1) Pengantar Algoritma Genetika

a) Contoh Sederhana Aplikasi Algoritma Genetika

- Algoritma Genetika memiliki kemampuan luar biasa dalam menemukan solusi optimal untuk masalah yang memiliki ruang pencarian yang sangat besar. Contoh klasik yang sering digunakan untuk mengilustrasikan cara kerja Algoritma Genetika adalah masalah memaksimalkan fungsi kuadrat sederhana, seperti $f(x) = x^2$ dalam rentang $0 \leq x \leq 31$. Dalam kasus ini, solusi optimalnya jelas, yaitu $X = 31$, menghasilkan $f(x) = 961$. Namun, masalah ini digunakan untuk mendemonstrasikan bagaimana Algoritma Genetika, meskipun dimulai dengan populasi acak (misalnya, empat kromosom biner 5-bit) , secara bertahap dapat memadukan fitur-fitur (building blocks) dari solusi yang ada untuk mencapai hasil yang lebih baik. Melalui siklus seleksi, crossover, dan mutasi, Algoritma Genetika menunjukkan bagaimana kinerja maksimal dan rata-rata populasi meningkat dari satu generasi ke generasi berikutnya. Dalam beberapa generasi, individu yang merepresentasikan $x = 31\ (biner\ 11111)$ akan mendominasi populasi.
- Contoh lain dari aplikasi algoritma genetika adalah Travelling Salesman Problem (TSP) di mana Algoritma Genetika digunakan untuk menemukan rute terpendek yang mengunjungi sejumlah kota, sebuah masalah optimasi kombinatorial yang sangat kompleks.

b) Algoritma Genetika untuk Penyelesaian Masalah Nyata

Algoritma Genetika menjadi semakin populer dalam rekayasa industri, manajemen sains, dan riset operasi karena kemampuannya dalam menangani masalah optimasi yang rumit. Banyak masalah rekayasa dunia nyata yang terlalu kompleks, non-linear, atau mengandung kendala yang rumit sehingga sulit diselesaikan dengan metode konvensional. Algoritma Genetika menawarkan metode yang robust dan dapat diterapkan secara luas untuk pencarian stokastik dan optimasi. Daripada hanya mencari satu titik solusi, Algoritma Genetika bekerja pada populasi solusi dan hanya memerlukan nilai fungsi objektif (fitness) tanpa memerlukan turunan atau pengetahuan awal yang mendalam tentang ruang pencarian. Hal ini menjadikan Algoritma Genetika sangat cocok untuk masalah, seperti

1. Penjadwalan (*Scheduling*)
   Masalah penjadwalan, seperti penjadwalan kuliah, produksi pabrik (*Job Shop Scheduling*), atau penerbangan, merupakan masalah optimasi kombinatorial yang dicirikan oleh sejumlah besar tugas, sumber daya terbatas, dan kendala waktu yang ketat. Dalam konteks ini, kromosom AG merepresentasikan satu urutan atau alokasi tugas (*schedule*). Fungsi *fitness* dirancang untuk meminimalkan kriteria seperti waktu penyelesaian total (*makespan*), jumlah keterlambatan, atau biaya operasional. Algoritma Genetika mampu menavigasi ruang solusi yang diskrit dan besar dengan sangat efisien, mencari urutan tugas terbaik yang memenuhi semua kendala yang rumit.

2. Desain jaringan telekomunikasi
   Dalam perancangan jaringan, Algoritma Genetika digunakan untuk mengoptimalkan topologi, kapasitas, dan penempatan node atau router untuk mencapai kinerja terbaik. Tujuannya seringkali adalah meminimalkan biaya pembangunan jaringan sambil memaksimalkan kualitas layanan, seperti meminimalkan delay atau memaksimalkan throughput. Kromosom dapat mengkodekan konfigurasi koneksi antar-node atau pilihan teknologi. Karena setiap perubahan konfigurasi dapat berdampak besar pada kinerja jaringan secara keseluruhan, Algoritma Genetika sangat berguna untuk menjelajahi berbagai arsitektur jaringan yang mungkin, yang secara matematis sulit diselesaikan dengan metode tradisional.
3. Perancangan VLSI (*Very Large Scale Integration)*
   VLSI adalah proses perancangan chip semikonduktor dengan jutaan transistor. Algoritma Genetika sangat relevan dalam dua sub-masalah utama, yaitu penempatan (placement) dan perutean (routing). Dalam masalah penempatan, Algoritma Genetika mengoptimalkan posisi blok-blok sirkuit di atas chip untuk meminimalkan panjang kawat total dan menghindari tumpang tindih. Dalam perutean, Algoritma Genetika mencari jalur terbaik untuk menghubungkan semua pin sirkuit tanpa melanggar batasan fisik. Tingginya dimensi dan kendala geometris (seperti area, daya, dan timing) membuat ruang pencarian VLSI eksplosif, yang menjadikan Algoritma Genetika sebagai teknik pencarian yang feasible untuk menghasilkan tata letak yang hampir optimal.

c) Alur Algoritma Genetika

Algoritma genetika memelihara sebuah populasi dari individu, yang dinotasikan sebagai $P(t)$ untuk generasi ke-t, di mana setiap individu merepresentasikan solusi potensial terhadap masalah yang sedang dihadapi. Siklus ini berjalan secara iteratif melalui langkah-langkah yang meniru proses evolusi. Setelah populasi awal $(P(0))$ diinisialisasi dan dievaluasi fitness-nya, proses seleksi dimulai, di mana individu-individu yang lebih fit dipilih secara probabilistik untuk memasuki mating pool. Individu yang terpilih kemudian mengalami transformasi stokastik melalui operator genetik. Operator crossover bertanggung jawab untuk exploiting informasi terbaik dari parents, sementara mutation bertugas untuk exploring ruang pencarian dengan memperkenalkan materi genetik baru. Individu baru yang dihasilkan, disebut offspring $C(t)$ kemudian dievaluasi fitness-nya. New population $(P(t+1)$ dibentuk dengan memilih individu-individu yang lebih fit, baik dari parent population maupun offspring population, melalui skema replacement. Proses ini diulang selama beberapa generasi hingga algoritma coverage menunjuk individu terbaik, yang diharapkan merepresentasikan sebuah solusi optimal atau suboptimal terhadap masalah.

```
Procedure: Genetic Algorithms
begin
 t ← 0;
 initialize P(t);
 evaluate P(t);
 while (not termination condition) do
 begin
  recombine P(t) to yield C(t);
  evaluate C(t);
  select P(t + 1) from P(t) and C(t);
  t ← t + 1;
 end
end
```

d) Variasi Algoritma Genetika

Meskipun Algoritma Genetika Generational Sederhana menjadi kerangka dasar, berbagai modifikasi telah dikembangkan untuk meningkatkan performa dalam menghadapi kompleksitas masalah. Beberapa dari modifikasi tersebut adalah:

1. Hybrid GA
   HGA menggabungkan Algoritma Genetika dengan teknik pencarian lokal (local search) konvensional. Dalam pendekatan hibrida, Algoritma Genetika melakukan eksplorasi global antar populasi, sementara pencarian lokal digunakan untuk penyempurnaan di sekitar solusi yang menjanjikan. Pendekatan ini sering mengungguli metode tunggal karena memanfaatkan keunggulan komplementer dari kedua teknik pencarian.
2. Adaptive GA
   AGA merupakan Algoritma Genetika yang memiliki parameter strategis (seperti probabilitas crossover atau mutasi) disesuaikan secara dinamis selama proses evolusi, seringkali menggunakan feedback dari kinerja populasi untuk menyeimbangkan eksploitasi dan eksplorasi secara efektif.
3. AG Paralel
   Paralel Algoritma Genetika merupakan Algoritma Genetika yang membagi populasi menjadi sub populasi yang berbeda di berbagai prosesor, memungkinkan pertukaran informasi secara periodik (migration) untuk meningkatkan keragaman dan kecepatan konvergensi.

Referensi

Gen, M., & Cheng, R. (2007). GENETIC ALGORITHMS AND ENGINEERING OPTIMIZATION. *Genetic Algorithms and Engineering Optimization*, 1–495. https://doi.org/10.1002/9780470172261

Mitchell, M. (1996). An Introduction to Genetic Algorithms. *An Introduction to Genetic Algorithms*. https://doi.org/10.7551/MITPRESS/3927.001.0001

2) Holland Schema

a) Pendahuluan

Dalam Algoritma Genetika, individu direpresentasikan dalam bentuk string (biasanya biner). Namun, fokus tidak hanya pada satu string, melainkan pada pola kesamaan di antara string-string yang memiliki tingkat *fitness* tinggi. Untuk menjelaskan kesamaan ini, John Holland (1968, 1975) memperkenalkan konsep schema.

b) Definisi Schema

Schema adalah *similarity template* yang mendeskripsikan subset dari string dengan kesamaan tertentu pada posisi tertentu. Dengan kata lain, schema adalah pola umum yang mewakili sekelompok string.

- *Binary alphabet*: {0,1}
- *Extended alphabet*: {0,1,*}, dengan tambahan simbol * (*don't care symbol*) yang berarti posisi tersebut bisa bernilai 0 atau 1.

Simbol * disebut *metasymbol* karena hanya digunakan untuk notasi pola dan tidak diproses secara eksplisit oleh algoritma genetika.

Sebagai contoh, string dengan panjang 5:

Schema	String yang Cocok	Jumlah String
*0000	{10000, 00000}	2
111	{01110, 01111, 11110, 11111}	4
0*1**	Semua string panjang 5 yang dimulai dengan 0 dan memiliki 1 di posisi ketiga	8

c) Karakteristik Penting Schema Holland

1. Order of Schema (o(S))

   Order dari sebuah S schema (dilambangkan dengan o(S)) menunjukkan jumlah posisi tetap bernilai 0 atau 1 (bukan posisi *don't care*) pada schema tersebut. *Order* ini menunjukkan tingkat *specialization* dari sebuah schema.

S1 = (* * * 0 0 1 * 1 1 0)	o(S1) = 6
S2 = (* * * * 0 0 * * 0 *)	o(S2) = 3
S3 = (1 1 1 0 1 * * 0 0 1)	o(S3) = 8

2. Defining Length (δ(S))

   Length limit dari sebuah S schema (dilambangkan dengan δ(S)) adalah jarak antara posisi digit pertama hingga digit terakhir yang bernilai 0 atau 1. Nilai ini menunjukkan *density of information* dalam sebuah schema.

S1 = (* * * 0 0 1 * 1 1 0)	δ(S1) = 10 - 4 = 6
S2 = (* * * * 0 0 * * 0 *)	δ(S2) = 9 - 5 = 4
S3 = (1 1 1 0 1 * * 0 0 1)	δ(S3) = 10 - 1 = 9

3. Fitness Schema (f(S))

   Holland membuktikan bahwa *schema* dengan (1) *fitness* rata-rata di atas rata-rata populasi, (2) *order* rendah, (3) *defining length* pendek memiliki peluang lebih besar untuk berkembang (propagasi) di generasi berikutnya.

   Formulasi dasar jumlah rata-rata kromosom dalam *schema* S pada generasi t+1:

$$E[m(S,t+1)] \geq m(S,t) \cdot \frac{f(S)}{f} \cdot [1 - P_c \frac{\delta(S)}{(l-1)} - o(S)P_m]$$

   Keterangan:

   - $m(S,t)$ = jumlah kromosom dalam populasi pada generasi t yang sesuai dengan *schema* S.
   - $f(S)$ = fitness rata-rata *schema* S.
   - $\bar{f}$ = *fitness* rata-rata populasi.
   - P_c = probabilitas *crossover*.
   - P_m = probabilitas mutasi per bit.
   - l = panjang kromosom.
   - $\delta(S)$ = *defining length*.
   - $o(S)$ = *order schema*.

d) Order Schema dalam Algoritma Genetika

1. Tingkat Kekhususan Pola

   Order suatu *schema* menunjukkan tingkat kekhususan pola yang diwakili dalam kromosom. Semakin tinggi *order*, semakin spesifik pola yang digambarkan. Misalnya, *schema* dengan *order* rendah seperti 1***** masih mewakili banyak kemungkinan kromosom karena hanya satu posisi yang tetap, sementara *schema* dengan *order* tinggi seperti 101011 sangat spesifik dan hanya sesuai dengan satu kromosom tertentu. Dengan demikian, *order* berperan dalam menentukan seberapa luas cakupan representasi sebuah *schema* dalam populasi.

2. Peluang Bertahan dalam Evolusi

Order suatu *schema* berpengaruh besar pada peluang schema untuk bertahan melalui proses evolusi. Pada algoritma genetika, dua operator utama yang sering menyebabkan perubahan adalah *crossover* dan mutasi.

Schema dengan *order* rendah relatif lebih aman terhadap perubahan ini karena hanya memiliki sedikit posisi tetap (*fixed positions*). Misalnya, *schema* 1***** hanya mengunci satu bit di posisi awal. Jika terjadi mutasi pada salah satu posisi lain atau *crossover* memotong bagian tengah, peluang kerusakan schema tersebut sangat kecil. Dengan kata lain, semakin sedikit bit tetap yang harus dipertahankan, semakin besar kemungkinan *schema* tersebut selamat dan diwariskan ke generasi berikutnya.

Sebaliknya, schema dengan *order* tinggi seperti 101011 memiliki banyak bit tetap yang harus persis sama agar tetap valid. Dalam kondisi seperti ini, bahkan satu mutasi kecil pada salah satu posisi tetap dapat merusak *schema* secara keseluruhan. Begitu pula pada proses *crossover*, peluang terpotong di antara posisi-posisi tetap menjadi lebih besar. Akibatnya, *schema* dengan *order* tinggi sering kali cepat hilang dari populasi karena sulit bertahan dari kombinasi variasi genetik yang terjadi.

3. Hubungan dengan Seleksi

   Meskipun tampak sederhana, *schema* dengan *order* rendah justru memiliki peran yang sangat penting dalam algoritma genetika. Kesederhanaan struktur ini memungkinkan schema untuk lebih tahan terhadap kerusakan akibat *crossover* maupun mutasi, sehingga pola-pola tersebut lebih sering bertahan dan diwariskan ke generasi berikutnya. Apabila schema sederhana tersebut mengandung pola yang relevan dengan solusi optimal misalnya kombinasi bit tertentu yang meningkatkan nilai *fitness*, maka *schema* itu akan muncul berulang kali pada berbagai kromosom dalam populasi.

   Fenomena ini sejalan dengan konsep *building block hypothesis* yang dikemukakan oleh Holland. Algoritma genetika tidak langsung mencari solusi kompleks secara keseluruhan, tetapi bekerja dengan mempertahankan dan menyusun blok-blok pola sederhana yang memiliki kontribusi positif terhadap fitness. Blok-blok ini kemudian digabungkan melalui seleksi, *crossover*, dan mutasi, sehingga membentuk struktur genetik yang lebih kompleks dan mendekati solusi optimal.

   Proses seleksi memegang peran sentral di sini. Individu yang memiliki *schema* dengan kontribusi tinggi terhadap *fitness* akan lebih sering dipilih untuk bereproduksi. Dengan demikian, *schema* sederhana yang bermanfaat dapat menyebar luas dalam populasi. Seiring waktu, kombinasi *building block* dari berbagai *schema* sederhana tersebut menghasilkan solusi yang tidak hanya lebih kompleks, tetapi juga lebih efisien dalam memecahkan masalah.

e) Relationship antara Holland Schema dan Genome

Jumlah *genome sequences* yang dapat direpresentasikan oleh sebuah *schema* bergantung pada banyaknya simbol *don't care* (*). Setiap simbol *don't care* dapat

bernilai 0 atau 1, sehingga sebuah *schema* dengan k simbol *don't care* akan menghasilkan 2^k kemungkinan *genome sequences*.

- S4 memiliki 1 *don't care*, sehingga terdapat $2^1 = 2$ kemungkinan *genome sequences*: 110010; 110110
- S5 memiliki 2 *don't care*, sehingga terdapat $2^2 = 4$ kemungkinan *genome sequences*: 1110000; 1110100; 0110000; 0110100
- S6 memiliki 3 *don't care*, sehingga terdapat $2^3 = 8$ kemungkinan *genome sequences*: 101100111; 101100110; 101100010; 101100011; 100100111; 100100110; 100100011; 100100111

f) Fungsi Schema dalam Algoritma Genetika

- Memberikan cara *powerful* dan *compact* untuk merepresentasikan *well-defined similarities* di antara string.
- Menjadi dasar teori dalam menjelaskan bagaimana pola yang baik (fit) dalam populasi dapat dipertahankan dan diperbanyak melalui crossover dan mutation.
- Membantu memahami bahwa Algoritma Genetika bekerja bukan hanya pada individu tunggal, tetapi juga pada kelas string yang berbagi pola (schema).

Referensi

Goldberg, D. E., & Holland, J. H. (1979). Genetic Algorithms in Search, Optimization, and Machine Learning David E. Goldberg The University of Alabama T. *Machine Learning*, *3*(2), 95–99. https://books.google.com/books/about/Genetic_Algorithms_in_Search_Optimizatio.html?id=2IIJAAAACAAJ

Sivanandam, S. N., & Deepa, S. N. (2008). Introduction to genetic algorithms. *Introduction to Genetic Algorithms*, 1–442. https://doi.org/10.1007/978-3-540-73190-0

Course Material Week 2 - Cycle of Genetic Algorithm and Holland Schema

3) Encoding

a) Problem Identification and Formulation

Salah satu hal yang perlu diperhatikan ketika menggunakan algoritma genetika adalah bagaimana kita merepresentasikan data yang kita teliti. Karena itulah, kita harus menentukan *encoding* (pengkodean) apa yang paling cocok dengan masalah yang kita hadapi. Seperti metode pencarian atau pembelajaran lain-lainnya, bagaimana encoding dilakukan terhadap kandidat-kandidat solusi merupakan salah satu, jika bukan faktor paling utama dari kesuksesan algoritma genetika itu. Kebanyakan aplikasi dari algoritma genetika menggunakan *string bit* dengan panjang dan urutan yang telah ditentukan. Namun, akhir-akhir ini ada banyak eksperimen yang telah dilakukan dengan jenis-jenis *encoding* lainnya

b) What is Encoding

Encoding adalah proses merepresentasikan gen-gen individual. Proses ini dapat dilakukan dengan *bit,* angka, *tree*, larik (*array*), *list*, maupun objek apapun selain itu. Encoding yang dilakukan utamanya bergantung terhadap cara menyelesaikan objek terkait. Sebagai contoh, encoding dapat dilakukan dengan bilangan bulat maupun real.

c) Encoding Types

1. Binary Encoding

Binary encoding, yaitu *string bit*, merupakan jenis encoding yang paling sederhana dan banyak digunakan. Banyak peneliti ternama yang menggunakan binary encoding dalam permasalahannya, contohnya Holland, seperti pada skema holland. Setiap kromosom dikodekan menjadi suatu string biner. Tiap-tiap bit didalamnya merepresentasikan suatu fitur atau karakteristik dari solusi tersebut. Binary encoding dapat menyimpan banyak informasi dengan jumlah alel yang minim (misal 0 atau 1, dibandingkan desimal yang memiliki banyak alel).

Berikut adalah beberapa contoh kromosom dengan binary encoding:

Kromosom 1	1 1 0 1 0 0 0 1 1 0 1 0
Kromosom 2	0 1 1 1 1 1 1 1 1 1 0 0

Di sisi lain, encoding ini tidak alami dan sulit untuk diimplementasikan bagi kebanyakan masalah yang umumnya memiliki fitur-fitur berupa bilangan real, dan seringkali koreksi harus dilakukan setelah operasi genetik selesai dilakukan.

2. Value Encoding

Permasalahan pada umumnya memiliki nilai nilai yang kompleks. Nilai-nilai seperti ini cukup sulit untuk direpresentasikan dalam bentuk biner. Pada permasalahan semacam

inilah Value encoding digunakan. Setiap kromosom merupakan *string of values,* dimana value pada tiap string sendiri dapat berupa apapun yang berkaitan dengan permasalahannya. Hal ini mencakup bilangan bulat, huruf, hingga bilangan real dan nilai-nilai lainnya.

Berikut adalah beberapa contoh kromosom dengan value encoding:

Kromosom A	1.2324 5.3242 0.4556 2.3293 2.4545
Kromosom B	ABDJEIFJDHDIERJFDLDFLEGT
Kromosom C	(back), (back), (right), (forward), (left)

3. Permutation Encoding

Ada beberapa permasalahan spesial yang tidak melihat kromosom dari nilai gen individualnya. Mereka melihat posisi kemunculan tiap gen, seperti pada rata-rata ordering problem. Setiap gen dapat berupa bilangan bulat maupun real, yang merepresentasikan angka-angka dalam suatu urutan.

Berikut adalah beberapa contoh kromosom dengan permutation encoding:

Kromosom A	1 5 3 2 6 4 7 9 8
Kromosom B	8 5 6 7 2 3 1 4 9

4. Tree Encoding

Tree encoding paling sering digunakan pada ekspresi program yang terus berubah dalam genetic programming. Tiap kromosom merupakan Tree dari suatu objek seperti fungsi atau perintah dari suatu bahasa pemrograman. Tree encoding memiliki sejumlah keunggulan, salah satunya adalah memungkinkan ruang pencarian yang terbuka, karena pada dasarnya Tree dengan ukuran apapun dapat dibentuk melalui crossover dan mutasi.

Akan tetapi, keterbukaan tersebut dapat menjadi salah satu kelemahan dari metode encoding ini. Tree yang dibentuk dapat menjadi terlalu besar tanpa bisa dikendalikan, sehingga menghalangi pembentukan kandidat solusi yang lebih terstruktur dan hierarkis. Selain itu, Tree yang dihasilkan cenderung sulit untuk diinterpretasikan dan disederhanakan karena ukurannya yang besar.

Referensi

Gen, M., & Cheng, R. (2007). GENETIC ALGORITHMS AND ENGINEERING OPTIMIZATION. *Genetic Algorithms and Engineering Optimization*, 1–495. https://doi.org/10.1002/9780470172261

Mitchell, M. (1996). An Introduction to Genetic Algorithms. *An Introduction to Genetic Algorithms*. https://doi.org/10.7551/MITPRESS/3927.001.0001

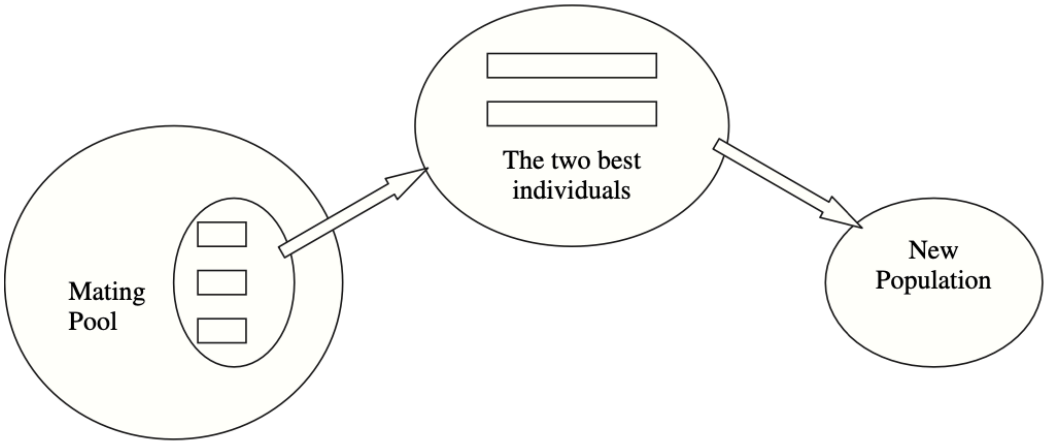
Sivanandam, S. N., & Deepa, S. N. (2008). Introduction to genetic algorithms. *Introduction to Genetic Algorithms*, 1–442. https://doi.org/10.1007/978-3-540-73190-0/COVER

4) Selection

a) Definisi dan Fungsi Operator Selection

Selection adalah proses memilih individu dari populasi untuk dijadikan *parent* dalam proses *crossover*. Proses ini menentukan individu mana yang akan menghasilkan *offspring* pada generasi berikutnya dan berapa banyak *offspring* yang akan dihasilkan.

Tujuan utama *selection* adalah memberi peluang lebih besar kepada individu dengan *fitness* lebih tinggi untuk dipilih, dengan harapan *offspring* yang dihasilkan juga memiliki *fitness* yang lebih tinggi. Dengan demikian, *selection* berfungsi sebagai mekanisme yang mendorong Algoritma Genetika menuju solusi optimal melalui peningkatan kualitas populasi dari generasi ke generasi.



Gambar 4.1 Menunjukkan proses dasar *selection*

Selection pressure adalah ukuran seberapa besar individu dengan *fitness* lebih tinggi diunggulkan untuk dipilih dibanding individu lain. Semakin tinggi *selection pressure*, semakin besar peluang individu terbaik terpilih. *Pressure* ini berperan penting karena mendorong Algoritma Genetika untuk terus meningkatkan rata-rata *fitness* populasi dari satu generasi ke generasi berikutnya.

Besarnya *selection pressure* sangat mempengaruhi *convergence rate* Algoritma Genetika. Jika *pressure* tinggi, Algoritma Genetika akan lebih cepat mencapai solusi, tetapi ada risiko berhenti terlalu cepat pada solusi yang salah (*premature convergence*). Sebaliknya, jika *pressure* terlalu rendah, proses evolusi menjadi lambat karena individu terbaik tidak cukup banyak disebarkan.

Oleh karena itu, diperlukan keseimbangan. *Selection* harus cukup kuat untuk mempercepat pencarian solusi, tetapi juga tetap menjaga *population diversity* agar variasi dalam populasi tidak hilang. Dengan cara ini, Algoritma Genetika dapat menghindari *premature convergence* dan tetap memiliki peluang menemukan solusi yang optimal atau mendekati optimal.

b) Jenis-Jenis Operator Selection

1. Fitness Proportionate Selection (FPS)
    - Roulette Wheel Selection
    - Baker's SUS (Stochastic Universal Sampling)
2. Rank-Based Selection
3. Tournament Selection

c) Fitness Proportionate Selection (FPS)

Algoritma Genetika yang dikembangkan oleh Holland menggunakan *Fitness Proportionate Selection* (FPS), di mana *expected value* dari sebuah individu (yaitu jumlah harapan berapa kali individu tersebut akan terpilih untuk reproduksi) dihitung sebagai *fitness* individu tersebut dibagi dengan rata-rata *fitness* populasi.

Dalam metode ini, setiap individu dapat terpilih menjadi *parent* dengan probabilitas yang sebanding dengan nilai *fitness*-nya. Oleh karena itu, individu dengan *fitness* lebih tinggi memiliki peluang lebih besar untuk melakukan reproduksi dan menyebarkan karakteristiknya ke generasi berikutnya. Dengan demikian, metode ini memberikan selection *pressure* pada individu-individu yang lebih *fit* dalam populasi, sehingga mendorong evolusi menuju individu yang lebih baik seiring waktu.

1. Roulette Wheel Selection

    Selection schema paling sederhana adalah *roulette-wheel selection*, yang juga disebut *stochastic sampling with replacement*. Ini merupakan algoritma stokastik dan melibatkan teknik berikut:

    Individu dipetakan ke segmen-segmen berurutan pada sebuah garis, di mana ukuran setiap segmen sama besar dengan nilai *fitness* individu tersebut. Sebuah bilangan acak dihasilkan, dan individu yang segmennya mencakup bilangan acak tersebut dipilih. Proses ini diulangi sampai jumlah individu yang diinginkan tercapai yang disebut *mating population*. Teknik ini dianalogikan seperti *roulette wheel*, di mana setiap irisan sebanding ukurannya dengan nilai fitness.

Number of Individual	1	2	3	4	5	6	7	8	9	10	11
Fitness Value	2.0	1.8	1.6	1.4	1.2	1.0	0.8	0.6	0.4	0.2	0.0
Selection Probability	0.18	0.16	0.15	0.13	0.11	0.09	0.07	0.06	0.03	0.02	0.0

Tabel 4.1 *Selection probability* dan *fitness value*

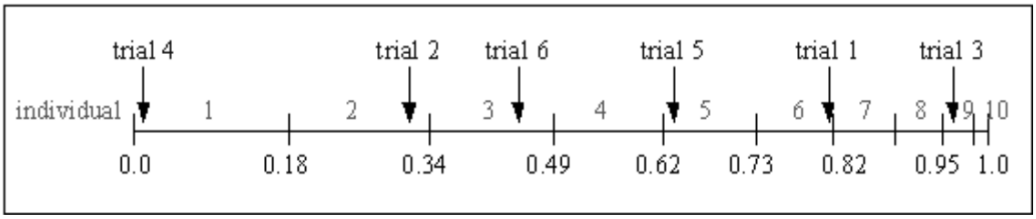
Tabel 4.1 menunjukkan probabilitas *selection* untuk 11 individu, *linear ranking* dengan *selective pressure* sebesar 2, bersamaan dengan nilai *fitness*-nya. Individu 1 adalah individu dengan *fitness* tertinggi dan menempati

interval terbesar, sedangkan individu 10 sebagai individu dengan *fitness* terendah kedua memiliki interval terkecil pada garis. Individu 11, dengan *fitness* paling rendah, memiliki nilai fitness = 0 dan tidak mendapat kesempatan untuk reproduksi.

Untuk memilih *mating population*, sejumlah bilangan acak yang terdistribusi uniformly (terdistribusi seragam antara 0.0 dan 1.0) dihasilkan secara independen.
Sampel dari 6 bilangan acak:

```
0.81, 0.32, 0.96, 0.01, 0.65, 0.42.
```

Gambar 4.2 menunjukkan proses *selection* individu untuk contoh pada tabel 4.1, beserta dengan *sample trial* di atas.



Gambar 4.2 Roulette-wheel selection

Setelah *selection*, *mating pool* terdiri dari individu:

```
1, 2, 3, 5, 6, 9.
```

Kekurangan pada *roulette-wheel selection* meskipun memberikan *zero bias*, tetapi tidak menjamin *minimum spread*.

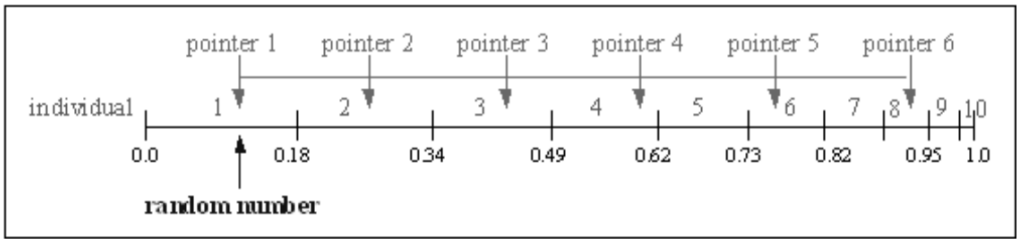
2. Baker's SUS (Stochastic Universal Sampling)

   Stochastic Universal Sampling (SUS) memberikan *zero bias* dan *minimum spread*. Individu dipetakan ke segmen-segmen berurutan pada sebuah garis, di mana ukuran setiap segmen sama dengan nilai *fitness*-nya, persis seperti pada *roulette-wheel selection*. Pada metode ini, *pointers* yang berjarak sama ditempatkan di atas garis sebanyak jumlah individu yang akan dipilih. Misalkan, **NPointer** adalah jumlah individu yang akan dipilih, maka jarak antar pointer adalah **1/NPointer**, dan posisi *pointer* pertama ditentukan oleh bilangan acak yang dihasilkan dalam rentang **[0, 1/NPointer]**.

   Sebagai contoh, untuk memilih 6 individu, jarak antar *pointer* adalah 1/6 = 0,167. Gambar 4.3 menunjukkan proses *selection* untuk contoh di atas.

   Sampel bilangan acak tunggal dalam rentang [0, 0,167]:

   ```
   0.1.
   ```



Gambar 4.3 *Stochastic universal sampling*

Setelah *selection*, *mating pool* terdiri dari individu:

```
1, 2, 3, 4, 6, 8.
```

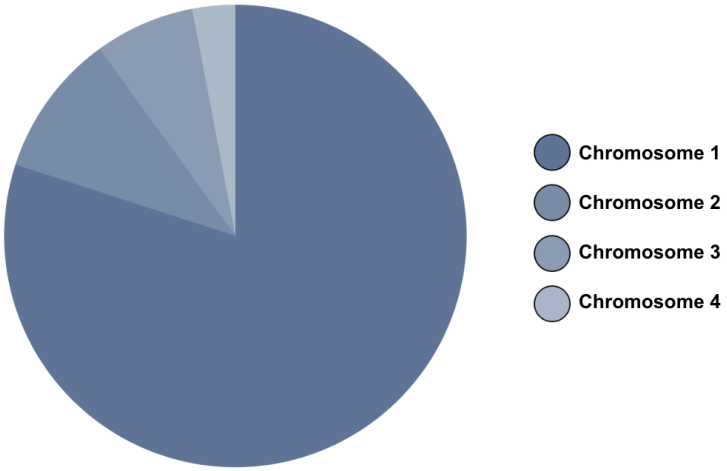
Stochastic universal sampling memastikan pemilihan *offspring* yang lebih mendekati nilai yang seharusnya dibandingkan dengan *roulette-wheel selection*.

d) Rank-Based Selection

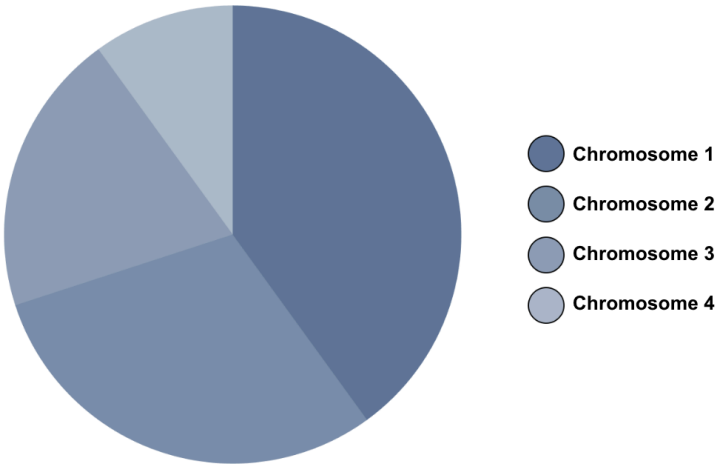
Ranked-Based Selection memperkenalkan pendekatan berbeda pada selection dari Algoritma Genetika. Alih-alih langsung menggunakan nilai *fitness* untuk menentukan probabilitas seleksi, individu dalam populasi terlebih dahulu diurutkan (*ranking*) berdasarkan nilai *fitness*-nya, kemudian diberikan *rank* pada masing-masing individu. Probabilitas seleksi kemudian dihitung berdasarkan *rank* tersebut, bukan nilai *fitness* aktual.

Pendekatan *rank-based* ini membantu mengurangi masalah yang terkait dengan seleksi langsung berbasis *fitness*, seperti *premature convergence* dan dominasi beberapa individu yang sangat *fit* pada tahap awal proses optimasi. Dengan memberikan *rank* dan menggunakannya untuk seleksi, *Ranked-Based Selection* menyediakan *selection pressure* yang lebih seimbang dan terkontrol, memungkinkan eksplorasi ruang pencarian yang lebih baik serta menjaga keragaman (*diversity*) dalam populasi.

Ranking biasanya diberikan secara linear atau eksponensial, di mana individu terbaik menerima *rank* tertinggi dan individu terburuk menerima *rank* terendah. Probabilitas seleksi kemudian dihitung berdasarkan *ranking* tersebut dengan menggunakan formula atau *mapping function* yang telah ditentukan. *Mapping function* ini dapat disesuaikan untuk mengontrol *selection pressure*, di mana *pressure* lebih tinggi akan lebih menguntungkan individu dengan *rank* tertinggi, sedangkan *pressure* lebih rendah memberikan distribusi probabilitas seleksi yang lebih merata.



Situasi Sebelum Ranking (*Graph of Fitness*)



Situasi Setelah Ranking (*Graph of Order Numbers*)

Gambar 4.4 Menunjukkan bagaimana situasi berubah setelah merubah *fitness* ke order *number*

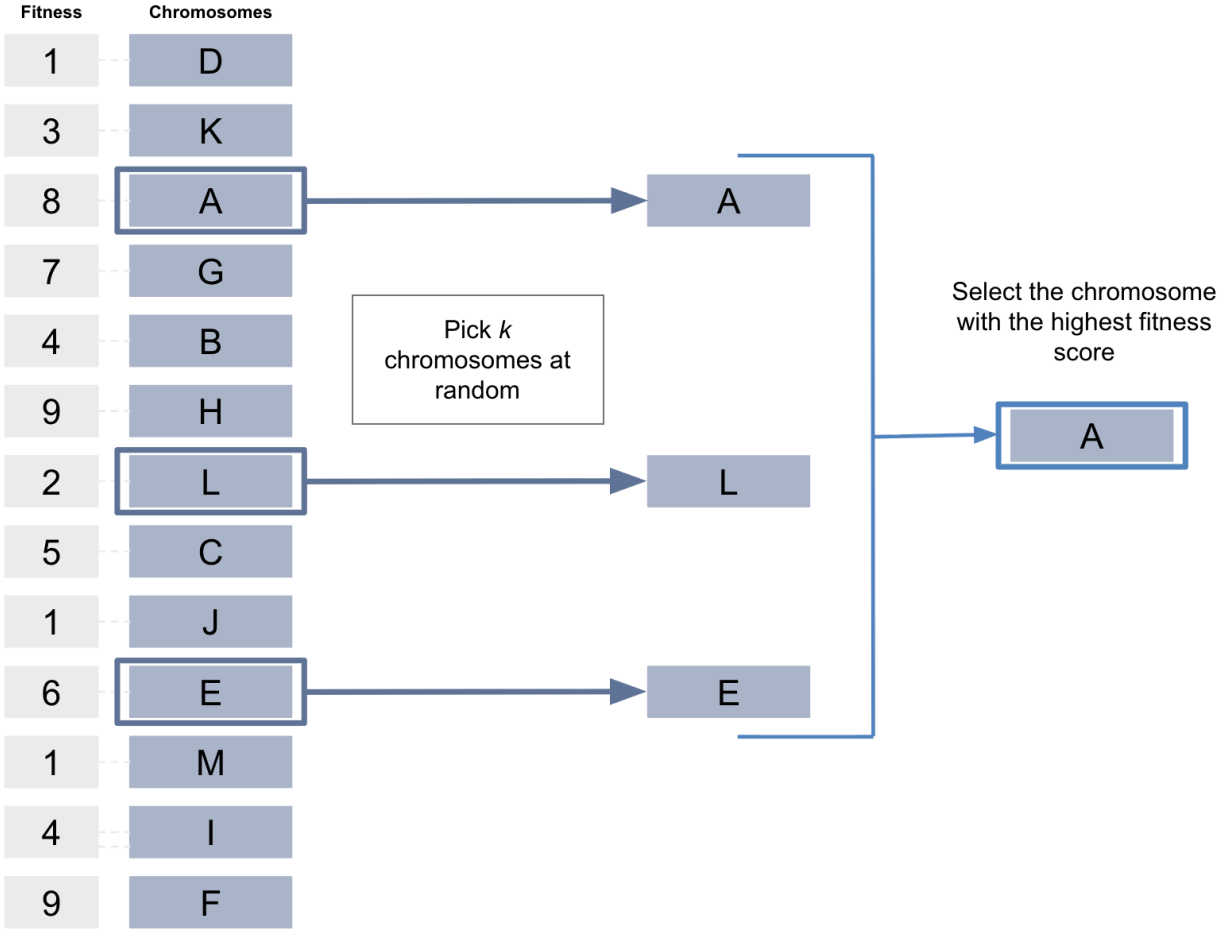
e) Tournament Selection

Tournament selection adalah mekanisme *selection* dalam yang kuat dan banyak digunakan dalam Algoritma Genetika karena mampu menjaga keseimbangan antara *diversity maintenance* dan *selective pressure*. Berbeda dengan *roulette-wheel selection* yang secara langsung bergantung pada *fitness* individu relatif terhadap total *fitness* populasi, *tournament selection* bekerja dengan cara mengadakan “turnamen” di antara subset individu, dan pemenang dari setiap turnamen dipilih untuk reproduksi.

Konsep utama *tournament selection* cukup sederhana, alih-alih mempertimbangkan seluruh populasi sekaligus, subset individu dipilih secara acak untuk saling berkompetisi. Individu dengan *fitness* tertinggi dalam “turnamen” tersebut

kemudian dipilih. Proses ini diulang hingga jumlah individu yang diinginkan untuk reproduksi tercapai.

Metode ini memiliki beberapa keunggulan, yaitu *tournament selection* tetap menjaga *diversity* karena individu dengan *fitness* rendah masih memiliki kesempatan untuk ikut serta dalam turnamen. Selain itu, metode ini memungkinkan *selective pressure* disesuaikan dengan mengatur ukuran turnamen.



Gambar 4.5 Tournament selection

Mekanisme Tournament Selection:

1. Tentukan ukuran turnamen (k), yaitu jumlah individu yang ikut dalam setiap turnamen.
2. Pilih k individu secara acak dari populasi.
3. Bandingkan nilai *fitness* individu-individu tersebut dan pilih individu dengan *fitness* tertinggi sebagai pemenang.
4. Masukkan pemenang ke dalam *mating pool*.
5. Ulangi langkah 2–4 hingga jumlah individu yang diinginkan tercapai.

Referensi

Sivanandam, S. N., & Deepa, S. N. (2008). Introduction to genetic algorithms. *Introduction to Genetic Algorithms*, 1–442. https://doi.org/10.1007/978-3-540-73190-0

GEATbx: Documentation - Genetic and Evolutionary Algorithm Toolbox for MATLAB. (n.d.). Retrieved September 30, 2025, from http://www.geatbx.com/docu/index.html

Chapter 4 - Selection Strategies | *Algorithm Afternoon*. (n.d.). Retrieved September 30, 2025, from https://algorithmafternoon.com/books/genetic_algorithm/chapter04/

Selection - Introduction to Genetic Algorithms - Tutorial with Interactive Java Applets. (n.d.). Retrieved September 30, 2025, from https://www.obitko.com/tutorials/genetic-algorithms/selection.php

Ranked Selection Genetic Algorithm | *Algorithm Afternoon*. (n.d.). Retrieved September 30, 2025, from https://algorithmafternoon.com/genetic/ranked_selection_genetic_algorithm/

Tournament Selection in Genetic Algorithms | Baeldung on Computer Science. (n.d.). Retrieved September 30, 2025, from https://www.baeldung.com/cs/ga-tournament-selection

5) Crossover untuk Binary Chromosome

a) Definisi dan Fungsi Operator Crossover

Crossover adalah *genetic operator* yang digunakan untuk memvariasikan susunan kromosom dari satu generasi ke generasi berikutnya. Metode *crossover* yang digunakan bergantung pada Metode *Encoding* yang diterapkan.

Crossover berlangsung dalam tiga tahap yaitu:

1. *Reproduction operator* memilih secara acak sepasang *individual string* untuk proses *mating*.
2. Memilih *cross site* secara acak di sepanjang panjang *string*.
3. Nilai-nilai posisi setelah *cross site* tersebut ditukar antara kedua string untuk membentuk *offspring* baru.

b) Jenis-Jenis Operator Crossover

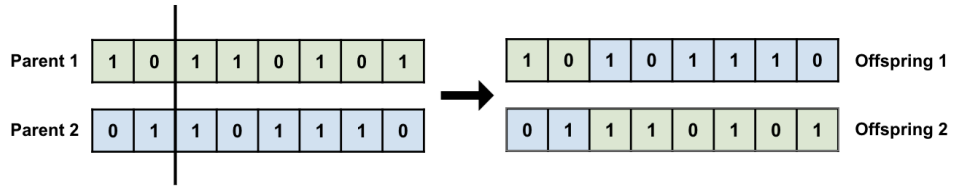
- *Crossover for Binary Chromosome*
- *Crossover for Integer Chromosome*
- *Crossover for Real Chromosome*
- *Crossover for Permutation Chromosome*
- *Crossover for Integer Chromosome*
- *Crossover Path Relinking*
- *Crossover Multi-parent*

Crossover for Binary Chromosome

Pada *binary chromosome representation*, setiap individu dalam populasi direpresentasikan sebagai rangkaian bit (0 dan 1) yang menyatakan solusi potensial terhadap suatu permasalahan.

1. Single-point Crossover

Single-point crossover merupakan bentuk paling sederhana dari *crossover*. Satu posisi *crossover* dipilih secara acak, kemudian bagian dari dua parent setelah posisi tersebut ditukarkan untuk membentuk dua *offspring* baru. *String* yang dihasilkan dari proses ini memiliki karakteristik *Positional Bias*.

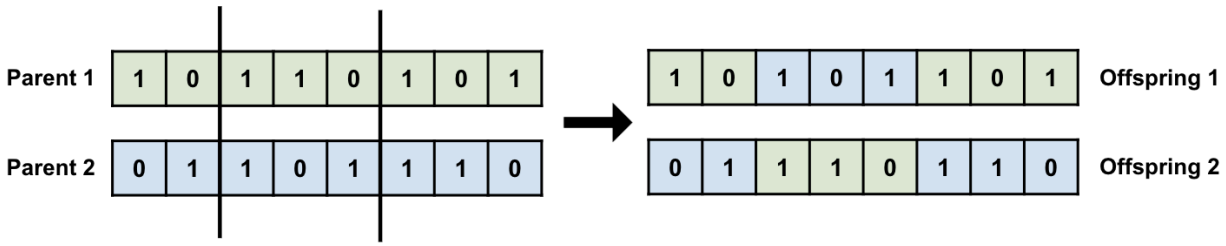


Gambar 5.1 Single-Point Crossover

2. Multi-point Crossover

Multi-point crossover memiliki mekanisme seperti *single-point crossover*, yang berbeda adalah jumlah posisi yang dipilih secara acak. Pada *Multi-point crossover*, sejumlah N posisi dipilih secara acak sepanjang panjang kromosom. Posisi-posisi tersebut ditukarkan untuk membentuk dua *offspring* baru.

Tujuan dari *multi-point crossover* antara lain meningkatkan keragaman genetik dalam populasi, mengurangi positional bias, dan meningkatkan peluang *recombination* dari berbagai *schemas* atau blok solusi yang berbeda. Namun, penggunaan terlalu banyak titik potong dapat menjadi terlalu disruptif, karena dapat merusak kombinasi gen yang sudah baik (*coadapted alleles*).

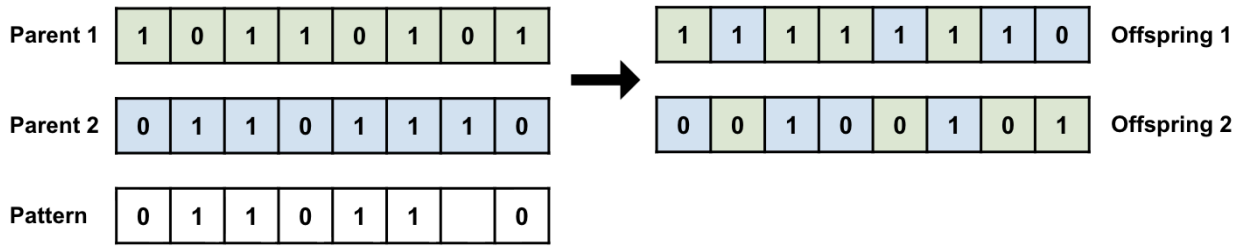


Gambar 5.2 Multi-point Crossover

3. Uniform Crossover

Pada *uniform crossover*, Setiap gen (bit) dipilih secara acak dari salah satu gen yang bersesuaian pada kromosom *parent*. Proses pemilihan ini dilakukan secara independen untuk setiap posisi gen. Metode ini dapat dianalogikan dengan melempar koin (*tossing a coin*). Jika hasilnya “head”, gen diambil dari *parent* 1; jika hasilnya “tail”, gen diambil dari *parent* 2.

Metode ini memberikan peluang yang sama bagi setiap gen untuk diwarisi dari salah satu parent, sehingga meningkatkan keragaman genetik (*genetic diversity*) dan menghilangkan *positional bias* yang biasanya muncul pada *single-point crossover* atau *multi-point crossover*.



Gambar 5.3 Uniform Crossover

/Referensi

Crossover in Genetic Algorithm - GeeksforGeeks. (n.d.). Retrieved November 3, 2025, from https://www.geeksforgeeks.org/machine-learning/crossover-in-genetic-algorithm/

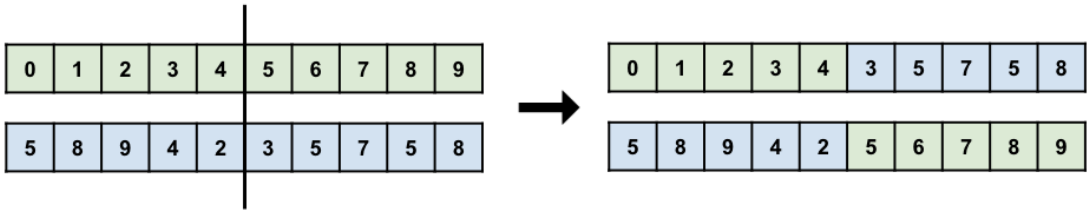
6) Crossover untuk Integer Chromosome

Crossover for Integer Chromosome

Berbeda dengan binary chromosome yang menggunakan bit 0 dan 1, representasi integer lebih sesuai untuk permasalahan yang melibatkan parameter diskrit atau nilai numerik yang memiliki makna kuantitatif, seperti penjadwalan, pengurutan, atau optimasi kombinatorial.

1. Single-Point Crossover

Single-Point Crossover merupakan bentuk paling sederhana dari *crossover*. Satu posisi *crossover* dipilih secara acak, kemudian bagian dari dua parent setelah posisi tersebut ditukarkan untuk membentuk dua *offspring* baru. *String* yang dihasilkan dari proses ini memiliki karakteristik *Positional Bias*.

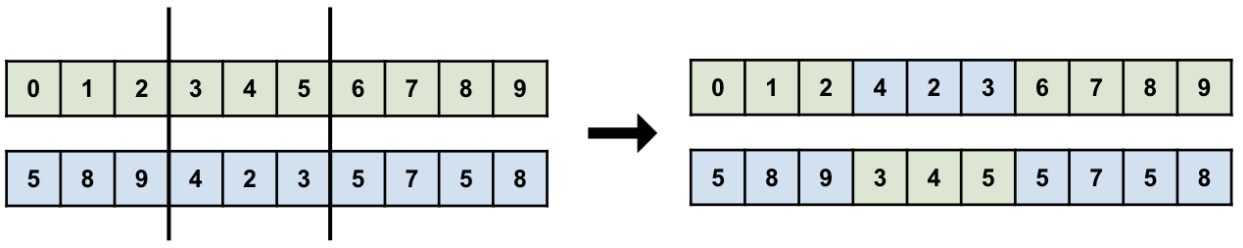


Gambar 6.1 Single-Point Crossover

2. Multi-point Crossover

Multi-point crossover memiliki mekanisme seperti *single-point crossover*, yang berbeda adalah jumlah posisi yang dipilih secara acak. Pada *Multi-point crossover*, sejumlah N posisi dipilih secara acak sepanjang panjang kromosom. Posisi-posisi tersebut ditukarkan untuk membentuk dua *offspring* baru.

Tujuan dari *multi-point crossover* antara lain meningkatkan keragaman genetik dalam populasi, mengurangi positional bias, dan meningkatkan peluang *recombination* dari berbagai *schemas* atau blok solusi yang berbeda. Namun, penggunaan terlalu banyak titik potong dapat menjadi terlalu disruptif, karena dapat merusak kombinasi gen yang sudah baik (*coadapted alleles*).



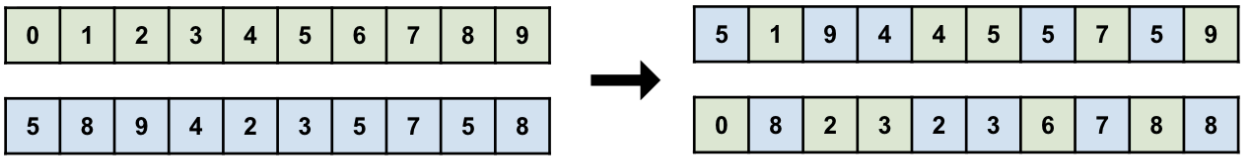
Gambar 6.2 Multi-point Crossover

3. Uniform Crossover

Pada *uniform crossover*, Setiap gen (bit) dipilih secara acak dari salah satu gen yang bersesuaian pada kromosom *parent*. Proses pemilihan ini dilakukan secara

independen untuk setiap posisi gen. Metode ini dapat dianalogikan dengan melempar koin (*tossing a coin*). Jika hasilnya “head”, gen diambil dari *parent* 1; jika hasilnya “tail”, gen diambil dari *parent* 2.

Metode ini memberikan peluang yang sama bagi setiap gen untuk diwarisi dari salah satu parent, sehingga meningkatkan keragaman genetik (*genetic diversity*) dan menghilangkan *positional bias* yang biasanya muncul pada *single-point crossover* atau *multi-point crossover*.



Gambar 6.3 Uniform Crossover

Referensi

Genetic Algorithms - Crossover. (n.d.). Retrieved November 3, 2025, from https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

7) Crossover untuk Real Chromosome

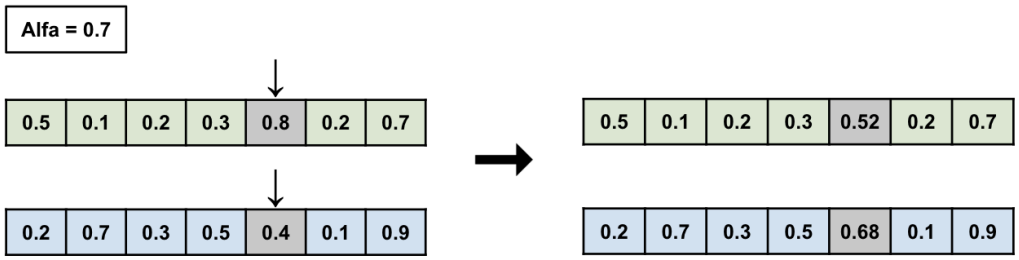
Crossover for Real Chromosome

Crossover pada *real chromosome* merupakan proses rekombinasi genetik dalam Algoritma Genetika yang diterapkan pada kromosom yang direpresentasikan dalam bentuk bilangan real (*floating-point representation*). Representasi ini umum digunakan untuk menyelesaikan permasalahan optimasi kontinu, di mana variabel keputusan memiliki nilai dalam rentang tertentu dan tidak terbatas pada bilangan bulat atau biner.

Berbeda dengan *crossover* pada *binary* atau *integer chromosome*, mekanisme crossover untuk *real chromosome* melibatkan operasi aritmetika terhadap nilai gen antar *parent*. Metode tersebut memungkinkan penciptaan *offspring* dengan nilai gen yang berada di antara atau sekitar nilai gen *parent*, sehingga menjaga kontinuitas dan stabilitas proses evolusi.

1. Single Arithmetic Crossover

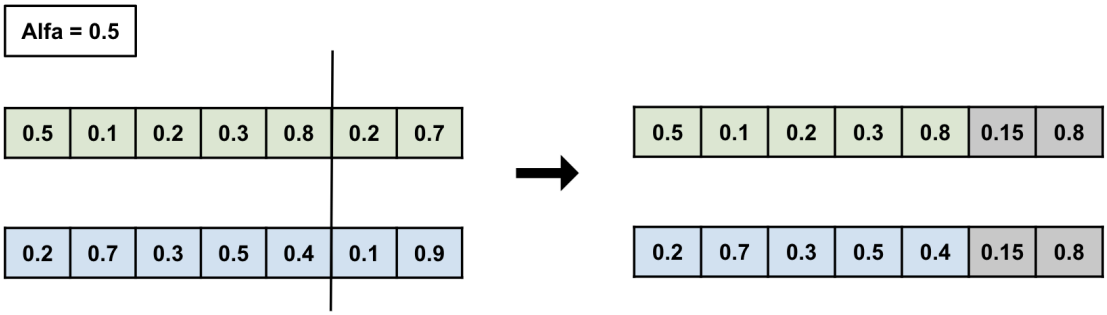
1) Dua *parent* direpresentasikan sebagai:
   Parent 1: $< x_1, \ldots, x_n >$
   Parent 2: $< y_1, \ldots, y_n >$
2) Pilih secara acak satu gen (k) yang akan dikenai operasi *crossover*
3) Hasilnya adalah dua *offspring* yang dibentuk berdasarkan kombinasi linier dari gen ke-k *parent* tersebut dengan parameter pengendali α, di mana $0 \leq \alpha \leq 1$:
   Offspring 1: $< x_1, \ldots, x_{k-1}, \alpha \cdot y_k + (1 - \alpha) \cdot x_k, \ldots, x_n >$
   Offspring 2: $< y_1, \ldots, y_{k-1}, \alpha \cdot x_k + (1 - \alpha) \cdot y_k, \ldots, y_n >$



Gambar 7.1 Single Arithmetic Crossover

2. Simple Arithmetic Crossover

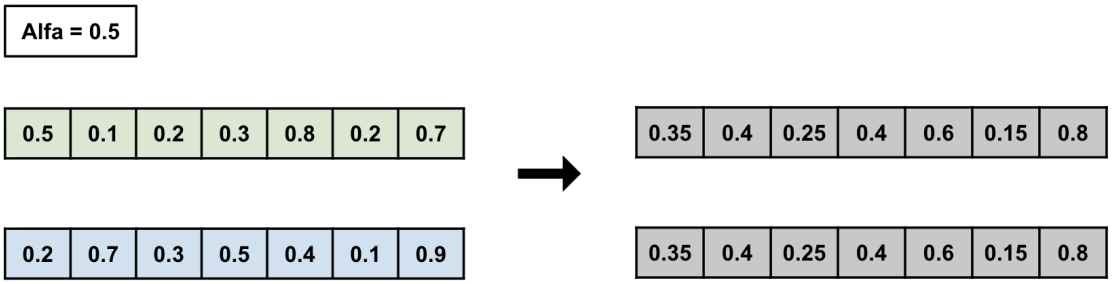
1) Dua *parent* direpresentasikan sebagai:
   Parent 1: $< x_1, \ldots, x_n >$
   Parent 2: $< y_1, \ldots, y_n >$
2) Pilih secara acak satu gen (k) yang akan menjadi titik pembatas *crossover*
3) Hasilnya adalah dua *offspring* yang dibentuk berdasarkan kombinasi linier dari gen ke-$k + 1$ hingga gen k-n dengan parameter pengendali α, di mana $0 \leq \alpha \leq 1$:
   Offspring 1: $< x_1, \ldots, x_k, \alpha \cdot y_{k+1} + (1 - \alpha) \cdot x_{k+1}, \ldots, \alpha \cdot y_n + (1 - \alpha) \cdot x_n >$
   Offspring 2: $< y_1, \ldots, y_k, \alpha \cdot x_{k+1} + (1 - \alpha) \cdot y_{k+1}, \ldots, \alpha \cdot x_n + (1 - \alpha) \cdot y_n >$



Gambar 7.2 Simple Arithmetic Crossover

3. Whole Arithmetic Crossover

1) Dua *parent* direpresentasikan sebagai:
   Parent 1: $< x_1, \ldots, x_n >$
   Parent 2: $< y_1, \ldots, y_n >$
2) Untuk setiap gen ke-i (i = 1, 2, …, n), *offspring* dibentuk dengan kombinasi linier dari gen pada kedua *parent* dengan parameter pengendali α, di mana 0 ≤ α ≤ 1:
   Offspring 1: $z_i^1 = \alpha \cdot y_i + (1 - \alpha) \cdot x_i$
   Offspring 2: $z_i^2 = \alpha \cdot x_i + (1 - \alpha) \cdot y_i$



Gambar 7.3 Whole Arithmetic Crossover

Referensi

Course Material Week 4 - Crossover

8) Crossover untuk Permutation Chromosome

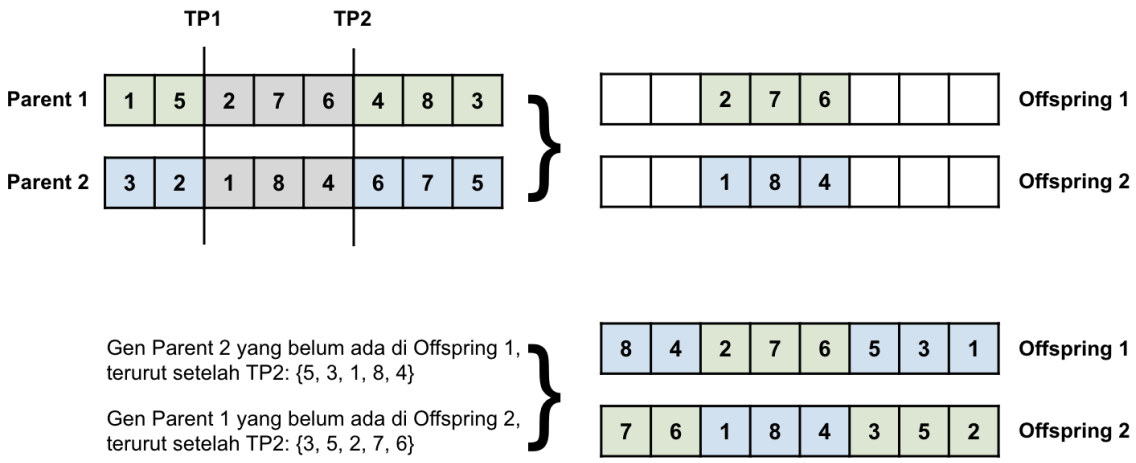
Crossover for Permutation Chromosome

Crossover pada *permutation chromosome* merupakan proses rekombinasi genetik dalam Algoritma Genetika yang diterapkan pada kromosom yang direpresentasikan sebagai permutasi, yaitu urutan elemen yang masing-masing harus muncul tepat satu kali. Representasi ini umum digunakan pada permasalahan optimasi kombinatorial seperti Travelling Salesman Problem (TSP), job scheduling, dan routing, di mana solusi berupa sebuah urutan yang valid.

Berbeda dengan crossover pada binary, integer, atau real chromosome, mekanisme crossover untuk permutation chromosome tidak dapat secara langsung menukar substring antar parent, karena tindakan tersebut berpotensi merusak sifat permutasi, seperti munculnya elemen duplikat atau hilangnya elemen tertentu. Oleh karena itu, berbagai metode crossover khusus dikembangkan untuk memastikan bahwa offspring tetap mempertahankan struktur permutasi sambil mewarisi sebanyak mungkin informasi penting dari parent, terutama pola urutan dan hubungan antar elemen.

1. Order Crossover

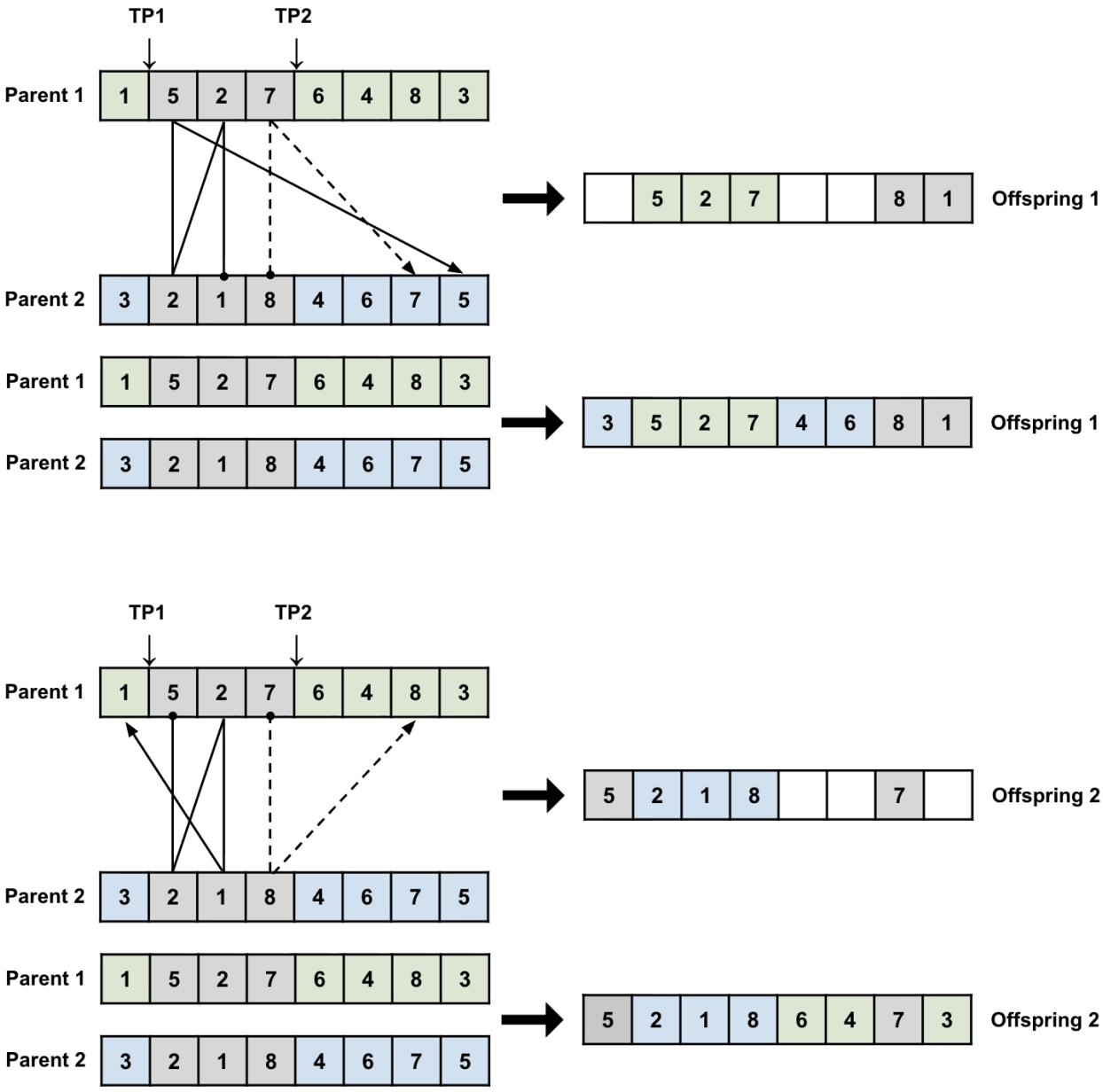
1) Pilih dua titik potong (TP1 dan TP2) pada kromosom.
   Dua posisi dalam kromosom dipilih sebagai batas segmen. Segmen gen di antara TP1 dan TP2 dari masing-masing parent disalin langsung ke posisi yang sama pada offspring.
2) Salin segmen dari parent ke offspring.
   - Pada Offspring 1, segmen dari Parent 1 ditempatkan pada posisi yang sama.
   - Pada Offspring 2, segmen dari Parent 2 ditempatkan pada posisi yang sama.
3) Mengisi sisa posisi dengan urutan parent lain.
   - Pengisian dilakukan dengan menempatkan gen-gen dari Parent 2 yang belum muncul pada segmen Offspring 1, dimulai dari posisi setelah TP2 dan bergerak melingkar sesuai urutan kemunculan gen pada Parent 2.
   - Pengisian dilakukan dengan menempatkan gen-gen dari Parent 1 yang belum muncul pada segmen Offspring 2, dimulai dari posisi setelah TP2 dan bergerak melingkar sesuai urutan kemunculan gen pada Parent 1.



Gambar 8.1 Order Crossover

2. Partially Mapped Crossover

1) Pilih dua titik potong (TP1 dan TP2) pada kromosom.
   Dua posisi dipilih secara acak sebagai batas segmen. Segmen gen di antara TP1 dan TP2 dari Parent 1 disalin langsung ke posisi yang sama pada Offspring 1.
2) Identifikasi elemen dari Parent 2 yang berada dalam segmen namun belum tersalin ke Offspring 1.
   Mulai dari TP1, periksa gen-gen pada segmen Parent 2. Setiap gen dalam segmen Parent 2 yang belum muncul dalam segmen Offspring 1 dicatat sebagai elemen yang perlu dipetakan.
3) Untuk setiap elemen tersebut (misal i), tentukan elemen pasangan dari Parent 1 (misal j).
   Pasangan i adalah gen Parent 1 yang berada pada posisi yang sama di dalam segmen TP1–TP2. Elemen j ini sudah berada dalam Offspring 1 sehingga posisinya tidak boleh ditempati lagi.
4) Tempatkan i ke posisi tempat j berada dalam Parent 2.
   Posisi j pada Parent 2 adalah posisi target untuk menempatkan i di Offspring 1, karena j sudah digunakan dalam segmen.
5) Jika posisi tersebut sudah terisi oleh elemen lain (misal k), lakukan pemetaan ulang.
   Jika posisi target telah terisi, maka i harus ditempatkan pada posisi elemen k dalam Parent 2. Proses ini diulangi sampai ditemukan posisi kosong bagi i dalam Offspring 1.

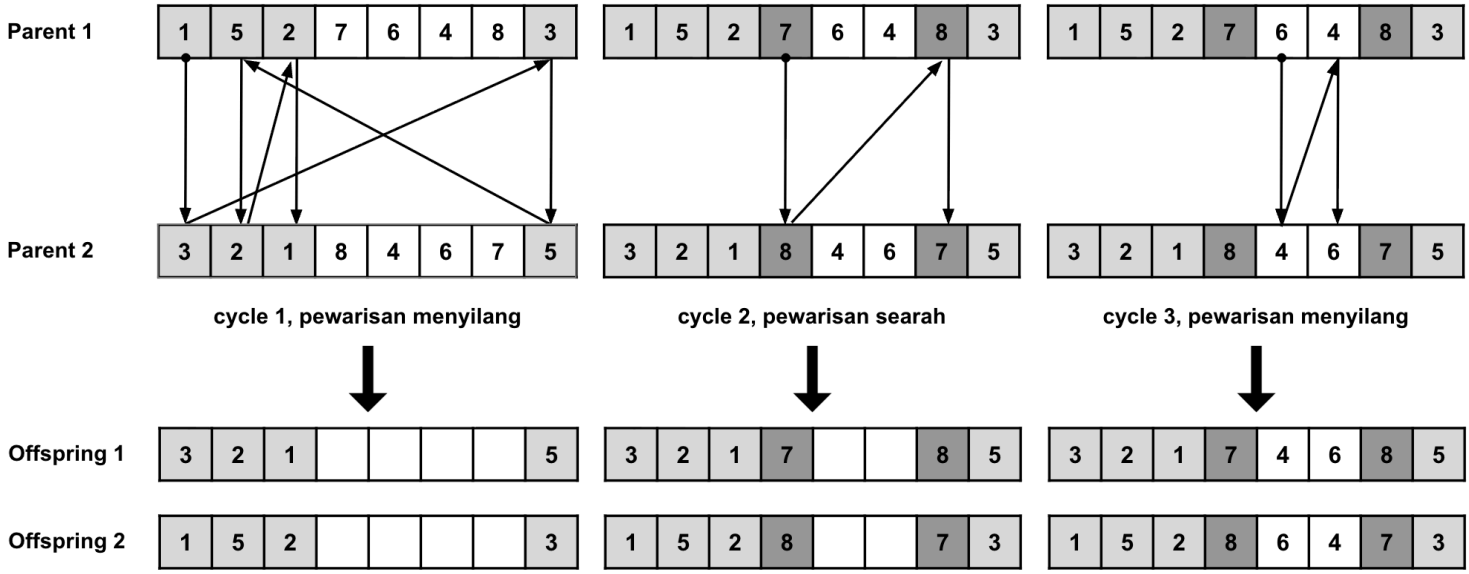


Gambar 8.2 Partially Mapped Crossover

3. Cycle Crossover

1) Mulai siklus dari posisi pertama yang belum digunakan.
   Pilih posisi pertama pada Parent 1 yang belum masuk siklus. Gen pada posisi ini menjadi elemen awal siklus.
2) Cari pasangan gen di Parent 2.
   Lihat gen yang berada di posisi yang sama pada Parent 2. Gen ini akan menunjukkan posisi berikutnya yang harus dimasukkan ke dalam siklus.

3) Tentukan posisi gen tersebut di Parent 1.
   Temukan posisi gen yang sama di Parent 1. Tambahkan gen ini ke siklus yang sedang dibuat.
4) Ulangi langkah 2–3 hingga siklus selesai.
   Proses dilanjutkan hingga kembali ke gen awal dari Parent 1, menandai satu siklus selesai.
5) Bentuk offspring dengan memilih siklus secara bergantian.
   Setelah semua siklus terbentuk, ambil siklus pertama dari Parent 1, siklus kedua dari Parent 2, dan seterusnya, hingga terbentuk Offspring baru.



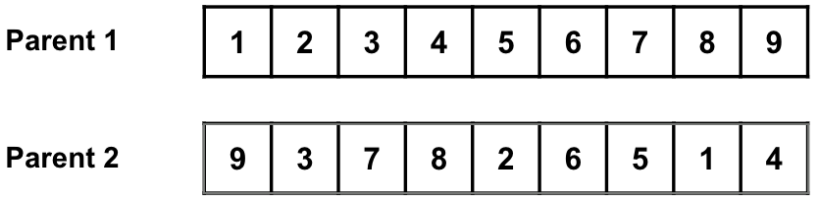
Gambar 8.3 Cycle Crossover

4. Edge Crossover

1) Buat edge table (adjacency list).
   Untuk setiap gen, catat semua gen yang terhubung dengannya pada kedua parent. Jika edge muncul di kedua parent, beri tanda + (common edge).
2) Pilih gen awal secara acak.
   Masukkan gen ini ke offspring dan tetapkan sebagai current element.
3) Hapus referensi gen saat ini dari tabel.
   Setelah gen ditambahkan ke offspring, hapus semua kemunculannya dari daftar tetangga semua gen.
4) Pilih gen berikutnya.
   - Jika ada common edge, pilih gen itu sebagai next element.
   - Jika tidak ada common edge, pilih gen dari daftar tetangga yang memiliki daftar tetangga paling pendek.
   - Jika ada beberapa kandidat sama, pilih secara acak.
5) Tangani daftar kosong.

Jika daftar gen berikutnya kosong, periksa ujung offspring lainnya untuk diperluas; jika tidak memungkinkan, pilih gen baru secara acak. Hanya dalam kasus terakhir inilah foreign edge bisa muncul.

6) Lanjutkan hingga offspring lengkap.
   Proses diulang hingga semua gen terisi.



Gambar 8.4 Parent untuk Edge Crossover

Element	Edges
1	2, 5, 4, 9
2	1, 3, 6, 8
3	2, 4, 7, 9
4	1, 3, 5, 9
5	1, 4, 6+
6	2, 5+, 7
7	3, 6, 8+
8	2, 7+, 9
9	1, 3, 4, 8

Tabel 8.1 Edge Table

Choices	Element Selected	Reason	Partial Result
All	1	Random	[1]
2, 5, 4, 9	5	Shortest List	[1 5]
4, 6	6	Common Edge	[1 5 6]
2, 7	2	Random Choice	[1 5 6 2]
3, 8	8	Shortest List	[1 5 6 2 8]

7, 9	7	Common Edge	[1 5 6 2 8 7]
3	3	Only Item in List	[1 5 6 2 8 7 3]
4, 9	9	Random Choice	[1 5 6 2 8 7 3 9]
4	4	Last Element	[1 5 6 2 8 7 3 9 4]

Tabel 8.2 Permutation construction

Referensi

Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing*. https://doi.org/10.1007/978-3-662-44874-8

Course Material Week 4 - Crossover

9) Crossover Path Relinking

Crossover dengan Path Relinking

Misalkan suatu kondisi dimana kita mempunyai 2 solusi yang *locally-optimal.* Maka akan masuk akal, jika dilakukan suatu *local search* yang menyusuri jalur antara satu solusi ke solusi lainnya, hal itu akan menemukan solusi-solusi baru, karena titik optimum lokal cenderung untuk ditemui di sekitar titik optimum lokal lainnya. Kalaupun pada jalur sedemikian rupa tidak ditemukan solusi yang lebih baik, setidaknya kita telah memperoleh sejumlah pemahaman tentang daerah-daerah berkumpulnya titik-titik optimum lokal tersebut pada 1 dimensi. Karena itulah, path relinking menjadi salah satu opsi yang dapat kita lakukan pada crossover

Path relinking digunakan untuk menciptakan solusi baru dengan menghubungkan jalur (*path*) dari 2 solusi berkualitas tinggi (2 *parent solutions*). Path relinking umumnya terdiri dari 2 langkah. Tahap pertama adalah membangun jalur yang menghubungkan kedua *parent solution*. Solusi pada awal dan akhir jalur tersebut masing-masing disebut *initial solution* dan *guiding solution*, sedangkan solusi-solusi lainnya di tengah jalur antara kedua solusi tersebut disebut *intermediate solution*. Langkah satunya adalah memilih solusi dari jalur yang telah dibangun sebagai referensi untuk peningkatan lebih lanjut.

Normalnya, parent dengan fitness lebih baiklah yang akan menjadi *guiding parent/solution*. Lalu, *initial solution* akan menjadi nilai awal dari *intermediate solution*. Cara kerja path relinking itu sendiri adalah dengan menelusuri *intermediate solution* dari depan ke belakang, dan membandingkan posisi yang sesuai antara *intermediate solution* dan *guiding solution*. Jika mereka berbeda, temukan posisi gen *intermediate solution* terkait pada *guiding solution* dan tukar gen *intermediate solution* pada kedua posisi itu. Jike mereka sama, lanjutkan menelusuri *intermediate solution* hingga mencapai akhirnya. *Initial solution* apapun dapat dibawa ke *guiding solution* dalam jumlah pertukaran tertentu (kurang dari atau sama dengan panjang kromosom).

Pada penelusuran ini, kita akan mencatat solusi dengan fitness terbaik, dengan nilai awal berupa *initial parent/solution*. Pada tiap iterasinya, kita hitung fitness dari *intermediate solution* yang dihasilkan. Jika lebih baik dari solusi-solusi sebelumnya, *overwrite* variabel solusi terbaik dengan *intermediate solution* baru itu. Versi lain mengatakan kita mencatat 2 solusi terbaik, dengan nilai awal berupa *initial parent* serta *guiding parent*. Ini dapat mengantisipasi kemungkinan munculnya *intermediate solution* dengan fitness yang lebih baik dari *guiding solution.* Berikut ini adalah langkah-langkah mendetail dari proses path relinking:

Procedure: PR operation
Input: the initial solution and the guiding solution
Output: updated offspring solution
Step 1. Produce the intermediate solutions through exchange
Step 2. Check each intermediate solution,
If the intermediate solution is the same with the *guide*,
Go to Step 3;
Else
If the intermediate solution is infeasible,
Go to Step 2;
Else
If the intermediate solution's objective is better than the initial solution,
Save the intermediate solution as the new solution.
Else
If there is no new solution for current intermediate solution yet,
Save the intermediate solution as the new solution with a probability of 0.5
Step 3. The exchange is over, and the new solution is returned.

Fig. 9.1 Pseudocode path relinking

Ada pula versi yang menjelaskan langkah-langkah secara berkebalikan, yaitu pada tiap iterasi, temukan posisi gen *guiding solution* terkait pada *intermediate solution* dan tukar gen pada kedua posisi itu. Versi manapun yang digunakan, pada akhir penelusuran, dengan sejumlah pertukaran, *intermediate solution* akan menjadi sama dengan *guiding solution*. Berikut adalah contoh *step-by-step* dari path relinking:

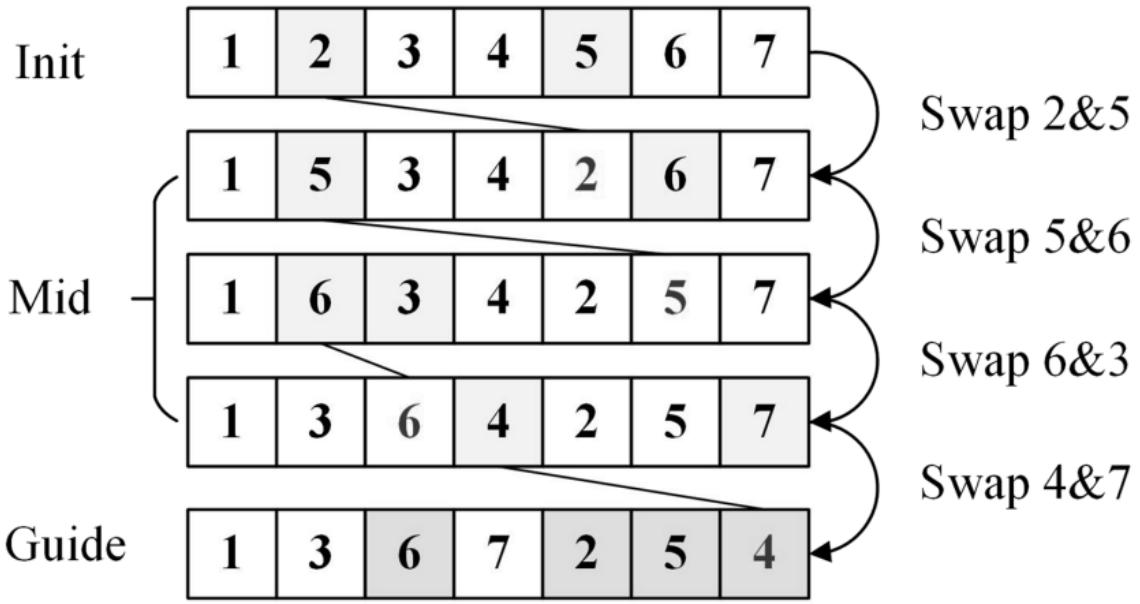


Fig. 9.2 Contoh path relinking

Referensi

Reeves, Colin & Yamada, Takeshi. (1998). Genetic Algorithms, Path Relinking and the Flowshop Sequencing Problem. Evolutionary Computation. 6. 10.1162/evco.1998.6.1.45.

Dou, J., Wang, S., Zhang, C. *et al.* A genetic algorithm with path-relinking for operation sequencing in CAPP. *Int J Adv Manuf Technol* 125, 3667–3690 (2023). https://doi.org/10.1007/s00170-023-10907-2

10) Crossover Multi-parent

Crossover Multi-parent

Operator crossover adalah inti dari algoritma genetika, yang digunakan untuk mengkombinasikan dua solusi dan melakukan pertukaran informasi antara mereka. Karena itu, input dari operator crossover adalah dua solusi, dan outputnya adalah dua offspring. Meskipun crossover dengan dua parent seperti itu telah membuktikan hasil yang sukses, penelitian dan riset pada akhir-akhir ini telah menunjukkan bahwa efisiensi algoritma genetika telah meningkat dengan menggunakan multi-parent crossover alih-alih dua parent.

Banyak percobaan akhir-akhir ini yang mencoba penggunaan multi-parent pada operator kombinasi, tidak hanya dua, untuk menciptakan offspring baru. Ide utama dari penggunaan multi-parent pada proses kombinasi adalah untuk meningkatkan informasi yang ditukar dan lebih mendorong diversifikasi dibanding dua parent. Menggunakan lebih dari dua parent dapat meningkatkan proses intensifikasi dengan memanfaatkan informasi yang sama-sama dimiliki oleh beberapa parent sekaligus.

Adanya parent tambahan di dalam proses crossover akan mengurangi tekanan pada proses selection dan meningkatkan keragaman pada populasi. Semakin tinggi tingkat keragaman dalam suatu populasi, maka kemungkinan terjadinya mating antara gen yang mirip atau berkaitan dekat juga akan semakin mengecil. Karena itulah digunakan mekanisme crossover multi-parent yang terdiri dari tiga atau lebih parent yang dihasilkan dari proses selection sebelumnya.

Ada dua karakteristik utama multi-parent crossover yang berbeda dibanding crossover biasa: *scanning order* dan *competition rule*. Scanning order memilih suatu gen dari gen-gen kandidat berdasarkan urutan dari tiap kromosom. Competition rule adalah suatu aturan yang bersifat *pre-determined*, yang mana memungkinkan offspring untuk mewarisi gen dari pilihan parent yang ada. Seperti crossover dua parent pada kebanyakan algoritma genetika, ada sangat banyak operator multi-parent crossover (MPX) yang dapat dipakai. Operator-operator tersebut antara lain, yang akan dibahas disini, adalah Scanning-based Crossover (SBC), Adjacency-based Crossover (ABC), and Diagonal-based Crossover (DBC).

1. Scanning-based Crossover

If the relation of genes is isolated in a chromosome, we can go through the sequence of the chromosome from the beginning to the end, and take the value of every parents as candidates. Then the occurrence competition rule is applied as selection criteria. The process is continued in the same manner until scanning to the last position.

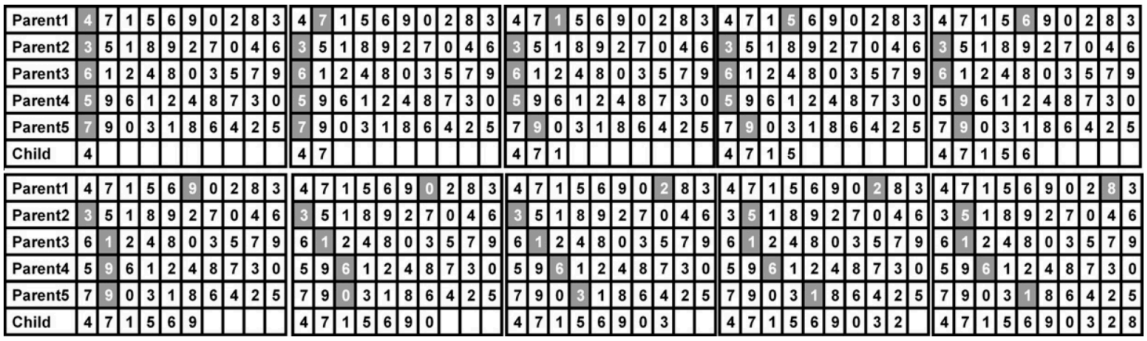


Fig. 10.1 Contoh scanning-based crossover

2. Adjacency-based Crossover

If the relative position of genes in a chromosome or the sequence of composition in a gene fragment is important (e.g., some chromosomes may share the same arcs in the traveling salesman problems), ABC would be suitable in this case. We interpret the ABC in the following way: the first gene value in the first parent is inherited in the beginning and we use the gene value to update the marker; then separately, we search the adjacent gene value from all chromosomes according to the marker value; the second gene value is inherited from competition and updates the marker value; the process is continued in the same manner until all the assignments are done.

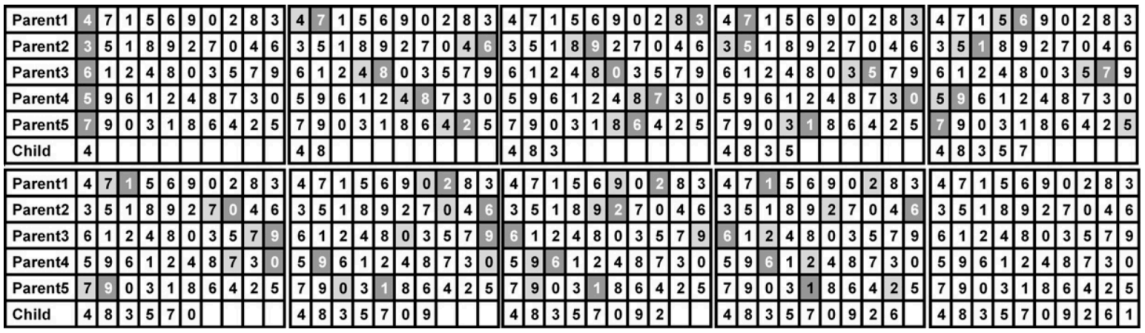


Fig. 10.2 Contoh adjacency-based crossover

3. Diagonal-based Crossover

Pada sebagian problem (paling sering ditemukan pada problem sequencing), ada kemungkinan bahwa banyak blok-blok gen yang tercipta didalam kromosom-kromosom yang ada. Beberapa peneliti berupaya untuk memanfaatkan informasi ini di dalam proses mating. DBC adalah suatu teknik yang membagi gen kedalam blok-blok sebanyak jumlah parent yang ada (misal 3 parent, maka kromosom dibagi menjadi 3 blok) dam kemudian membentuk kromosom yang utuh mengikuti aturan diagonal.

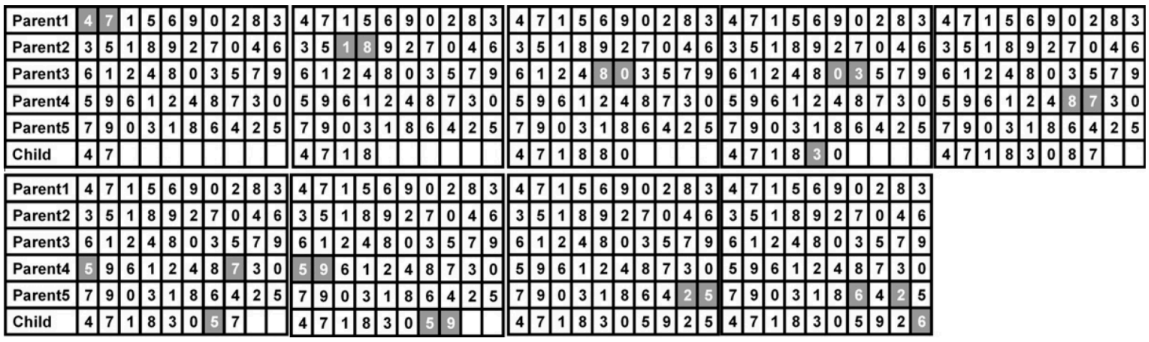


Fig. 10.3 Contoh diagonal-based crossover

Referensi

E. T. Yassen, M. Ayob, M. Z. A. Nazri and N. R. Sabar, "Multi-parent insertion crossover for vehicle routing problem with time windows", *2012 4th Conference on Data Mining and Optimization (DMO)*, Langkawi, Malaysia, 2012, pp. 103-108, doi: 10.1109/DMO.2012.6329806

Shih-Hsin Chen, Min-Chih Chen, Pei-Chann Chang, V. Mani, "Multiple parents crossover operators: A new approach removes the overlapping solutions for sequencing problems", *Applied Mathematical Modelling*, Volume 37, Issue 5, 2013, Pages 2737-2746, ISSN 0307-904X, https://doi.org/10.1016/j.apm.2012.06.005.

S. M. Elsayed, R. A. Sarker and D. L. Essam, "GA with a new multi-parent crossover for constrained optimization," *2011 IEEE Congress of Evolutionary Computation (CEC)*, New Orleans, LA, USA, 2011, pp. 857-864, doi: 10.1109/CEC.2011.5949708.

Fajrin A.M., Fatichah C. "Multi-parent order crossover mechanism of genetic algorithm for minimizing violation of soft constraint on course timetabling problem". (2020) Register: Jurnal Ilmiah Teknologi Sistem Informasi, 6 (1), pp. 43 - 51, DOI: 10.26594/register.v6i1.1663

11) Operator Mutasi

a) Pendahuluan

Setelah proses crossover, Algoritma Genetika menjalankan operator mutasi terhadap setiap kromosom pada *mating pool*. Mutasi didefinisikan sebagai proses mengubah nilai satu atau beberapa gen dalam suatu kromosom untuk mendapatkan solusi baru. Operator ini digunakan untuk menjaga dan memperkenalkan *diversity* dalam populasi genetik, biasanya diterapkan dengan probabilitas rendah P_m. Jika probabilitas terlalu tinggi, Algoritma Genetika akan berubah menjadi pencarian acak.

Jika crossover berfungsi untuk mengeksploitasi solusi yang ada untuk menemukan solusi yang lebih baik, maka mutasi berfungsi membantu eksplorasi *search space*. Mutasi dipandang sebagai *background operator* untuk menjaga *genetic diversity* dalam populasi dengan memperkenalkan struktur genetik baru melalui modifikasi secara acak beberapa building blocks-nya. Dengan demikian, mutasi membantu algoritma keluar dari perangkap local minima sekaligus memastikan *mating pool* tetap kaya sehingga memenuhi sifat ergodisitas, yaitu kondisi di mana terdapat probabilitas non-zero untuk menghasilkan solusi apa pun dari keadaan populasi apa pun.

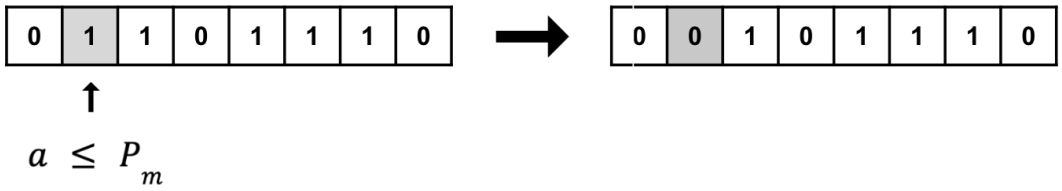
Namun demikian, populasi baru hasil mutasi tidak selalu lebih baik daripada populasi sebelumnya, karena tujuan utama mutasi adalah memperluas *search space*, bukan secara langsung meningkatkan kualitas solusi.

b) Jenis-Jenis Operator Mutasi

Pemilihan metode mutasi tidak dapat dilakukan secara sembarangan, karena setiap representasi kromosom memiliki aturan encoding yang berbeda. Oleh karena itu, metode mutasi yang digunakan harus disesuaikan dengan representasi kromosom agar modifikasi gen tidak merusak struktur solusi dan menghasilkan solusi baru yang valid.

1. Mutasi untuk Representasi Biner

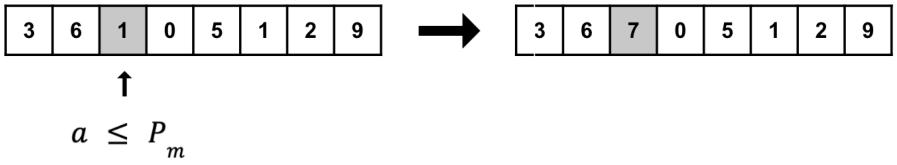
   Mutasi pada representasi biner dilakukan dengan cara mempertimbangkan setiap gen (bit) secara independen. Setiap gen memiliki peluang untuk berubah (flip) dari 0 menjadi 1 atau 1 menjadi 0 berdasarkan sebuah probabilitas mutasi (P_m) yang telah ditentukan. Untuk setiap gen, algoritma menghasilkan sebuah nilai acak a dalam rentang 0 hingga 1. Jika nilai acak tersebut lebih kecil atau sama dengan probabilitas mutasi P_m, maka gen tersebut akan mengalami mutasi.



Gambar 11.1 Bit Flip

2. Mutasi untuk Representasi Integer
    - Random Resetting

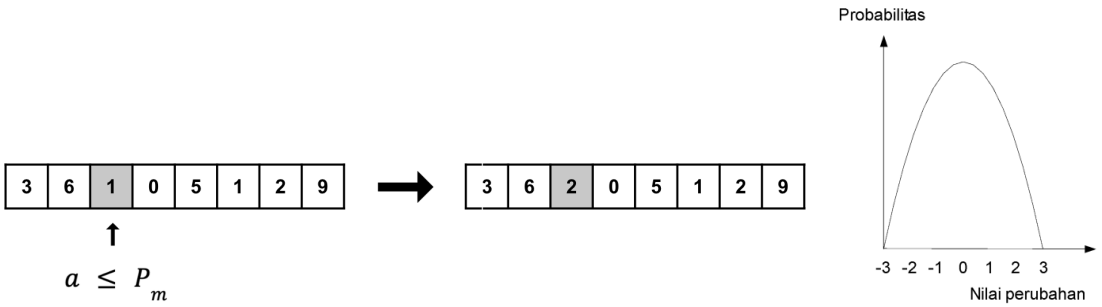
      Mutasi dengan metode Random Resetting dilakukan dengan mempertimbangkan setiap gen secara independen. Setiap gen memiliki peluang untuk diganti dengan nilai baru berdasarkan probabilitas mutasi (P_m) yang telah ditentukan. Untuk setiap gen, algoritma menghasilkan sebuah nilai acak a dalam rentang 0 hingga 1. Jika nilai acak tersebut lebih kecil atau sama dengan probabilitas mutasi P_m, maka gen tersebut akan mengalami dimutasi dengan cara memilih nilai baru secara acak dari domain nilai yang ditentukan.



Gambar 11.2 Random Resetting dengan domain gen {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

- Creep Mutation

  Mutasi dengan metode Creep Mutation dilakukan dengan mempertimbangkan setiap gen secara independen. Setiap gen memiliki peluang untuk mengalami perubahan kecil (increment atau decrement) berdasarkan probabilitas mutasi (Pm) yang ditentukan. Untuk setiap gen, algoritma menghasilkan sebuah nilai acak a dalam rentang 0 hingga 1. Jika nilai acak tersebut lebih kecil atau sama dengan Pm, maka gen tersebut dimutasi dengan menambahkan nilai kecil (positif atau negatif) yang biasanya diambil secara acak dari distribusi yang simetris terhadap nol. Perubahan dibuat kecil agar struktur solusi tetap stabil dan tidak melompat terlalu jauh dari nilai awal.



Gambar 11.3 Creep Mutation

3. Mutasi untuk Representasi Real
    - Uniform Mutation

Mutasi dengan metode Uniform Mutation dilakukan dengan mempertimbangkan setiap gen secara independen. Setiap gen memiliki peluang untuk diganti dengan nilai baru berdasarkan probabilitas mutasi (Pm) yang ditentukan. Untuk setiap gen, algoritma menghasilkan sebuah nilai acak a dalam rentang 0 hingga 1. Jika nilai acak tersebut lebih kecil atau sama dengan Pm, maka gen tersebut dimutasi dengan cara mengambil nilai baru secara acak dari distribusi uniform dalam rentang yang diizinkan [L_i , U_i]. Karena nilai baru dipilih secara seragam dari seluruh rentang, gen yang dimutasi dapat berubah menjadi nilai apa pun di domainnya, tanpa memperhatikan nilai awal.

Gambar 11.4 Uniform Mutation dengan domain gen [0, 10]

- Nonuniform Mutation

  Mutasi dengan metode Non Uniform Mutation dilakukan dengan mempertimbangkan setiap gen secara independen. Setiap gen memiliki peluang untuk mengalami perubahan kecil berdasarkan probabilitas mutasi (Pm). Untuk setiap gen, algoritma menghasilkan sebuah nilai acak a dalam rentang 0 hingga 1. Jika nilai a lebih kecil atau sama dengan Pm, maka gen tersebut dimutasi dengan menambahkan nilai perubahan (Δ) yang diambil dari distribusi nonuniform, biasanya distribusi Gaussian (normal) dengan mean 0 dan standar deviasi σ. Setelah nilai baru dihitung, hasil mutasi dipotong (clamped) agar tetap berada dalam batas domain yang diizinkan [L_i , U_i].

  Distribusi Gaussian menghasilkan nilai yang cenderung kecil sehingga sebagian besar mutasi hanya menggeser gen sedikit dari nilai awalnya, sehingga menjaga stabilitas solusi, tetapi tetap memungkinkan eksplorasi yang lebih luas ketika perubahan besar kadang muncul.

4. Mutasi untuk Representasi Permutasi
   - Swap Mutation

     Swap Mutation dilakukan dengan memilih dua posisi gen dalam kromosom secara acak. Kemudian, menukar nilai pada kedua posisi gen tersebut.

Gambar 11.6 Swap Mutation

   - Insert Mutation

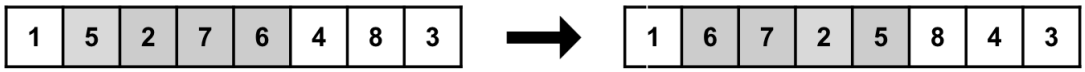
Insert Mutation dilakukan dengan memilih dua posisi gen dalam kromosom secara acak. Kemudian, gen pada posisi terpilih kedua dipindahkan dan disisipkan tepat setelah posisi gen terpilih pertama. Sementara itu, gen-gen lain di antara kedua posisi tersebut digeser untuk memberi ruang.

Gambar 11.7 Insert Mutation

- Scramble Mutation
  Scramble Mutation dilakukan dengan memilih sebuah subset gen dalam kromosom secara acak. Setelah subset tersebut dipilih, urutan gen di dalam subset diacak (scrambled), tetapi nilai gennya tidak diubah. Gen di luar subset tetap tidak berubah.

Gambar 11.8 Scramble Mutation

- Inversion Mutation
  Inversion Mutation dilakukan dengan memilih sebuah subset gen dalam kromosom secara acak. Namun, alih-alih mengacak urutan gen di dalam subset, seluruh urutan gen pada subset tersebut dibalik (inverted), gen pada bagian yang dipilih ditulis ulang dari belakang ke depan. Gen di luar subset tetap tidak berubah.



Gambar 11.9 Inversion Mutation

Referensi

Sivanandam, S. N., & Deepa, S. N. (2008). Introduction to genetic algorithms. *Introduction to Genetic Algorithms*, 1–442. https://doi.org/10.1007/978-3-540-73190-0

Genetic Algorithms - Mutation. (n.d.). Retrieved November 22, 2025, from https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm

Course Material Week 9 - Mutation and Update Generation

Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing*. https://doi.org/10.1007/978-3-662-44874-8

12) Perbaikan Generasi pada Algoritma Genetika

a) Pendahuluan

Perbaikan generasi (*survivor selection*) pada Algoritma Genetika merupakan tahap penting yang menentukan kualitas populasi dari satu generasi ke generasi berikutnya. Setelah individu baru (offspring) dihasilkan melalui operasi seleksi, crossover, dan mutasi, diperlukan mekanisme yang dapat memilih individu mana yang layak dipertahankan untuk membentuk populasi baru. Tahap ini berfungsi menjaga ukuran populasi tetap konstan sekaligus memastikan bahwa individu yang memiliki kualitas lebih baik berdasarkan fitness maupun kriteria lain seperti usia memiliki peluang lebih besar untuk bertahan. Dengan demikian, perbaikan generasi berperan sebagai filter evolusioner yang menjaga stabilitas, keberagaman, dan arah pencarian solusi, sehingga menjadi salah satu komponen inti dalam keberhasilan algoritma genetika.

Dua pendekatan utama dalam perbaikan generasi adalah *generational update* dan *steady-state update.*

b) Generational Update

Generational update merupakan suatu skema perbaikan generasi dalam Algoritma Genetika yang menggantikan seluruh populasi lama dengan populasi baru pada setiap siklus evolusi. Dalam bentuk dasarnya, sejumlah N offspring dihasilkan dari populasi parent berukuran N, dan seluruh offspring tersebut langsung membentuk populasi pada generasi berikutnya. Dengan demikian, individu dalam suatu generasi hanya dapat melakukan reproduksi dengan individu lain yang berada pada generasi yang sama.

c) Steady-State Update

Steady-State Update merupakan suatu skema perbaikan generasi dalam Algoritma Genetika yang memperbarui populasi secara bertahap, tidak sekaligus seperti pada Generational Update. Pada pendekatan ini, setiap offspring yang dihasilkan melalui proses reproduksi segera dimasukkan ke dalam populasi, sehingga perubahan komposisi populasi terjadi secara kontinu sepanjang siklus evolusi.

Karena penambahan individu baru mengharuskan populasi tetap berada pada ukuran yang konstan, setiap proses penyisipan offspring disertai dengan penghapusan satu individu lama. Pemilihan individu yang akan digantikan dapat dilakukan dengan beberapa mekanisme, antara lain:

- Menggantikan individu dengan fitness terburuk.
- Menggantikan individu tertua.
- Menggantikan individu yang paling mirip dengan offspring baru guna mempertahankan keragaman populasi.
- Tournament replacement, yaitu mekanisme yang bekerja dengan cara serupa seperti tournament selection, namun dengan tujuan kebalikannya untuk lebih sering memilih individu berkualitas rendah sebagai kandidat penggantian.

Pendekatan Steady-State Update menghasilkan perubahan populasi yang lebih halus dan kontinu karena perubahan populasi tidak terjadi secara drastis pada batas generasi, melainkan berlangsung perlahan dari satu reproduksi ke reproduksi berikutnya. Dengan demikian, skema ini sering digunakan ketika diperlukan adaptasi bertahap dan pemeliharaan keragaman populasi selama proses pencarian solusi.

d) Age-Based Replacement

Age-Based Replacement merupakan suatu strategi penggantian individu dalam Algoritma Genetika di mana usia menjadi dasar pemilihan individu yang akan dikeluarkan dari populasi. Setiap individu dirancang untuk memiliki masa hidup yang sama panjangnya, sehingga ia akan digantikan setelah melewati sejumlah iterasi tertentu dalam Evolutionary Algorithm (EA).

Karena proses penggantian tidak mempertimbangkan fitness, fitness rata-rata ataupun best fitness pada suatu generasi dapat saja lebih rendah dibandingkan generasi sebelumnya. Kondisi ini tidak menjadi masalah selama tidak terjadi terlalu sering, dan bahkan dapat membantu ketika populasi terjebak di sekitar local optimum. Perbaikan fitness secara keseluruhan tetap bergantung pada adanya selection pressure yang memadai saat memilih parent, serta penggunaan variation operators yang tidak terlalu destruktif.

Strategi ini digunakan dalam simple Genetic Algorithm, di mana banyaknya offspring sama dengan banyaknya parent ($\mu = \lambda$), sehingga setiap individu hanya bertahan selama satu generasi sebelum sepenuhnya digantikan oleh offspring. Variasi dari strategi ini dapat diterapkan pada Steady-State Update dengan skenario di mana individu tertua dihapus ketika individu baru dimasukkan seperti mekanisme first-in-first-out (FIFO).

e) Fitness-Based Replacement

Fitness-Based Replacement merupakan suatu strategi penggantian individu dalam Algoritma Genetika yang mempertimbangkan nilai fitness individu. Berbagai strategi dikembangkan, antara lain:

1. Replace Worst (GENITOR)
   Replace Worst memilih individu dengan fitness terburuk untuk digantikan. Strategi ini mempercepat peningkatan fitness rata-rata, tetapi berisiko menyebabkan premature convergence karena populasi cenderung dengan cepat berfokus pada anggota paling fit yang ada saat ini.
2. Elitism
   Elitism memastikan individu terbaik tetap dipertahankan di populasi. Jika individu terbaik masuk ke kelompok yang akan diganti, maka individu tersebut dipertahankan dan membuang salah satu offspring.

3. Round-Robin Tournament
   Round-Robin Tournament mengadakan turnamen berpasangan di mana setiap individu dievaluasi terhadap q individu lain yang dipilih secara acak dari populasi gabungan parent dan offspring. Individu dengan jumlah kemenangan terbanyak dipilih sebagai survivor. Strategi ini bersifat stokastik dan memungkinkan individu yang kurang fit tetap bertahan apabila mendapat lawan yang lebih lemah.
4. (μ + λ) Selection
   Pada skema (μ + λ) selection, populasi parent berukuran μ menghasilkan λ offspring. Seluruh parent dan offspring digabungkan, kemudian di-ranking berdasarkan nilai fitness. Dari gabungan tersebut, μ individu dengan fitness terbaik dipilih untuk membentuk generasi berikutnya. Karena individu lama masih dapat bertahan jika kualitasnya tinggi, pendekatan ini memberikan selection pressure yang kuat, terutama saat λ jauh lebih besar dari μ.
5. (μ, λ) Selection
   Pada skema (μ, λ) selection, populasi parent berukuran μ menghasilkan λ offspring dengan ketentuan λ ≥ μ. Setelah offspring di-ranking berdasarkan nilai fitness, hanya μ individu terbaik yang dipilih untuk menjadi generasi berikutnya. Semua parent langsung dibuang, sehingga tidak ada individu yang hidup lebih dari satu generasi. Dengan demikian, strategi ini menggabungkan aspek age (parent selalu diganti) dan fitness (offspring tetap diseleksi berdasarkan kualitas).

Referensi

Eiben, A. E., & Smith, J. E. (2015). *Introduction to Evolutionary Computing*. https://doi.org/10.1007/978-3-662-44874-8

Sivanandam, S. N., & Deepa, S. N. (2008). Introduction to genetic algorithms. *Introduction to Genetic Algorithms*, 1–442. https://doi.org/10.1007/978-3-540-73190-0

13) Parameter-Parameter Algoritma Genetika

1. Pendahuluan

Pemilihan parameter merupakan aspek krusial dalam desain dan implementasi GA yang efektif. Parameter yang dipilih dengan baik dapat membedakan antara algoritma yang konvergen cepat dengan solusi optimal dengan algoritma yang terjebak dalam local optima atau gagal konvergen sama sekali.

Konvergensi mengacu pada kemampuan algoritma untuk memusatkan pencarian pada solusi berkualitas tinggi. Parameter seperti tekanan seleksi (selection pressure) memainkan peran dominan dalam mengendalikan kecepatan konvergensi. Tekanan seleksi yang terlalu tinggi bisa menyebabkan konvergensi prematur, sedangkan tekanan terlalu rendah membuat algoritma sulit konvergen dan cenderung berputar tanpa arah (Shams, 2025).

Selain konvergen, eksplorasi dan eksploitasi menjadi penting dalam proses GA. Eksplorasi merujuk pada kemampuan algoritma untuk menjelajahi ruang pencarian dan menemukan region baru yang menjanjikan. Eksploitasi merujuk pada kemampuan untuk memanfaatkan informasi dari solusi yang sudah ditemukan untuk memperbaikinya lebih lanjut.

2. Representasi (Encoding)

Representasi atau encoding adalah cara kita mengkodekan solusi kandidat dalam GA. Pilihan representasi sangat mempengaruhi operator yang digunakan dan efektivitas algoritma (Smith dan Vavak, 1998).

2.1 Binary Encoding

Setiap solusi direpresentasikan sebagai string bit (0 dan 1).

Contoh:

Individu 1	1	0	0	1	1	0	1	0
Individu 2	0	1	1	0	0	1	0	1

Kelebihan:

- Sederhana dan universal
- Mudah implementasi operator
- Fast computation

Kekurangan:

- Precision terbatas
- Tidak natural untuk continuous domains

Aplikasi: Knapsack problem, feature selection, kombinatorial optimization sederhana.

2.2 Real Valuded (Floating-Point) Encoding

Solusi direpresentasikan sebagai vektor bilangan real.

Contoh:

Individu 1: [2.54, -1.32, 0.87, 3.14]

Individu 2: [1.89, 2.01, -0.45, 2.67]

Kelebihan:

- Natural untuk continuous optimization
- Precision lebih tinggi
- Operator lebih intuitif

Kekurangan:

- Perlu operator khusus
- Schema theorem tidak langsung berlaku

Aplikasi: Function optimization, neural network weight optimization, engineering design.

2.3 Integer Encoding
Vektor bilangan bulat
Contoh:
Individu 1: [3, 7, 2, 5, 1]
Individu 2: [4, 2, 6, 1, 8]
Aplikasi: Scheduling, bin packing, resource allocation.

2.4 Permutation Encoding
Urutan elemen unik (setiap elemen muncul tepat sekali).
Contoh:
Individu 1: [2, 4, 1, 5, 3] Urutan kunjungan: kota 2 → 4 → 1 → 5 → 3
Individu 2: [1, 3, 5, 2, 4]
Aplikasi: Traveling Salesman Problem (TSP), vehicle routing, job shop scheduling.

3. Ukuran populasi (Population Size)

Ukuran populasi (biasanya dilambangkan N atau $Psize$) adalah jumlah individu (solusi kandidat) yang dipertahankan dalam setiap generasi GA. Pilihan ukuran populasi memiliki dampak yang sangat krusial terhadap performa GA karena berkaitan erat dengan aspek keberagaman genetik (diversity), kapasitas eksplorasi dan eksploitasi algoritma, serta besaran komputasi yang harus dikeluarkan (Shams, 2025).

Jika ukuran populasi terlalu kecil, kelebihannya terletak pada kecepatan konvergensi dan efisiensi waktu komputasi per generasi. Ukuran ini cocok untuk masalah sederhana, namun populasinya berisiko tinggi mengalami premature convergence yang berarti mudah terjebak pada solusi lokal yang bukan solusi optimal, selain itu juga diversity menjadi rendah sehingga struktur genetik penting bisa hilang dari populasi. Untuk populasi sedang, terdapat keseimbangan yang baik antara kecepatan konvergensi dan eksplorasi, juga diversity yang cukup untuk mayoritas aplikasi. Namun pada masalah yang sangat kompleks, premature convergence masih mungkin terjadi. Sementara itu untuk populasi besar memiliki kelebihan berupa diversity yang sangat tinggi, eksplorasi ruang solusi yang menyeluruh, dan kemampuan robust terhadap jebakan local optima. Namun, besarnya populasi membuat proses komputasi berjalan lebih lambat dan lebih mahal secara sumber daya, bahkan kadang menjadi overkill untuk masalah sederhana.

4. Fungsi fitness

Pada algoritma evolusioner, fungsi fitness berperan sebagai core yang menentukan seberapa baik setiap individu memenuhi tujuan optimasi. Prinsip perancangan fungsi fitness yang baik meliputi beberapa kriteria utama. Pertama, fungsi tersebut harus meaningful atau benar-benar mencerminkan kualitas solusi yang dihasilkan. Kedua, skala nilai fitness harus reasonable, yaitu terdistribusi dalam rentang yang logis agar bisa digunakan untuk seleksi. Ketiga, evaluasi fitness sebaiknya tidak terlalu mahal secara komputasi karena akan dimanfaatkan ribuan kali dalam satu siklus evolusi. Keempat, sangat dianjurkan agar fungsi fitness bersifat smooth, yaitu peningkatan kualitas solusi harus sejalan dengan peningkatan nilai fitness.

4.1 Linear Scaling

$f'(x) = a \cdot f(x) + b$

Di mana a dan b dipilih sehingga:

- Mean scaled fitness = mean raw fitness
- Max scaled fitness = C x mean

4.2 Sigma Scaling

$f'(x) = f(x) - (\bar{f} - c \cdot \sigma f)$

Di mana:

$\bar{f}$ = mean fitness populasi

σf = standard deviation fitness

c = konstanta

4.3 Rank-Based Scaling

$f'(i) = 2 - s + \frac{2(s-1)(i-1)}{N-1}$

Di mana:

- S = selection pressure (1 <s<=2)
- I = rank (1=worst, N=best)

5. Seleksi

Selection operator memilih individu dari populasi untuk menjadi parent yang akan menghasilkan offspring generasi berikutnya. Selection adalah mekanisme yang memodelkan "survival of the fittest" dalam evolusi alami.

5.1 Roulette Wheel Selection

Prinsip: Probabilitas seleksi proporsional dengan fitness

$P(select\ i) = \frac{fi}{\Sigma_{j=1}^{N} fi}$

Algoritma:

1. Hitung total fitness
2. Generate random number $r \in [0, S]$
3. Sum fitness dari individu 1,2,... hingga sum >= r
4. Pilih individu di mana sum melewati r

5.2 Rank Selection

$P(i) = \frac{1}{N}(2 - s + \frac{2(s-1)(i-1)}{N-1})$

Di mana

- I = rank (1=worst, N=best)
- s=selection pressure

6. Crossover

Crossover (atau recombination) adalah operator utama dalam GA yang menggabungkan informasi genetik dari dua (atau lebih) parent untuk menghasilkan offspring. Crossover adalah mekanisme eksploitasi utama: memanfaatkan building blocks yang sudah ditemukan untuk menciptakan solusi lebih baik.

6.1 One point crossover

Algoritma:

- Pilih random crossover point $k \in [1, L - 1]$ di mana L=chromosome length
- Offspring 1 = Parent1[1..k] + Parent2[k+1..L]
- Offspring 2 = Parent2[1..k] + Parent1[k+1..L]

6.2 Two point crossover

Algoritma:

- Pilih dua random points k1, k1
- Swap segment di antara k1 dan k2

6.3 Uniform crossover

Algoritma:

- Untuk setiap gene position, flip coin (probabilitas 0.5)
- Jika heads → ambil dari Parent1, jika tails → ambil dari Parent2

6.4 Whole arithmetic crossover

$Offspring1 = a \cdot parent1 + (1 - a) \cdot parent2$

$Offspring2 = (1 - a) \cdot parent1 + a \cdot parent2$

6.4 Partially Mapped Crossover

Algoritma:

- Pilih dua crossover points
- Copy segment dari Parent1 ke Offspring1
- Establish mapping dari segment
- Fill sisanya berdasarkan mapping, avoid duplicates

6.5 Cycle crossover
Setiap gene diambil dari salah satu parent, berdasarkan cycle structure untuk menghindari conflict.

7. Mutasi

Mutasi adalah operator yang memperkenalkan variasi acak kecil pada individu. Mutasi adalah mekanisme eksplorasi utama dalam GA.
Tujuan mutasi:
- Maintain genetic diversity: Hindari premature convergence
- Escape local optima: Random perturbation dapat keluar dari basin
- Introduce new genetic material: Genes yang hilang bisa muncul kembali
- Fine-tuning: Small mutations untuk perbaikan lokal

7.1 Bit Flip Mutation
Algoritma:
```
for each gene in chromosome:
    if random() < pm:
        flip bit (0→1 or 1→0)
```

7.2 Uniform Mutation
Algoritma:
```
for each gene in chromosome:
        if random() < pm:
                gene = random_value in [lower_bound, upper_bound]
```

7.3 Gaussian Mutation
Algoritma:
```
for each gene in chromosome:
        if random() < pm:
                gene = gene + N(0, σ)
```

7.4 Swap Mutation
Algoritma:
- Pilih dua posisi acak
- Tukar nilai pada kedua posisi

7.5 Insert Mutation
Algoritma:
- Pilih satu gene

- Remove dari posisi saat ini
- Insert di posisi random lain

7.6 Inversion Mutation
Algoritma:

- Pilih dua posisi
- Invert (reverse) substring di antara keduanya

8. Elitism dan Replacement

Elitisme adalah strategi mempertahankan sejumlah individu terbaik dari generasi saat ini ke generasi berikutnya tanpa modifikasi. Tujuan utama elitisme adalah memastikan solusi terbaik tidak pernah mengalami penurunan fitness selama evolusi, sehingga konvergensi menjadi lebih terjamin dan stabil. Namun, jumlah elitisme yang terlalu besar beresiko mengurangi keberagaman populasi, berujung pada premature convergence.

Strategi penggantian populasi (replacement) terbagi menjadi generational replacement, steady-state replacement, dan beberapa varian lain seperti (μ + λ) dan (μ, λ). Generational replacement mengganti seluruh populasi setiap generasi, sementara steady-state mengganti 1 atau 2 individu saja per iterasi, menjaga diversity dan elitisme secara implisit. Pada strategi (μ + λ), offspring bersaing dengan parent, sedangkan pada (μ, λ) hanya offspring yang dipertahankan. Setiap strategi replacement memiliki trade-off antara eksplorasi, elitisme, dan kecepatan konvergensi.

9. Stopping Criteria

Termination criteria atau kriteria penghentian menentukan kapan GA harus berhenti beroperasi. Tanpa kriteria yang tepat, risikonya adalah algoritma berhenti terlalu dini sebelum mencapai solusi optimal atau berjalan terlalu lama sehingga pemborosan sumber daya.

Beberapa tipe utama termination criteria antara lain jumlah generasi maksimum, threshold fitness, stagnasi (tidak ada perbaikan dalam N generasi berturut-turut), tingkat konvergensi populasi, total waktu/jumlah evaluasi, serta kombinasi dari beberapa kriteria.

10. Tuning Parameter

Parameter tuning pada GA dapat dilakukan menggunakan berbagai pendekatan, mulai dari manual trial-and-error, hingga eksperimen terstruktur seperti grid search, random search, factorial design, dan response surface methodology (RSM). Proses meta-optimization dapat memanfaatkan algoritma GA sendiri atau metaheuristik lain

seperti PSO dan Bayesian optimization untuk mencari nilai parameter optimal secara otomatis dan adaptif.

Strategi adaptif dan self-adaptive pada parameter kontrol (misal mutation, crossover rate) mengubah parameter secara dinamis mengikuti feedback dari performa atau keadaan populasi. Metode self-adaptive meng-encode parameter dalam gen dan ikut berevolusi bersama solusi.

11. Referensi

Shams, E. (2025). Resolving the Exploration-Exploitation Dilemma in Evolutionary Algorithms: A Novel Human-Centered Framework. arXiv (Cornell University). doi:https://doi.org/10.48550/arxiv.2501.02153.

Smith, J. and Vavak, F. (1998). Replacement Strategies in Steady State Genetic Algorithms: Static Environments. Pp.219–234.

Mitchell, M. (1998). An introduction to genetic algorithms. Cambridge, Mass. ; London: Mit.

Hybrid GA

1. Pendahuluan

Hybridization pada Genetic Algorithm (GA) merupakan paradigma pengembangan metaheuristik yang mengkombinasikan eksplorasi ruang solusi dengan algoritma lain yang memiliki keunggulan dalam eksploitasi. Tujuan utama hybridization adalah meningkatkan efisiensi pencarian, mengurangi premature convergence, serta memperbaiki kualitas solusi dalam masalah-masalah optimasi nyata yang kompleks dan multimodal. Fenomena ini sangat berkembang pesat seiring tuntutan aplikasi dunia nyata yang membutuhkan solusi presisi tinggi dan efisiensi komputasi lebih baik dibanding GA standar (Gu et al., 2022).

2. Konsep Dasar Hybridization dalam GA

Konsep hybridization dalam GA didasarkan pada prinsip bahwa kombinasi dua atau lebih metode optimasi dapat menyeimbangkan kekuatan eksplorasi (global search) dari GA dan eksploitasi (local search atau problem-specific operator) dari algoritma lain. Contoh nyata adalah penggabungan GA dengan local search membentuk algoritma Memetic yang mampu memanfaatkan keunggulan evolusi populasi serta perbaikan solusi secara simultan.

3. Klasifikasi Hybrid Genetic Algorithm

3.1. Low Level Hybridization

Integrasi terjadi pada level operator atau komponen spesifik GA (misal, local search menggantikan atau melengkapi mutation/crossover dalam pipeline generasi offspring).

3.2. High Level Hybridization

Komponen utama tetap berjalan terpisah dan hasil/solusi dipertukarkan di interval tertentu (misal, GA dan Simulated Annealing bekerja paralel serta bertukar solusi terbaik).

3.3. Tight Coupling

Komponen hybrid berjalan secara erat (ingrained) di satu kerangka algoritma, di mana solusi/representasi dapat berubah secara sinkron dalam satu proses evolusi.

3.4. Loose Coupling

Komponen digabungkan longgar, di mana setiap metode memiliki ruang/cara berjalan sendiri sebelum hasilnya saling dipertukarkan dalam siklus tertentu.

4. Common Hybridization Partners

4.1. GA + Local Search (Memetic Algorithm)

Local search (misal, Hill Climbing, 2-Opt) diaplikasikan pada sebagian atau seluruh offspring/populasi setelah tahap GA. Terbukti sangat efektif dalam combinatorial optimization seperti TSP, JSSP.

4.2. GA + Simulated Annealing
Hybrid ini biasanya menggunakan SA sebagai post-processing untuk solusi yang dihasilkan GA, atau integrasi kedua skema secara berselang-seling.

4.3. GA + Hill Climbing
Mirip memetic algorithm, Hill Climbing diterapkan untuk mencari local optima setelah langkah mutasi/crossover.

4.4. GA + Machine Learning
GA digunakan untuk optimasi hyperparameter, arsitektur neural network, atau proses feature selection pada model ML (Majhi et al., 2025).

4.5. GA + Particle Swarm Optimization
Keduanya dikombinasi untuk memanfaatkan keunggulan eksplorasi PSO dan rekombinasi GA, misal pada pembangkitan populasi baru.

4.6. GA + Fuzzy Logic
Fuzzy logic digunakan untuk representasi fitness ataupun penentuan adaptif operator serta pengambilan keputusan dalam hybrid GA.

4.7. GA + Hybrid Constraint Solvers
Misal kombinasi GA dengan Dynamic Programming (DP) untuk optimasi terstruktur, atau Ant Colony Optimization (ACO) untuk routing problems.

5. Arsitektur Hybrid Genetic Algorithm

Terdapat beberapa model arsitektur utama dalam HGA:

5.1. Model Sequential
Proses GA dan mitra hybrid dijalankan bergantian atau secara serial (misal, setiap generasi GA diikuti local search, atau sebaliknya).

5.2. Model Parallel
GA dikombinasikan dengan metode lain dalam arsitektur paralel menggunakan multiple population/islands yang tiap-nya menjalankan strategi berbeda, biasanya terhubung lewat mekanisme migrasi solusi (Majhi et al., 2025).

5.3. Model Adaptive (Self Adaptive)
Algoritma dapat menyesuaikan sendiri kapan dan metode apa yang digunakan untuk hibridisasi berdasarkan indikator performa seperti stagnasi, diversity, atau sejarah konvergensi sebelumnya

6. Desain Hybrid GA

6.1. Memilih teknik lokal yang sesuai
Pilihan mitra hybrid harus kompatibel dengan karakteristik masalah (misal, TSP efektif dengan 2-Opt).

6.2. Menentukan parameter integrasi
Harus didefinisikan parameter seperti seberapa sering menjalankan local search pada individu, interval pertukaran solusi, atau tingkat pengacakan pada crossover/mutasi.

6.3. Menentukan kapan hybrid dijalankan
Apakah di setiap generasi, setelah stagnasi, pada subset elit, dsb.

6.4. Menilai trade-off
Hybrid meningkatkan kualitas solusi dan mengurangi premature convergence namun seringkali menambah penggunaan waktu komputasi, overhead kompleksitas implementasi, dan parameter tuning.

7. Studi Kasus

Pada studi terbaru di bidang kombinasi Genetic Algorithm dan K-Nearest Neighbors, sebuah paper oleh Pandey et al. (2025) menunjukkan bahwa hybrid berbasis genetic algorithm untuk mengoptimalkan hyperparameter dan feature selection pada model KNN secara signifikan meningkatkan akurasi klasifikasi pada prediksi penyakit jantung. Dengan tuning otomatis parameter K dan seleksi fitur berbasis GA, mereka berhasil menaikkan akurasi KNN hingga 95,38%, lebih unggul dari model KNN tanpa tuning. Demikian juga, untuk aplikasi hybrid Genetic Algorithm dan Simulated Annealing, Xiang et al. (2023) membuktikan pada problem penjadwalan multi-objective serta variant job shop scheduling bahwa integrasi GA dengan SA menghasilkan solusi dengan best-fitness lebih baik dan waktu konvergensi lebih stabil dibandingkan metode tunggal, serta mampu menghindari jebakan local optima lebih efektif. Pada domain parameter estimation dan high-dimensional optimization, Majhi dkk. (2025) menghadirkan GANMA, yaitu hybrid antara GA dan Nelder-Mead. Hasil benchmarking dan studi kasus parameter estimation menunjukkan hybrid ini menawarkan robustness yang tinggi, konvergensi cepat, dan solution quality yang melewati single GA maupun Nelder-Mead saja.

8. Kelebihan dan Kekurangan Hybrid GA

8.1 Kelebihan:

- Meningkatkan eksploitasi sekaligus menjaga eksplorasi sehingga dapat menghindari premature convergence.
- Sering kali lebih robust terhadap complex/multimodal problems.
- Mencapai performa lebih tinggi pada banyak benchmark dan aplikasi dunia nyata.
- Adaptif, mudah dikustomisasi dan paralelisasi.

8.2 Kekurangan:

- Parameter tuning menjadi lebih sulit karena adanya parameter hybrid tambahan.
- Komputasi lebih intensif, overhead waktu, dan biaya pengembangan lebih besar.

- Desain dan pengujian arsitektur hybrid memerlukan pengetahuan mendalam tentang karakteristik dan interaksi antar algoritma partner.
- Rentan terhadap “over-hybridization” yang dapat menambah kompleksitas tapi tidak selalu meningkatkan performa.

9. Referensi

Betul Sultan Yıldız, Kumar, S., Natee Panagant, Mehta, P., Sait, S.M., Ali Riza Yildiz, Nantiwat Pholdee, Sujin Bureerat and Seyedali Mirjalili (2023). A novel hybrid arithmetic optimization algorithm for solving constrained optimization problems. Knowledge-Based Systems, 271, pp.110554–110554. doi:https://doi.org/10.1016/j.knosys.2023.110554.

Gu, H., Lam, H.C. and Zinder, Y. (2022). A hybrid genetic algorithm for scheduling jobs sharing multiple resources under uncertainty. EURO Journal on Computational Optimization, 10, pp.100050–100050. doi:https://doi.org/10.1016/j.ejco.2022.100050.

Majhi, N. and Mishra, R. (2025). A novel hybrid genetic algorithm and Nelder-Mead approach and it’s application for parameter estimation. F1000Research, 13, p.1073. doi:https://doi.org/10.12688/f1000research.154598.3.

Murad, S.H., Tayfor, N.B., Mahmood, N.H. and Arman, L. (2025). Hybrid genetic algorithms-driven optimization of machine learning models for heart disease prediction. MethodsX, [online] 15, p.103510. doi:https://doi.org/10.1016/j.mex.2025.103510.