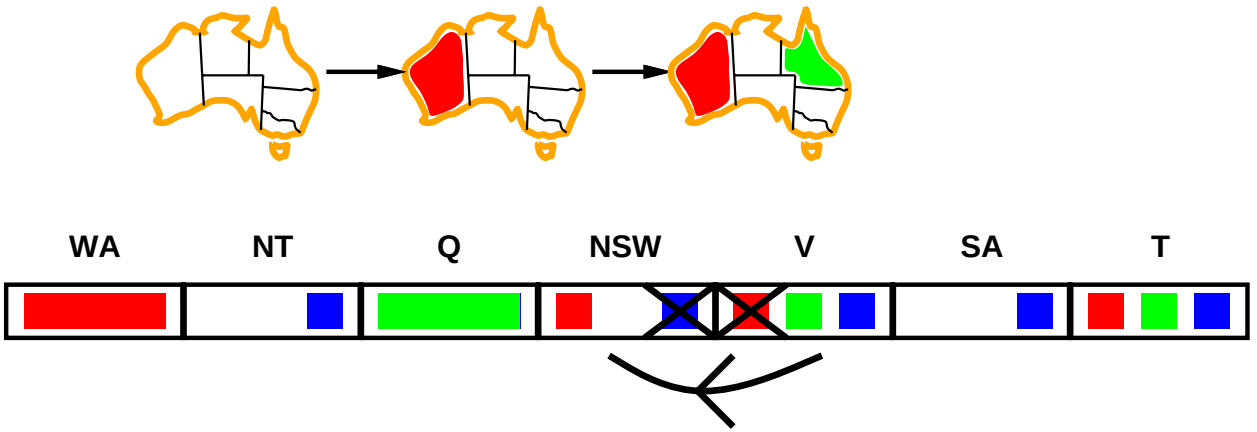


Arc Consistency

- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan arc-consistent jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.

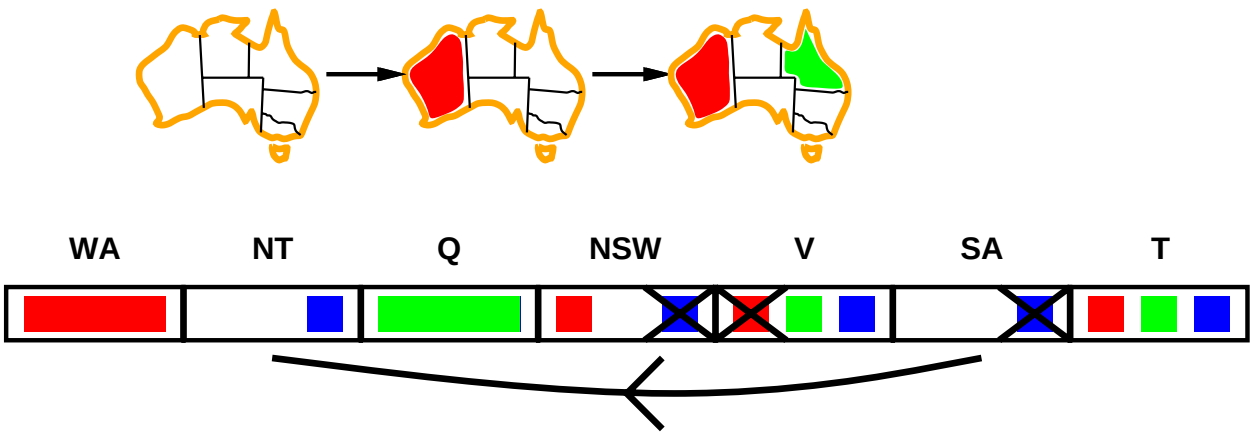


Arc Consistency

- Arc $X \rightarrow Y$ adalah edge satu arah dari variable X ke Y dalam *constraint graph*.

Prinsip Arc Consistency

$X \rightarrow Y$ dikatakan arc-consistent jika dan hanya jika untuk setiap nilai sah x dari X ada nilai sah y dari Y.



Sebagai pre-processor, arc-consistency mampu mendeteksi lebih awal kegagalan pemenuhan constraint.

Algoritma AC3

function AC3(*csp*) **returns** false (jika inkonsistensi ditemukan) atau true

queue ← semua *arc* dalam *csp*
loop while *queue* belum kosong **do**
  (X_i, X_j) ← REMOVE-FIRST(*queue*)
  if REVISE(*csp*, X_i, X_j) **then**
    if size of $D_i = 0$ **then return** *false*
    for each $X_k \in$ NEIGHBOURS[X_i] - $\{X_j\}$ **do**
      tambahkan (X_k, X_i) ke *queue*

function REVISE(*csp*, X_i, X_j) **returns** *true* iff domain X_i direvisi

revised ← *false*
loop for each $x \in D_i$ **do**
  if tidak ada $y \in D_j$ shg (x, y) memenuhi constraint (X_i, X_j)
    then buang x dari D_i
      revised ← *true*
return *revised*

Outline

Menyelesaikan CSP dengan search biasa

Mari kita mulai dengan pendekatan sederhana...

Menyelesaikan CSP dengan depth-limited search

- State: partial *assignment*
- Initial state: *assignment* kosong: $\{\}$
- RESULT function: pilih nilai untuk sebuah variabel yang belum di-assign
- Goal test: *assignment* yang complete dan consistent

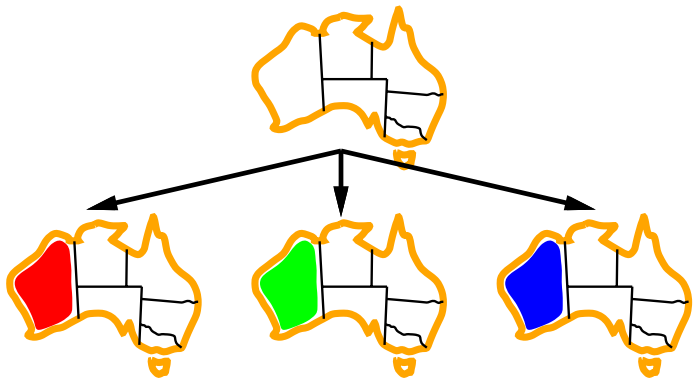
- Bisa menyelesaikan semua masalah CSP
- Solusi berada di depth n untuk n variabel
- Jika ukuran domain d, maka $b = (n - \ell)d$ pada depth $\ell \in \{0, 1, 2, \ldots, n - 1\}$
  $\rightarrow$ ada $n!d^n$ leaves (complete assignment)

Backtracking search

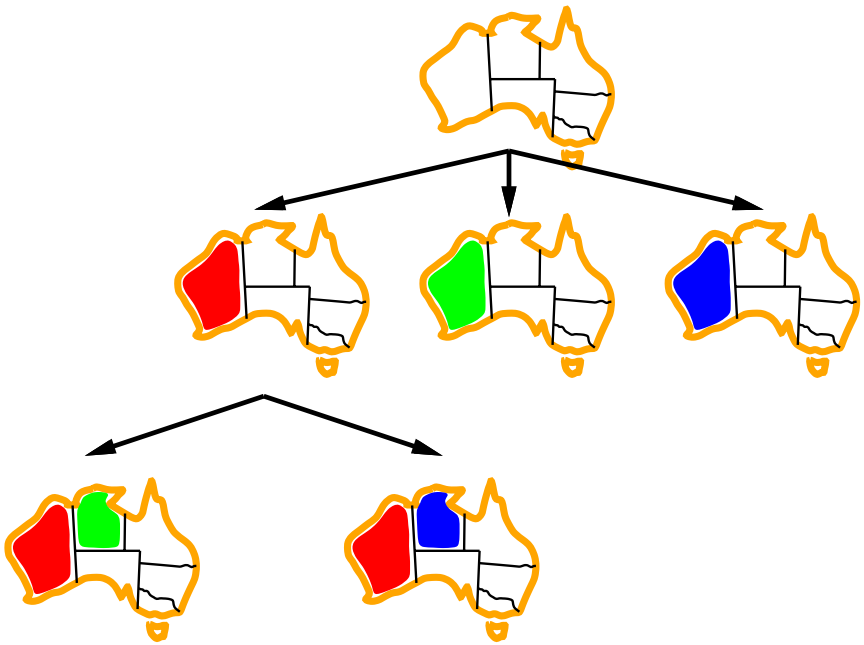
- *Variable assignment* berlaku komutatif, dalam arti: [*WA*=*red* lalu *NT*=*green*] sama saja [*NT*=*green* lalu *WA*=*red*]
- Pada tiap level, hanya perlu meng-assign satu variabel: $b = d$ $\rightarrow$ ada d^n leaves.
- Depth first search pada CSP dengan assignment satu variabel tiap level disebut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

Contoh backtracking

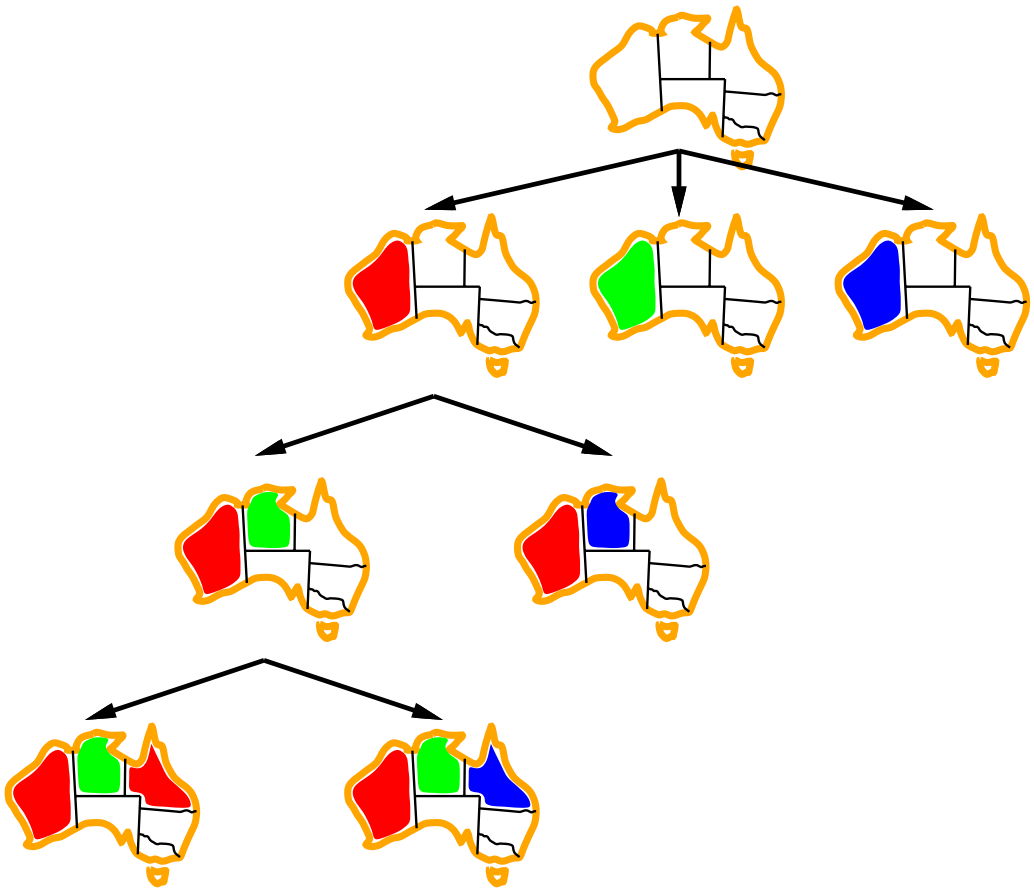
Contoh backtracking



Contoh backtracking



Contoh backtracking



Backtracking search

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

return BACKTRACKING({},*csp*)

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

if *assignment* sudah lengkap **then return** *assignment*
var ← SELECTUNASSIGNEDVARIABLE(*csp*)
for each *value* **in** ORDERDOMAINVALUES(*var*,*assignment*,*csp*) **do**
  if *value* konsisten dengan *assignment* **then**
    tambahkan {*var* = *value*} ke *assignment*
    inferences ← INFERENCE(*csp*, *var*, *value*)
    if *inferences* ≠ *failure* **then**
      tambahkan *inferences* ke *assignment*
      result ← BACKTRACKING(*assignment*, *csp*)
      if *result* ≠ *failure* **then**
        return *result*
  buang {*var* = *value*} dan *inferences* dari *assignment*
return *failure*

Isu-isu backtracking pada CSP

- Terdapat beberapa isu backtracking pada CSP (tidak domain-specific) yang dapat membantu menyelesaikan CSP secara efisien:
  1. Inferensi apa yang dapat dilakukan pada proses search? (INFERENCE)
  2. Jika search mendapatkan assignment yang melanggar constraint, dapatkah search menghindari untuk mengulangi kesalahan ini? (constraint learning dan conflict-directed backjumping)
  3. Variable mana yang perlu di-assign terlebih dulu? (SELECTUNASSIGNEDVARIABLE)
  4. Nilai apakah yang perlu dicoba terlebih dulu? (ORDERDOMAINVALUES)

Outline

Inferensi pada search: "Forward checking"

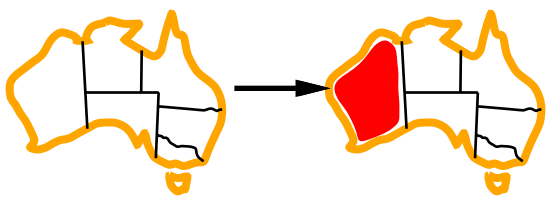
Forward checking

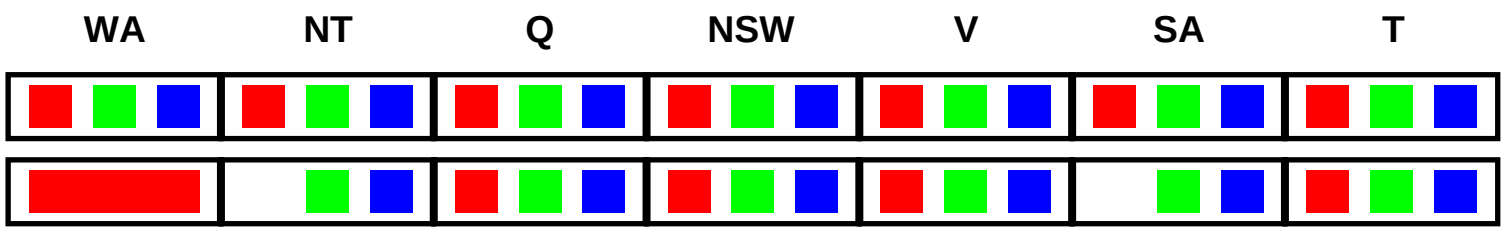
Ketika variabel X di-assign, forward checking mengupayakan arc-consistency terhadap X: untuk setiap variabel Y yang belum di-assign dan terhubung ke X melalui constraint, hapus nilai pada Y yang inkonsisten dengan nilai yang telah dipilih untuk X.

Inferensi pada search: "Forward checking"

Forward checking

Ketika variabel X di-assign, forward checking mengupayakan arc-consistency terhadap X: untuk setiap variabel Y yang belum di-assign dan terhubung ke X melalui constraint, hapus nilai pada Y yang inkonsisten dengan nilai yang telah dipilih untuk X.

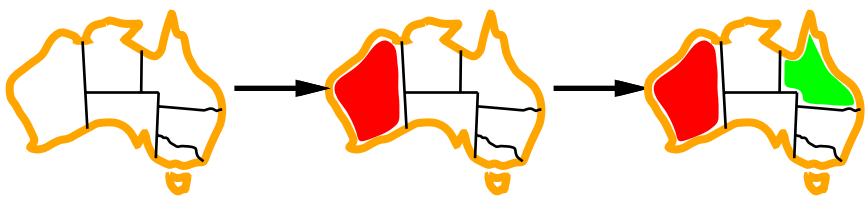




Inferensi pada search: "Forward checking"

Forward checking

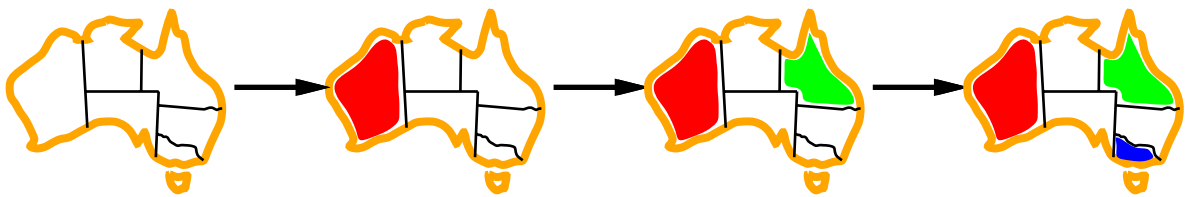
Ketika variabel X di-assign, forward checking mengupayakan arc-consistency terhadap X: untuk setiap variabel Y yang belum di-assign dan terhubung ke X melalui constraint, hapus nilai pada Y yang inkonsisten dengan nilai yang telah dipilih untuk X.

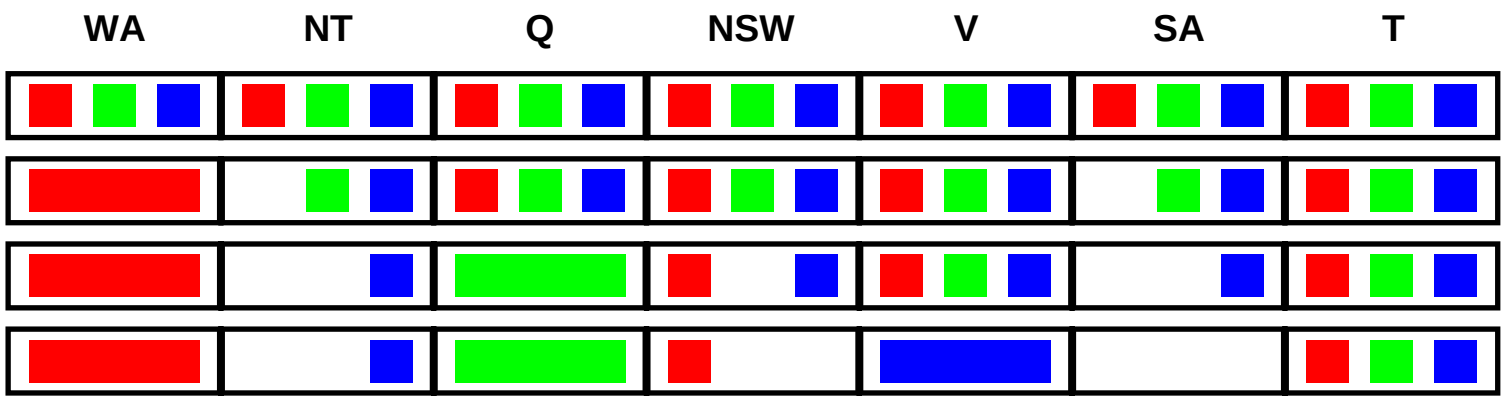


Inferensi pada search: "Forward checking"

Forward checking

Ketika variabel X di-assign, forward checking mengupayakan arc-consistency terhadap X: untuk setiap variabel Y yang belum di-assign dan terhubung ke X melalui constraint, hapus nilai pada Y yang inkonsisten dengan nilai yang telah dipilih untuk X.





Menghindari pelanggaran constraint

Partial assignment $\{Q$=*red*, NSW=*green*, V=*blue*, T=*red*$\}$, assignment selanjutnya adalah T=*red*, kemudian SA.

- Backjumping V, karena perubahan assignment T tidak menyelesaikan masalah untuk assignment SA.

Partial assignment $\{WA$=*red*, NSW=*red*$\}$, assignment selanjutnya adalah T=*red*, kemudian NT, Q, V, SA.

- Backjumping tidak menyelesaikan masalah, karena NT memiliki nilai yang konsisten dengan WA, NSW, T
- Conflict-directed backjumping, dengan conflict set $\{WA, NSW\}$

Conflict-directed Backjumping

Dengan X_j adalah *current variable* dan conflict set $conf(X_j)$, jika setiap nilai X_j melanggar constraint, backjump ke most recent variable X_i pada $conf(X_j)$, dan set: $conf(X_i) \leftarrow conf(X_i) \cup conf(X_j) - \{X_i\}$

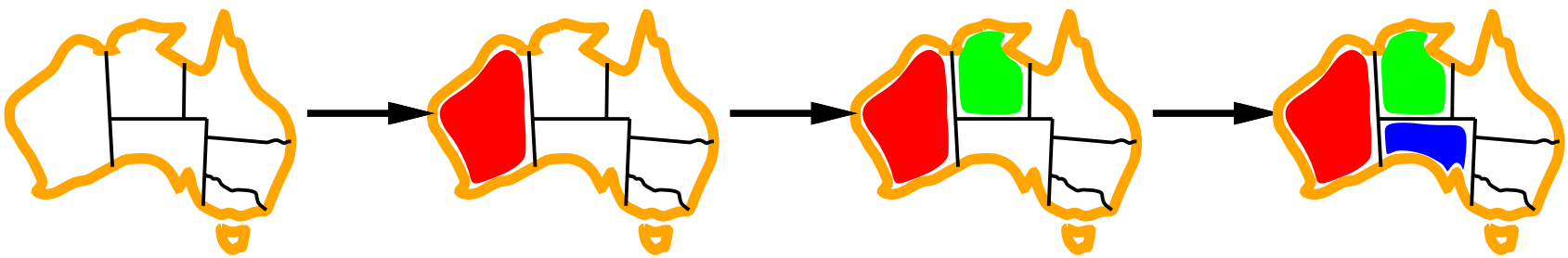
Outline

Prinsip "Most Constrained Variable"

Dikenal juga dengan nama minimum-remaining-values (MRV) heuristic.

Variabel paling dibatasi

Pilih variabel yang memiliki kemungkinan nilai sah paling sedikit



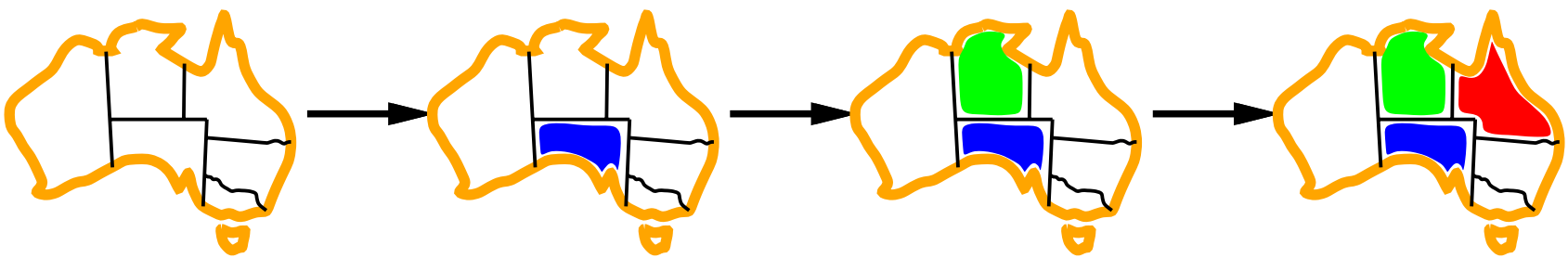
Prinsip "Most Constraining Variable"

Dikenal juga dengan name degree heuristic.

Variable paling membatasi

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

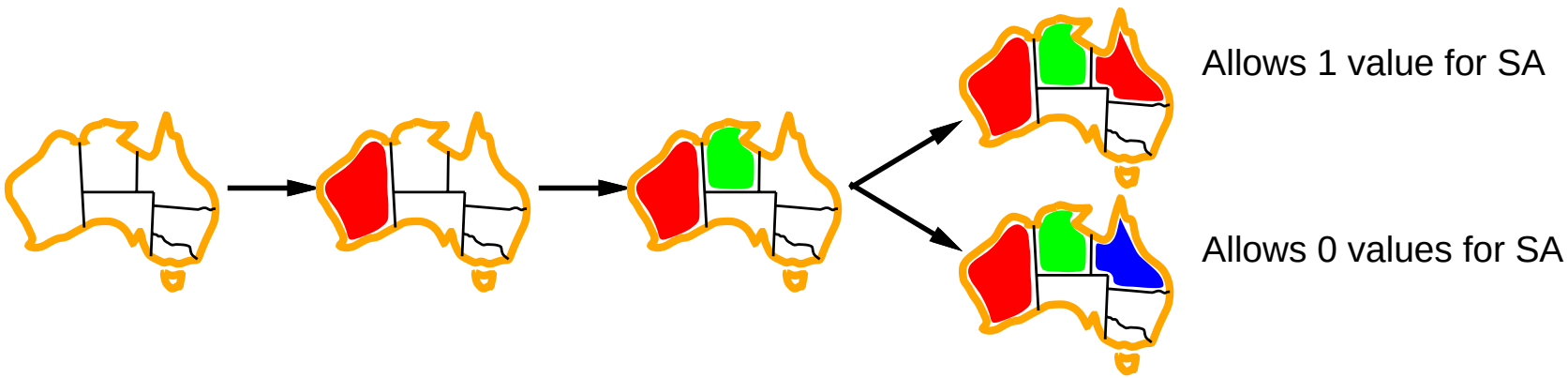
Berfungsi sebagai *tie-breaker*: gunakan kalau ada 2 atau lebih variable yang sama bagusnya berdasarkan prinsip most constrained variable.



Prinsip "Least Constraining Value"

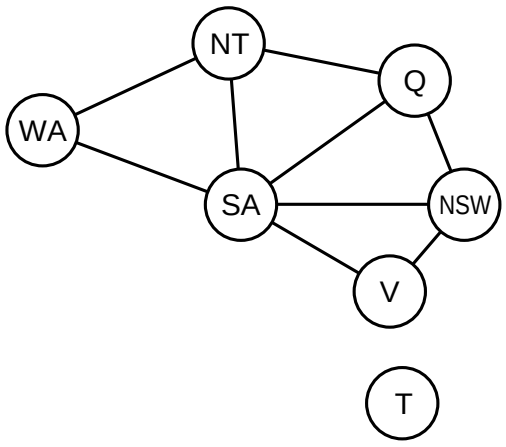
Nilai paling membebasi

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



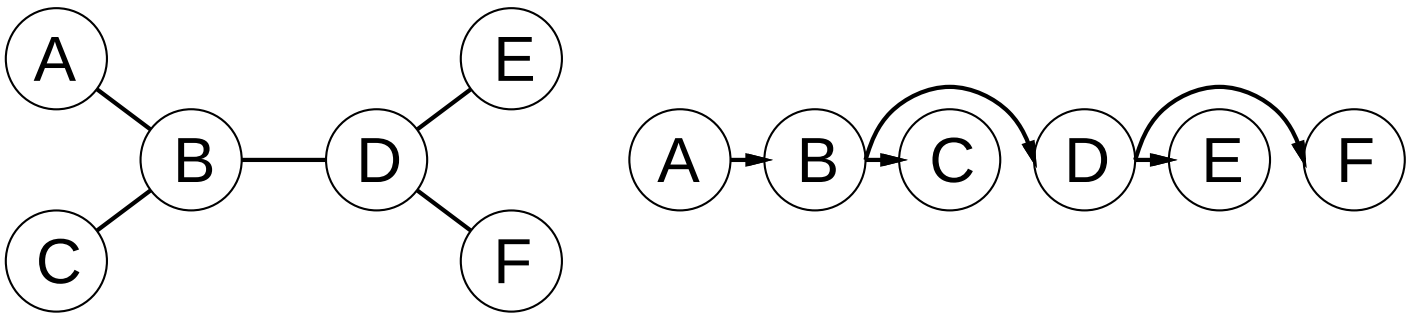
Outline

Submasalah independen



- *Assignment* T (Tasmania) adalah submasalah independen.
- Andaikan CSP dengan n variabel $\rightarrow$ submasalah masing-masing 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

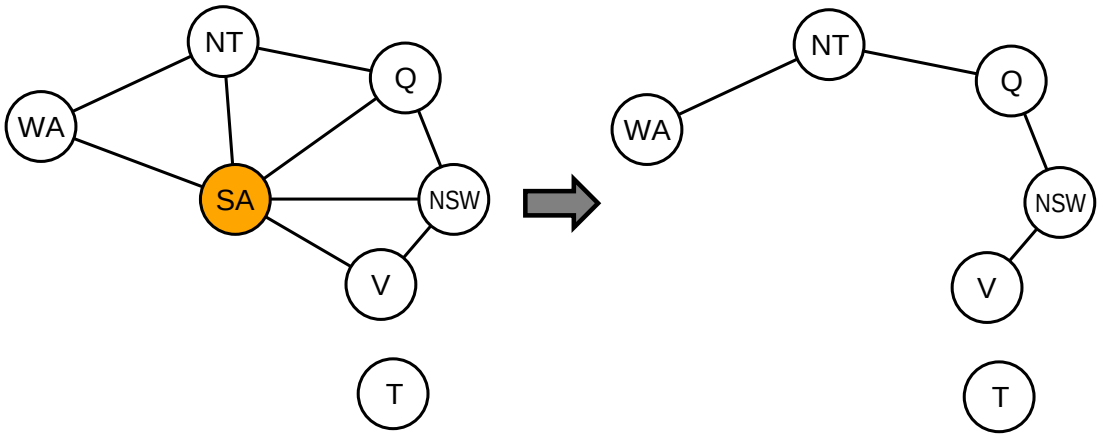


- 2 variable terhubung melalui maks. 1 path. (dkl: tidak ada loop)
- Bisa diselesaikan dalam $O(nd^2)$ (bandingkan kasus umum: $O(d^n)$)

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 MAKE-ARC-CONSISTENT($Parent(X_i), X_i$).
3. For $i = 1$ to n, *assign* nilai konsisten X_i dari D_i.

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 ukuran C adalah 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: hard problem.

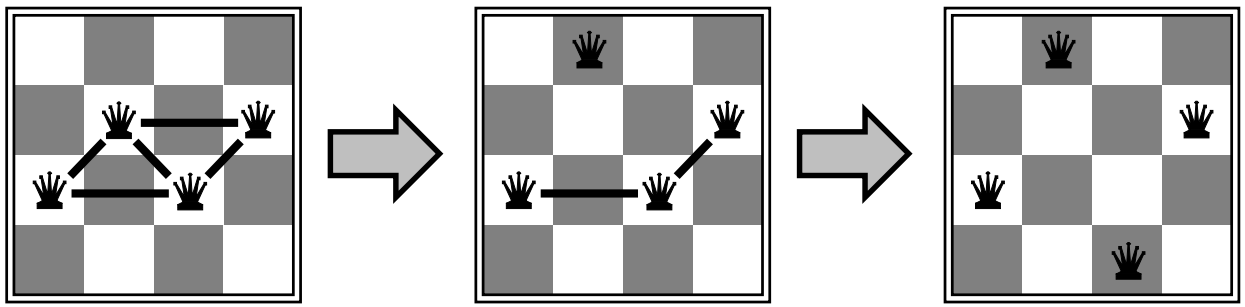
Outline

Pendekatan local search untuk solusi CSP

- Dalam praktek, local search cocok untuk CSP.
- Initial 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 (gradient descent, 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
- Constraint: tidak ada menteri saling makan
- Evaluation function: $h(n) =$ jumlah pasangan menteri saling makan
- Bisa menyelesaikan 1000000-queens problem!



Ringkasan

- CSP adalah masalah yang bentuknya spesifik:
  - State berupa assignment nilai terhadap variabel
  - Solusi berupa assignment nilai terhadap semua variabel (complete) sehingga semua constraint terpenuhi (consistent)
- Menyelesaikan CSP tidak hanya melalui search, tetapi melibatkan juga inferensi: constraint propagation.
- Backtracking: depth-first search mempertimbangkan variable satu per satu dan backtrack jika tidak ada nilai yang bisa di-assign pada suatu variabel
- Urutan pemilihan variable dan nilai sangat mempengaruhi kinerja backtracking
- Representasi CSP yang spesifik memungkinkan analisis masalah
- Metode local search untuk CSP (meminimalkan conflicts) dalam praktek cukup efektif.