BAB 2 LANDASAN TEORI

Pada bab ini dibahas mengenai penjadwalan kuliah secara umum pada subbab 2.1, permodelan penjadwalan kuliah sebagai *constraint satisfaction problem* (CSP) pada subbab 2.2, dan penyelesaian CSP dengan *local search* dan *genetic algorithm* pada subbab 2.3.

2.1 Penjadwalan Kuliah

Penjadwalan adalah permasalahan dalam menentukan jadwal yang tepat atas suatu pekerjaan terhadap sumber daya yang tersedia sesuai dengan *constraint* yang harus dipenuhi (Soraya, 2007). Dalam penjadwalan kuliah, terdapat sumber daya seperti mata kuliah (MK), dosen, ruangan, dan waktu atau slot perkuliahan. Dari sumber daya yang tersedia ini dibuatlah jadwal pekerjaan, dalam hal ini kegiatan perkuliahan, sesuai dengan *constraint* yang harus dipenuhi. Contoh *constraint* dalam masalah penjadwalan kuliah yang harus dipenuhi yaitu dosen tidak mengajar lebih dari satu MK pada saat yang sama. Tentang definisi *constraint* yang berhubungan dengan penjadwalan kuliah secara lebih detil dibahas pada subbab 3.2.

Jadwal untuk kegiatan perkuliahan biasanya mingguan dan berlaku untuk satu semester. Sebuah jadwal disebut lengkap jika setiap MK yang ditawarkan pada suatu semester memiliki slot kapan kuliah tersebut berlangsung, ruang di mana kuliah tersebut berlangsung, dan dosen yang mengajar kuliah tersebut. Jadwal yang lengkap belum tentu valid. Jadwal yang valid adalah jadwal yang sesuai dengan *constraint* yang harus dipenuhi agar kegiatan perkuliahan dapat benar-benar berlangsung. Contoh kegiatan perkuliahan tidak dapat berlangsung misalnya jika ada jadwal di mana dosen harus mengajar 2 MK yang berbeda pada saat yang sama. Dosen tersebut tidak akan bisa mengajar 2 MK bersamaan. Jadwal seperti ini tidak valid dan tidak bisa dijalankan.

Penjadwalan kuliah sebagai kegiatan yang rutin diadakan universitas merupakan faktor pendukung nyamannya perkuliahan yang diadakan. Banyaknya

kemungkinan kombinasi nilai untuk penjadwalan membuat permasalahan ini termasuk ke dalam kategori *NP-Hard problem* (Soraya 2007, Iorga 2006). Berbagai penelitian diadakan untuk memecahkan *problem* ini, berbagai pendekatan dan algoritma pun dicoba. Salah satu penelitian mengenai penjadwalan kuliah dilakukan oleh Dania Tigarani Soraya menggunakan pendekatan *constraint programming*.

Pada penelitiannya, Soraya membagi *constraint* ke dalam 2 kategori yaitu *hard constraint* dan *soft constraint* berdasarkan kondisi yang dimiliki Fakultas Ilmu Komputer Universitas Indonesia. *Hard constraint* adalah kondisi dasar yang harus terpenuhi. Sedangkan *soft constraint* adalah kondisi yang mungkin terganggu (tidak terpenuhi dengan baik), tetapi sebaiknya terpenuhi. Berikut ini definisi masing-masing *hard constraint* dan *soft constraint*.

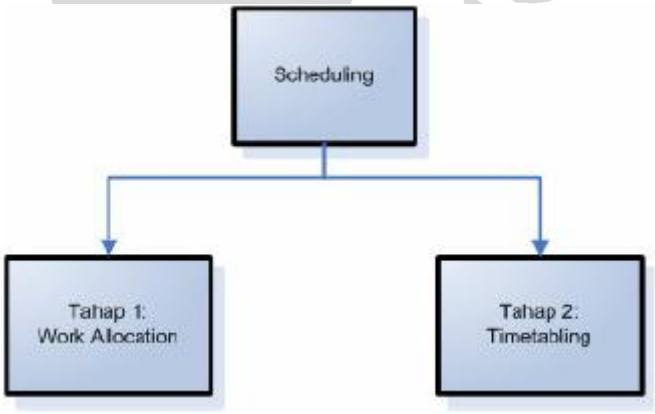
Hard constraint:

- satu MK dapat diajar satu (atau lebih) orang dosen,
- beban kuliah yang dimiliki oleh setiap dosen tidak boleh melebihi batas maksimum (*workload*) yang ditentukan,
- seorang dosen tidak boleh memiliki jadwal mengajar yang *overlap* (mengajar lebih dari satu MK dalam jadwal (jam dan hari kuliah) yang bersamaan),
- MK yang berbeda pada tingkat yang sama tidak boleh *overlap* (memiliki jadwal (jam dan hari kuliah) yang sama),
- jam kuliah dimulai dari jam 08.00 hingga jam 15.00, kecuali jam 12.00-13.00 (jam istirahat),
- hari kuliah dalam satu minggu dimulai sejak hari Senin hingga hari Jumat,
- hari Rabu jam 13.00-14.00 bukan termasuk jam kuliah karena dialokasikan untuk seminar Reboan,
- hari Jumat jam 13.00-15.00 bukan termasuk jam kuliah karena dialokasikan untuk rapat akademik.

Soft constraint:

- dosen mengajar MK karena MK tersebut adalah keahlian dosen, permintaan dosen, ataupun karena ditunjuk oleh Fakultas melalui bagian akademis,
- permintaan khusus dari dosen, seperti hari, jam, dan ruang kelas yang diinginkan terpenuhi,
- jadwal MPKT, olahraga, atau seni tidak dapat diubah,
- semester ajar yang berlaku adalah semester I, II, III, IV, V, VI, VII, dan VIII,
- antar MK prasyarat boleh dibentrokan, misalnya SDA boleh dibentrokan dengan DDP (selama dosen pengajar bukanlah orang yang sama),
- pemilihan ruang kelas dipengaruhi oleh jumlah pengikut kuliah.

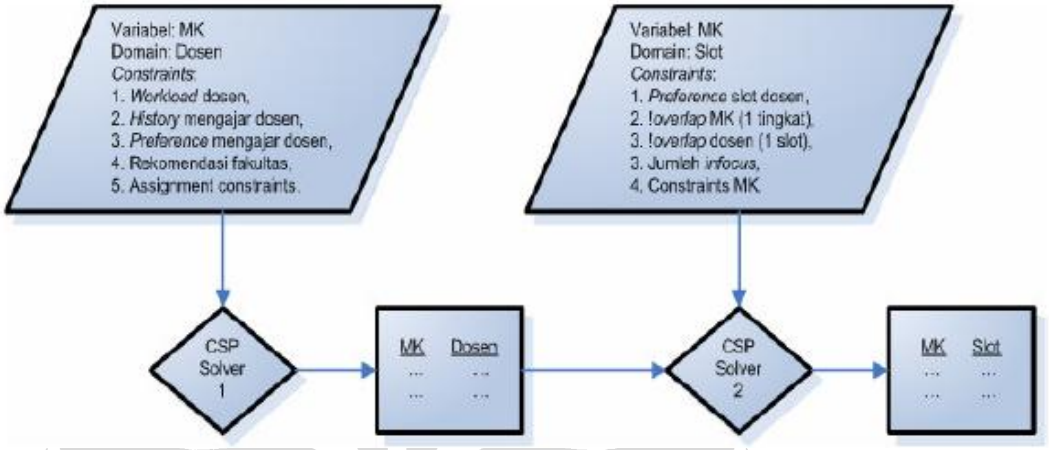
Penyelesaian penjadwalan kuliah dibagi ke dalam 2 tahap. Tahap pertama *work allocation*, yaitu pemetaan MK dan dosen yang memenuhi *constraint*. Tahap kedua *timetabling*, yaitu pemetaan MK ke slot (hari dan jam kuliah) yang memenuhi *constraint*. Tahapan model penjadwalan kuliah dapat dilihat pada Gambar 2.1.



Gambar 2.1 Tahapan model penjadwalan kuliah (Soraya, 2007)

Tahap awal yang dilakukan adalah pemetaan MK yang diajar kepada dosen yang mengajar dalam tahun ajar yang bersangkutan. Permasalahan pada tahap 1 ini dimodelkan ke dalam bentuk *constraint satisfaction problem* (CSP). Definisi variabel, *domain*, dan *constraint* tahap 1 dapat dilihat pada Gambar 2.2. Pada

tahap kedua dilakukan pemetaan MK yang diajar kepada slot kuliah dalam tahun ajaran yang bersangkutan. Permasalahan pada tahap 2 ini juga dimodelkan ke dalam bentuk CSP. Definisi variabel, *domain*, dan *constraint* tahap 2 dapat dilihat pada Gambar 2.2. Dalam mencari solusi dari model CSP pada tahap 1 dan tahap 2 digunakan teknik *branch and bound*.



Gambar 2.2 Alur model penjadwalan kuliah (Soraya, 2007)

2.2 Penjadwalan Kuliah Sebagai CSP

Constraint satisfaction problem (CSP) didefinisikan sebagai sebuah himpunan variabel, X_1, X_2, …, X_n, dan sebuah himpunan *constraint*, C_1, C_2, …, C_m. Setiap variabel X_i memiliki *domain* yang tidak kosong D_i. Setiap *constraint* C_i melibatkan beberapa *subset* dari variabel dan menentukan kombinasi nilai yang diperbolehkan dari *subset* tersebut. Sebuah *state problem* didefinisikan sebagai *assignment* nilai ke beberapa atau semua variabel, $\{X_i = v_i, X_j = v_j, \ldots\}$. *Assignment* yang tidak menyalahi *constraint* disebut *consistent* atau legal *assignment*. Solusi dari CSP adalah *assignment* lengkap yang memenuhi semua *constraint* yang ada (Russell & Norvig, 2003).

Seperti yang sudah dijelaskan di subbab 2.1, sebuah jadwal disebut lengkap jika setiap mata kuliah (MK) yang ditawarkan pada suatu semester memiliki slot kapan kuliah tersebut berlangsung, ruang di mana kuliah tersebut berlangsung, dan dosen yang mengajar kuliah tersebut. Oleh karena itu, penjadwalan kuliah secara umum sebagai CSP dapat dimodelkan dengan variabel X_1, X_2, …, X_n di mana n = jumlah MK yang ditawarkan pada suatu semester dan X_i = { (m_i, D_i, R_i,

S_i) | $1 \leq i \leq n$, $i \in$ bilangan bulat, $m_i \in$ himpunan MK, $D_i \in$ himpunan dosen-dosen, $R_i \in$ himpunan ruangan-ruangan, $S_i \in$ himpunan slot-slot}. Dengan pengertian MK m_i diajar oleh dosen-dosen D_i di ruangan-ruangan R_i pada slot-slot S_i. Contoh *constraint* yang harus dipenuhi dalam penjadwalan kuliah yaitu dosen tidak mengajar lebih dari satu MK pada saat yang sama. Tentang definisi *constraint* yang berhubungan dengan penjadwalan kuliah secara lebih detil dibahas pada subbab 3.2.

Pada penelitiannya, Soraya memecah masalah menjadi 2 tahap yang masing-masing tahapnya dimodelkan dengan CSP seperti dijelaskan pada subbab 2.1. Masalah pada tahap 1 dan tahap 2 adalah masalah yang *dependent*, dalam arti hasil dari solusi tahap 1 mempengaruhi *search space* masalah tahap 2. Misalnya dari tahap 1 diperoleh solusi MK A dan B diajar oleh dosen C. Karena ada *constraint* seorang dosen tidak boleh memiliki jadwal mengajar yang *overlap*, pemetaan MK ke slot pada tahap 2 akan menghindari MK A dan B jatuh pada slot yang sama. Dengan demikian, *search space* pada tahap 2 diperkecil karena adanya hasil di tahap 1. Padahal, mungkin saja solusi optimal berada pada serach space yang lebih besar yang tidak tercakup *search space* pada tahap 2, sehingga solusi optimal tidak dapat ditemukan. Dalam kaitannya dengan contoh, mungkin saja solusi optimal tercapai ketika MK A dan B tidak diajar oleh dosen C. Di sisi lain, keuntungan dari pemecahan masalah menjadi 2 tahap adalah kecepatan karena mencari solusi di *search space* yang lebih kecil akan lebih cepat dibanding mencari di *search space* yang besar.

2.3 Penyelesaian CSP dengan Local Search dan Genetic Algorithm

Local search merupakan salah satu algoritma dalam mencari solusi atau *goal*. Berbeda dengan *search* yang sistematik, *local search* tidak peduli dengan *path*, sehingga untuk *problem* yang *path* ke *goal*-nya tidak dicari, *local search* dapat digunakan. Cara kerja *local search* secara singkat yaitu membuat suatu *state random*, lalu melakukan operasi terhadap *state* tersebut menghasilkan sejumlah *state* baru, lalu dipilih *state* terbaik berdasarkan fungsi tertentu, lalu dari *state*

terpilih tersebut dilakukan operasi lagi sampai ditemukan solusi. Informasi *path* mengenai *state* yang dipilih tidak disimpan (Russell & Norvig, 2003).

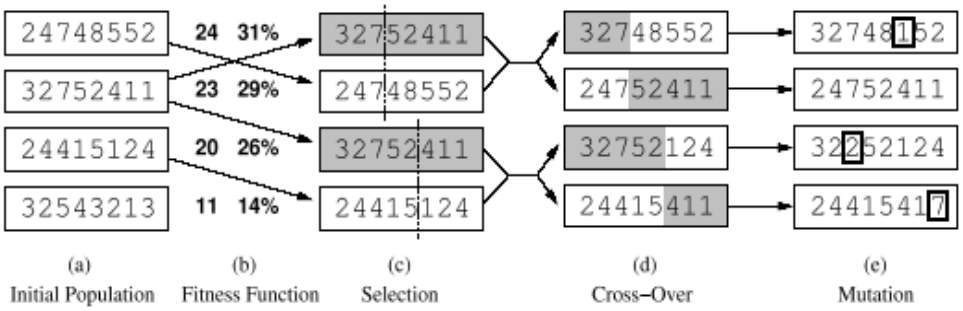
Keuntungan menggunakan algoritma ini adalah *local search* bekerja dengan memori yang kecil dan biasanya konstan. Selain itu, *local search* sering kali dapat menemukan solusi yang masuk akal dalam *state space* yang sangat besar yang tidak bisa ditemukan oleh algorima yang sistematik. Namun, pencarian solusi dengan algoritma *local search* tidak menjamin ditemukannya solusi (Russell & Norvig, 2003).

Dalam mencari *goal*, *local search* cocok digunakan untuk *pure optimization problem*, yang mana tujuannya adalah untuk menemukan *state* terbaik berdasarkan fungsi objektif. Contoh algoritma *local search* yaitu *hill-climbing search*, *simulated annealing search*, *local beam search*, *stochastic beam search* dan *genetic algorithm*. *Genetic algorithm* (GA) menggunakan prinsip seleksi alam dan pewarisan gen, di mana individu yang terkuat yang akan *survive* dan dapat menghasilkan keturunan yang mewarisi sifat genetik dari orang tuanya. Penggunaan GA yang berhasil memerlukan pembuatan representasi yang cermat. (Russell & Norvig, 2003).

Beberapa istilah yang biasa digunakan dalam GA yaitu sebagai berikut.

1. Individu. Merupakan representasi permasalahan, menggambarkan suatu *state*, berupa satu *chromosome*.
2. *Chromosome*. Merupakan kumpulan *genome* yang merepresentasikan suatu individu.
3. *Genome*. Merupakan suatu bagian dari *chromosome* yang memiliki *domain* tertentu.
4. Populasi. Merupakan kumpulan individu.
5. *Fitness function*. Merupakan fungsi untuk menilai suatu individu .
6. *Selection*. Merupakan cara memilih individu yang akan dilibatkan dalam operasi genetik.

7. *Crossover*. Merupakan operasi genetik yang melibatkan dua individu sehingga anaknya merupakan kombinasi dari kedua inidividu induk.
8. *Mutation*. Merupakan operasi genetik yang melibatkan satu individu dengan mengubah *genome* dari individu tersebut secara *random*. Bisanya operasi ini dilakukan dengan probabilitas kecil setelah operasi *crossover*.
9. *Genotype*. Merupakan representasi individu berupa *genome-genome* (*chromosome*). Contoh: DNA mausia. DNA manusia terdiri dari kombinasi gen ATCG sedemikian rupa. Suatu konfigurasi ATCG pada DNA manusia dapat keluar (dipetakan) sebagai sifat-sifat yang dimiliki manusia tesebut seperti hidung mancung, badan tinggi, dan warna rambut hitam. Sifat-sifat inilah yang disebut *fenotype*.
10. *Fenotype*. Merupakan representasi individu yang dapat dinilai kebagusannya. Contoh: bentuk hidung manusia, tinggi badan manusia, dan warna rambut manusia.



Gambar 2.3 *Genetic algorithm*. Populasi awal (a) direngking oleh *fitness function* (b), menghasilkan pasangan untuk reproduksi (c). Mereka menghasilkan keturunan (d), yang merupakan subjek untuk dimutasi (e) (Russell & Norvig, 2003).

Biasanya individu (*chromosome*) direpresentasikan sebagai *string*. Sebagai contoh, *state* 8-*queens* menggambarkan posisi 8 *queen* di 8 kolom, dalam setiap kolom bisa bernilai antara 1 sampai dengan 8. Ini dapat direpresentasikan sebagai *string* 8 digit, di mana nilai setiap digitnya berkisar antara 1 sampai dengan 8. Gambar 2.3 (a) menunjukkan populasi dari 4 individu (*string* 8 digit) yang

merepresentasikan *state* dalam 8-*queens problem*. Pembuatan individu baru dari generasi selajutnya ditunjukkan di Gambar 2.3 (b)-(e) (Russell & Norvig, 2003).

Pada (b) setiap individu dinilai dengan sebuah *fitness function*. *Fitness function* harus bernilai semakin tinggi jika state yang dihasilkan semakin baik. Oleh karena itu, digunakan jumlah *queen* yang tidak saling menyerang sebagai *fitness function*. Jadi jika tidak ada *queen* yang saling menyerang nilai *fitness function* = 28. Setelah dihitung, nilai *fitness* dari masing-masing individu adalah 24, 23, 20, dan 11. Di samping nilai *fitness* ada persentase dari nilai *fitness*. Nilai *fitness* ini yang akan menentukan individu yang mana yang akan dilibatkan dalam operasi genetik, semakin besar nilai *fitness* suatu individu semakin besar juga kemungkinannya untuk dipilih (Russell & Norvig, 2003).

Pada (c), 2 pasang individu dipilih secara *random* untuk reproduksi berdasarkan probabilitas di (b). Perhatikan bahwa individu kedua dipilih dua kali, sementara individu keempat tidak dipilih. Hal ini wajar karena probabilitas individu keempat paling kecil. Untuk setiap pasang yang dikawinkan dipilih *crossover point* secara *random*. Perhatikan bahwa *crossover point* pasangan pertama ada pada setelah digit ketiga dan *crossover point* pasangan kedua ada pada setelah digit kelima (Russell & Norvig, 2003).

Pada (d), keturunan terbentuk dari persilangan orang tua pada *crossover point*. Perhatikan keturunan pertama terbentuk dari digit 1-3 orang tua pertama dan sisanya dari digit 4-8 orang tua kedua. Sementara keturunan kedua terbentuk dari digit 1-3 orang tua pertama dan sisanya dari orang tua kedua. Akhirnya, pada (e) dilakukan mutasi secara *random* pada keturunan yang terbentuk di (d) dengan probabilitas tertentu. Perhatikan pada keturunan pertama terjadi mutasi pada digit keenam, mengubah nilai digit tersebut dari 5 menjadi 1. Keseluruhan proses pada Gambar 2.3 menggambarkan *pseudocode* GA pada Gambar 2.4 (Russell & Norvig, 2003).

Algoritma *local search* bisa sangat efektif dalam memecahkan masalah CSP. *State* awal bisa didapat dari pemberian nilai seluruh variabel dan *successor function* bisa dengan mengubah nilai satu variabel. Dalam GA *successor function* bisa didapat dengan mengombinasikan nilai variabel (*crossover*) dan mengubah nilai satu variabel (*mutation*). Sebagai contoh yaitu 8-*queens problem*, *state* awal dapat berupa konfigurasi *random* dari 8 *queens* di 8 kolom dan *successor function* didapat dengan mengubah posisi *queen* di suatu kolom. Dalam memilih nilai baru dari suatu variabel, *heuristic* yang paling pasti yaitu memilih nilai yang menghasilkan jumlah konflik dengan variabel lain minimum (*min-conflicts heuristic*). *Min-conflicts* ini efektif untuk banyak *problem* CSP, terutama ketika diberikan *state* awal yang masuk akal. Bahkan, jutaan-*queens problem* dapat diselesaikan dengan rata-rata 50 langkah dari *state* awal (Russell & Norvig, 2003).

```
function GENETIC-ALGHORITHM(population, FITNESS-FN) returns an individual
   inputs:   population, a set of individuals
             FITNESS-FN, a function that measures the fitness of an individual
   repeat:
     new_population ← empty set
     loop for i from 1 to SIZE(population) do
       x ← RANDOM-SELECTION(population,FITNESS-FN)
       y ← RANDOM-SELECTION(population,FITNESS-FN)
       child ← REPRODUCE(x,y)
       if(small random probability) then child ← MUTATE(child)
       add child to new_population
   until some individual is fit enough, or enough time has elapsed
   return the best individual in population, according to FITNESS-FN


fuction REPRODUCE(x,y) returns an individual
   inputs: x,y, parent individuals
   n ← LENGTH(x)
   c ← random number from 1 to n
   return APPEND(SUBSTRING(x,1,x),SUBSTRING(y,c+1,n))
```

Gambar 2.4 *Pseudocode Genetic Algorithm*. Algoritma ini sama dengan algoritma pada Gambar 2.3 kecuali pada algoritma ini setiap pasang induk menghasilkan satu keturunan, bukan dua (Russell & Norvig, 2003).

2.3.1 Multiobjective Optimization

Penjadwalan kuliah biasanya termasuk masalah yang *multiobjective* karena banyak aspek yang menentukan baik buruknya suatu jadwal kuliah, misalnya dalam suatu jadwal MK wajib di tingkat yang sama tidak boleh bentrok atau dosen tidak mengajar lebih dari satu MK pada slot yang sama. Setiap kriteria (*constraint*) yang menentukan baik buruknya suatu jadwal kuliah dapat dipandang sebagai satu fungsi objektif. Masalahnya, kadang-kadang tidak semua fungsi objektif dapat dimaksimalkan, bisa ada fungsi-fungsi objektif yang saling bertentangan. Jika kita maksimalkan fungsi objektif yang satu, fungsi objektif yang lain menurun, ada *trade off* di antara keduanya (Zitzler et al., 2004). Hal yang diinginkan adalah memaksimalkan semua fungsi objektif seoptimal mungkin atau disebut *multiobjective* optimization.

Dalam menilai *fitness function* pada masalah *multiobjective* ada dua pendekatan yaitu *aggregation based* dan Pareto *based*. *Fitness aggregation based* didapatkan dari agregasi nilai semua fungsi objektif menjadi satu nilai atau dengan kata lain, kombinasi linear dari semua fungsi objektif. Sementara itu, Pareto *based* menggunakan prinsip dominasi dalam menentukan *fitness* suatu individu terhadap populasinya. Pengertian individu A mendominasi B adalah jika semua fungsi objektif A >= B. Fungsinya untuk membuang individu yang sangat buruk, yaitu individu yang didominasi. Contoh algoritma Pareto *based* adalah SPEA2, dijelaskan di subbab 2.3.2 (Zitzler et al., 2004).

2.3.2 SPEA2

Algoritma Pareto *based* yang terbaik belakangan ini adalah *Strength Pareto Optimization Algorithm* (SPEA) 2. Algoritma ini memaksimalkan semua fungsi objektif, tidak ada satu fungsi objektif pun yang diabaikan, semua fungsi objektif sama pentingnya. Dengan demikian, algoritma ini mempertahankan keragaman genetik dalam populasi. Algoritma ini bekerja dengan langkah-langkah seperti pada Gambar 2.5 (Zitzler et al., 2004).

Input:	M (offspring population size) N (archive size) T (maximum number of generations)
Output:	A* (nondominated set)
Step 1:	Initialization: Generate an initial population P_0 and create the empty archive(external set) $A_0 = \emptyset$. Set t = 0.
Step 2:	Fitness assignment: Calculate fitness values of individuals in P_t and A_t.
Step 3:	Environtmental selection: Copy all nondominated individuals in P_t and A_t to A_{t+1}. If size of A_{t+1} exceeds N then reduce A_{t-1} by means of the truncation operator, otherwise if size of A_{t+1} is less than N then fill A_{t+1} with dominated individual in P_t and A_t.
Step 4:	Termination: If $t \geq T$ or another stopping criterion is satisfied then set A* to the set of decision vectors represented by the nondominated individuals in A_{t+1}. Stop.
Step 5:	Mating selection: Perform binary tournament selection with replacement on A_{t+1} in order to fill the mating pool.
Step 6:	Variation: Apply recombination and mutation operators to the mating pool and set P_{t+1} to the resulting population. Increment generation counter (t = t+1) and go to Step 2.

Gambar 2.5 Algoritma SPEA2 (Zitzler et al., 2004).

Perhitungan *fitness function* pada SPEA2 adalah sebagai berikut. Setiap individu *i* di dalam *archive* A_t dan populasi P_t diberikan nilai strength *S(i),* yang merepresentasikan jumlah solusi yang didominasinya.

$$S(i) = |\{j \mid j \in P_t + A_t \wedge i \succ j\}|$$

dimana | . | menyatakan jumlah anggota himpunan, + menyatakan *multiset union* dan simbol $\succ$ menyatakan Pareto *dominance* ($i \succ j$ jika *decision vector* yang dihasilkan *i* mendominasi *decision vector* yang dihasilkan *j*). Berdasarkan nilai *S*, *raw fitness R(i)* dari suatu individu *i* dihitung dengan cara:

$$R(i) = \sum_{j \in P_t + A_t \, j \succ i} S(j)$$

Jadi, *raw fitness* suatu individu ditentukan oleh nilai *strength* dari pendominasinya di *archive* dan populasi. Untuk membedakan nilai *fitness* individu yang tidak saling mendominasi, dihitunglah *density* dari suatu inidvidu *D(i)*. Untuk setiap individu *i* jarak ke semua individu *j* di *archive* dan populasi dihitung, dan diurutkan ke dalam daftar. Setelah mengurutkan ke dalam daftar secara terurut menaik, elemen ke-*k* memberikan jarak yang dicari, dinyatakan dengan σ_i^k. Setelah itu, *density* dari suatu individu *i* didefinisikan sebagai

$$D(i) = \frac{1}{\sigma_i^k + 2}$$

Dalam penyebut, 2 ditambahkan untuk memastikan bahwa nilainya lebih besar dari 0 dan *D(i)* < 1. Akhirnya, *fitness* dari suatu individu *F(i)* didapat dengan menambahkan *D(i)* dan nilai *raw fitness R(i)* dari individu tersebut (Zitzler et al., 2004).

$$F(i) = R(i) + D(i)$$