BAB II
LANDASAN TEORI

2.1. Sidang Skripsi Prodi Informatika UMN

Dalam proses sidang skripsi di Prodi Informatika UMN, terdapat empat komponen utama, yaitu, mahasiswa, dosen pembimbing, dosen penguji, dan ketua sidang skripsi. Mahasiswa adalah pihak yang akan diuji dalam sidang skripsi tersebut. Dosen pembimbing adalah dosen yang membimbing mahasiswa. Dosen penguji adalah dosen yang menguji dalam sidang skripsi. Ketua sidang adalah dosen yang memimpin jalannya sidang skripsi (Christopher & Hansun, 2019).

Selama ini, sidang skripsi dijadwalkan oleh Kaprodi Informatika UMN secara manual (Christopher & Hansun, 2019). Jadwal yang dihasilkanpun harus memenuhi sejumlah kriteria yang tidak boleh terlanggar (*hard-constraint*) seperti:

- Jadwal dari dosen yang terlibat tidak berbentrokan
- Beban kerja dari masing-masing dosen yang terlibat
- Jenjang Jabatan Akademik (JJA) bagi dosen penguji dan ketua sidang
- Pengalaman memimpin sidang skripsi bagi ketua sidang

Selain itu terdapat kriteria yang masih boleh dilanggar apabila kondisi tidak memungkinkan (*soft-constraint*), yaitu:

- Kesesuaian area penelitian dosen dengan kode area penelitian skripsi

Jadwal sidang skripsi baru dapat dikatakan optimal apabila memenuhi semua kriteria yang ada tersebut (Christopher & Hansun, 2019).

2.2. Penjadwalan

Penjadwalan merupakan alokasi dari sumber daya terhadap waktu untuk menghasilkan sebuah kumpulan pekerjaan (Baker, 1974). Permasalahan penjadwalan merupakan permasalahan kombinatorial yang rumit karena memiliki daerah alternatif solusi yang luas dan banyak dijumpai lokal optimal (Yu, 2006). Permasalahan tersebut menjadi salah satu permasalahan kombinatorial yang mendapatkan banyak perhatian dari para peneliti (Amirthagadeswaran, dkk., 2006). Beberapa diantaranya membuktikan bahwa permasalahan tersebut bertipe *non-deterministic polynomial-time hard* (NP-hard) atau tipe permasalahan yang sulit untuk diselesaikan untuk ukuran yang besar (Wu, dkk., 2006).

Di dalam permasalahan penjadwalan, terdapat 2 jenis *constraint*, yaitu *hard constraint* dan *soft constraint*. *Hard constraint* merupakan batasan yang tidak boleh dilanggar dalam penjadwalan. Sedangkan *soft constraint* merupakan batasan yang memiliki kemungkinan untuk dilanggar, tetapi pelanggaran tersebut dapat diminimalkan agar penjadwalan menjadi maksimal (Sani, dkk., 2016).

Di dalam dunia ilmu komputer, terdapat beberapa jenis permasalahan klasik yang berkaitan dengan permasalahan penjadwalan dan optimasi. Beberapa permasalahan penjadwalan dan optimasi tersebut, yaitu, *Job Shop Problem* dan *Knapsack Problem* (Bansal & Deep, 2012; Choudhary & Dave, 2016).

2.2.1. Job Shop Problem

Job Shop Problem merupakan salah satu permasalahan optimasi di dunia ilmu komputer dan riset operasi dimana suatu pekerjaan akan ditugaskan ke salah satu sumber daya dalam kurun waktu tertentu (Arisha, dkk., 2001). Pekerjaan yang ditugaskan tersebut harus berjalan sesuai dengan urutannya dan sumber daya yang menerima pekerjaan tersebut harus menyelesaikan pekerjaan tersebut terlebih dahulu apabila ingin mendapatkan pekerjaan baru lainnya (Google, 2019).

Terdapat beberapa batasan yang harus ditaati dan tidak boleh dilanggar dalam permasalahan Job Shop, yaitu (Google, 2019):

- Pekerjaan baru bisa dikerjakan oleh suatu sumber daya apabila pekerjaan sebelumnya telah diselesaikan
- Sumber daya yang ada hanya dapat mengerjakan satu pekerjaan saja dalam waktu tertentu. Atau dalam kata lain, sumber daya tersebut tidak dapat multitasking.
- Ketika suatu pekerjaan sudah dimulai, maka sumber daya tersebut berkewajiban untuk menyelesaikan pekerjaan tersebut.

Tujuan utama yang ingin dihasilkan lewat permasalahan ini adalah untuk menjadwalkan pekerjaan bagi semua sumber daya yang ada. Tujuan lainnya dari permasalahan ini adalah untuk mengoptimalkan pekerjaan yang dikerjakan oleh masing-masing sumber daya agar mendapatkan hasil yang paling optimal (Google, 2019).

2.2.2. Knapsack Problem

Knapsack Problem merupakan salah satu permasalahan *combinatorial optimization* dengan tujuan untuk mendapatkan hasil terbaik dari suatu kumpulan parameter agar dapat mencapai suatu kriteria tertentu. *Knapsack Problem* banyak diaplikasikan dalam kasus-kasus di dunia nyata seperti pada bidang industri, transportasi, logistik, dan ilmu komputer (Hembecker, dkk., 2007).

Permasalahan *Knapsack* dapat dijabarkan sebagai berikut. Terdapat sejumlah objek yang memiliki nilai atau bobot. Sejumlah objek-objek tersebut harus dimasukkan kedalam suatu wadah yang memiliki batasan dalam menampung objek-objek tersebut tanpa melebihi kemampuan dari wadah tersebut dalam menampung objek-objek yang ada. Dalam batasan tersebut, maka harus dihasilkan jumlah objek yang dapat diletakkan di dalam wadah tersebut tanpa melebihi kapasitas dari wadah tersebut (Bansal & Deep, 2012).

Permasalahan *Knapsack* sendiri memiliki sejumlah variasi, yaitu (Bansal & Deep, 2012):

- *Single Knapsack*

  Semua objek harus dimasukkan kedalam satu wadah
- *Multidimensional Knapsack*

  Terdapat lebih dari satu wadah untuk menampung objek-objek
- *Multiple-choice Knapsack*

  Objek yang ada dikelompokkan menjadi beberapa subset dan hanya satu object yang dapat dipilih

– *Bounded Knapsack*

Terdapat batasan untuk memilih objek yang tersedia

2.3. Particle Swarm Optimization

Particle swarm optimization (PSO) didasarkan pada perilaku kawanan serangga, seperti semut, rayap, lebah atau burung. Algoritma PSO meniru perilaku sosial organisme ini. Perilaku sosial terdiri dari tindakan individu dan pengaruh dari individu-individu lain dalam suatu kelompok. Kata partikel menunjukkan, misalnya, seekor burung dalam kawanan burung. (Li, dkk., 2006)

Algoritma PSO ini awalnya diusulkan oleh J. Kennedy dan R. C. Eberahrdt tahun 1995 (Kennedy, dkk., 1995). Dalam konteks optimasi multivariabel, kawanan diasumsikan mempunyai ukuran tertentu atau tetap dengan setiap partikel posisi awalnya terletak di suatu lokasi yang acak dalam ruang multidimensi. Setiap partikel diasumsikan memiliki dua karakteristik: posisi dan kecepatan. Setiap partikel bergerak dalam ruang tertentu dan mengingat posisi terbaik yang pernah dilalui atau ditemukan terhadap sumber makanan atau nilai fungsi objektif. Setiap partikel menyampaikan informasi atau posisi terbaiknya kepada partikel yang lain dan menyesuaikan posisi dan kecepatan masing-masing berdasarkan informasi yang diterima mengenai posisi yang bagus tersebut. Jadi PSO dikembangkan dengan berdasarkan pada model berikut (Kennedy, dkk., 1995):

1. Ketika seekor burung mendekati target atau makanan (atau bisa minimum atau maksimum suatu fungsi tujuan) secara cepat mengirim informasi kepada burung-burung yang lain dalam kawanan tertentu

2. Burung yang lain akan mengikuti arah menuju ke makanan tetapi tidak secara langsung
3. Ada komponen yang tergantung pada pikiran setiap burung, yaitu memorinya tentang apa yang sudah dilewati pada waktu sebelumnya.

Algoritma dasar PSO terdiri dari tiga tahap, yaitu pembangkitan posisi serta kecepatan partikel, *update velocity* (*update* kecepatan), *update position* (*update* posisi). Partikel berubah posisinya dari suatu perpindahan (iterasi) ke posisi lainnya berdasarkan pada *update velocity*. Pertama posisi x_k^i, dan kecepatan v_k^i dari kumpulan partikel dibangkitkan secara acak menggunakan batas atas (x_{max} atau v_{max}) dan batas bawah (x_{min} atau v_{min}) dari *design variable*, seperti yang ditunjukkan pada persamaan (2.1) dan (2.2) (Ariani, dkk., 2010).

$$x_0^i = x_{min} + rand\ (x_{max} - x_{min}) \quad \ldots(2.1)$$

$$v_0^i = v_{min} + rand\ (v_{max} - v_{min}) \quad \ldots(2.2)$$

Posisi dan kecepatan direpresentasikan dalam bentuk vektor dimana *n* dimensi vektor merepresentasikan jumlah dari desain variabel partikel, dengan *superscript* dan *subscript* menotasikan partikel ke-*i* pada waktu ke-*k*. Dengan proses inisialisasi ini maka kumpulan partikel dapat terdistribusi secara acak pada *design space*. Vektor seperti ditunjukkan di bawah ini (Ariani, dkk., 2010):

$$x_k^i = \left(x_k^{i1}, x_k^{i2}, \ldots, x_k^{in}\right)^T \quad \ldots(2.3)$$

$$v_k^i = \left(v_k^{i1}, v_k^{i2}, \ldots, v_k^{in}\right)^T \quad \ldots(2.4)$$

Langkah berikutnya adalah melakukan *update* kecepatan untuk semua partikel pada waktu *k+1* menggunakan fungsi objektif atau nilai *fitness* partikel saat ini pada *design space* saat waktu ke-*k*. Perumusan *update* kecepatan mencakup

beberapa parameter acak (*rand*) , untuk mendapatkan cakupan yang baik pada *design space*, tiga parameter yang mempengaruhi arah pencarian, yaitu *inertia factor (w*), *self-confidence* (*c1*), dan *swarm-confidence* (*c2*) akan digabungkan dalam satu penyajian, seperti yang ditunjukkan persamaan berikut (Ariani, dkk., 2010) :

$$v_{k+1}^{i} = w * v_k^i + c1 * rand * \left(p^i - x_k^i\right) + c2 * rand * (p_k^g - x_k^i) \quad ...(2.5)$$

Langkah terakhir dari setiap iterasi adalah *update* posisi tiap partikel dengan vektor kecepatan, seperti yang ditunjukkan pada persamaan berikut ini (Ariani, dkk., 2010):

$$x_{k+1}^{i} = x_k^i + v_{k+1}^i \quad ...(2.6)$$

Tiga tahapan diatas akan diulang sampai kriteria kekonvergenan terpenuhi, kriteria kekonvergenan sangat penting dalam menghindari penambahan fungsi evaluasi setelah solusi optimum didapatkan, namun, kriteria kekonvergenan tidak selalu mutlak diperlukan, penetapan jumlah iterasi maksimal juga dapat digunakan sebagai kondisi berhenti dari algoritma (Ariani, dkk., 2010).

Algoritma PSO juga telah berhasil diimplementasikan untuk berbagai masalah penjadwalan yang ditemui di kehidupan sehari-hari dan menghasilkan hasil perhitungan yang lebih cepat dibandingkan algoritma penjadwalan yang sudah ada sebelumnya (Pongchairerks, 2009). Algoritma PSO juga telah diimplementasikan untuk membuat *timetable* penjadwalan matakuliah di suatu universitas (Ariani, dkk., 2010; Rochman, dkk., 2013).

```
Inisialisasi posisi partikel
Repeat
   Evaluasi f(xi)
      For i = 1 : N
         Update kecepatan:
             v_i(t) = v_i(t − 1) + φC_1(p_i − x_i(t − 1)) + φC_2(g − x_i(t − 1))
         Bergerak ke posisi yang baru
             x_i(t) = x_i(t − 1) + v_i(t)
         If f(x_i) < f(p_i)
            p_i = x_i
         EndIf
         if f(x_i) < f(g)
            g = x_i
         endIf

         Update (xi,vi)
      EndFor

EndRepeat
```

Gambar 2.1 Pseudocode Algoritma PSO (Purnomo, 2013)

Untuk mengevaluasi *fitness* dari masing-masing partikel, dibutuhkan fungsi untuk mengevaluasinya. Fungsi untuk mengevaluasi *fitness* tersebut dinamakan *fitness function* atau *cost function*. Tidak ada aturan baku dalam penentuan *cost function* yang digunakan dalam algoritma PSO karena *cost function* yang digunakan harus sesuai dengan kasus yang diangkat dalam penelitian (Hu, 2006).

Algoritma PSO dipilih karena hasil jadwal yang dihasilkan memiliki hasil yang sama optimalnya dengan algoritma *heuristic* lainnya tetapi memiliki *convergence speed* yang lebih cepat dibandingkan algoritma lainnya (Nzanywayingoma & Yang, 2017). Hal tersebut membuat waktu yang dibutuhkan untuk mengeksekusi menjadi lebih cepat dan *computation cost* menjadi berkurang (Nzanywayingoma & Yang, 2017).

2.4. End-User Computing Satisfaction (EUCS)

End-User Computing Satisfaction (EUCS) merupakan salah satu metode evaluasi yang dikembangkan oleh Doll dan Torkzadeh pada tahun 1988 (Doll & Torkzadeh, 1988). Metode ini melakukan evaluasi secara keseluruhan dari sistem informasi yang ada berdasarkan pengalaman *end-user* yang menggunakan sistem informasi tersebut (Chin & Matthew, 2001).

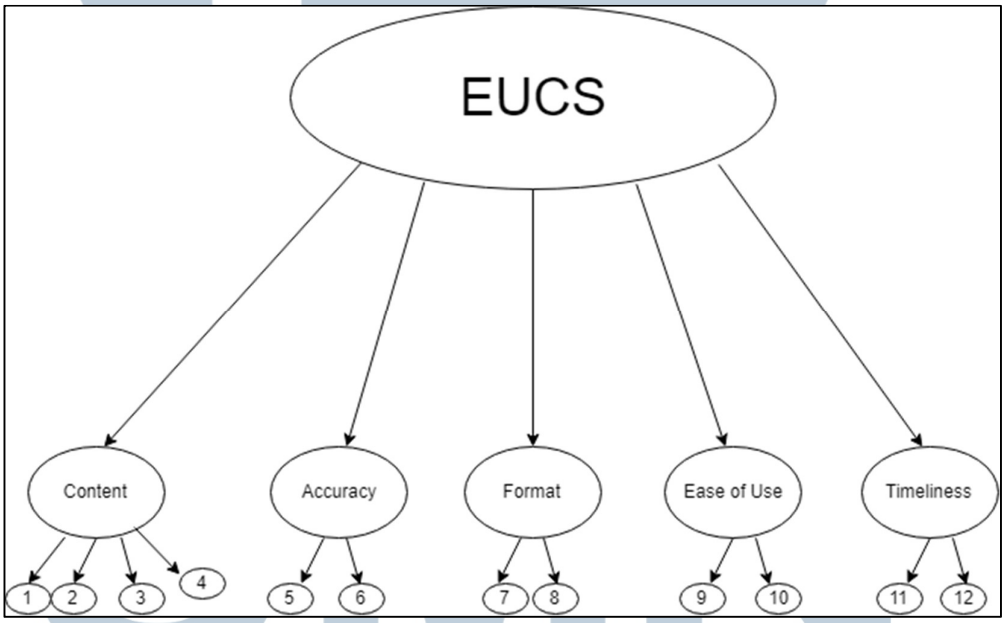
Tujuan dari metode evaluasi EUCS adalah untuk mendapatkan kesimpulan apakah sistem yang telah dikembangkan memiliki nilai guna dan dapat diterima oleh umum. Beberapa faktor atau dimensi kepuasan dalam metode evaluasi EUCS adalah sebagai berikut (Doll & Torkzadeh, 1988):

1. *Content*
2. *Accuracy*
3. Format
4. *Ease of Use*
5. *Timeliness*

Isi EUCS yang direncanakan akan digunakan dalam penelitian ini adalah sebagai berikut (Doll & Torkzadeh, 1988):

1. Apakah *web service* ini menghasilkan jadwal sebagaimana mestinya?
2. Apakah jadwal yang dihasilkan memenuhi kriteria yang ada?
3. Apakah jadwal yang dihasilkan sesuai dengan yang Anda butuhkan?
4. Apakah *web service* ini memberikan informasi yang mencukupi?
5. Apakah *web service* ini akurat?

6. Apakah Anda puas dengan akurasi dari *web service* ini?
7. Apakah *output* yang ditampilkan dalam format yang baik?
8. Apakah informasi yang ditampilkan jelas?
9. Apakah *web service* ini ramah terhadap pengguna?
10. Apakah *web service* ini mudah untuk digunakan?
11. Apakah Anda mendapatkan informasi yang Anda butuhkan tepat pada waktunya?
12. Apakah *web service* ini memberikan informasi yang teraktual?



Gambar 2.2 Model EUCS (Doll & Torkzadeh, 1988)

Gambar 2 merupakan model dari EUCS yang akan digunakan dalam penelitian ini. Pertanyaan nomor 1 sampai 4 masuk dalam dimensi *Content* dari EUCS. Pertanyaan nomor 5 dan 6 masuk dalam dimensi *Accuracy* dari EUCS. Pertanyaan nomor 7 dan 8 masuk dalam dimensi Format dari EUCS. Pertanyaan

nomor 9 dan 10 masuk dalam dimensi *Ease of Use* dari EUCS. Pertanyaan nomor 11 dan 12 masuk dalam dimensi *Timeliness* dari EUCS.

2.4.1. Skala Likert

Skala Likert merupakan salah satu skala yang digunakan secara luas untuk mengukur opini, persepsi, dan sikap secara ilmiah agar dapat diterima dan divalidasi. Skala Likert diperkenalkan oleh seorang ilmuwan sosial Amerika Serikat Rensis Likert pada tahun 1932 (Jamieson, 2017).

Cara untuk mengukur opini, persepsi, dan sikap dengan menggunakan skala Likert adalah dengan menetapkan terlebih dahulu variabel penelitian dalam penelitian tersebut. Setelah ditetapkan, maka langkah selanjutnya adalah membentuk variabel tersebut menjadi beberapa pertanyaan atau pernyataan. Jawaban yang ada tersebut memiliki rentang nilai dari negatif hingga positif (Joshi, dkk., 2015).