CSCM603130: Sistem Cerdas
Beyond Classical Search

Fariz Darari, Aruni Yasmin Azizah

Fakultas Ilmu Komputer
Universitas Indonesia

2019/2020 • Semester Ganjil

Outline

Outline

Sebuah pendekatan yang berbeda . . .

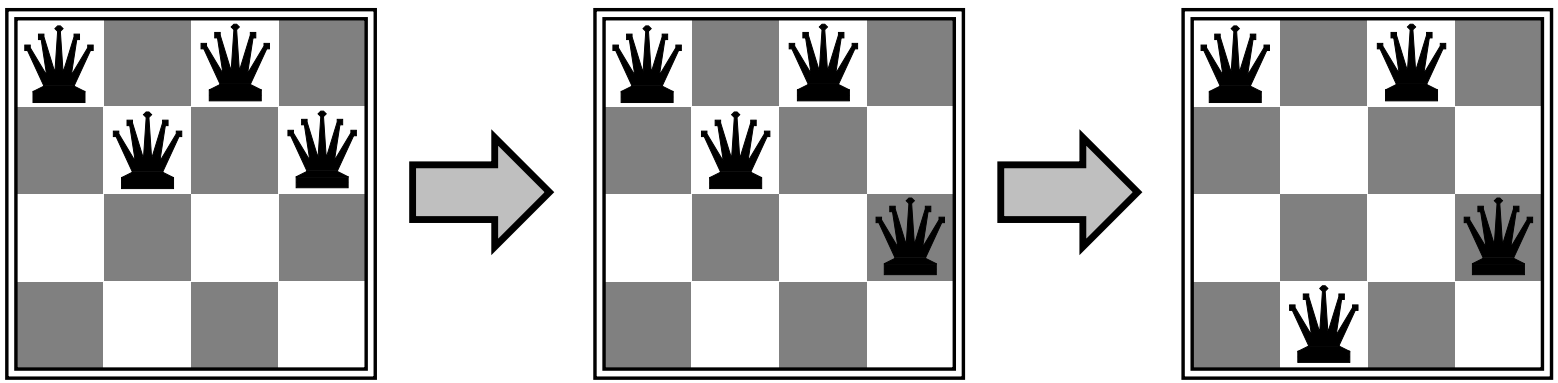
- Algoritma search yang telah kita pelajari: sistematis
  - Teoretis: seluruh *search space* ditelusuri
  - Kiat-kiat melakukan *pruning* $\rightarrow$ *heuristics*
- Ada banyak masalah di mana solusinya adalah konfigurasi goal state. Path-nya tidak penting:
  - Contoh "mainan": *n*-queens, mewarnai wilayah, ...
  - Contoh "betulan": penjadwalan kereta, integrated circuit (IC) layout, ...
- Perlu pendekatan yang berbeda: local search.
- Definisi masalah sedemikian sehingga *state space* = himpunan konfigurasi lengkap.
- Solusi:
  - Menemukan goal state yang memenuhi semua syarat: *constraint satisfaction*.
  - Menemukan state terbaik berdasarkan suatu *objective function*: *optimization problem*.

Konsep dasar local search

- Untuk masalah seperti ini, kita bisa melakukan algoritma *random search* berikut:
  1. Pilih secara *random* suatu state (berupa konfigurasi lengkap).
  2. Lakukan modifikasi terhadap *current state*.
  3. Ulangi langkah 2 sampai *goal* ditemukan (atau waktu habis).
- Yang dibutuhkan:
  - Menghasilkan calon solusi secara acak
  - Nilai evaluasi calon tersebut (heuristics atau objective functions)
  - Memodifikasi calon tersebut (pindah ke state lain)

Contoh: n-queens problem ($n = 4$)

- Tujuan: pasang n menteri pada papan $n \times n$ sehingga tidak ada yang saling menyerang.
- Heuristics (cost) function h: banyaknya pasangan menteri yang saling menyerang.
- Mulai dengan n menteri secara acak (per kolom), pindahkan sebuah menteri untuk meminimumkan h.



Outline

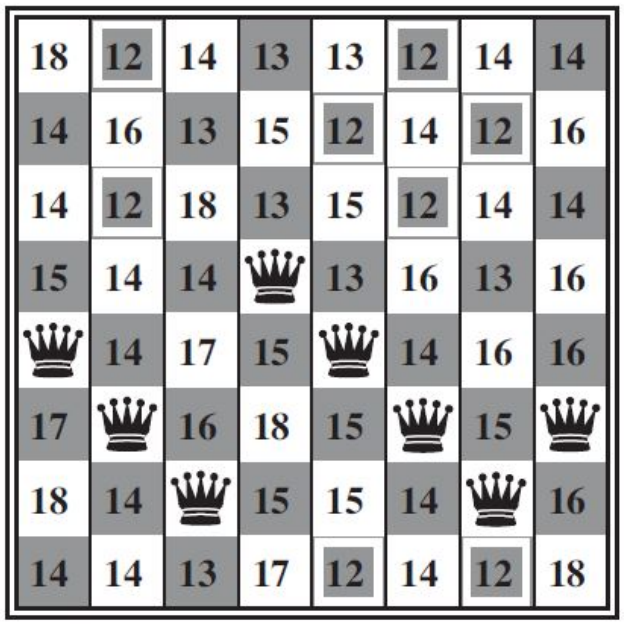
Hill-climbing search

function HILL-CLIMBING (*problem*) **returns** *local maximum state*

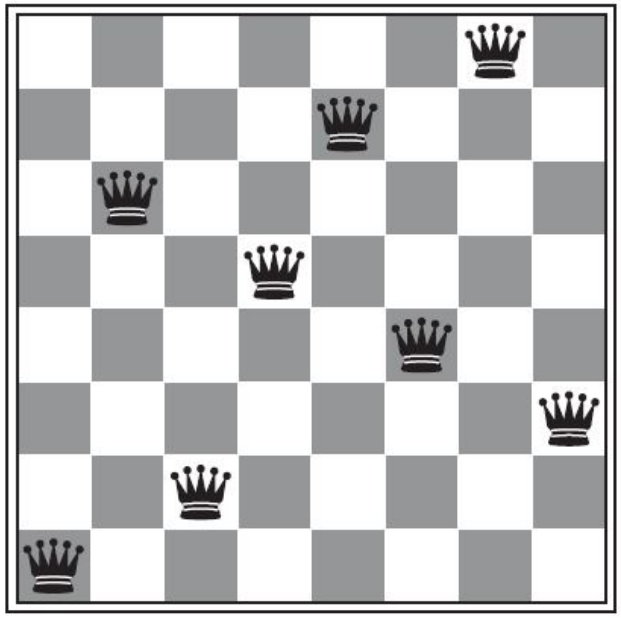
current ← MAKENODE(INITIALSTATE[*problem*])
loop do
  neighbour ← a highest-valued successor of *current*
  if *neighbour*.VALUE ≤ *current*.VALUE **then return** *current*.STATE
  current ← *neighbour*
end

- NILAI sebuah node n: $h(n)$ (heuristic function)
- Bayangkan seorang penderita amnesia mendaki gunung dengan kabut tebal ...
  - State: posisi koordinat (X,Y)
  - $h(n)$: ketinggian pendaki
- Disebut juga *greedy local search*

Contoh: n-queens problem ($n = 8$)



(a)



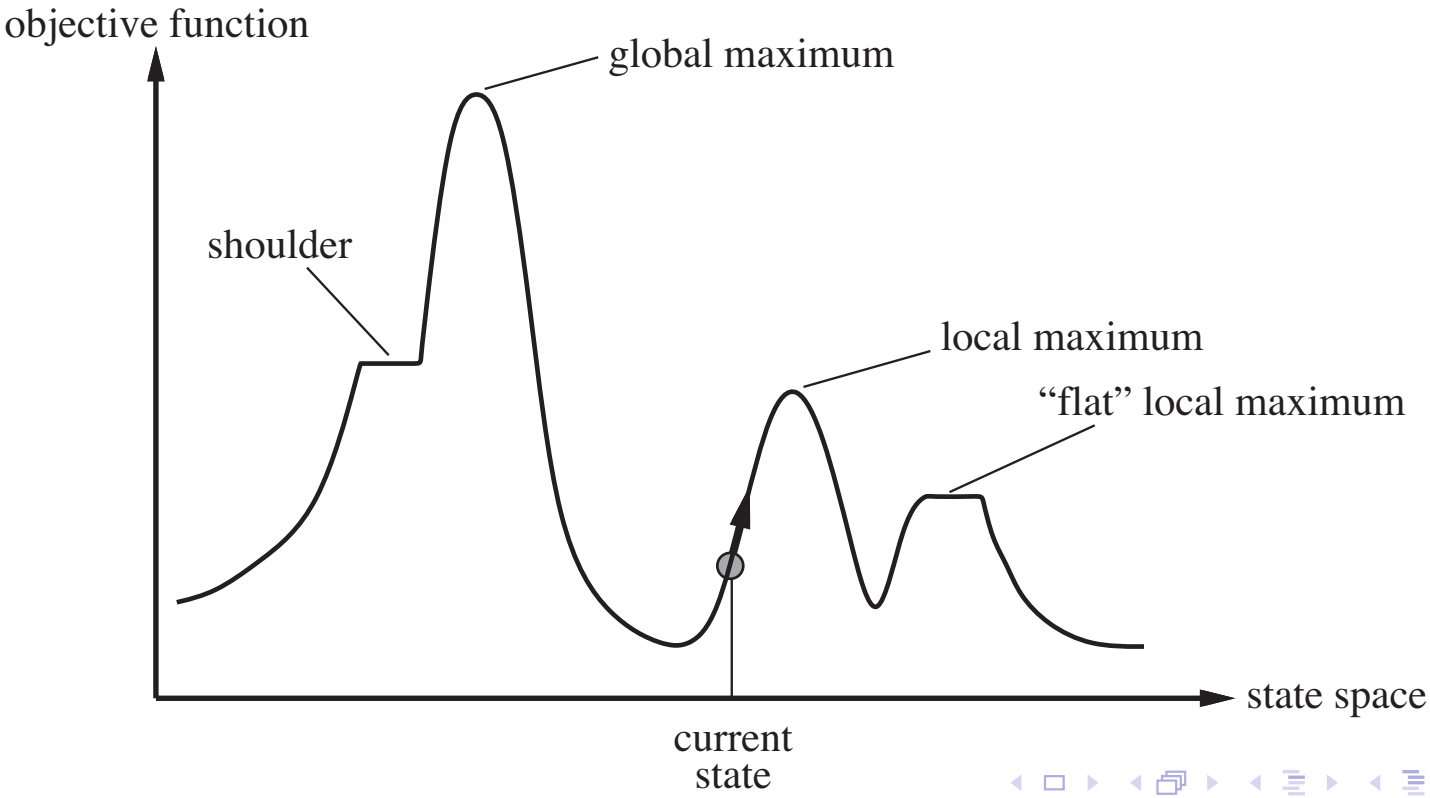
(b)

Figure 4.3 (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.

Outline

State-space landscape

State-space landscape: "location" (state) dan "elevation" (heuristics or objective functions).
Tergantung pilihan *initial state*, hill-climbing bisa terperangkap dalam *local maximum*.



Masalah hill-climbing

- *Local maximum*: tidak ada tetangga yang lebih baik, tetapi bukan solusi optimal.
- *Plateau* (dataran): semua tetangga sama baiknya
- Untuk *8-queens problem*, hill-climbing dari konfigurasi lengkap acak gagal 86%, sukses 14%. Rata-rata solusi 4 langkah, dan 3 langkah kalau gagal (padahal ada $8^8 \approx 17$ juta state!).
- Kalau kita perbolehkan "bergerak di dataran" (dibatasi 100 langkah), peluang untuk sukses naik jadi 94% (rata-rata solusi 21 langkah).

Beberapa pendekatan alternatif

- Stochastic hill-climbing: dari semua kemungkinan tindakan yang arahnya naik, pilih salah satu secara acak.
- Random-restart hill-climbing: kalau gagal, ulangi dengan *initial state* yang baru.
  "If at first you don't succeed, try and try again"
- Local beam search: Dimulai dari k buah *initial state*. Cari semua successors dari k states tersebut. Pilih k buah *successor* terbaik, jika goal belum ditemukan.
  $\rightarrow$ Berbeda dengan k kali *random-restart*: Mengapa? Local beam search tidak independen!
- Simulated annealing: kombinasi hill-climbing dan random walk untuk mendapatkan efisiensi maupun completeness.

Outline

Genetic algorithm (GA)

- Berangkat dari *stochastic beam search*: pilih k successor secara acak dengan probabilitas berbanding lurus dengan "kebaikannya".
- Algoritma genetika = sejenis algoritma optimisasi fungsi secara probabilistik: Simpanlah beberapa calon solusi (*population*). Lakukan "evolusi" secara bertahap dengan menerapkan operator genetika (stochastic).
- Terinspirasi dari proses evolusi biologis → *natural selection* dan *genetic inheritance* (Darwin 1859)
- GA dikembangkan pertama kali oleh John Holland (1975)

Algoritma

1. Initialization: Buat k buah calon solusi (*individual*) secara acak = *population*.

Algoritma

1. Initialization: Buat k buah calon solusi (*individual*) secara acak = *population*.
2. Evaluation: Nilai setiap *individual* berdasarkan *fitness function*.

Algoritma

1. Initialization: Buat k buah calon solusi (*individual*) secara acak = *population*.
2. Evaluation: Nilai setiap *individual* berdasarkan *fitness function*.
3. Selection: Pilih *individual* terbaik untuk "beranak": proportional sampling (roulette), tournament, etc.

Algoritma

1. Initialization: Buat k buah calon solusi (*individual*) secara acak = *population*.
2. Evaluation: Nilai setiap *individual* berdasarkan *fitness function*.
3. Selection: Pilih *individual* terbaik untuk "beranak": proportional sampling (roulette), tournament, etc.
4. Reproduction: Buat "generasi" *population* berikut dengan operasi *crossover* dan *mutation*.
   - *Crossover*: pilih dua (atau lebih) calon solusi, kombinasikan secara acak untuk menghasilkan calon solusi baru.
5. *Mutation*: dengan probabilitas acak (kecil), modifikasi secara acak sebuah calon solusi.

Algoritma

1. Initialization: Buat k buah calon solusi (*individual*) secara acak = *population*.
2. Evaluation: Nilai setiap *individual* berdasarkan *fitness function*.
3. Selection: Pilih *individual* terbaik untuk "beranak": proportional sampling (roulette), tournament, etc.
4. Reproduction: Buat "generasi" *population* berikut dengan operasi *crossover* dan *mutation*.
   - *Crossover*: pilih dua (atau lebih) calon solusi, kombinasikan secara acak untuk menghasilkan calon solusi baru.
5. *Mutation*: dengan probabilitas acak (kecil), modifikasi secara acak sebuah calon solusi.
6. Ulangi langkah 2-5 sampai berhenti.

Kapan berhenti?

- Solusi yang "mencukupi" ditemukan.
- Sudah menjalankan N buah generasi.

Genetic Algorithm

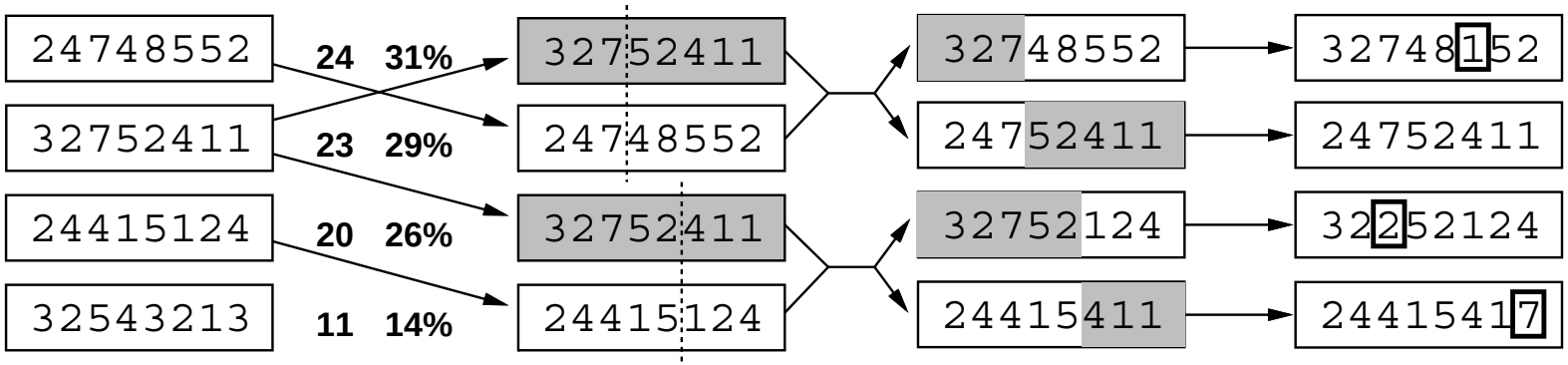
function GENETIC-ALGORITHM (*population*,FITNESS-FN) **returns** *an individual*

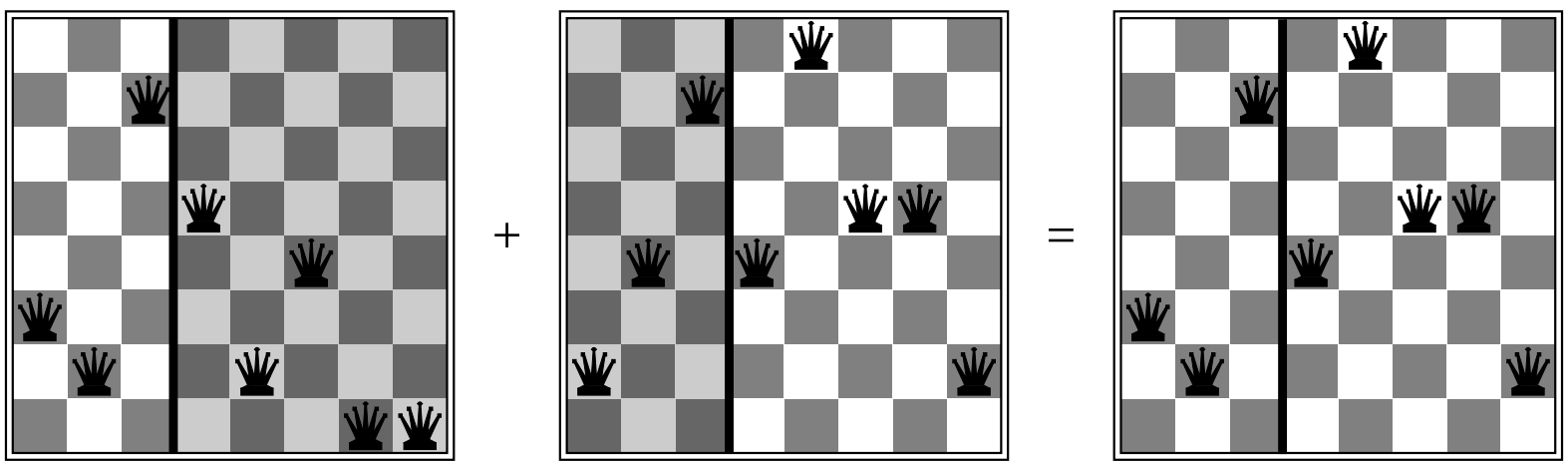
repeat
  new_population ← empty set
  for $i = 1$ **to** SIZE(*population*) **do**
    x ← RANDOM-SELECTION(*population*,FITNESS-FN)
    y ← RANDOM-SELECTION(*population*,FITNESS-FN)
    child ← REPRODUCE(x, y)
    if (small random probability) **then** *child* ← MUTATE(*child*)
    add *child* to *new_population*
  population ← *new_population*
until some individual is fit enough, or enough time has elapsed
return the best individual in *population*, according to FITNESS-FN

function REPRODUCE (x, y) **returns** *an individual*

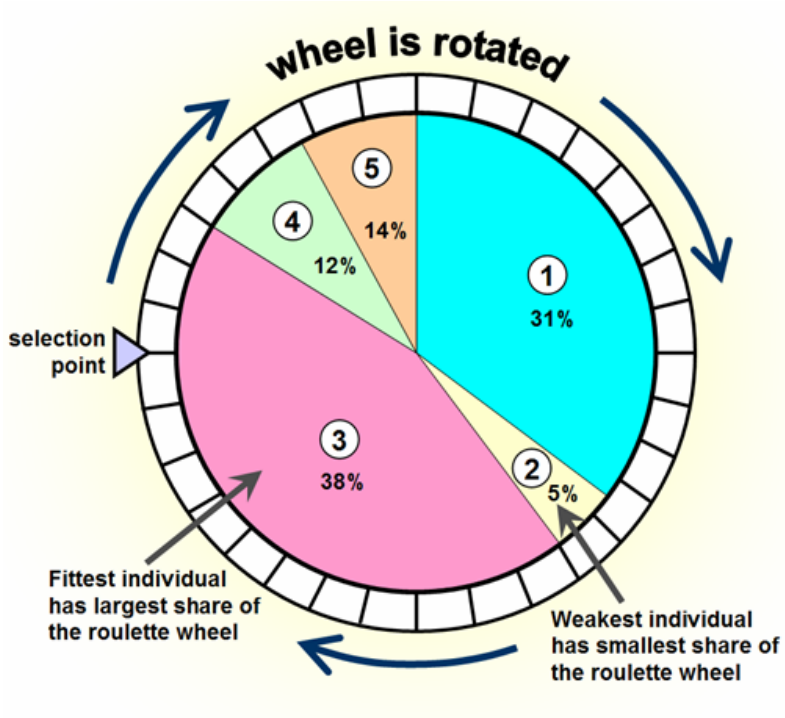
n ← LENGTH(x); c ← random number from 1 to n
return APPEND(SUBSTRING($x, 1, c$),SUBSTRING($y, c + 1, n$))

Contoh: *8-queens problem*





Teknik *Selection*: Roulette-wheel



- Masing-masing individu mendapatkan satu *slice*.
- Individu yang paling *fit* ("sehat") mendapatkan *slice* yang lebih besar daripada yang kurang *fit* $\rightarrow$ Peluang lebih besar untuk dipilih.

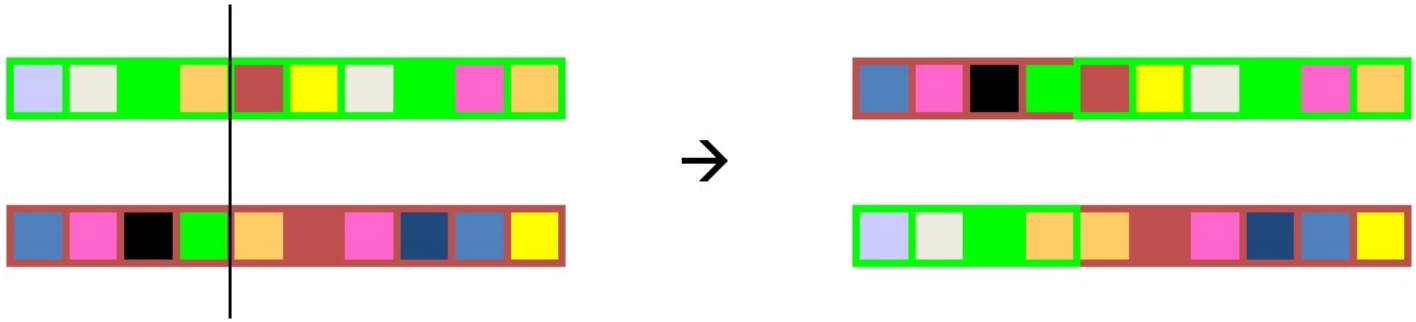
Teknik *Selection*: Tournament

Tournament: Mengadakan k pertandingan pada n individu. Individu pemenang (dengan nilai *fitness* terbaik) dari masing-masing pertandingan dipilih.

Example: Suppose you want to pick 20 individuals from 100. Randomly choose (with uniform probability) a small number of individuals (typically fewer than 10) from the 100 (with replacement). Keep the fittest one. Do this again and again until you have got your 20.

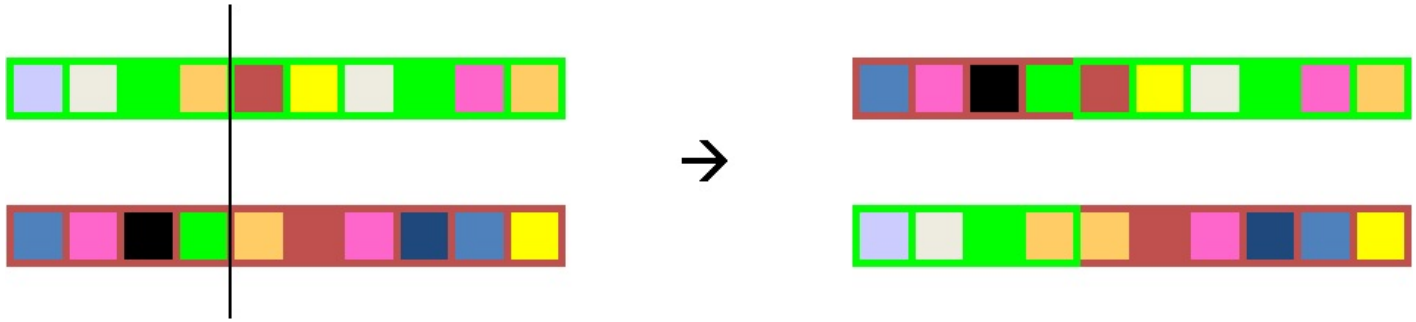
Teknik *Crossover*

- One-point Crossover: Memilih satu posisi pada kromosom calon solusi secara acak.

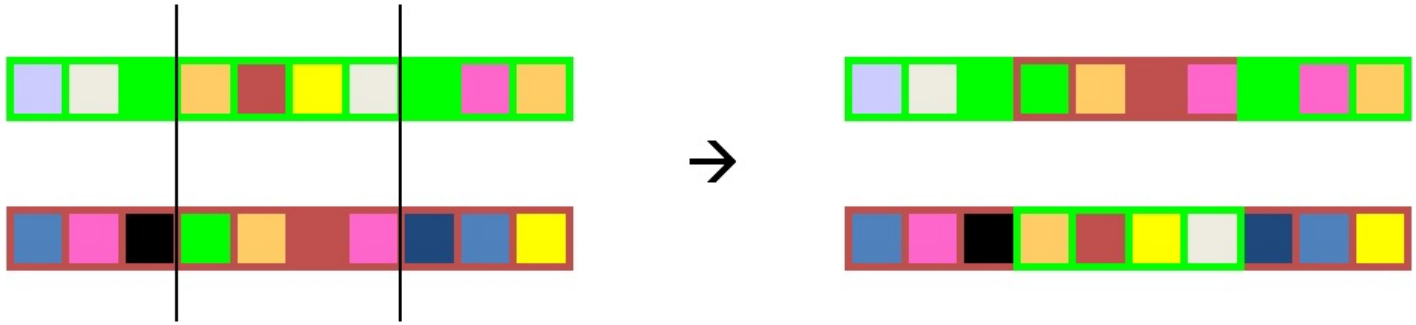


Teknik *Crossover*

- One-point Crossover: Memilih satu posisi pada kromosom calon solusi secara acak.



- Two-point Crossover: Memilih dua posisi pada kromosom calon solusi secara acak.

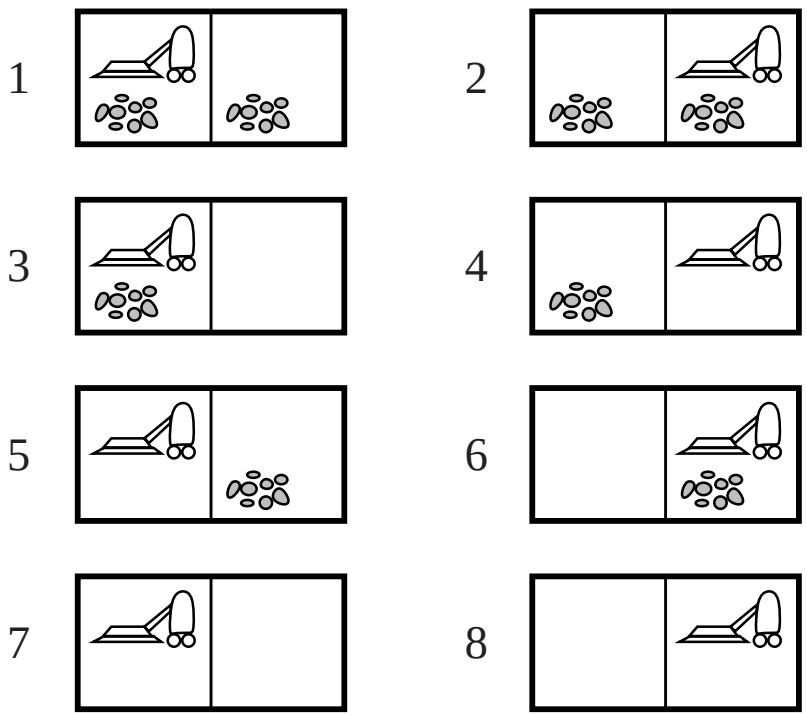


Kunci dari aplikasi GA yang sukses

- Definisi masalah $\rightarrow$ representasi kromosom (bit-string? tree? struktur data kompleks?)
- Fitness function $\rightarrow$ menyatakan semua *domain knowledge*. Masalah baru: *multi-objective optimization*.
- Operator genetika $\rightarrow$ bisa mencapai semua *state*?

Outline

Environment yang tidak deterministik



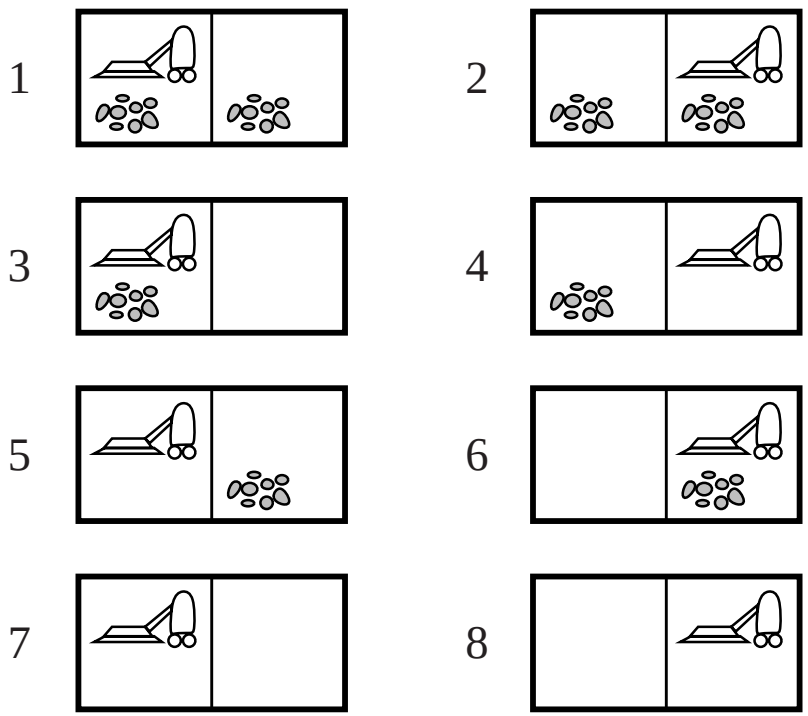
Environment yang tidak deterministik (lanj.)

- Bayangkan *problem solving agent* kita berada di lingkungan yang *fully-observable* dan *non-deterministic*.
- Bayangkan robot pembersih kita cacat, untuk operasi *DoSedot*:
  - Jika dilakukan pada ruang yang kotor, *DoSedot* membersihkan ruangan tersebut dan kadang-kadang membersihkan ruangan di sebelahnya.
  - Jika dilakukan di ruangan bersih, kadang-kadang ia malah membuatnya kotor!
- Results Function mengembalikan himpunan *state* yang mungkin dicapai dengan melakukan (*non-deterministic*) *action*.
- Solusi sekarang bukanlah rangkaian tindakan (*action sequence*), tetapi *action tree*, mis: jika mulai dari state 1, dihasilkan *contingency plan* sbb.

  [*DoSedot*, **if** *State* = 5 **then** [*DoKanan*, *DoSedot*] **else** []].
- Solusi harus menangani semua kemungkinan results.

Environment yang tidak *observable*

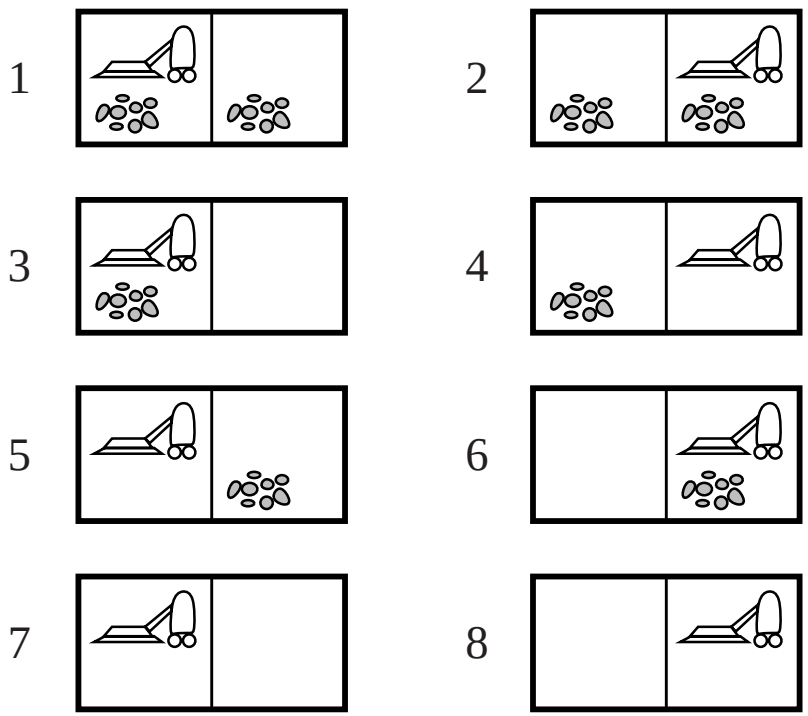
- Selama ini, kita berasumsi bahwa *environment* di mana *problem solving agent* kita berada *fully observable*.
- Apa yang terjadi jika si *agent* tidak memiliki sensor?



- *Initial state* bisa di mana saja: $\{1, 2, 3, 4, 5, 6, 7, 8\}$
- Setelah *DoKanan*, bisa di: $\{2, 4, 6, 8\} \rightarrow$ kemungkinan state mengecil!

Environment yang tidak *observable*

- Selama ini, kita berasumsi bahwa *environment* di mana *problem solving agent* kita berada *fully observable*.
- Apa yang terjadi jika si *agent* tidak memiliki sensor?



- *Initial state* bisa di mana saja: $\{1, 2, 3, 4, 5, 6, 7, 8\}$
- Setelah *DoKanan*, bisa di: $\{2, 4, 6, 8\} \rightarrow$ kemungkinan state mengecil!
- Solusi adalah rangkaian tindakan [*DoKeKanan*, *DoSedot*, *DoKeKiri*, *DoSedot*]

Sensorless problem

- Si agent harus mencatat himpunan *physical state* (S_p) yang mungkin sedang terjadi $\rightarrow$ *belief state* (S_b).
- *Search* dilakukan dalam *space* yang terdiri dari *belief state*, bukan *physical state*.
- *Action* pada S_b dapat ditentukan dengan mengambil union atau intersection dari action-action *physical state* yang membentuk S_b.
- *Belief state* S_b' yang dihasilkan suatu *action* terhadap *belief state* S_b adalah *union* dari semua *physical state* S_p' yang dihasilkan *action* tersebut terhadap semua *physical state* $S_p \in S_b$.
- Sebuah solusi adalah path yang menuju *belief state* di mana semua member *physical state*-nya adalah goal.

Contoh *belief state* VacuumCleanerWorld