

Catatan Kuliah

IF3170 Artificial Intelligence

Semester I 2025/2026

6 Juli 2025

IF '23

Razi Rachman Widyadhana - *@zirachw*

Muhammad Fathur Rizky - *@fathurwithyou*

Apabila terdapat kesalahan pada file, silakan kontak penulis :)

Dilarang membuka dan membawa berkas ini saat ujian berlangsung

This page intentionally left blank

Daftar Isi

0 Preliminary	5
0.1 Dosen Pengampu	6
0.2 Workload	6
0.3 Important Dates	6
0.4 Aturan Perkuliahan	6
0.5 Resources	6
1 Introduction to AI	7
1.1 AI in a Nutshell	8
1.2 4 Approaches in AI Definition	8
1.3 Acting Humanly	9
1.4 Thinking Humanly	9
1.5 Acting Rationally	10
1.6 Thinking Rationally	10
1.7 Latihan	11
2 Agents in AI	12
2.1 What is an Agent?	13
2.2 Agent Model (Studi Kasus)	14
2.3 Rational Agent	15
2.4 PEAS	16
2.5 Task Environment	17
2.6 Agent Structure & Types	18
2.7 Agent Level (Studi Kasus)	21
3 Local Search	23
3.1 Classical Search Review	24
3.2 Local Search	24
3.3 Landscape Ruang State pada Local Search	26
3.4 State	26
3.5 Successor dan Neighbor	27
3.6 Hill Climb	27
3.6.1 Basic Hill Climb	27
3.6.2 Sideways	29
3.6.3 Stochastic	30
3.6.4 Random Restart	30
3.7 Simulated Annealing	30
3.8 Genetic Algorithm	31
3.8.1 Individu	32
3.8.2 Populasi	33
3.8.3 Selection	33
3.8.4 Cross Over & Mutation	34

4 Adversarial Search	35
4.1 Minimax Algorithm	36
4.2 Alpha-Beta Pruning	38
4.2.1 Mekanisme Kerja	38
4.2.2 Keunggulan dan Efisiensi	39
5 Constraint Satisfaction Problem	40
5.1 What is CSP?	41
5.2 Visualisasi CSP	41
5.3 Variasi dari Formulasi CSP	42
5.4 Constraint Propagation & Inference	43
5.5 Improving Backtracking Efficiency	44
5.6 Interleaving Search	45
5.7 Local Search for CSP	46

Bab 0

Preliminary

0.1	Dosen Pengampu	6
0.2	Workload	6
0.3	Important Dates	6
0.4	Aturan Perkuliahan	6
0.5	Resources	6

Mata kuliah “Inteligensi Buatan” merupakan salah satu mata kuliah wajib prodi di Program Studi Teknik Informatika (S1) STEI ITB.

Mata kuliah ini berada dalam lingkup lab AI (Artificial Intelligence) dengan KK Informatika.

Memiliki beban 4 SKS yang terbagi menjadi 4 jam tatap muka di kelas, 4 jam mengerjakan berbagai tugas, dan 4 jam belajar mandiri. Selain itu, terdapat prerequisite LogKom, Stima, TBFO, dan ProbStat untuk Mata Kuliah ini.

0.1 Dosen Pengampu

- K1: Dr. Nur Ulfa Maulidevi, S.T, M.Sc.
- K2: Dr. Fariska Zakhralatifa Ruskanda, S.T., M.T.
- K3: Dr. Eng. Ayu Purwarianti, S.T, M.T.

0.2 Workload

- Latihan Edunex
- 3 Praktikum
- 2 Tugas kecil, 2 Tugas Besar
- Belajar Mandiri (2 Kuis, UTS, UAS)

0.3 Important Dates

- Kuis 1 (Minggu ke-4): 22 September 2025
- UTS (Minggu ke-8)
- Prak 1 (Minggu ke-9): 29 Oktober 2025
- Kuis 2 (Minggu ke-12): 10 November 2025
- Prak 1 (Minggu ke-9): 24 November 2025
- UAS (Minggu ke-16)

0.4 Aturan Perkuliahan

- Syarat lulus IF3170: nilai setiap komponen > 0
- Segala bentuk kecurangan akademik akan mengakibatkan nilai E, bekerja sama antar kelompok berbeda, mengutip dari sumber lain tapi tidak disertakan acuannya (Google, ChatGPT, dll).
- Susulan Kuis, UTS, UAS hanya untuk yang dirawat di rumah sakit/terkena musibah besar.

0.5 Resources

Stuart J Russell & Peter Norvig, Artificial Intelligence: A Modern Approach, 4th Edition, Prentice-Hall International, Inc, 2022, Textbook

Bab 1

Introduction to AI

1.1	AI in a Nutshell	8
1.2	4 Approaches in AI Definition	8
1.3	Acting Humanly	9
1.4	Thinking Humanly	9
1.5	Acting Rationally	10
1.6	Thinking Rationally	10
1.7	Latihan	11

1.1 AI in a Nutshell

Is this an AI?



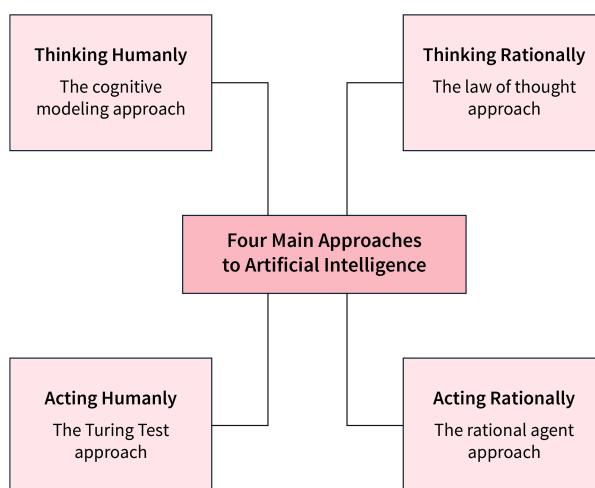
Gambar 1.1: Calculator, obviously

Try to answer these (yes/no) - Quiz IF3170 2024/2025:

1. Lampu yang menyala otomatis ketika ada orang di ruangan yang keberadaannya ditangkap oleh sensor pada lampu tsb
2. Aplikasi penjadwalan kuliah yang menggunakan algoritma pencarian
3. Robot pengantar makanan ke ruangan pasien yang dikendalikan dengan remote control
4. Aplikasi akuntansi yang menggunakan beragam fungsi matematika deterministik dan pelaporan otomatis
5. Aplikasi yang memilah barang catat berdasar gambar yang ditangkap kamera

1.2 4 Approaches in AI Definition

By Russel and Norvig



Gambar 1.2: 4 Approach dalam definisi AI

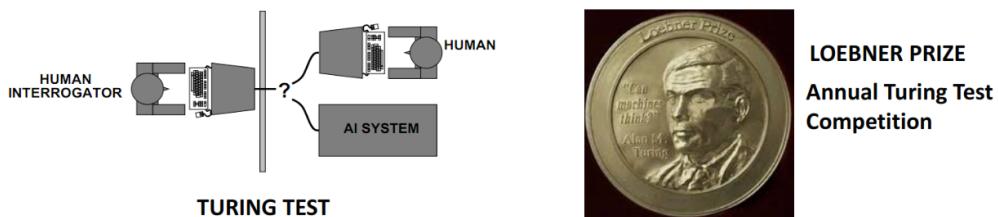
1.3 Acting Humanly

Konsep: The Turing Test Approach

"The art of creating machines that perform functions that require intelligence when performed by people (Kurzweil, 1990)

The study of how to make computers do things at which, at the moment, people are better (Rich and Knight, 1991)

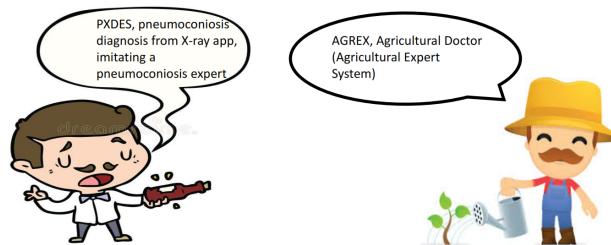
Artinya apa bang messi? Intinya, berfokus pada kemampuan *machines* untuk *mimic* seperti *human* pada umumnya. Bahkan, bisa aja ga bisa bedain ini yang *peragain* apakah *machine* atau *human*.



Gambar 1.3: Turing Test

Turing Test melibatkan seorang juri manusia yang berinteraksi dalam percakapan bahasa alami dengan sebuah mesin dan seorang manusia, **tanpa mengetahui mana di antara keduanya**. Jika mesin berhasil meyakinkan sang juri bahwa ia adalah manusia, maka mesin tersebut dianggap telah lulus Turing Test. Dengan kata lain, merupakan AI dengan pendekatan *Acting Humanly*

Jadi, contohnya apa? Dari *turing test*, istilah yang sekarang ramai menjadi *chatbot*.



Gambar 1.4: Contoh lain dari AI dengan pendekatan *Acting Humanly* selain *chatbot*

1.4 Thinking Humanly

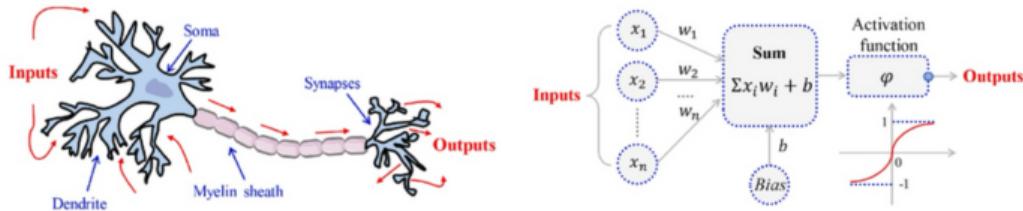
Konsep: The Cognitive Approach

[The automation of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning ... (Bellman, 1978)

The exciting new effort to make computers think ... machines with minds, in the full and literal sense (Haugeland, 1985)

Pendekatan ini berkaitan dengan proses kognitif internal manusia. Dengan kata lain, bagaimana **manusia berpikir** dan bagaimana kita dapat merepresentasikan proses tersebut **dalam komputer**.

Contohnya? Model *artificial neural network* terinspirasi oleh *neural networks* dalam otak manusia. Meskipun model-model ini tidak sepenuhnya mencerminkan cara kerja otak, mereka berusaha meniru bagaimana *neuron* dalam otak manusia dapat memproses informasi.



Gambar 1.5: (a) Biological Neuron, (b) Artificial Neuron

1.5 Acting Rationally

Konsep

Computational intelligence is the study of the design of intelligent agents(Poole et al., 1998)

AI...is concerned with intelligent behavior in artifacts (Nilsson, 1998)

Berfokus pada hasil akhir dari tindakan AI, yaitu bertindak dengan **cara yang paling optimal atau efisien** untuk mencapai suatu tujuan.

Salah satunya adalah *Pathfinding algorithms* dalam permainan catur. Meskipun mereka mungkin tidak memikirkan permainan dengan cara yang sama seperti manusia, mereka bertujuan untuk membuat langkah terbaik berdasarkan posisi saat ini dan kemungkinan langkah di masa depan.



Gambar 1.6: Contoh AI dengan pendekatan *Acting Rationally*

1.6 Thinking Rationally

Konsep

The study of mental faculties through the use of computational models (Charniak and McDermott, 1985)

The study of the computations that make it possible to perceive, reason, and act (Winston, 1992)

Pendekatan ini berkaitan dengan berpikir **secara logis** dan membuat keputusan berdasarkan penalaran. Dengan kata lain, **menggunakan logika** dalam pengambilan keputusan.

Gimana sih bentuknya? seperti ***Rule-based systems*** yang menggunakan ***formal logic*** untuk membuat keputusan. Misalnya, expert systems dalam bidang medis yang menganalisis gejala dan data pasien untuk memberikan diagnosis atau rekomendasi perawatan.

Ataupun *theorem solver*, Kenapa?



Gambar 1.7: Contoh AI dengan pendekatan *Acting Rationally*

1.7 Latihan

Berdasarkan empat pendekatan AI, tentukan pendekatan yang digunakan pada aplikasi/teknologi berikut, ataukah aplikasi tersebut tidak menggunakan pendekatan AI.

1. NuPIC, platform perangkat lunak yang berdasarkan pada model struktur dan operasi pada *neocortex* (bagian pada otak manusia)
2. PXDES, aplikasi yang melakukan diagnosis X-ray layaknya seorang pakar melakukan diagnosis, untuk penentuan *pneumoconiosis* (penyakit paru-paru yang disebabkan oleh penghisapan debu)
3. Pc-Nqthm, aplikasi 'proof-checker' yang berlandaskan pada teori automated reasoning, berkaitan dengan aturan formal logika.
4. AceMoney, aplikasi yang membantu mengorganisasikan dan mengatur keuangan individu (mencatat pemasukan dan pengeluaran)
5. Vampire, *automatic theorem prover* untuk *first order logic* yang menjuarai 11 kali *world cup in theorem prover* sejak 1999
6. Robot melakukan eksplorasi pada suatu lingkungan yang belum pernah dikenali sebelumnya. Robot tersebut bekerja untuk sampai pada lokasi tertentu yang diinginkan oleh pemiliknya dari posisi awal mereka diletakkan
7. Logic Problem Solver, aplikasi yang dapat membantu menyelesaikan persoalan logika yang ada di buku atau majalah logic puzzle
8. AGREX, aplikasi sistem pakar yang membantu petani/pembisnis agrikultur dengan memberikan saran yang benar pada saat yang tepat mengenai penjadwalan irigasi, diagnosis penyakit padi, pemupukan, dan perlindungan tanaman
9. Vitamin D, perangkat lunak yang digunakan untuk mendeteksi manusia atau objek bergerak pada video streams. Aplikasi ini memanfaatkan teknologi yang memodelkan *neocortex* (bagian dari otak manusia yang bertanggung jawab untuk high level perception)

Bab 2

Agents in AI

2.1	What is an Agent?	13
2.2	Agent Model (Studi Kasus)	14
2.3	Rational Agent	15
2.4	PEAS	16
2.5	Task Environment	17
2.6	Agent Structure & Types	18
2.7	Agent Level (Studi Kasus)	21

2.1 What is an Agent?

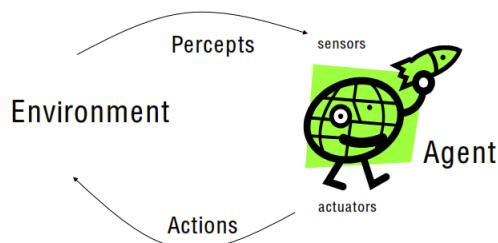
Definisi 2.1

Segala sesuatu yang dapat dipandang sebagai **persepsi** terhadap **environment** melalui **sensors** dan **bertindak** terhadap **environment** tersebut melalui **actuators**.

Dapat berupa:

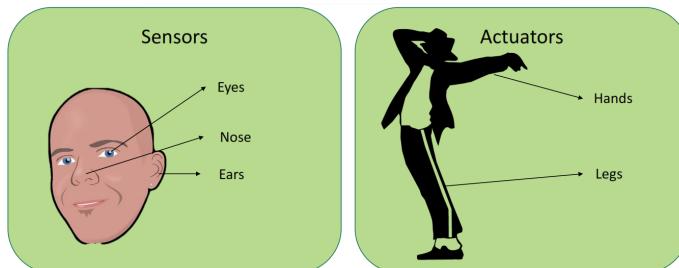
- Robot
- Pabrik
- Program Web Shopping

Computational agents yang berperilaku secara otonom.



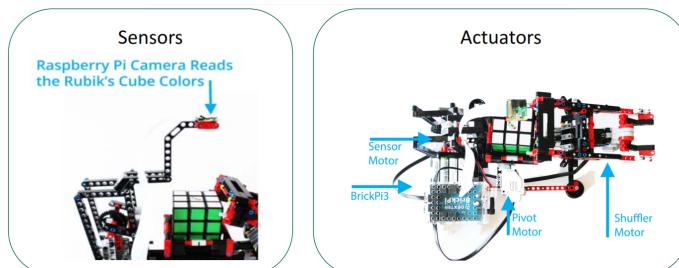
Gambar 2.1: Agent dan Envorinment

Sebagai contoh pada manusia:



Gambar 2.2: Human Agent

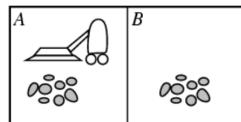
Adapun contoh pada robot:



Gambar 2.3: Rubik Solver Robot Agent

2.2 Agent Model (Studi Kasus)

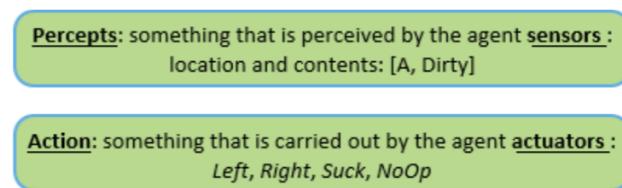
Berikut persoalan yang sederhana, yaitu Agent Penghisap Debu. Terdapat 2 buah ruangan, A dan B. Agent pada awalnya berada pada ruangan A.



Gambar 2.4: Vacuum-Cleaner World

Dari persoalan, maka dapat diperoleh:

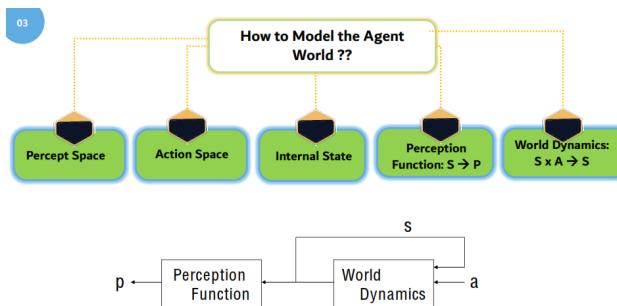
1. Agent dapat menangkap dua hal dari lingkungannya, yaitu lokasi dan keadaan lingkungannya.
2. Aksi yang dapat dilakukan oleh agen adalah bergerak, menghisap, dan *do nothing*



Gambar 2.5: Persepsi dan Aksi Agent

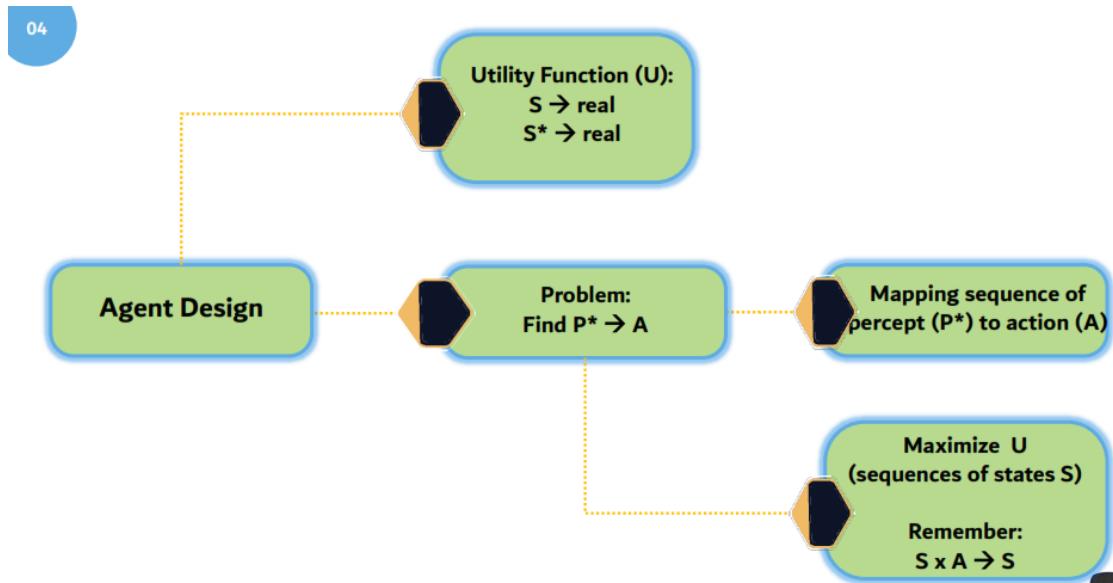
Dari analisis di atas:

1. Agen memiliki kumpulan persepsi atau kondisi atau state yang bisa ditangkap oleh agen yang dirancang ada pada agen (Percept Space)
2. Aksi apa saja yang bisa dilakukan oleh agen untuk mencapai tujuannya juga harus dirancang dari awal (Action Space)
3. Kondisi Internal atau state dalam program atau proses berpikir agen untuk menentukan apakah goal telah tercapai atau belum (Internal State)
4. Representasi state mungkin berbeda dari kondisi yang ditangkap oleh agen, maka diperlukan fungsi yang memetakan state ke dalam suatu kondisi yang bisa ditangkap oleh agen (Perception Function: $S \rightarrow P$)
5. Dinamika dunia agen, jika pada suatu state lingkungan dilakukan aksi tertentu, maka state lingkungan akan berubah atau tetap (World Dynamics: $S \times A \rightarrow S$)



Gambar 2.6: Pemodelan Dunia Agent

Setelah itu, perlu suatu angka kualitas yang menunjukkan seberapa baik agen yang dirancang dengan menggunakan nilai utilitas.



Gambar 2.7: Evaluasi Agent

2.3 Rational Agent

Agen perlu untuk bekerja dengan baik dengan selalu berpikir untuk memilih aksi terbaik dalam mencapai tujuan yang diinginkan. Ketika agen memilih aksi terbaik, perlu didefinisikan *performance measure*.

Definisi 2.2: Rational Agent

Untuk tiap *sequence percepts* yang mungkin dan modal/bekal pengetahuan yang dimiliki oleh agen, **Rational Agent** harus memilih aksi yang tepat yang dapat memaksimalkan *performance measure*

Notes

Rational tidak berarti maha tahu (omscience).

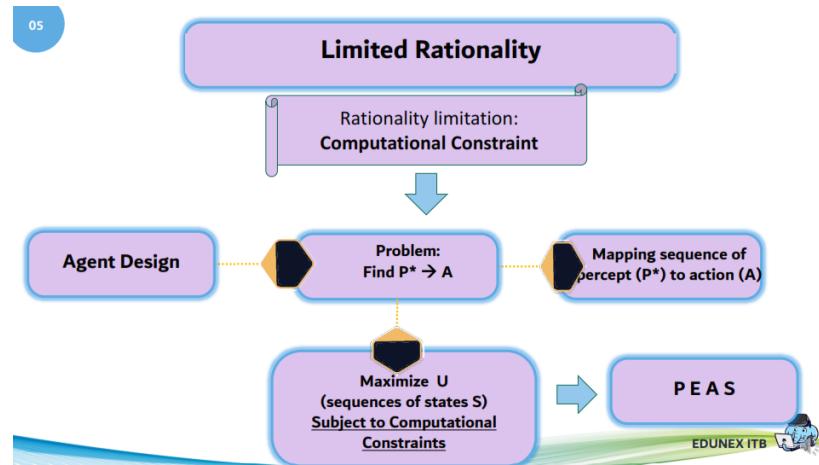
Jika seseorang tidak mendengarkan ramalan cuaca yang menyatakan hari ini akan curah dengan tidak ada hujan. Namun, seseorang tersebut masih membawa payung. Aksi tersebut masih dikatakan rational karena seseorang tidak tahu mengenai ramalan cuaca.

Rational juga tidak sama dengan sukses.

Ketika seseorang tersebut membawa payung agar tidak basah, tetapi seseorang tersebut bertemu dengan anjing di tengah jalan dan menggunakan payung tersebut untuk menghalau anjing.

Seseorang tersebut sukses menggunakan payung untuk menghindari digigit anjing, tetapi tidak rasional karena payung tersebut untuk menghindari hujan, bukan untuk menghalau anjing.

Akibat batasan komputasi, maka definisi desain agen akan berubah pada pemilihan aksi untuk mendapatkan *sequence of states* yang memiliki nilai utilitas maksimal.



Gambar 2.8: Definisi dengan PEAS

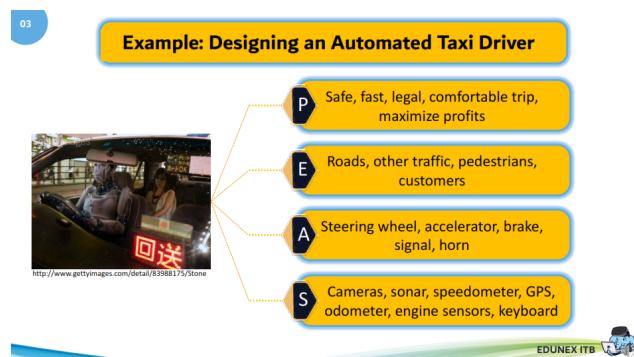
2.4 PEAS

Konsep

PEAS adalah kerangka kerja yang digunakan untuk mendefinisikan lingkungan tugas dalam AI, terdiri dari empat komponen:

1. Performance Measure - Kriteria untuk mengevaluasi keberhasilan agent, seperti akurasi.
2. Environment - Kondisi atau konteks di mana agent beroperasi, seperti dunia nyata atau simulasi.
3. Actuators - Mekanisme yang digunakan agent untuk bertindak, seperti motor atau output teks.
4. Sensors - Alat yang digunakan agent untuk memahami lingkungan, seperti kamera.

PEAS membantu merancang agent dengan memahami kebutuhan dan batasan lingkungan tugas secara terstruktur.

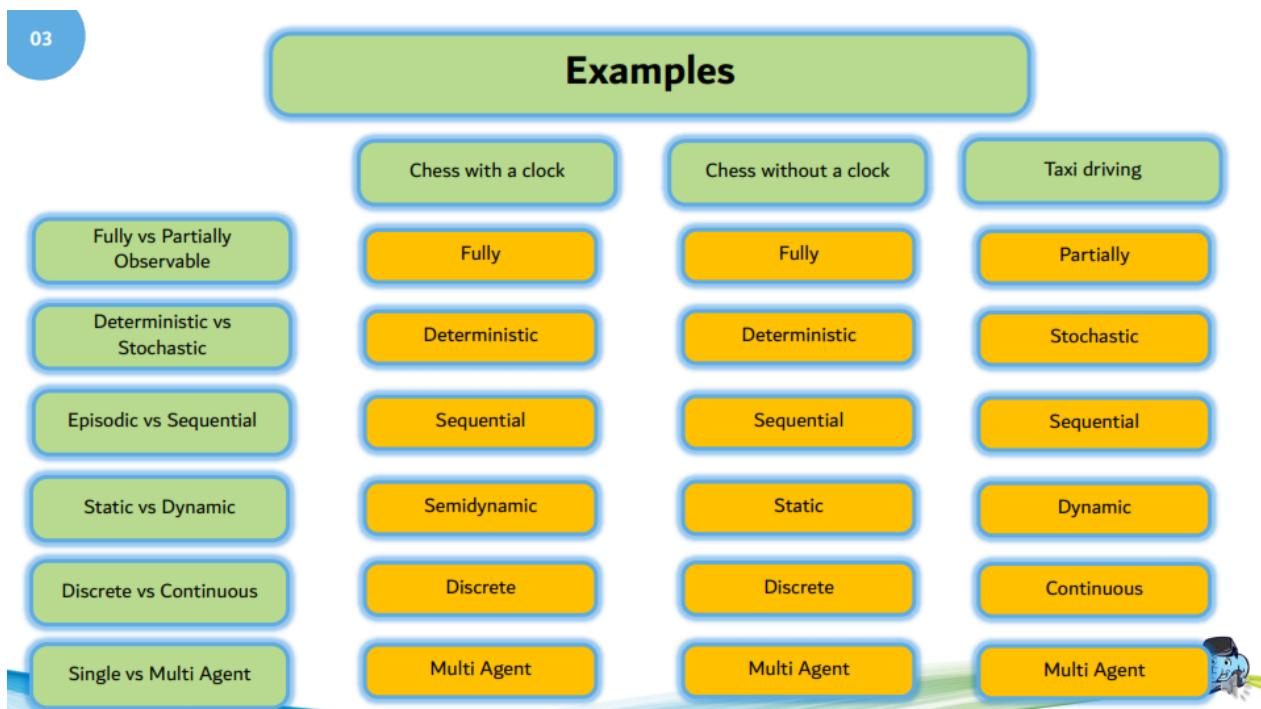


Gambar 2.9: Contoh Abstraksi PEAS

2.5 Task Environment

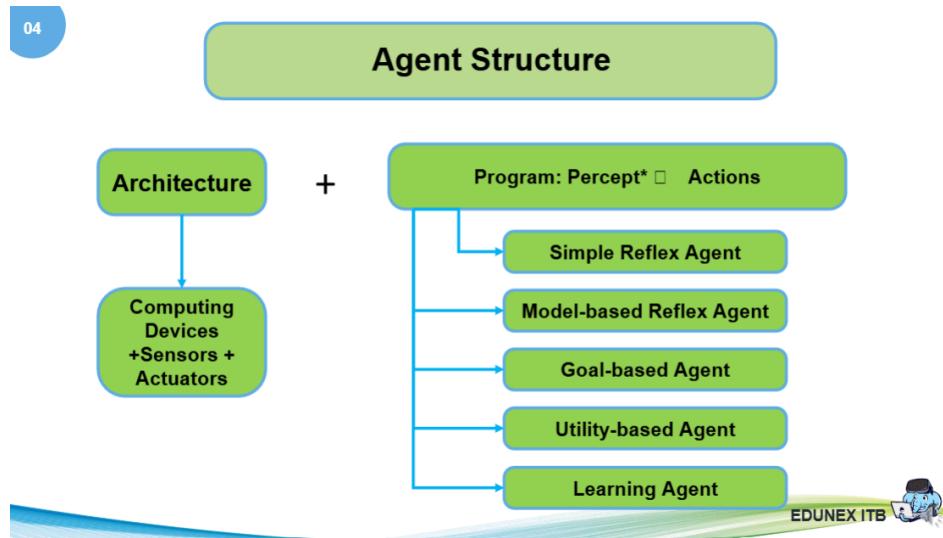
Enam properti lingkungan tugas dalam AI meliputi:

1. Fully Observable vs Partially Observable: Apakah agent memiliki akses penuh ke informasi lingkungan (fully) atau hanya sebagian (partially).
 2. Deterministic vs Stochastic: Apakah hasil tindakan dapat diprediksi (deterministic) atau mengandung elemen acak (stochastic).
 3. Episodic vs Sequential - Apakah tugas terdiri dari episode independen (episodic) atau langkah-langkah yang saling bergantung (sequential).
 4. Static vs Dynamic vs Semidynamic: Apakah lingkungan tetap (static) atau berubah seiring waktu (dynamic) atau lingkungan tetap tetapi performanya dinilai berdasarkan waktu yang digunakan (semidynamic) seperti catur dengan batasan waktu.
 5. Discrete vs Continuous: Apakah state, aksi, dan waktu bersifat diskrit (discrete) atau kontinu (continuous).
 6. Single Agent vs Multi-Agent: Apakah hanya ada satu agent atau beberapa agent yang berinteraksi.
- X. Known vs Unknown: Lebih pada pengetahuan agen terhadap cara kerja lingkungan, keluaran diberikan (known) atau harus dipelajari agen (unknown)



Gambar 2.10: Studi Kasus Task Environment

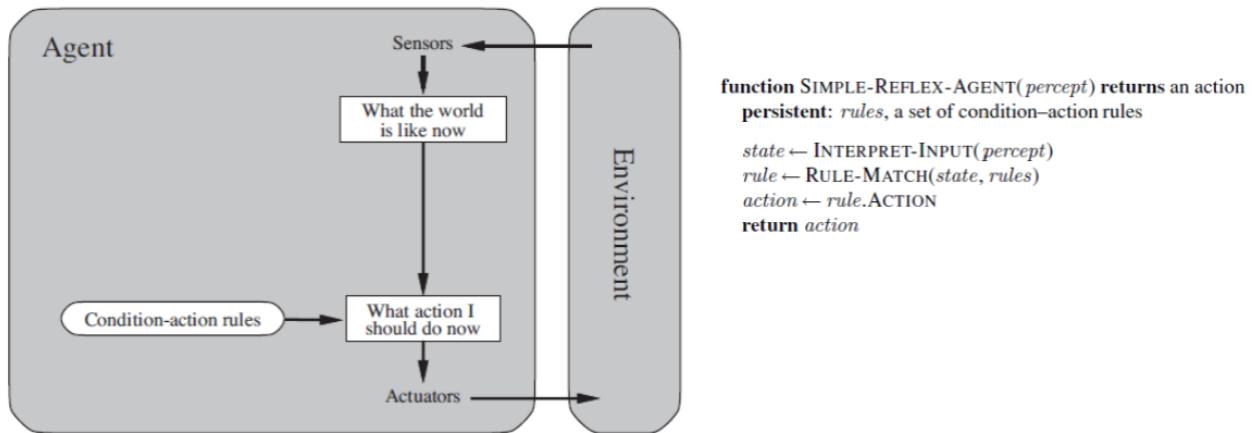
2.6 Agent Structure & Types



Gambar 2.11: Arsitektur dan Tipe-tipe Agent

Konsep: Simple Reflex Agent

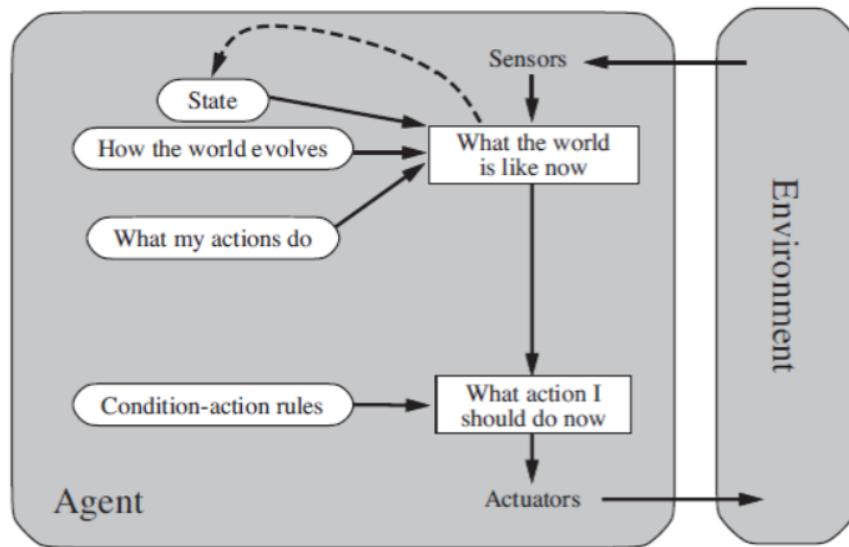
Agent yang bertindak berdasarkan aturan kondisi-aksi sederhana, hanya menggunakan persepsi saat ini (contoh: termostat).



Gambar 2.12: Simple Reflex Agent

Konsep: Model-Based Reflex Agent

Agent yang menggunakan model internal lingkungan untuk melacak state berdasarkan persepsi masa lalu dan saat ini (contoh: vacuum cleaner cerdas).



```

function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
    model, a description of how the next state depends on current state and action
    rules, a set of condition-action rules
    action, the most recent action, initially none

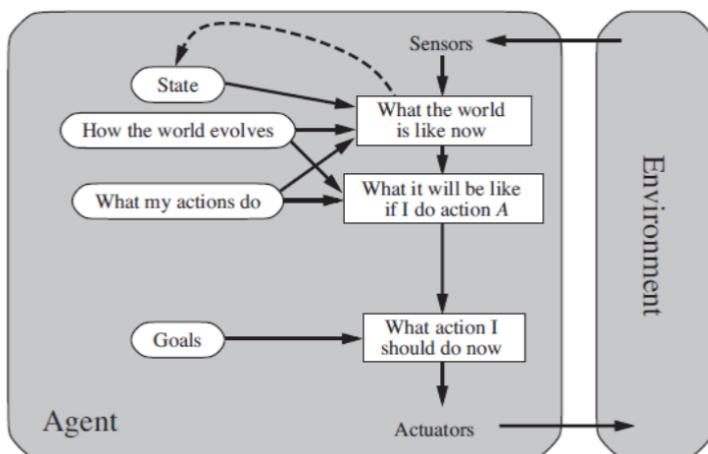
  state  $\leftarrow$  UPDATE-STATE(state, action, percept, model)
  rule  $\leftarrow$  RULE-MATCH(state, rules)
  action  $\leftarrow$  rule.ACTION
  return action

```

Gambar 2.13: Model-Based Reflex Agent

Konsep: Goal-Based Agent

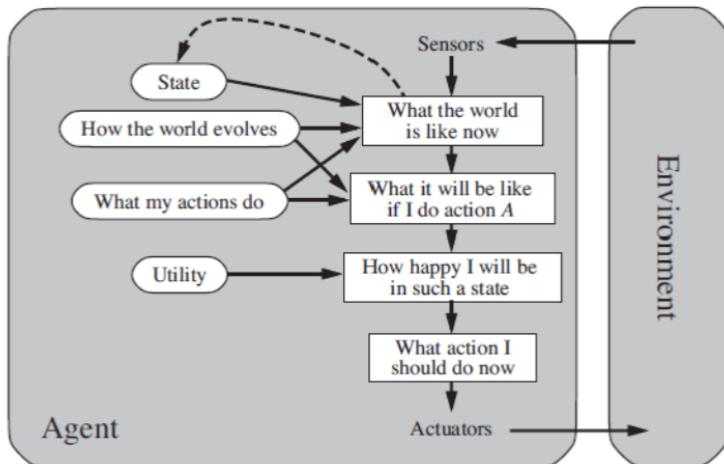
Agent yang bertindak untuk mencapai tujuan tertentu dengan mempertimbangkan konsekuensi aksi (contoh: sistem navigasi GPS).



Gambar 2.14: Goal-Based Agent

Konsep: Utility-Based Agent

Agent yang memaksimalkan utilitas atau kepuasan berdasarkan fungsi utilitas, mempertimbangkan preferensi (contoh: agen e-commerce yang merekomendasikan produk).



Gambar 2.15: Utility-Based Agent

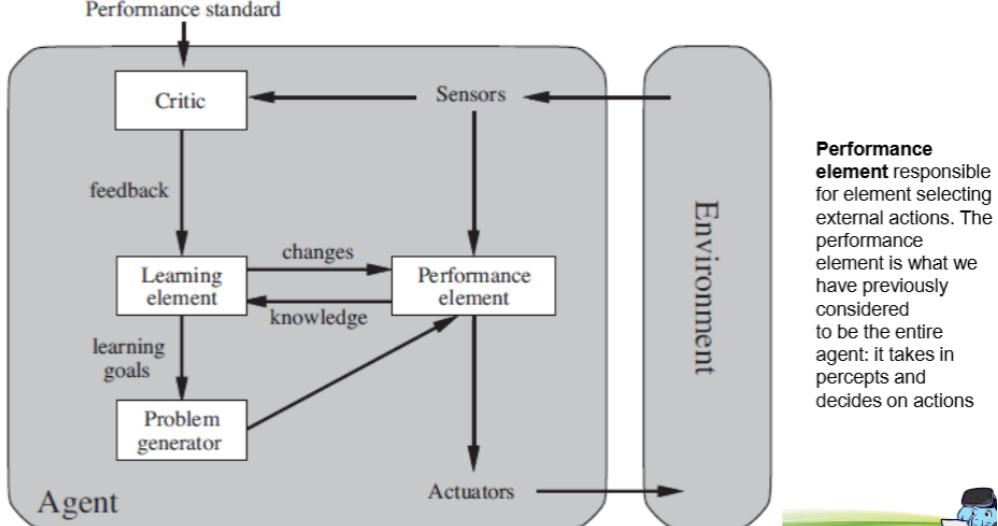
Konsep: Learning Agent

Agent yang dapat belajar dari pengalaman untuk meningkatkan performa seiring waktu, menggunakan komponen seperti kritik dan learning (contoh: agen reinforcement learning untuk permainan).

Critic tells the learning element how well the agent is doing with respect to a fixed performance standard

The **learning element** (responsible for making improvement) uses feedback from the critic on how the agent is doing and determines how the performance element should be modified to do better in the future

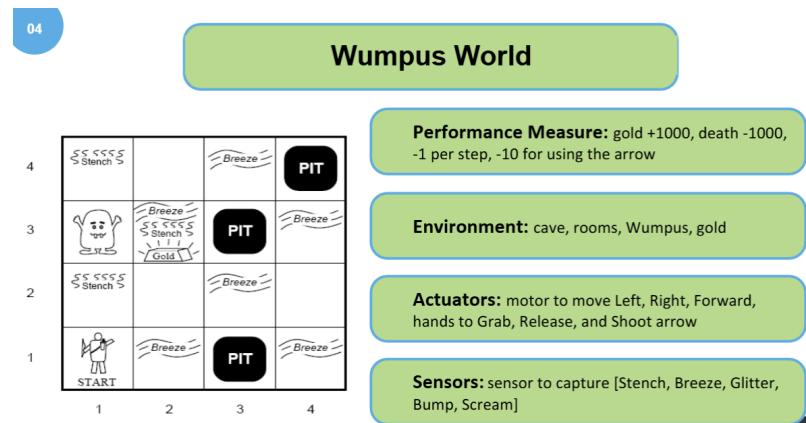
Problem Generator is responsible for suggesting actions that will lead to new and informative experiences



Gambar 2.16: Learning Agent

2.7 Agent Level (Studi Kasus)

Untuk memperkenalkan tingkatan pada agent, terdapat persoalan klasik, yaitu Wumpus World. Terdapat goa yang memiliki banyak ruangan, Wumpus akan memakan apa saja dan siapa saja yang masuk ke ruangannya. Wumpus dapat dipanah oleh agen, tetapi agen hanya memiliki satu anak panah. Pada beberapa ruangan goa, terdapat lubang tidak berdasar yang akan menjebak siapa saja kecuali Wumpus. Tujuan dari agent adalah mendapatkan emas yang ada pada salah satu ruangan goa.



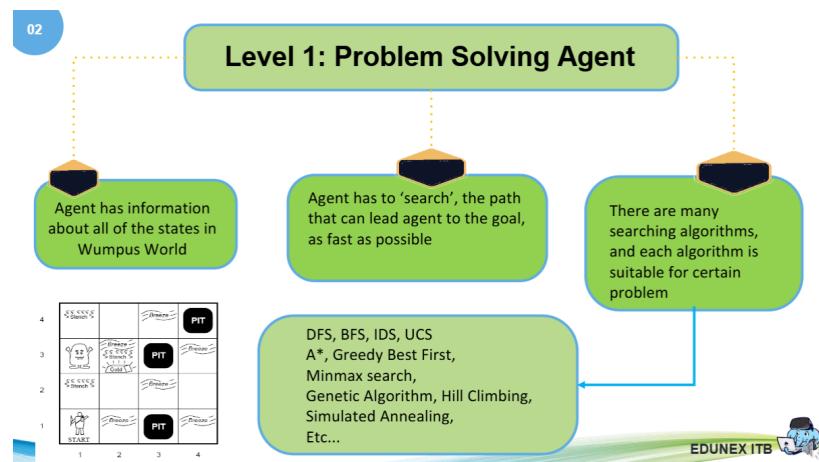
Gambar 2.17: Persoalan Wumpus World dan Abstraksi PEAS-nya

Konsep: Problem Solving Agent (Level 1)

Semua state persoalan yang bisa ditangkap oleh agent dan semua aksi yang bisa dipilih untuk mencapai tujuan telah diberikan.

Tugas agent adalah melakukan pencarian aksi apa yang harus dilakukan agar agent dapat mendapatkan goal dari start.

Agent paling sederhana dalam pemrosesan pencarian solusi dan tidak ada penalaran khusus.

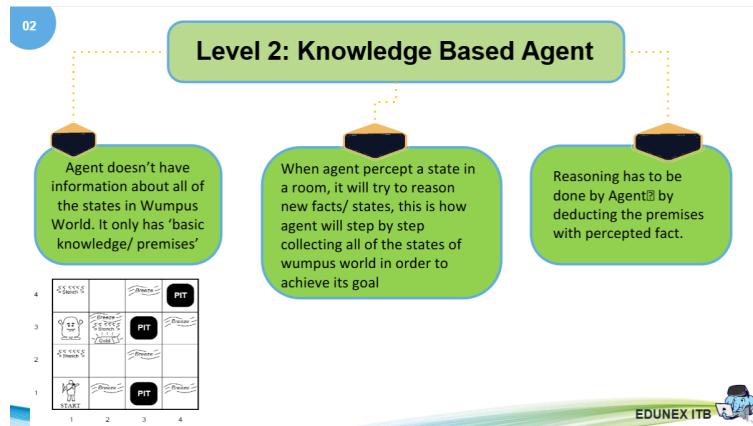


Gambar 2.18: Problem Solving Agent pada persoalan Wumpus World

Konsep: Knowledge Based Agent (Level 2)

Tidak semua state diberikan di awal, namun yang diberikan adalah pengetahuan dasar atau premis yang bisa dimanfaatkan pada proses penalaran untuk mendapatkan fakta atau state yang baru.

Agent telah dilengkapi oleh pengetahuan. Oleh karena itu, agent memiliki kemampuan penalaran untuk memperoleh fakta atau state baru dari dunia persoalan yang dihadapi dalam rangka untuk menyelesaikan persoalan tersebut.

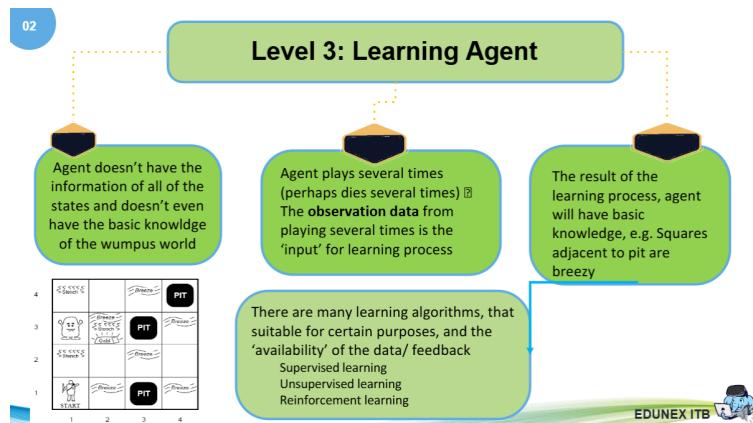


Gambar 2.19: Knowledge Based Agent pada persoalan Wumpus World

Konsep: Learning Agent (Level 3)

Tidak ada informasi yang diberikan kepada agent mengenai states yang ada di persoalan. Modal yang diberikan kepada agent adalah kumpulan data hasil observasi selama beberapa waktu.

Dari kumpulan data tersebut, agent harus dapat membentuk pengetahuan dasar dan menyelesaikan persoalan.



Gambar 2.20: Learning Agent pada persoalan Wumpus World

Bab 3

Local Search

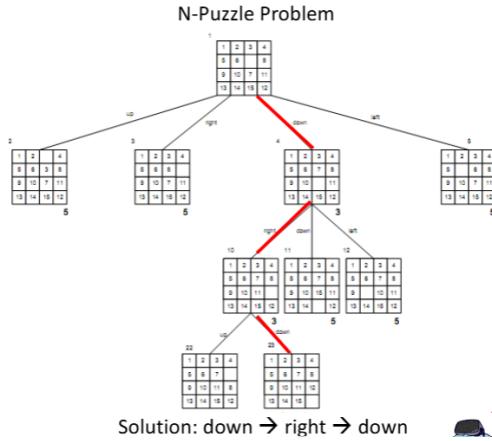
3.1	Classical Search Review	24
3.2	Local Search	24
3.3	Landscape Ruang State pada Local Search	26
3.4	State	26
3.5	Successor dan Neighbor	27
3.6	Hill Climb	27
3.7	Simulated Annealing	30
3.8	Genetic Algorithm	31

Local Search merupakan metode pencarian generik yang berusaha mengoptimisasi pencarian berdasarkan *local changes* saja, bukan space solusi secara keseluruhan. Ditulis oleh Muhammad Fathur Rizky saat liburan.

3.1 Classical Search Review

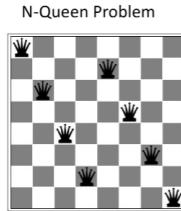
Definisi 3.1

Dirancang untuk lingkungan yang *observable* dan *deterministic* dengan melakukan eksplorasi pencarian secara sistematis sehingga dihasilkan solusinya berupa path atau urutan aksi.



Gambar 3.1: Classical Search pada persoalan N-puzzle

Akan tetapi, pada beberapa persoalan solusi berupa *path to goal* tidak relevan. Sebagai contoh, persoalan N-queen yang solusinya berupa konfigurasi akhir posisi masing-masing *queen*.



Gambar 3.2: Persoalan N-queen

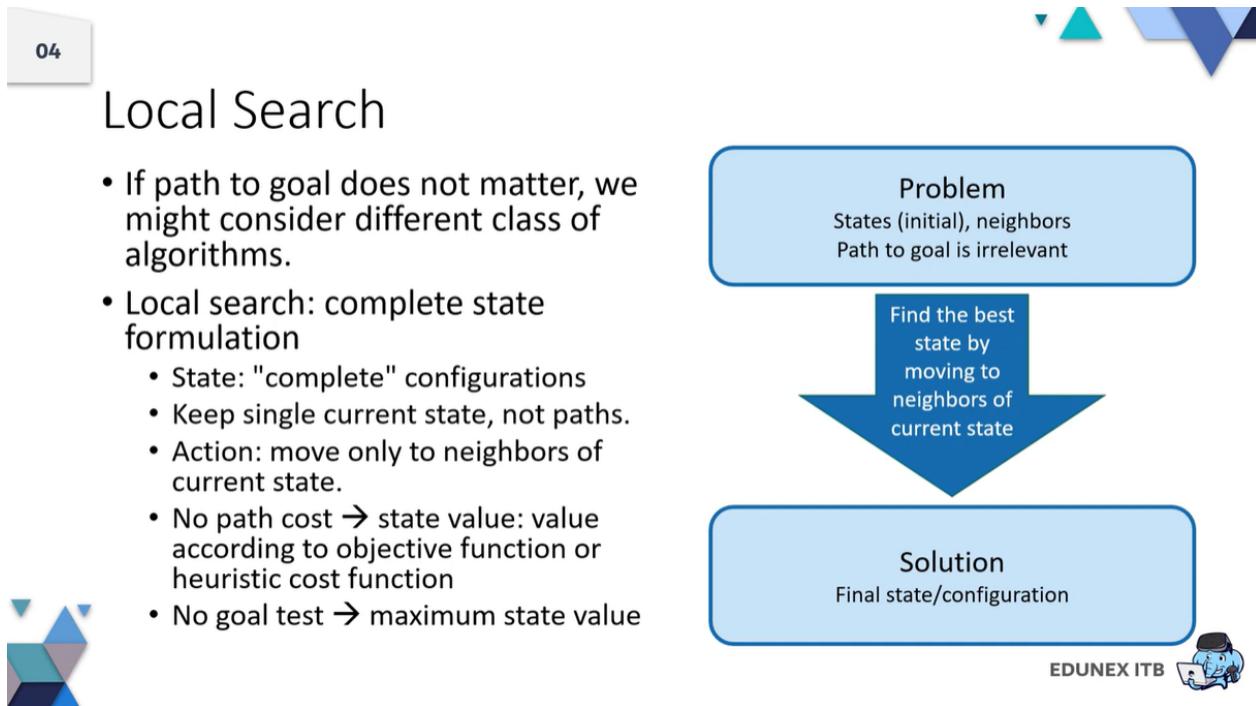
3.2 Local Search

Definisi 3.2

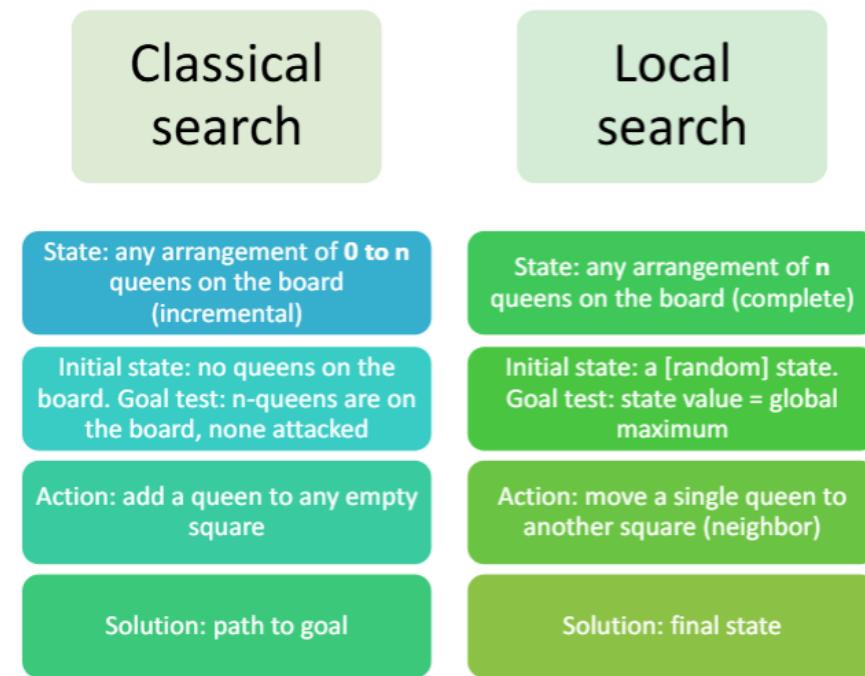
Pencarian Lokal adalah kelas algoritma di mana jalur menuju tujuan tidak penting. Algoritma ini beroperasi pada formulasi keadaan lengkap, artinya setiap keadaan merepresentasikan solusi yang utuh.

Pencarian dimulai dari satu keadaan saat ini dan hanya bergerak ke tetangga dari keadaan tersebut. Solusinya adalah konfigurasi keadaan akhir, bukan urutan tindakan. Nilai keadaan ditentukan oleh fungsi objektif atau fungsi biaya heuristik.

Tujuannya adalah untuk menemukan keadaan terbaik, atau optimum global, dalam lanskap ruang keadaan.

**Gambar 3.3:** Local Search pada N-queen

Terus bagaimana perbedaan antara kedua pencarian tersebut pada persoalan N-queen?

**Gambar 3.4:** Perbedaan kedua pencarian pada persoalan N-queen

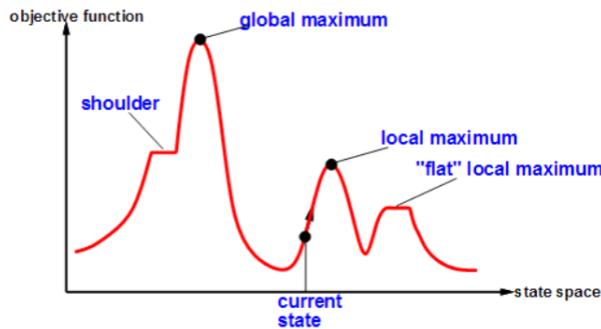
3.3 Landscape Ruang State pada Local Search

Konsep

Sebuah landscape memiliki “location” (didefinisikan oleh state) dan “elevation” (nilai dari **objective** atau **heuristic cost value**).

Local search bertujuan untuk menemukan **global optimum**

Masalah: tergantung pada initial state, dapat **terjebak** dalam **local optimum**.

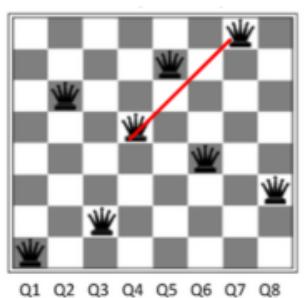


Gambar 3.5: Grafik Local Search

3.4 State

Konsep

Berupa konfigurasi lengkap atau semua variabelnya telah diberi nilai.



Each state has state value based on heuristic cost function.

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	14	13	16	13	16
17	14	17	15	14	16	16	16
18	14	16	18	15	14	15	16
14	14	13	17	12	14	12	18
14	14	13	17	12	14	12	18

Gambar 3.6: Contoh sebuah *state* pada persoalan N-queen

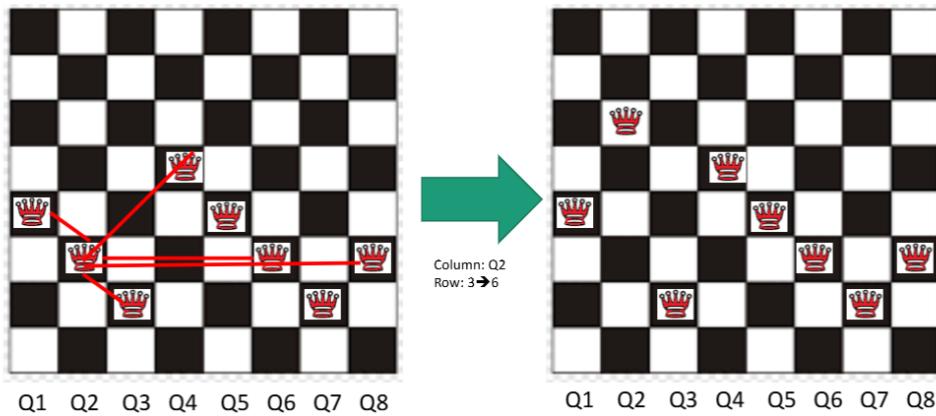
3.5 Successor dan Neighbor

Konsep

Successor function akan menghasilkan seluruh state yang mungkin dari melakukan hanya satu aksi (**Successor**).

Neighbor simpelnya adalah **successor** yang akhirnya dipilih, bisa dengan random ataupun dipilih berdasarkan nilai heuristiknya yang tertinggi.

Neighbor: Random Successor



Gambar 3.7: Neighbor jika dari random successor pada persoalan N-queen

3.6 Hill Climb

Konsep

Hill-Climbing adalah algoritma pencarian lokal yang secara terus-menerus bergerak ke arah peningkatan nilai (untuk fungsi objektif) atau penurunan nilai (untuk biaya). Pencarian dimulai dari keadaan awal yang dibuat secara acak dan berhenti ketika mencapai "puncak" di mana tidak ada tetangga yang memiliki nilai lebih tinggi. Algoritma ini bisa terjebak di maksimum lokal, tergantung pada keadaan awalnya.

3.6.1 Basic Hill Climb

Konsep

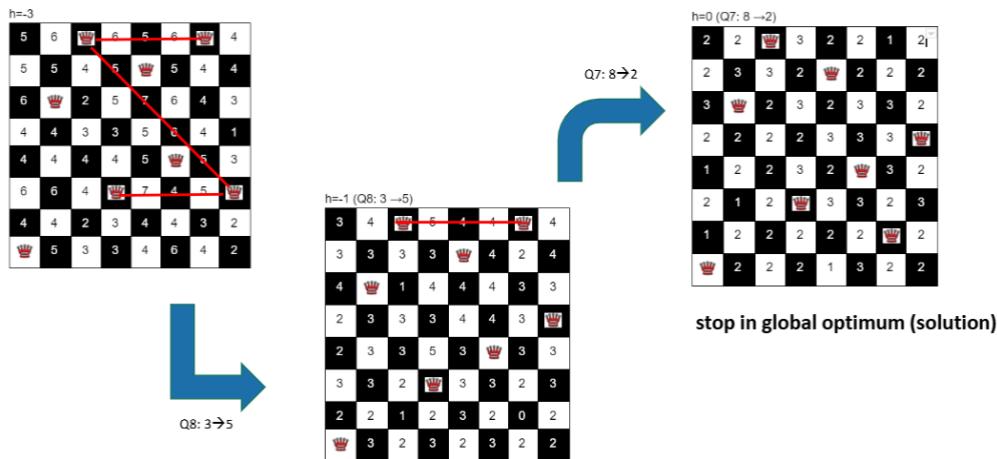
Variasi ini juga dikenal sebagai *steepest-ascent*. Algoritma ini akan memilih tetangga dengan nilai terbaik dari keadaan saat ini. Jika nilai tetangga terbaik tidak lebih baik dari nilai saat ini, algoritma akan berhenti.

Algorithm 1 Basic Hill-Climbing (Steepest-Ascent)

```

1: function BASICHILLCLIMB(keadaan_awal)
2:   keadaan_sekarang  $\leftarrow keadaan_awal
3:   loop
4:     tetangga  $\leftarrow tetangga dengan nilai tertinggi dari keadaan_sekarang
5:     if nilai(tetangga)  $\leq nilai(keadaan_sekarang) then
6:       return keadaan_sekarang
7:     end if
8:     keadaan_sekarang  $\leftarrow tetangga
9:   end loop
10: end function$$$$ 
```

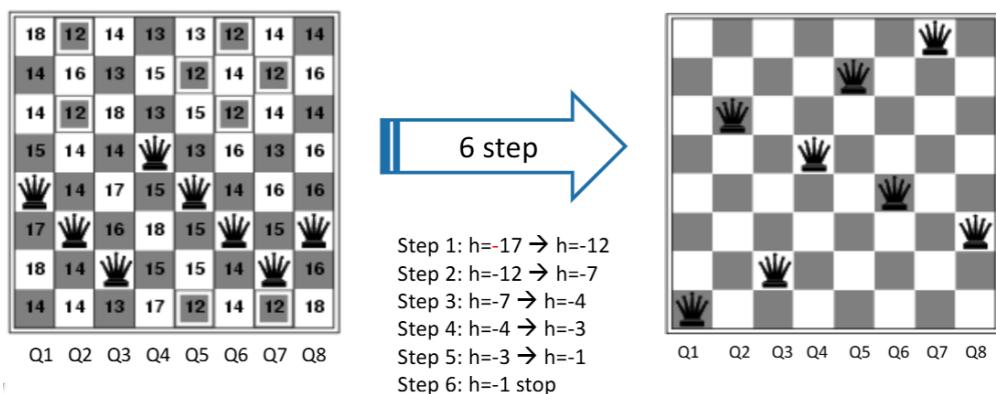
Hill-climbing: Illustration



Gambar 3.8: Ilustrasi *steepest-ascent* Hill Climb Search yang berhasil menuju *global optimum*

Akan tetapi, tidak semua kasus akan berujung dengan *ending* baik.

Hill-climbing: Stuck In Local Optimum



Gambar 3.9: Ilustrasi *steepest-ascent* Hill Climb Search yang nge-stuck

Oleh karena itu, terdapat varian-varian dari Hill Climb Search dengan tujuan menghindari *stuck* pada *local optimum*.

Hill-climbing for 8-Queen Problem

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current  $\leftarrow$  MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor  $\leftarrow$  a highest-valued successor of current
        if neighbor.VALUE  $\leq$  current.VALUE then return current.STATE
        current  $\leftarrow$  neighbor
```

State space:
 $8^8 \approx 16.8$ million states

Average case: works quickly
when **success**: avg 4 steps
when **stuck**: avg 3 steps

Best case:
initial state = goal state
Prob: $92 / 8^8 = 0.00054\%$

Get **stuck** 86%,
solving only 14% of
problem instances

Gambar 3.10: Evaluasi Basic Hill Climb Search

3.6.2 Sideways

Konsep

Varian ini mengizinkan gerakan menyamping, yaitu pindah ke tetangga yang memiliki nilai yang sama dengan keadaan saat ini, dengan batasan jumlah langkah untuk menghindari perulangan.

Algorithm 2 Hill-Climbing with Sideways Moves

```
1: function HILL_CLIMB_SIDWAYS(keadaan_awal, batas_menyamping)
2:   keadaan_sekarang  $\leftarrow$  keadaan_awal
3:   jumlah_menyamping  $\leftarrow$  0
4:   loop
5:     tetangga  $\leftarrow$  tetangga dengan nilai tertinggi dari keadaan_sekarang
6:     if nilai(tetangga) < nilai(keadaan_sekarang) then
7:       keadaan_sekarang  $\leftarrow$  tetangga
8:       jumlah_menyamping  $\leftarrow$  0
9:     else if nilai(tetangga) = nilai(keadaan_sekarang) and jumlah_menyamping <
batas_menyamping then
10:      keadaan_sekarang  $\leftarrow$  tetangga
11:      jumlah_menyamping  $\leftarrow$  jumlah_menyamping + 1
12:    else
13:      return keadaan_sekarang
14:    end if
15:  end loop
16: end function
```

3.6.3 Stochastic

Konsep

Pada **Stochastic Hill-Climbing**, algoritma memilih satu penerus secara acak dan hanya pindah jika nilainya lebih baik.

Algorithm 3 Stochastic Hill-Climbing

```

1: function STOCHASTIC_HILL_CLIMB(keadaan_awal, langkah_maks)
2:   keadaan_sekarang  $\leftarrow$  keadaan_awal
3:   for i  $\leftarrow 1$  to langkah_maks do
4:     tetangga_acak  $\leftarrow$  pilih tetangga acak dari keadaan_sekarang
5:     if nilai(tetangga_acak)  $>$  nilai(keadaan_sekarang) then
6:       keadaan_sekarang  $\leftarrow$  tetangga_acak
7:     end if
8:   end for
9:   return keadaan_sekarang
10: end function
```

3.6.4 Random Restart

Konsep

Strategi ini menjalankan serangkaian pencarian hill-climbing dari keadaan awal acak yang berbeda.

Algorithm 4 Random-Restart Hill-Climbing

```

1: function RANDOM_RESTART_HILL_CLIMB(masalah, hill_climb_func, restart_maks)
2:   for i  $\leftarrow 1$  to restart_maks do
3:     keadaan_awal  $\leftarrow$  masalah.buat_keadaan_acak()
4:     solusi  $\leftarrow$  hill_climb_func(keadaan_awal)
5:     if solusi adalah keadaan tujuan then
6:       return solusi
7:     end if
8:   end for
9:   return "Gagal menemukan solusi"
10: end function
```

3.7 Simulated Annealing

Konsep

Simulated Annealing adalah varian dari stochastic hill-climbing yang mengizinkan beberapa gerakan "menurun" untuk menghindari terjebak di maksimum lokal.

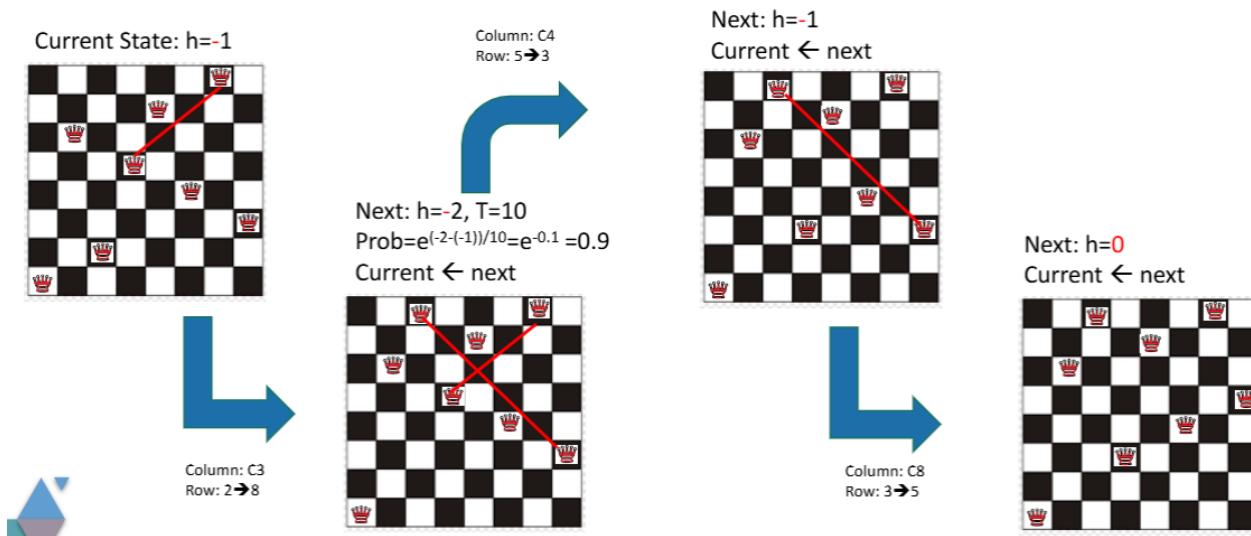
Algorithm 5 Simulated Annealing

```

1: function SIMULATED_ANNEALING(masalah, jadwal)
2:   keadaan_sekarang  $\leftarrow$  masalah.keadaan_awal
3:   for  $t \leftarrow 1$  to  $\infty$  do
4:      $T \leftarrow$  jadwal( $t$ )
5:     if  $T = 0$  then
6:       return keadaan_sekarang
7:     end if
8:     keadaan_selanjutnya  $\leftarrow$  pilih penerus acak dari keadaan_sekarang
9:      $\Delta E \leftarrow$  nilai(keadaan_selanjutnya) - nilai(keadaan_sekarang)
10:    if  $\Delta E > 0$  then
11:      keadaan_sekarang  $\leftarrow$  keadaan_selanjutnya
12:    else
13:      keadaan_sekarang  $\leftarrow$  keadaan_selanjutnya dengan probabilitas  $e^{\Delta E/T}$ 
14:    end if
15:   end for
16: end function

```

Simulated Annealing: Illustration



Gambar 3.11: Ilustrasi Simulated Annealing pada persoalan N-queens

3.8 Genetic Algorithm

Konsep

Algoritma Genetika (GA) adalah metode pencarian yang terinspirasi dari evolusi biologis. Algoritma ini memelihara populasi individu dan menggunakan operator seperti seleksi, pindah silang, dan mutasi untuk menghasilkan generasi baru.

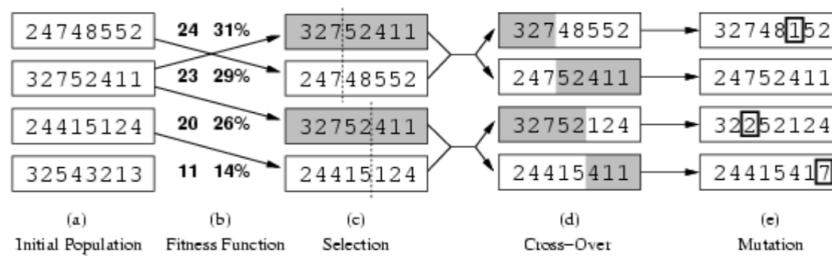
Algorithm 6 Genetic Algorithm

```

1: function GENETIC_ALGORITHM(populasi, fungsi_kebugaran)
2:   repeat
3:     populasi_baru  $\leftarrow$  himpunan kosong
4:     for i  $\leftarrow$  1 to ukuran(populasi) do
5:       x  $\leftarrow$  Pilih_Aacak(populasi, fungsi_kebugaran)
6:       y  $\leftarrow$  Pilih_Aacak(populasi, fungsi_kebugaran)
7:       anak  $\leftarrow$  Reproduksi(x, y)
8:       if (probabilitas acak kecil) then
9:         anak  $\leftarrow$  Mutasi(anak)
10:      end if
11:      tambahkan anak ke populasi_baru
12:    end for
13:    populasi  $\leftarrow$  populasi_baru
14:  until kondisi berhenti terpenuhi
15:  return individu terbaik dalam populasi
16: end function

```

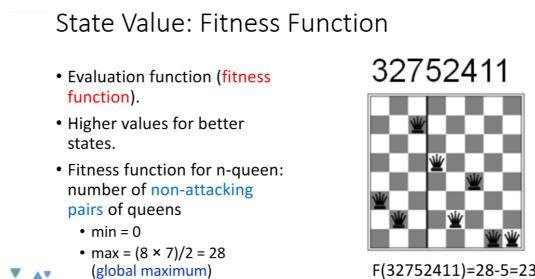
Genetic Algorithm: Illustration (Russel & Norvig, 2010)



Gambar 3.12: Ilustrasi Genetic Algorithm pada persoalan N-queens

3.8.1 Individu**Definisi 3.3**

Sebuah **state** / **individu** direpresentasikan sebagai string pada finite alphabet (seringkali berupa string dari 0s dan 1s). Satu karakter - satu variabel.



Gambar 3.13: Fitness Function untuk State Value

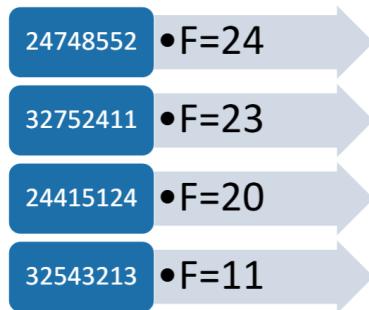
3.8.2 Populasi

Konsep

GA dimulai dengan k randomly generated states.

Sebuah successor state dihasilkan dengan menggabungkan dua parent states.

Menghasilkan next generation dari states melalui selection, crossover, dan mutation.

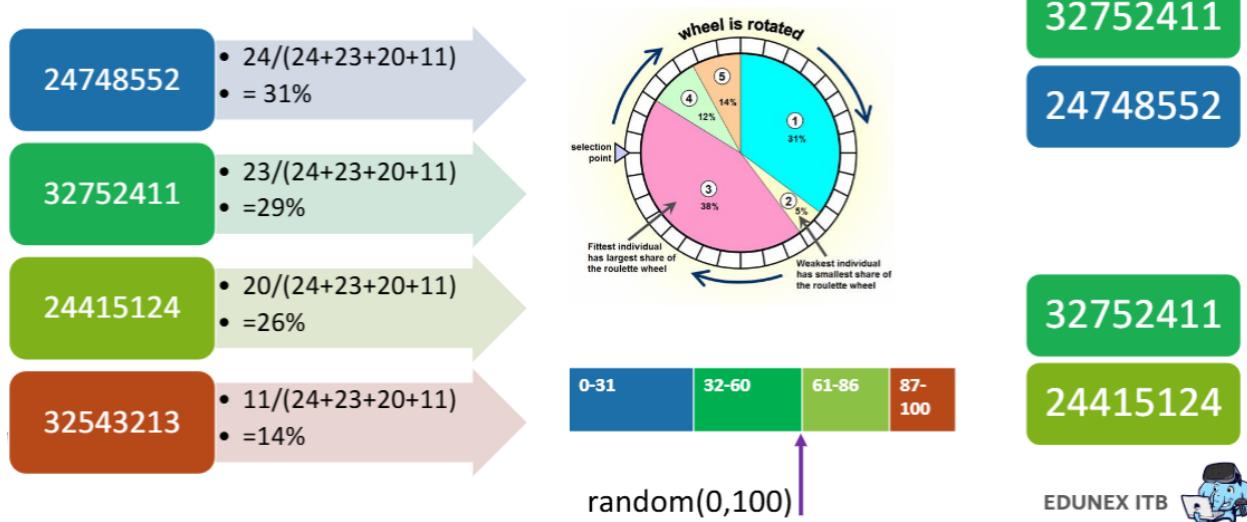


Gambar 3.14: Populasi dengan $k = 4$ individu

3.8.3 Selection

Random Selection of Parent States

Probability of random selection



Gambar 3.15: Proses Selection dengan random pada Genetic Algorithm

3.8.4 Cross Over & Mutation

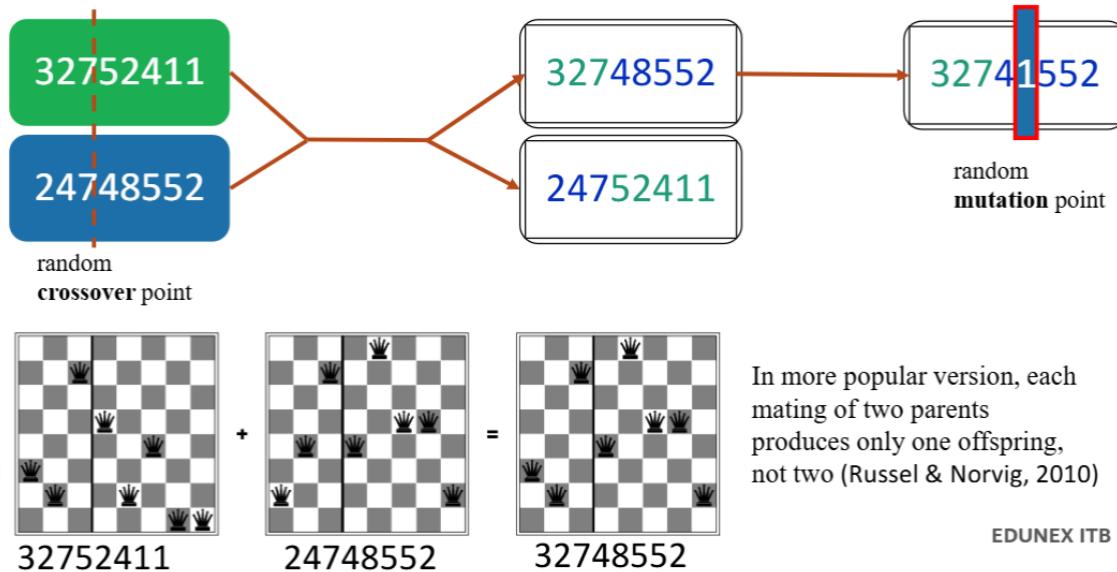
Konsep

Diambil *index/point* random sebagai titik pisah antar dua individu.

Tukar hasil pemisahan antar dua individu

Kemudian, untuk mutasi diambil *index/point* random sebagai posisi yang akan diganti dengan nilai random sesuai domain.

Cross Over / Reproduce & Mutation



Gambar 3.16: Proses Cross-over dan Mutation pada Genetic Algorithm

Bab 4

Adversarial Search

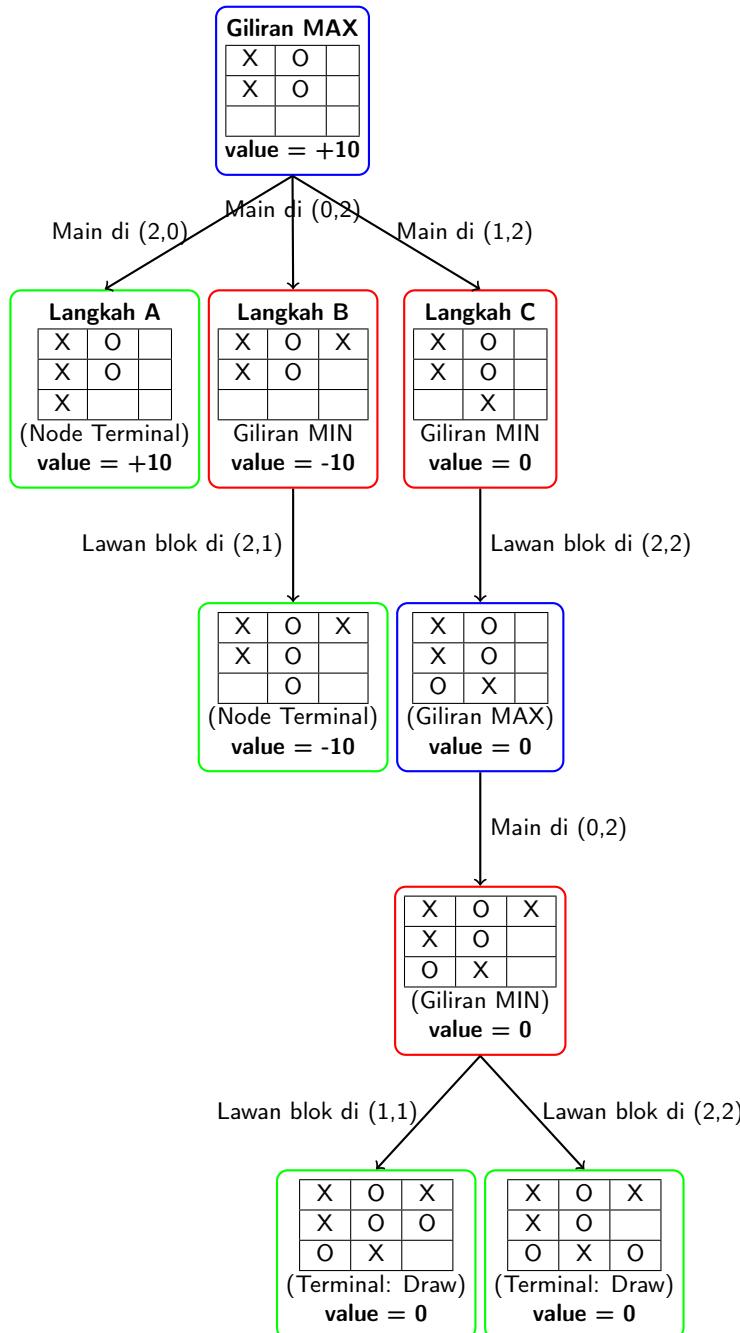
4.1 Minimax Algorithm	36
4.2 Alpha-Beta Pruning	38

Algoritma pencarian adversarial merupakan tulang punggung pengambilan keputusan strategis dalam kecerdasan buatan. Algoritma ini memungkinkan agen untuk menavigasi skenario kompetitif secara efektif.

Adversarial search sangat cocok digunakan pada kondisi di mana terdapat dua atau lebih agen yang berkompetisi. Dengan kata lain, ini biasanya diterapkan pada permainan strategi. Dengan menggunakan adversarial search, agen dapat membuat sebuah keputusan sambil mengantisipasi countermove musuh.

Ditulis oleh Muhammad Fathur Rizky.

4.1 Minimax Algorithm



Gambar 4.1: Diagram pohon keputusan Minimax. AI (MAX) akan memilih langkah A karena memiliki nilai tertinggi (+10).

Konsep

Algoritma Minimax adalah algoritma pengambilan keputusan yang digunakan dalam permainan dua pemain dengan informasi sempurna dan hasil zero-sum (kemenangan satu pemain adalah kekalahan pemain lain). Contoh klasiknya adalah Catur, Tic-Tac-Toe, dan Go. Tujuannya adalah untuk menemukan langkah optimal dengan mengasumsikan bahwa lawan juga akan bermain secara optimal.

$$\begin{cases} \text{evaluate}(n), & \text{jika } n \text{ adalah node terminal} \\ \max_{s \in \text{successors}(n)} \text{value}(s), & \text{jika } n \text{ adalah node MAX} \\ \min_{s \in \text{successors}(n)} \text{value}(s), & \text{jika } n \text{ adalah node MIN} \end{cases}$$

Algoritma ini bekerja dengan menetapkan dua peran: pemain Maximizer (MAX), yang tujuannya adalah memaksimalkan skornya, dan pemain Minimizer (MIN), yang diasumsikan akan selalu memilih langkah untuk meminimalkan skor MAX. Prosesnya dimulai dari keadaan akhir permainan (node terminal), di mana sebuah nilai numerik (disebut evaluate) diberikan, misalnya +1 untuk kemenangan MAX, -1 untuk kekalahan, dan 0 untuk seri.

Kemudian, secara rekursif mundur, pada setiap giliran MAX, algoritma akan memilih langkah yang mengarah ke keadaan dengan nilai maksimum. Sebaliknya, pada setiap giliran MIN, algoritma akan memilih langkah yang mengarah ke keadaan dengan nilai minimum. Dengan mensimulasikan pilihan optimal lawan, algoritma ini memungkinkan pemain untuk memilih langkah yang paling menguntungkan.

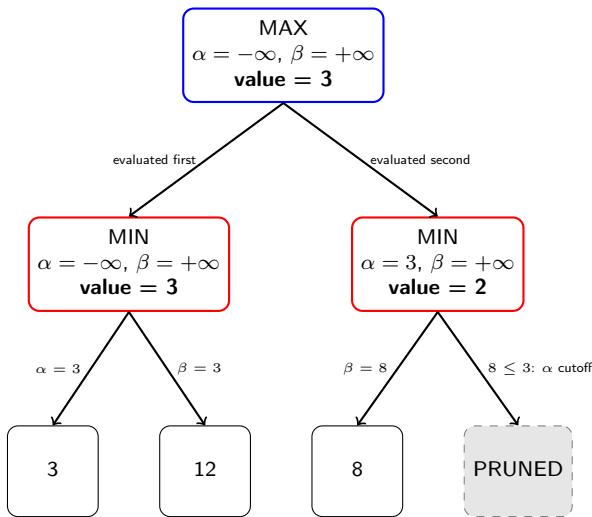
Dalam konteks contoh Tic-Tac-Toe pada Gambar 4.1, Langkah A dipilih karena langsung menghasilkan kemenangan dengan nilai +10 tanpa memberikan kesempatan kepada lawan untuk merespons. Sementara itu, Langkah B dan Langkah C hanya menghasilkan nilai 0 (seri) karena pada giliran berikutnya, lawan (MIN) akan memilih langkah yang menghalangi kemenangan MAX, yaitu memblokir jalur yang dapat menyebabkan kekalahan.

Keoptimalan algoritma Minimax terjamin karena beberapa alasan fundamental. Pertama, algoritma ini melakukan eksplorasi menyeluruh terhadap semua kemungkinan langkah hingga mencapai keadaan terminal, sehingga tidak ada kemungkinan yang terlewat. Kedua, asumsi rasionalitas lawan memastikan bahwa evaluasi dilakukan terhadap skenario terburuk yang mungkin dihadapi, sehingga keputusan yang diambil akan tetap optimal meskipun lawan bermain dengan strategi terbaik mereka. Ketiga, sifat deterministik dari algoritma memastikan bahwa untuk kondisi permainan yang sama, hasil yang diperoleh akan selalu konsisten.

4.2 Alpha-Beta Pruning

Konsep: Definisi dan Konsep Dasar

Alpha-Beta Pruning merupakan teknik optimasi yang digunakan untuk meningkatkan efisiensi algoritma Minimax dengan mengeliminasi cabang-cabang pohon pencarian yang tidak perlu dievaluasi. Teknik ini mempertahankan hasil yang sama dengan algoritma Minimax standar namun dengan kompleksitas waktu yang jauh lebih rendah. Prinsip fundamental Alpha-Beta Pruning adalah menghentikan evaluasi suatu cabang ketika sudah dapat dipastikan bahwa cabang tersebut tidak akan mempengaruhi keputusan akhir.



Gambar 4.2: Ilustrasi proses Alpha-Beta Pruning. Node kedua dari kanan tidak perlu dievaluasi karena kondisi α cutoff terpenuhi.

4.2.1 Mekanisme Kerja

Alpha-Beta Pruning bekerja dengan memelihara dua nilai parameter selama proses pencarian:

Alpha (α): Merepresentasikan nilai terbaik yang dapat dijamin oleh pemain MAX pada jalur yang telah dievaluasi. Nilai ini dimulai dari negatif tak hingga dan akan meningkat seiring dengan ditemukannya langkah yang lebih baik bagi MAX.

Beta (β): Merepresentasikan nilai terbaik yang dapat dijamin oleh pemain MIN pada jalur yang telah dievaluasi. Nilai ini dimulai dari positif tak hingga dan akan menurun seiring dengan ditemukannya langkah yang lebih baik bagi MIN.

Proses pruning terjadi ketika kondisi $\alpha \geq \beta$ terpenuhi. Kondisi ini mengindikasikan bahwa cabang saat ini tidak akan dipilih karena sudah terdapat alternatif yang lebih baik di tempat lain dalam pohon pencarian.

Algorithm 7 Alpha-Beta Pruning

```

1: function ALPHA-BETA-SEARCH(state)
2:    $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$ 
3:   return the action in ACTIONS(state) with value  $v$ 
4: end function

5:
6: function MAX-VALUE(state,  $\alpha, \beta$ )
7:   if TERMINAL-TEST(state) then
8:     return UTILITY(state)
9:   end if
10:   $v \leftarrow -\infty$ 
11:  for each  $a$  in ACTIONS(state) do
12:     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(\text{state}, a), \alpha, \beta))$ 
13:    if  $v \geq \beta$  then
14:      return  $v$                                  $\triangleright \beta$  cutoff
15:    end if
16:     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
17:  end for
18:  return  $v$ 
19: end function

20:
21: function MIN-VALUE(state,  $\alpha, \beta$ )
22:   if TERMINAL-TEST(state) then
23:     return UTILITY(state)
24:   end if
25:    $v \leftarrow +\infty$ 
26:   for each  $a$  in ACTIONS(state) do
27:      $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\text{state}, a), \alpha, \beta))$ 
28:     if  $v \leq \alpha$  then
29:       return  $v$                                  $\triangleright \alpha$  cutoff
30:     end if
31:      $\beta \leftarrow \text{MIN}(\beta, v)$ 
32:   end for
33:   return  $v$ 
34: end function

```

4.2.2 Keunggulan dan Efisiensi

Alpha-Beta Pruning memberikan peningkatan performa yang signifikan dibandingkan dengan algoritma Minimax standar. Dalam skenario optimal, di mana langkah-langkah terbaik selalu dievaluasi terlebih dahulu, Alpha-Beta Pruning dapat mengurangi kompleksitas waktu dari $O(b^d)$ menjadi $O(b^{d/2})$, di mana b adalah branching factor dan d adalah kedalaman pencarian.

Efisiensi pruning sangat bergantung pada urutan evaluasi node. Pengurutan yang baik dapat meningkatkan jumlah pruning yang terjadi, sementara pengurutan yang buruk dapat mengurangi efektivitas teknik ini. Oleh karena itu, heuristik pengurutan langkah menjadi faktor penting dalam implementasi Alpha-Beta Pruning yang efisien.

$$\text{Minimax tanpa pruning : } O(b^d) \quad (4.1)$$

$$\text{Alpha-Beta Pruning (optimal) : } O(b^{d/2}) \quad (4.2)$$

$$\text{Alpha-Beta Pruning (rata-rata) : } O(b^{3d/4}) \quad (4.3)$$

Bab 5

Constraint Satisfaction Problem

5.1	What is CSP?	41
5.2	Visualisai CSP	41
5.3	Variasi dari Formulasi CSP	42
5.4	Costraint Propagation & Inference . .	43
5.5	Improving Backtracking Efficiency . .	44
5.6	Interleaving Search	45
5.7	Local Search for CSP	46

5.1 What is CSP?

Teorema 5.1

Sebuah *constraint satisfaction problem* terdiri dari tiga komponen, yaitu:

X sebuah himpunan variabel, $\{X_1, \dots, X_n\}$

D sebuah himpunan domain, satu untuk setiap variabel, $\{D_1, \dots, D_n\}$

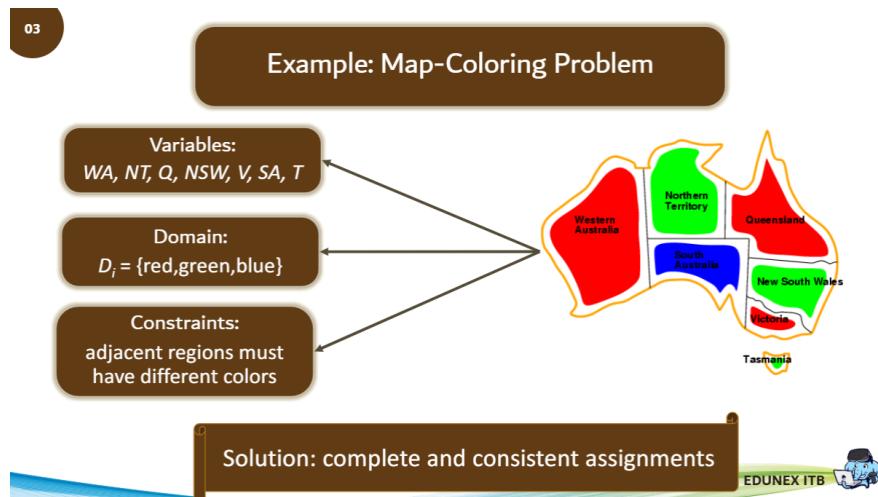
C sebuah himpunan constraints yang menentukan kombinasi yang diperbolehkan.

Konsep

CSP search algorithms memanfaatkan struktur dari states dan menggunakan *heuristics* yang bersifat general daripada *domain-specific* untuk memungkinkan penyelesaian masalah yang kompleks.

Ide utamanya adalah mengeliminasi bagian besar dari search space sekaligus dengan mengidentifikasi kombinasi *variable/value* yang melanggar *constraints*.

CSPs memiliki keuntungan tambahan bahwa *actions* dan *transition* model dapat disimpulkan dari *problem description*.



Gambar 5.1: Contoh persoalan pewarnaan peta yang dapat diselesaikan dengan CSP

5.2 Visualisasi CSP

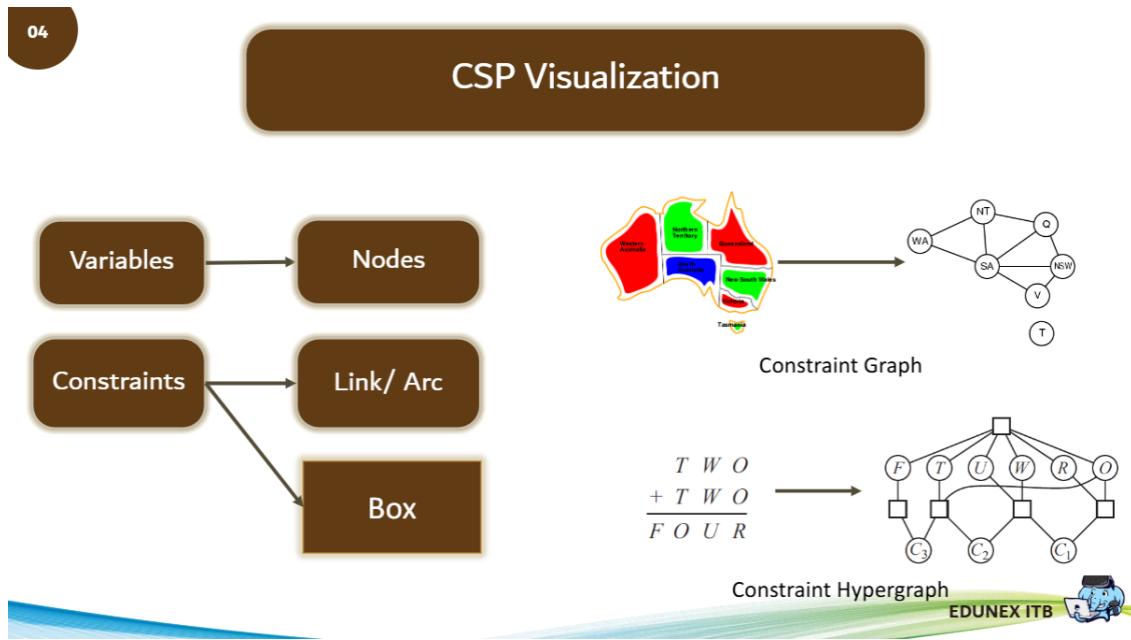
Konsep

Variabel: direpresentasikan sebagai nodes.

Constraints: direpresentasikan sebagai *link* ataupun *box* tergantung pada *graph* yang digunakan.

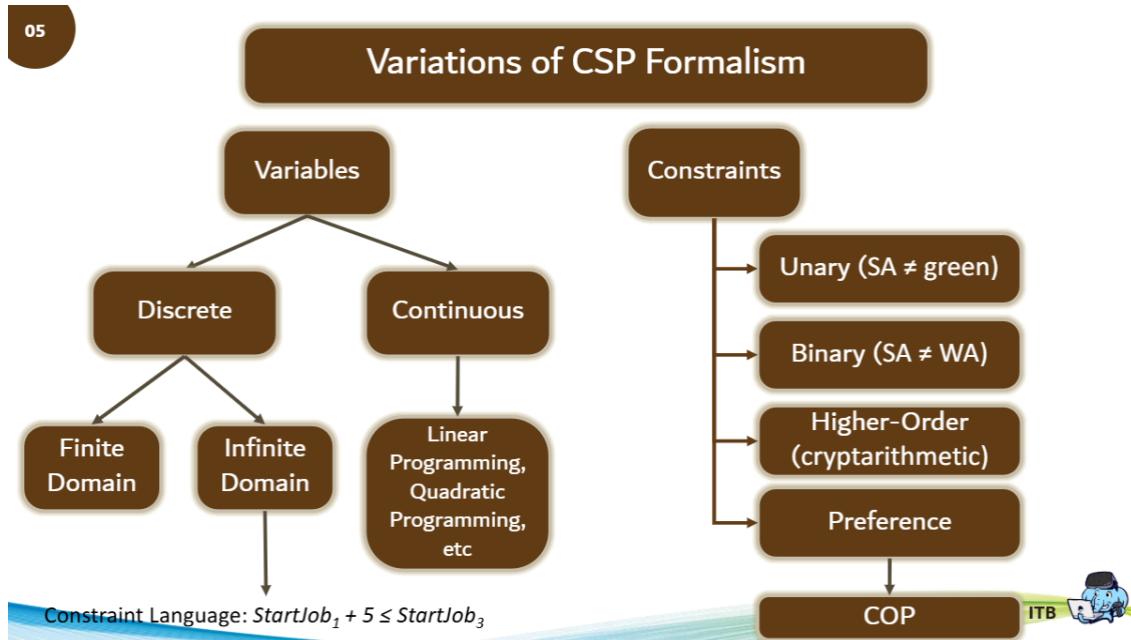
Constraint Graph: sebuah edge menghubungkan dua variables yang terlibat dalam sebuah constraint. (Lebih baik dengan jumlah *constraint* yang sedikit)

Constraint Hypergraph: sebuah box yang terhubung oleh dua nodes yang terlibat dalam sebuah constraint. (Lebih baik dengan jumlah *constraint* yang banyak)



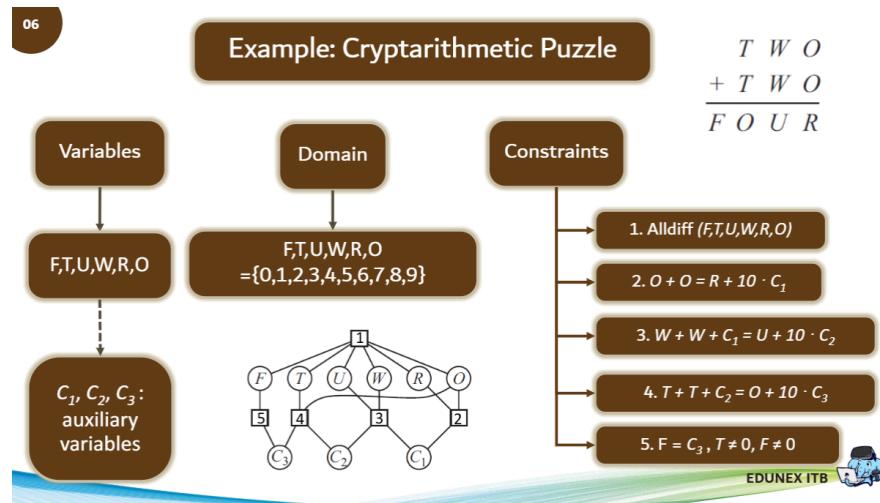
Gambar 5.2: Contoh persoalan pewarnaan peta yang dapat diselesaikan dengan CSP

5.3 Variasi dari Formulasi CSP



Gambar 5.3: Variasi dari Formulasi CSP

Contoh persoalan:



Gambar 5.4: Contoh persoalan cryptaritmetic puzzle

5.4 Constraint Propagation & Inference

Teorema 5.2

Menggunakan *constraint* untuk mengurangi nilai yang diperbolehkan pada suatu variabel, yang pada akhirnya bisa mengurangi nilai yang diperbolehkan pada variabel lain.

Ide kuncinya adalah menjaga kekonsistensiannya lokal. Maka, prosesnya dengan menghilangkan nilai yang menyebabkan graf tersebut menjadi tidak konsisten

Konsep: Node Consistency

Jika semua nilai dalam domain variabel tersebut memenuhi batasan dari *unary constraints*.

Sebuah *network* dikatakan *node consistent* jika untuk tiap variabel yang terdapat pada *network* tersebut, variabelnya bersifat *node consistent*.

Konsep: Arc Consistency

Jika setiap nilai pada domain variabel tersebut memenuhi / tidak melanggar *binary constraint* yang ada / melibatkan variabel tersebut.

Konsep: Path Consistency

X_i, X_j adalah path-consistent terhadap X_m jika:

Assignment X_i = a, X_j = b konsisten dengan constraints pada X_i, X_j.

Ada assignment ke X_m yang memenuhi constraints pada X_i, X_m dan X_j.

Konsep: K-Consistency

Sebuah CSP adalah k-consistent jika: untuk setiap himpunan dari $k - 1$ variabel dan setiap *consistent assignment* pada variabel tersebut, ada sebuah *consistent value* yang dapat diberikan pada variabel ke- k .

1-consistency: diberikan empty set, dapat membuat himpunan dari satu variable menjadi consistent.

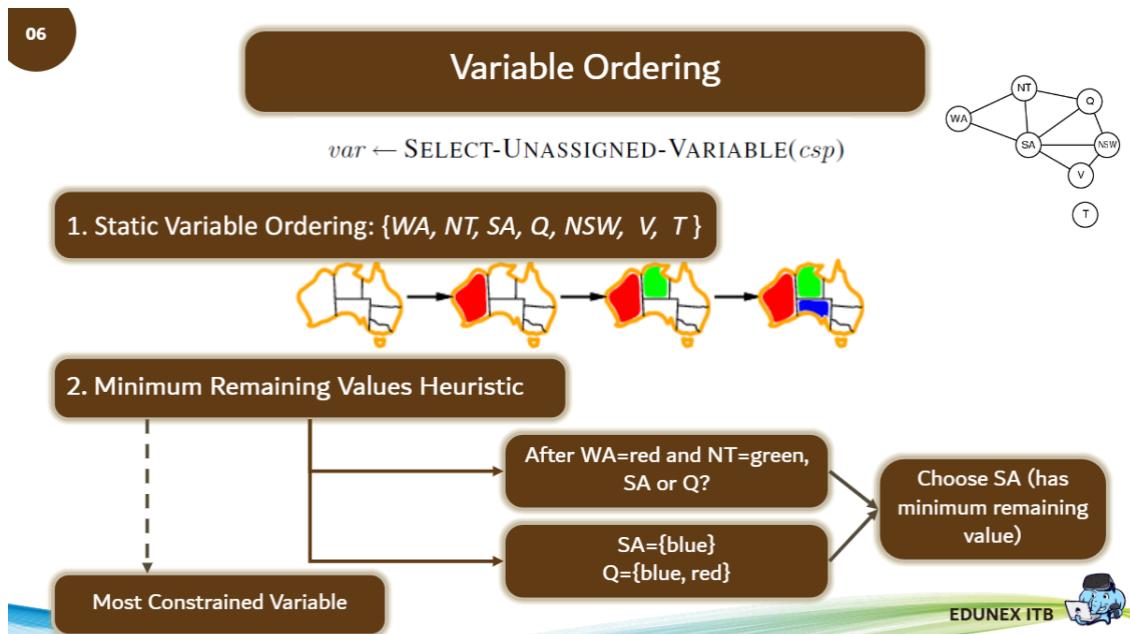
2-consistency = Arc Consistency.

3-consistency = Path Consistency.

5.5 Improving Backtracking Efficiency

Konsep: Variable Ordering

1. **Static:** Menetapkan dari awal urutan yang diinginkan.
 2. **Minimum Remaining Values Heuristic:** Memilih variabel dengan nilai kemungkinannya lebih sedikit untuk urutan selanjutnya.
- Degree Heuristic:** Memilih variabel dengan nilai kemungkinannya lebih sedikit untuk urutan selanjutnya.

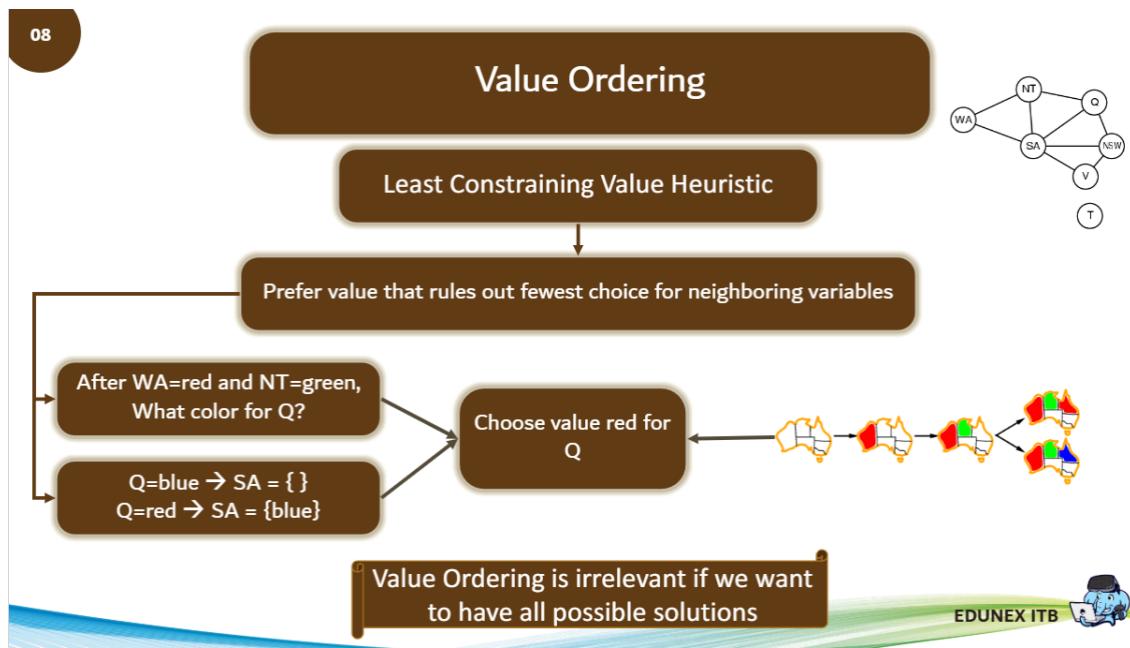


Gambar 5.5: Variable Ordering

Konsep: Value Ordering

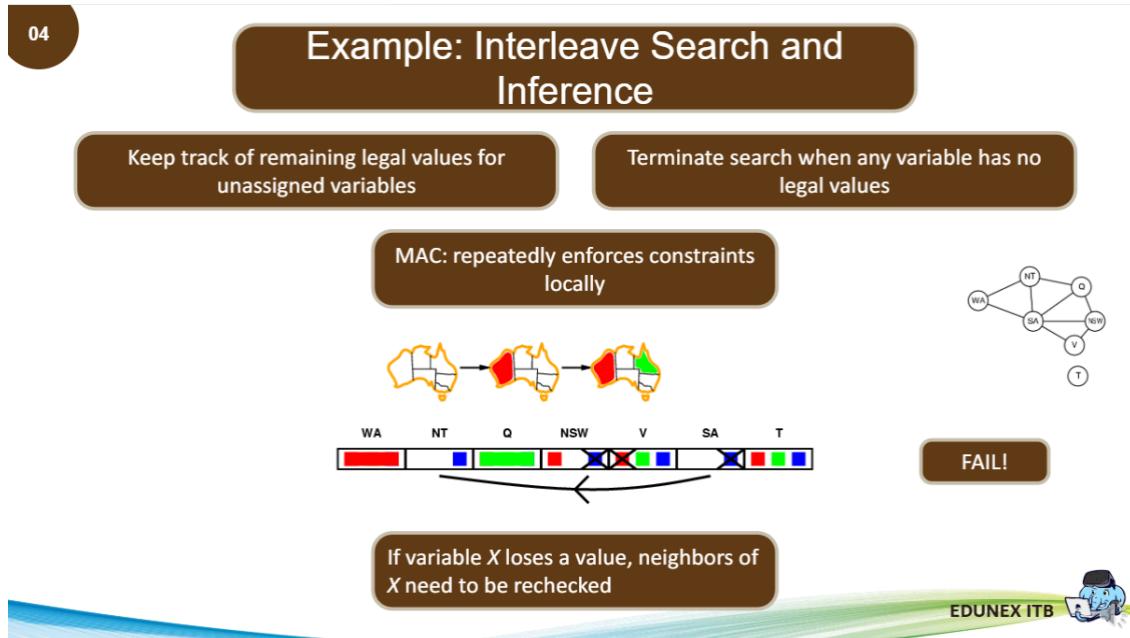
Least Constraining Value Heuristics - Lebih memilih nilai yang akan memberikan kebebasan yang lebih besar kepada variabel berikutnya yang belum di-*assign* nilai.

Tidak relevan ketika menginginkan semua kemungkinan solusi untuk sebuah persoalan CSP



Gambar 5.6: Value Ordering

5.6 Interleaving Search



Gambar 5.7: Value Ordering

5.7 Local Search for CSP

Konsep: Min-Conflict Heuristic

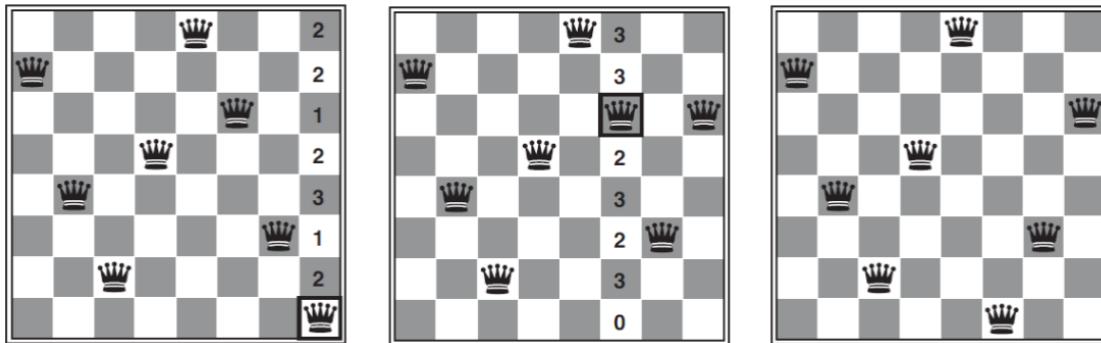
Merupakan strategi pencarian lokal untuk menyelesaikan Constraint Satisfaction Problem (CSP) dengan cara memilih variabel yang bermasalah dan mengganti nilainya agar meminimalkan jumlah konflik. Heuristik ini efisien pada masalah besar karena langsung berfokus mengurangi pelanggaran, bukan mengeksplorasi semua kemungkinan.

Algorithm 8 Min-Conflicts

```

1: function MIN-CONFLICTS(csp, max_steps)
2:   current  $\leftarrow$  sebuah penugasan lengkap awal untuk csp
3:   for i  $\leftarrow 1$  to max_steps do
4:     if current adalah solusi untuk csp then
5:       return current
6:     end if
7:     var  $\leftarrow$  pilih secara acak variabel yang konflik dari csp.VARIABLES
8:     value  $\leftarrow$  nilai v untuk var yang meminimalkan CONFLICTS(var, v, current, csp)
9:     set var  $\leftarrow$  value dalam current
10:   end for
11:   return failure
12: end function

```



Gambar 5.8: Min-Conflict Heuristic pada persoalan N-queen