TUGAS AKHIR – IS184853

PENJADWALAN MATA KULIAH OTOMATIS MENGGUNAKAN ALGORITMA *GREAT DELUGE HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

AUTOMATED COURSE TIMETABLING USING GREAT DELUGE HYPER-HEURISTICS ALGORITHM WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

KHARISMA DIAH PUSPITASARI
NRP 05211540000008

Dosen Pembimbing
Ahmad Muklason, S.Kom., M.Sc., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

TUGAS AKHIR – IS184853

PENJADWALAN MATA KULIAH OTOMATIS MENGGUNAKAN ALGORITMA *GREAT DELUGE HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

KHARISMA DIAH PUSPITASARI
NRP 05211540000008

Dosen Pembimbing
Ahmad Muklason, S.Kom., M.Sc., Ph.D.

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2019

Halaman ini sengaja dikosongkan

FINAL PROJECT – IS184853

AUTOMATED COURSE TIMETABLING USING GREAT DELUGE HYPER-HEURISTICS ALGORITHM WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

KHARISMA DIAH PUSPITASARI
NRP 05211540000008

Supervisor
Ahmad Muklason, S.Kom., M.Sc., Ph.D.

INFORMATION SYSTEMS DEPARTMENT
Faculty of Information and Communication Technology
Sepuluh Nopember Institut of Technology
Surabaya 2019

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

PENJADWALAN MATA KULIAH OTOMATIS MENGGUNAKAN ALGORITMA *GREAT DELUGE HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:

KHARISMA DIAH PUSPITASARI
NRP. 05211540000008

Surabaya, Juli 2019

KEPALA
DEPARTEMEN SISTEM INFORMASI

Mahendrawathi ER, S.T., M.Sc., Ph.D
NIP. 19761011 200604 2 001

Halaman ini sengaja dikosongkan

LEMBAR PERSETUJUAN

PENJADWALAN MATA KULIAH OTOMATIS MENGGUNAKAN ALGORITMA *GREAT DELUGE HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Departemen Sistem Informasi
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :

KHARISMA DIAH PUSPITASARI
NRP. 05211540000008

Disetujui Tim Penguji : Tanggal Ujian : Juli 2019
Periode Wisuda : September 2019

Ahmad Muklason, S.Kom., M.Sc., Ph.D. **(Pembimbing I)**

Wiwik Anggraeni, S.Si., M.Kom. **(Penguji I)**

Retno Aulia Vinarti, S.Kom., M.,Kom., Ph.D. **(Penguji II)**

Halaman ini sengaja dikosongkan

PENJADWALAN MATA KULIAH OTOMATIS MENGGUNAKAN ALGORITMA *GREAT DELUGE HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

Nama Mahasiswa : Kharisma Diah Puspitasari
NRP : 05211540000008
Departemen : Sistem Informasi
Dosen Pembimbing : Ahmad Muklason, S.Kom., M.Sc., Ph.D

ABSTRAK

Permasalahan penjadwalan merupakan salah satu topik yang menarik untuk diteliti, khususnya pada bidang pendidikan seperti perguruan tinggi. Salah satu permasalahan penjadwalan yang terjadi di perguruan tinggi adalah penjadwalan mata kuliah yang erat kaitannya dengan pengalokasian sumber daya pada satu semester. Penjadwalan dilakukan untuk mengoptimalkan satu atau beberapa tujuan dengan memperhatikan berbagai batasan yang ditetapkan. Secara teoritis permasalahan optimasi penjadwalan tergolong sebagai nondeterministic polynomial-hard (NP-hard), yang mana belum ada algoritma eksak yang dapat menemukan solusi optimal dalam waktu polynomial. Tugas akhir ini membahas penyelesaian permasalahan penjadwalan mata kuliah menggunakan benchmark dataset pada International Timetabling Competition (ITC) 2019. ITC 2019 merupakan kompetisi keempat sejak pertama kali diselenggarakan pada tahun 2002. Kompetisi ini berfokus pada penjadwalan mata kuliah di perguruan tinggi dengan tujuan untuk menciptakan

dataset dunia nyata yang melimpah dan akan mendorong ke arah penelitian baru dalam teori dan praktik penjadwalan otomatis. Dalam penelitian ini digunakan algoritma Great Deluge dengan pendekatan metode hyper-heuristics untuk menyelesaikan permasalahan penjadwalan mata kuliah. Pada tugas akhir ini dihasilkan sebuah jadwal mata kuliah yang telah memenuhi batasan-batasan yang telah ditetapkan. Hasil dari tugas akhir ini menunjukkan bahwa algoritma Great Deluge dapat digunakan untuk membuat jadwal mata kuliah yang lebih optimal dibandingkan dengan penjadwalan solusi awal dan penjadwalan menggunakan algoritma hill climbing, dibuktikan dengan menghasilkan rata-rata nilai penalti yang lebih rendah yaitu 674,5 sedangkan rata-rata nilai penalti yang dihasilkan dari penjadwalan mata kuliah menggunakan algoritma hill climbing adalah 1085,3.

Kata Kunci: Penjadwalan Mata Kuliah, Penjadwalan Otomatis, Hyper-Heuristics, Algoritma Great Deluge

AUTOMATED COURSE TIMETABLING USING GREAT DELUGE HYPER-HEURISTICS ALGORITHM WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

Name	**:**	**Kharisma Diah Puspitasari**
NRP	**:**	**05211540000008**
Department	**:**	**Information Systems**
Supervisior	**:**	**Ahmad Muklason, S.Kom., M.Sc., Ph.D**

ABSTRACT

Timetabling problems are one of interesting topics to study, especially in the field of education such as universities. One of the scheduling problems that occur in college is the course scheduling that are closely related to the allocation of resources in one semester. Scheduling is done to optimize one or several objectives by taking into account the various limits set. Theoretically, the scheduling optimization problem is classified as nondeterministic polynomial-hard (NP-hard), where there is no exact algorithm that can find the optimal solution in polynomial time. This final project discusses solving scheduling problems using the benchmark dataset at the International Timetabling Competition (ITC) 2019. ITC 2019 is the fourth competition since it was first held in 2002. The competition focuses on scheduling courses in universities with the aim of creating abundant real world datasets and will lead to new research in the theory and practice of automatic scheduling. In this study, the Great Deluge algorithm is used with the hyper-

heuristics method approach to solve course scheduling problems. In this final assignment a course schedule is produced that meets the prescribed limits. The results of this final assignment show that the Great Deluge algorithm can be used to create a more optimal course schedule compared to the initial solution and scheduling using Hill Climbing algorithms, as evidenced by the lower average penalty value of 674,5 while the average penalty value resulting from the scheduling of courses using the Hill Climbing algorithm is 1085,3.

Keywords: Course Timetabling Problem, Automated Timetabling, Hyper-Heuristics, Great Deluge Algorithm

KATA PENGANTAR

Alhamdulillahi robbil 'alamin, segala puji bagi Allah SWT atas limpahan nikmat, rahmat, petunjuk, dan karunia-Nya, sehingga penulis dapat menyelesaikan laporan penelitian tugas akhir dengan judul, "**PENJADWALAN MATA KULIAH OTOMATIS MENGGUNAKAN ALGORITMA *GREAT DELUGE HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019***" yang menjadi salah satu syarat kelulusan pada Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya.

Pengerjaan tugas akhir ini tidak lepas dari bantuan, bimbingan, dukungan, maupun doa dari berbagai pihak. Oleh karena itu, izinkan penulis menyampaikan rasa terima kasih yang sebesar-besarnya kepada,

1. Kedua orang tua penulis, Bapak Setyo Satuhu Putro dan Ibu Widji Utami yang tidak pernah lelah dan tak pernah henti memberikan wejangan, dukungan, motivasi, serta doa yang tulus dan ikhlas demi yang terbaik untuk penulis.
2. Kakak penulis, Syscha Pratama Paramita, yang telah memberikan dukungan, baik secara moril maupun materil, serta arahan untuk kebaikan penulis.
3. Bapak Ahmad Muklason, S.Kom., M.Sc., Ph.D. selaku dosen pembimbing yang telah meluangkan banyak waktu dan selalu sabar membimbing serta mengarahkan penulis dalam pengerjaan tugas akhir ini.

4. Ibu Wiwik Anggraeni, S.Si., M.Kom. dan Ibu Retno Aulia Vinarti, S.Kom., M.,Kom., Ph.D. selaku dosen penguji yang telah memberikan kritik dan saran yang membangun dalam proses pengerjaan tugas akhir ini.
5. Kawan-kawan seperjuangan Cut Alna Fadhilla, Umar Rizki Kusumo Widayu, dan Narendra Puspa Adi Negara yang telah berjuang bersama, memberikan dukungan dan membantu dalam menyelesaikan tugas akhir.
6. Rekan-rekan pejuang Laboratorium RDIB dan Safira Lady Al Islami yang selalu menemani dan mendukung serta menjadi tempat diskusi penulis dalam pengerjaan tugas akhir hingga tak kenal waktu.
7. Keluarga Lannister yang selalu memberikan dukungan dan hiburan tersendiri bagi penulis. Tetaplah menjadi *beton terbang bersinar*.
8. Dan seluruh pihak yang telah membantu penulis dalam mengerjakan tugas akhir ini yang tidak mungkin disebutkan satu per satu.

Semoga semua bentuk dukungan maupun doa menjadi catatan amal kebaikan di hadapan Tuhan Yang Maha Esa. Penulis juga menyadari pengerjaan penelitian tugas akhir ini masih jauh dari kata sempurna. Oleh karena itu, penulis menerima pertanyaan, saran, dan kritik yang membangun untuk menjadi masukan penulis dan masukan penelitian selanjutnya. Semoga penelitian tugas akhir dapat memberikan manfaat bagi pembaca.

Surabaya, 17 Juli 2019

Penulis

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR KODE

BAB I
PENDAHULUAN

Dalam bab ini akan membahas mengenai latar belakang masalah, perumusan masalah, batasan masalah tujuan tugas akhir, manfaat tugas akhir, dan relevansi tugas akhir. Berdasarkan uraian pada bab ini, diharapkan mampu memberikan gambaran umum permasalahan dan pemecahan masalah pada tugas akhir.

1.1. Latar Belakang

Dalam industri pendidikan khususnya perguruan tinggi, masalah penjadwalan masih menjadi isu yang menarik dan secara luas diteliti oleh banyak peneliti di dunia. Di perguruan tinggi, penjadwalan menjadi salah satu hal penting dalam proses belajar mengajar karena seluruh kegiatan dosen dan mahasiswa akan bergantung pada jadwal yang ada, oleh sebab itu jadwal tersebut harus disusun dengan tepat sehingga tidak akan mengganggu aktivitas belajar mengajar [1] [2]. Persoalan penjadwalan erat kaitannya dengan pengalokasian sumber daya ke dalam tugas-tugas atau fungsi-fungsi tertentu. Tujuan dari dilakukannya penjadwalan adalah untuk mengoptimalkan satu atau beberapa tujuan [3].

Penjadwalan mata kuliah pada suatu perguruan tinggi merupakan masalah yang sulit untuk dipecahkan [4]. Dengan semakin bertambahnya jumlah mata kuliah yang akan dijadwalkan, sementara itu disisi lain banyak keterbatasan yang ada, maka semakin kompleks pula persoalan penjadwalan yang

dihadapi [3]. Oleh sebab itu, masalah penjadwalan ini digolongkan sebagai *NP-Hard* (*nondeterministic polynomial-hard*) yang berarti bahwa waktu yang diperlukan untuk perhitungan pencarian solusi cukup tinggi, terlebih lagi apabila ukuran dari permasalahan semakin besar dengan bertambahnya jumlah komponen dan batasan yang ditetapkan oleh institusi dimana jadwal tersebut diterapkan [1] [5]. Selain itu, secara teoritis permasalahan optimasi penjadwalan ini telah dibuktikan pada penelitian [6] sebagai *NP-Hard* yang mana belum ada algoritma eksak yang mampu menyelesaikan permasalahan tersebut dalam waktu polinomial [7] [8]. Untuk itu diperlukan adanya suatu solusi untuk membentuk penjadwalan mata kuliah yang dapat secara terstruktur membantu menyusun jadwal mata kuliah berdasarkan ketersediaan sumber daya dan tentunya dengan memenuhi batasan-batasan yang telah ditetapkan.

Selama puluhan tahun belakangan ini, permasalahan penjadwalan dalam bidang pendidikan masih mendapatkan perhatian dari banyak peneliti, khususnya di bidang riset operasi dan kecerdasan buatan [7], beberapa diantara peneliti tersebut adalah Tomas Muller, Hana Rudova dan Zuzana Mullerova. Pada tahun 2019 ini, ketiga peneliti tersebut bersama dengan beberapa sponsor menyelenggarakan kompetisi *International Timetabling Competition (ITC)* 2019. ITC 2019 merupakan kompetisi keempat yang diadakan setelah kompetisi pertama di tahun 2002, kompetisi kedua tahun 2007 dan yang ketiga pada tahun 2011 meraih kesuksesan. ITC 2019 dirancang dengan berfokus pada penjadwalan mata kuliah di perguruan tinggi dengan tujuan untuk menciptakan *datasets* dunia nyata yang melimpah dan akan mendorong ke arah penelitian baru dalam teori dan praktik penjadwalan otomatis [9].

Dalam beberapa penelitian, metode yang efektif untuk menyelesaikan permasalahan penjadwalan didominasi oleh algoritma *meta-heuristics*. Salah satu algoritma *meta-heuristics* yang pernah diterapkan untuk menyelesaikan permasalahan ini adalah *Great Deluge Algorithm* [7] [10]. Namun metode *meta-heuristics* memiliki kelemahan yaitu diperlukannya parameter *tuning* yang intensif dan memerlukan pengetahuan terkait dengan *problem domain* yang spesifik. Sehingga apabila terdapat beberapa *problem instances* yang berbeda maka diperlukan parameter *tuning* yang berbeda pula. Apabila parameter *tuning* yang digunakan sama, maka performa algoritma yang diterapkan akan bagus pada suatu *problem instance* namun sangat buruk untuk *problem instances* lainnya. Oleh sebab itu, diperlukan penerapan metode *hyper-heuristics* untuk mengatasi permasalahan penjadwalan [7].

Dalam penelitian ini digunakan algoritma *Great Deluge* dengan pendekatan metode *hyper-heuristics* untuk menyelesaikan permasalahan penjadwalan mata kuliah. *Great Deluge* merupakan algoritma *meta-heuristics* yang dianalogikan seperti ketika dalam banjir besar seseorang yang mendaki bukit akan mencoba untuk bergerak ke arah manapun yang tidak membuat kakinya basah dengan harapan menemukan jalan naik ketika permukaan air juga naik [11]. Algoritma *Great Deluge* dipilih karena dalam beberapa penelitian [10] [11] [12] terbukti menghasilkan performa yang lebih baik dibandingkan dengan penelitian terdahulu dengan metode dan algoritma yang berbeda.

Sementara itu pada penelitian ini data yang digunakan untuk menguji coba metode *Great Deluge Hyper-heuristics* adalah

benchmark data pada ITC 2019. Dataset pada kompetisi ini dipilih karena berfokus pada permasalahan penjadwalan mata kuliah di perguruan tinggi dengan tujuan utamanya adalah untuk meminimalisasi biaya yang diperlukan untuk semua sumber daya yang tersedia dengan memenuhi batasan-batasan yang telah ditetapkan. Selain itu dataset ini dipilih karena memiliki banyak *problem instances* yang mana masing-masingnya mempunyai jumlah dan jenis data yang berbeda-beda [9], sehingga diharapkan hasil uji coba metode yang akan diterapkan lebih akurat. Melalui penelitian ini dengan penerapan algoritma *Great Deluge* disertai pendekatan metode *hyper-heuristics* diharapkan dapat membantu menyelesaikan permasalahan penjadwalan mata kuliah di perguruan tinggi dengan ukuran data yang kompleks dan hasil yang kompetitif dibandingkan dengan solusi menggunakan algoritma lainnya.

1.2. Perumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, permasalahan yang akan dibahas pada tugas akhir ini adalah sebagai berikut.

1. Bagaimana pembuatan model matematis untuk permasalahan penjadwalan mata kuliah pada studi kasus ITC 2019?
2. Bagaimana penerapan algoritma *Great Deluge Hyper-heuristics* untuk menyelesaikan permasalahan penjadwalan mata kuliah pada studi kasus ITC 2019?
3. Bagaimana performa algoritma *Great Deluge Hyper-heuristics* dalam menyelesaikan permasalahan penjadwalan mata kuliah?

1.3. Batasan Pengerjaan Tugas Akhir

Batasan masalah yang dikaji dalam penelitian ini adalah sebagai berikut:

1. Data yang digunakan dalam penjadwalan mata kuliah adalah *benchmark dataset* pada kompetisi ITC 2019 dengan tipe kelompok data adalah *test instances* dan *early data instances*.
2. Penjadwalan yang dilakukan hanya terbatas pada penjadwalan mata kuliah, tanpa menjadwalkan mahasiswa ke masing-masing mata kuliah yang diambil.
3. Penyelesaian penjadwalan mata kuliah menggunakan algoritma *Great Deluge Hyper-heuristics*.
4. Aplikasi dibangun dengan menggunakan Bahasa Java.

1.4. Tujuan Tugas Akhir

Berdasarkan latar belakang dan rumusan masalah yang telah diuraikan, tujuan dari penelitian ini adalah sebagai berikut:

1. Membuat model matematis untuk permasalahan penjadwalan mata kuliah pada studi kasus ITC 2019.
2. Mengetahui hasil implementasi algoritma *Great Deluge Hyper-heuristics* untuk penyelesaian penjadwalan mata kuliah pada studi kasus ITC 2019.
3. Menganalisis performa algoritma *Great Deluge Hyper-heuristics* dalam menyelesaikan permasalahan penjadwalan mata kuliah.

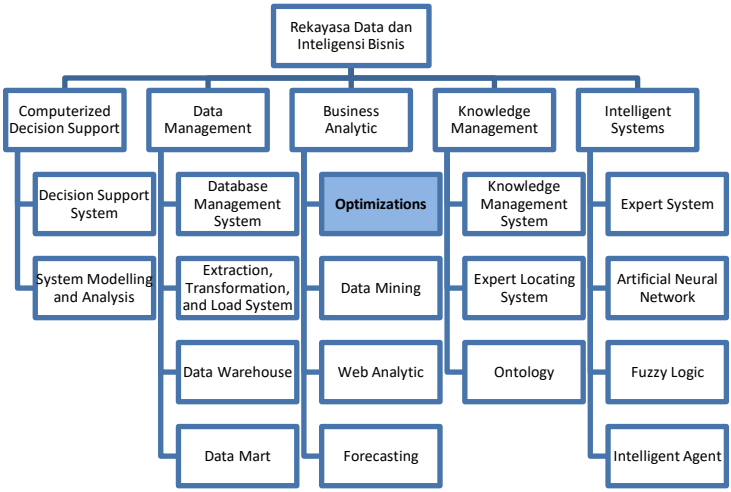
1.5. Manfaat Tugas Akhir

Dari pengerjaan tugas akhir ini, manfaat yang dapat diberikan antara lain:

1. Menjadi referensi atau rujukan bagi penelitian selanjutnya terkait dengan penjadwalan mata kuliah.
2. Membantu menyelesaikan permasalahan penjadwalan mata kuliah dengan menggunakan algoritma *Great Deluge Hyper-heuristics*.
3. Menghasilkan solusi untuk ITC 2019 yang kompetitif dengan solusi *benchmark* dan algoritma yang digunakan dapat diterapkan untuk menyelesaikan permasalahan penjadwalan lainnya.

1.6. Relevansi

Penelitian tugas akhir ini memiliki relevansi dengan *roadmap* penelitian Laboratorium Rekayasa Data dan Inteligensi Bisnis (RDIB) pada pokok penelitian *Business Analytics*, khususnya bidang optimasi. Sementara itu dalam pengerjaan tugas akhir ini didukung dan berkaitan dengan mata kuliah yang diajarkan di Departemen Sistem Informasi ITS yaitu mata kuliah Bahasa Pemrograman dan mata kuliah Optimasi Kombinatorial dan *Heuristic*. Laboratorium RDIB memiliki tujuan untuk menjadi pusat penelitian terkait pemanfaatan data yang mendukung analisis bisnis dan organisasi untuk bisa ditransformasi menjadi informasi bermakna serta pengetahuan sehingga berguna dalam pengambilan keputusan.



Gambar 1.1 Bidang Keilmuan Laboratorium Rekayasa Data dan Inteligensi Bisnis

Halaman ini sengaja dikosongkan

BAB II
TINJAUAN PUSTAKA

Tinjauan pustaka ini berisikan mengenai penelitian sebelumnya dan penjelasan terkait dengan dasar teori yang dijadikan sebagai acuan dalam pengerjaan tugas akhir ini.

2.1. Studi Sebelumnya

Dalam studi ini digunakan beberapa penelitian terdahulu sebagai acuan, pedoman, dan referensi penulis untuk memperkaya teori dalam melaksanakan proses-proses dalam pengerjaan tugas akhir. Beberapa penelitian sebelumnya disajikan dalam Tabel 2.1 yang berisi informasi mengenai deskripsi umum dan keterkaitan penelitian terhadap tugas akhir.

Tabel 2.1 Penelitian Terdahulu

Acuan 1	
Judul	A Time-Predefined Approach To Course Timetabling
Penulis;Tahun	Edmund Burke, Yuri Bykov, James Newall, Sanja Petrovic; 2003
Deskripsi Umum	Penelitian [10] menjelaskan mengenai bagaimana menyelesaikan permasalahan penjadwalan mata kuliah dengan pengembangan variasi lanjutan dari algoritma *Great Deluge local search*. Berdasarkan percobaan yang dilakukan menggunakan

	benchmark instances dari permasalahan penjadwalan mata kuliah didapatkan bahwa 8 dari 23 *datasets* yang ada, algoritma *Great Deluge* menghasilkan hasil terbaik diantara 21 algoritma lainnya.
Keterkaitan Penelitian	Penelitian [10] dapat menjadi referensi penelitian terkait dengan penjadwalan mata kuliah menggunakan algoritma *Great Deluge*.
Acuan 2	
Judul	New Optimization Heuristics – The Great Deluge Algorithm and The Record-to-Record Travel
Penulis;Tahun	Gunter Dueck; 1991
Deskripsi Umum	Penelitian [11] membahas terkait metode *threshold accepting* (TA) dan *simulated annealing* (SA) untuk optimasi diskrit. Dengan menggunakan prinsip TA, peneliti menemukan dua *heuristics* baru untuk optimasi yang memiliki performa lebih baik yaitu menggunakan *Great Deluge Algorithm* (GDA) dan *Record-to-Record Travel* (RRT). Struktur algoritma ini menyerupai metode

	threshold accepting (TA) dan *simulated annealing* (SA). Kualitas hasil komputasi yang diperoleh menunjukkan bahwa algoritma yang diterapkan berperilaku sama baiknya dengan *threshold accepting* (TA) dan lebih baik daripada *simulated annealing* (SA).
Keterkaitan Penelitian	Penelitian [11] dapat menjadi referensi penelitian terkait dengan metode optimasi menggunakan algoritma *Great Deluge*.
Acuan 3	
Judul	An Extended Implementation of the Great Deluge Algorithm for Course Timetabling
Penulis;Tahun	Paul McMullan; 2007
Deskripsi Umum	Penelitian [13] memperkenalkan versi perluasan dari algoritma *Great Deluge* untuk menyelesaikan permasalahan penjadwalan mata kuliah, menggunakan *heuristic* pencarian *neighbourhood* demi mendapatkan solusi dalam waktu yang relatif singkat. Dengan menggunakan *benchmark datasets*, pada penelitian [13] dihasilkan peningkatan sebesar 60% dari

	beberapa hasil yang telah dipublikasikan saat ini.
Keterkaitan Penelitian	Penelitian [13] dapat menjadi referensi penelitian terkait penerapan algoritma *Great Deluge* untuk penyelesaian permasalahan penjadwalan mata kuliah.
Acuan 4	
Judul	A Survey of Hyper-heuristics
Penulis;Tahun	Edmund K. Burke, Matthew Hyde, Graham Kendall, Ender Ozcan, Gabriela Ochoa, Rong Qu; 2009
Deskripsi Umum	Penelitian [14] membahas terkait metode *hyper-heuristics* termasuk asal dan akar intelektualnya, penjelasan terperinci mengenai jenis-jenis pendekatan utama, dan tinjauan umum dari beberapa bidang terkait.
Keterkaitan Penelitian	Penelitian [14] dapat menjadi referensi penelitian terkait dengan metode *hyper-heuristics.*
Acuan 5	
Judul	Solver Penjadwal Ujian Otomatis Dengan Algoritma *Maximal Clique* dan *Hyper-heuristics*

Penulis;Tahun	Ahmad Muklason;2017
Deskripsi Umum	Penelitian [7] membahas usulan metode baru yaitu metode heuristic sekuensial berdasarkan konsep *maximal clique* pada teori graf digabung dengan metode *hyper-heuristic* untuk menyelesaikan permasalahan penjadwalan ujian (*examination timetabling*).
Keterkaitan Penelitian	Penelitian [7] dapat menjadi referensi penelitian terkait penjadwalan otomatis melalui pendekatan *hyper-heuristics*.

Tabel 2.1 menampilkan penelitian yang menjadi acuan dalam pengerjaan tugas akhir. Hasil identifikasi menjelaskan bahwa penelitian penelitian tersebut dapat digunakan sebagai acuan penulis karena memiliki keterkaitan dengan kasus dan metode yang digunakan.

2.2. Dasar Teori

Dasar teori berisi teori-teori yang mendukung dan berkaitan dengan pengerjaan tugas akhir.

2.2.1. Penjadwalan

Suatu jadwal merupakan sekumpulan pertemuan dalam waktu tertentu yang telah ditetapkan sebelumnya. Sementara itu pertemuan merupakan kombinasi dari sumber daya yang beberapa diantaranya ditentukan oleh masalah dan beberapa lainnya mungkin dialokasikan sebagai bagian dari solusi [15].

Penjadwalan sendiri merupakan salah satu kajian dalam bidang permasalahan optimasi kombinatorik yang mana secara umum mengkaji secara matematis dalam hal penyusunan, pengelompokan, pengurutan, atau pemilihan objek diskrit yang biasanya terdapat dalam jumlah yang terbatas (*finite number*) [7] [16] dan tentunya memenuhi batasan-batasan tertentu.

Pada literatur penjadwalan, batasan biasanya dikategorikan dalam dua tipe yaitu *hard constraints* (*required*) dan *soft constraints*. *Hard constraints* tidak dapat dilanggar dalam kondisi apa pun, misalnya mata kuliah yang saling bertentangan (yang melibatkan sumber daya yang sama seperti ruangan atau mahasiswa) tidak dapat dijadwalkan secara bersamaan. Suatu jadwal yang memenuhi seluruh *hard constraints* dapat dikatakan layak (*feasible*) [9]. Sementara itu *soft constraints* merupakan batasan yang diinginkan namun tidak bersifat mutlak karena dalam praktiknya biasanya tidak mungkin suatu solusi yang layak (*feasible*) dapat memenuhi seluruh *soft constraints*. Hal ini dikarenakan *soft constraints* sangat bervariasi bergantung pada tipe dan kepentingan dari setiap instansi. Kualitas dari suatu jadwal diukur dengan cara memeriksa sejauh mana *soft constraints* dilanggar pada solusi yang dihasilkan [9] [17].

2.2.2. *Course Timetabling Problem*

Course Timetabling Problem merupakan masalah *NP-Hard* yang berarti bahwa waktu yang diperlukan untuk perhitungan pencarian solusi meningkat secara eksponensial seiring dengan bertambahnya ukuran dari permasalahan tersebut [5]. Masalah dalam mengembangkan penjadwalan mata kuliah untuk institusi pendidikan terdiri atas alokasi sekumpulan mata kuliah

yang ditawarkan oleh perguruan tinggi pada waktu dan ruang tertentu dimana setiap pengajar, mahasiswa dan ruangan yang ada tidak boleh digunakan lebih dari sekali pada periode waktu tertentu dan jumlah mahasiswa yang mengikuti kelas tidak boleh melebihi kapasitas ruangan [18].

2.2.3. *International Timetabling Competition (ITC) 2019*

ITC terbuka untuk semua calon *timetabler*. ITC 2019 merupakan kompetisi keempat yang diadakan setelah kompetisi pertama di tahun 2002, kompetisi kedua tahun 2007 dan yang ketiga pada tahun 2011 meraih kesuksesan [9]. Kompetisi ITC diperkenalkan untuk menarik para peneliti dalam mengembangkan dan menguji teknik-teknik terdepan pada arena yang kompetitif [19]. Tujuan penting dari kompetisi ini adalah menghasilkan pendekatan baru untuk permasalahan yang berkaitan, dengan cara menarik peneliti dari seluruh bidang penelitian. Seperti halnya banyak kasus di masa lalu, kemajuan yang signifikan telah didapatkan dalam bidang penelitian melalui cara pendekatan terhadap multi-disiplin. Tujuan penting lainnya yaitu untuk menutup kesenjangan yang saat ini ada antara penelitian dan praktik dalam bidang penelitian operasional [9]. Sementara itu ITC 2019 dibangun dengan tujuan untuk memotivasi penelitian selanjutnya terkait penjadwalan mata kuliah yang kompleks pada perguruan tinggi [20].

ITC 2019 dirancang dengan berfokus pada penjadwalan mata kuliah di perguruan tinggi dengan tujuan untuk menciptakan *dataset* dunia nyata yang melimpah dan akan mendorong ke arah penelitian baru dalam teori dan praktik penjadwalan otomatis. Pada ITC 2019 ini, kompetisi berfokus pada

permasalahan yang harus diselesaikan yaitu membentuk jadwal mata kuliah (*course timetable*) perguruan tinggi yang optimal dalam hal waktu (*times*), penggunaan ruangan (*rooms*) dan mahasiswa (*students*) yang berkaitan dengan sekumpulan mata kuliah (*courses*) [9].

Pada ITC 2019, permasalahan yang diangkat terdiri atas penjadwalan mata kuliah dan permintaan mata kuliah oleh mahasiswa. Setiap mata kuliah memiliki satu atau lebih kelas yang bisa saja terdiri atas satu perkuliahan dan beberapa seminar. Penyusunan mata kuliah yang terstruktur sangat diperlukan untuk menentukan bagaimana mahasiswa menghadiri kelas. Dalam seminggu setiap kelas dapat memiliki beberapa pertemuan yang harus dijadwalkan pada waktu dan ruangan yang sama [9].

Tujuan utama dari permasalahan ini adalah untuk menemukan waktu, ruang dan mahasiswa yang tepat untuk seluruh kelas yang mana setiap kelas memiliki sekumpulan waktu dan ruang yang boleh digunakan dengan meminimalisasi biaya yang diperlukan. Semua mahasiswa harus dijadwalkan secara tepat pada kelas-kelas yang mereka minta. Pada penjadwalan semacam ini tidak dipungkiri bahwa dapat terjadi konflik mahasiswa yaitu ketika seorang mahasiswa tidak dapat menghadiri kelas-kelas mereka dikarenakan jadwal kelas yang tumpang tindih maka sudah seharusnya konflik seperti ini dapat diminimalkan [9].

2.2.4. *Dataset* ITC 2019

Dalam *dataset* ITC 2019 terdapat permasalahan yang terdiri atas ruangan (*rooms*), kelas-kelas (*classes*) dengan struktur mata kuliah (*courses*), batasan distribusi (*distribution*

constraints), dan mahasiswa (*students*) dengan permintaan mata kuliah mereka. Tujuan yang akan dicapai disini adalah menempatkan mata kuliah pada waktu dan ruangan yang tersedia sebagaimana membagi mahasiswa dalam kelas-kelas berdasarkan mata kuliah yang mereka minta, tentunya dengan memenuhi berbagai batasan yang ada [9]. Namun sebagaimana yang telah dijelaskan pada subbab 1.3 bahwa dalam penelitian tugas akhir ini hanya terbatas pada penetapan mata kuliah di waktu dan ruangan yang tersedia, tanpa melakukan penjadwalan untuk masing-masing mahasiswa. Terdapat 4 kelompok *data instances* yang digunakan dalam ITC 2019 yaitu *test*, *early*, *middle*, dan *late*. Sementara itu, ruang lingkup penelitian ini hanya mengacu pada kelompok *test instance* dan *early data instance*. Karakteristik dari *datasets* ITC 2019 dapat dilihat pada Tabel 2.2.

Tabel 2.2 Karakteristik *Dataset* ITC 2019

Nama *Instances*	**Jumlah *Courses***	**Jumlah *Classes***	**Jumlah *Rooms***	**Jumlah *Students***	**Jumlah *Distribution Constraints***
Test Instance 2 (Tiny)	19	20	62	0	2
Test Instance 3 (Small 1)	48	127	46	0	144
Test Instance 4 (Small 2)	44	174	13	2002	102
Test Instance 5 (Medium)	602	802	56	27881	329

Nama *Instances*	**Jumlah *Courses***	**Jumlah *Classes***	**Jumlah *Rooms***	**Jumlah *Students***	**Jumlah *Distribution Constraints***
Test Instance 6 (Large)	1035	2417	207	29514	2002
Early Instance 1	340	1239	80	1641	1220
Early Instance 2	272	1852	44	2116	2690
Early Instance 3	353	983	62	3018	1251
Early Instance 4	1206	2641	214	0	2902
Early Instance 5	544	882	90	3666	3947
Early Instance 6	228	575	35	1543	740
Early Instance 7	226	561	44	865	400
Early Instance 8	1089	2526	70	2938	2026
Early Instance 9	687	1001	75	27018	634

Nama *Instances*	**Jumlah *Courses***	**Jumlah *Classes***	**Jumlah *Rooms***	**Jumlah *Students***	**Jumlah *Distribution Constraints***
Early Instance 10	36	711	15	0	501

Pada permasalahan ini, seluruh biaya dispesifikasikan sebagai penalti yang diinisiasi dalam bentuk bilangan bulat tidak negatif (*non-negative integer penalties*). Semakin kecil nilai penalti berarti menandakan semakin kecil pelanggaran yang dilakukan dan hal tersebut mengindikasikan hasil yang semakin baik, sementara itu nilai nol pada penalti merepresentasikan terpenuhinya kepuasan [9]. Dalam *dataset* ITC 2019 terdapat 5 *test instances* dan 10 *early instances* dengan 4 variabel keputusan yaitu *times*, *rooms*, *courses*, dan *students* serta *distribution constraints* [9].

2.2.5. *Hyper-Heuristics*

Hyper-heuristics terdiri atas serangkaian pendekatan untuk mengotomatisasi desain guna menyelesaikan masalah pencarian komputasi yang rumit. Istilah *hyper-heuristics* pertama kali diciptakan pada awal tahun 2000 yang merujuk pada gagasan "heuristik untuk memilih heuristik" dalam konteks optimasi kombinatorial. Namun gagasan untuk mengotomatisasi desain heuristik gabungan bukanlah hal baru karena pada tahun 1960 sudah ada [14].

Beberapa waktu terakhir definisi *hyper-heuristics* mulai diperluas, yang merujuk pada metode pencarian atau mekanisme pembelajaran untuk memilih atau menghasilkan heuristik yang dapat digunakan pada pemecahan masalah

pencarian komputasional [21]. Terdapat dua tujuan penggunaan pendekatan *hyper-heuristics* yaitu (1) memilih dan mengkombinasikan *heuristic* yang lebih sederhana, dan (2) menghasilkan *heuristic* baru dengan komponen *heuristic* yang sudah ada [7].

Selain pendekatan *hyper-heuristics*, terdapat pula pendekatan *meta-heuristics*. Fitur yang membedakan adalah *hyper-heuristics* beroperasi pada ruang pencarian *heuristics* (tidak bersinggungan langsung dengan *solution space*) yang hanya bersinggungan dengan *lower-level heuristics*, sementara itu *meta-heuristics* beroperasi secara langsung pada ruang pencarian solusi (*solution space*). Dengan *search space* yang lebih tinggi membuat pendekatan *hyper-heuristics* tidak bergantung pada *problem domain* tertentu saja, sehingga tidak diperlukan *parameter tuning* untuk setiap *problem domain* [7]. Ketika menggunakan pendekatan *hyper-heuristics* lebih diutamakan untuk menemukan metode atau urutan heuristik yang tepat daripada mencoba memecahkan masalah secara langsung [14].

2.2.6. Algoritma *Great Deluge*

Algoritma *Great Deluge* diperkenalkan oleh Dueck [22] sebagai suatu alternatif dari *Simulated Annealing*. Standar algoritma ini telah diperluas agar mirip dengan *Simulated Annealing* dalam penjadwalan. Tujuan dari pendekatan ini adalah meningkatkan kecepatan dimana suatu solusi optimal ditemukan dan di waktu yang sama memanfaatkan mekanismenya untuk menghindari jebakan optima lokal dan berusaha memberikan pencarian solusi yang lebih luas [13] [23].

Pada algoritma *Great Deluge* digunakan kondisi *boundary* untuk menerima solusi yang lebih buruk. *Boundary* atau batas awal tersebut ditetapkan dengan nilai yang sedikit lebih tinggi daripada biaya solusi awal, dan akan dikurangi secara bertahap melalui proses perbaikan. Solusi baru hanya bisa diterima apabila solusi tersebut meningkat dalam hal evaluasi biaya atau setidaknya masih dalam batas nilai yang ditetapkan (*boundary value*) [13]. Kode Program 2.1 merupakan *pseudocode* umum algoritma *Great Deluge* [24].

```
1: Set initial solution, Sol;
2: Calculate the initial cost function value, f(Sol);
3: Set best solution, Sol_best ← Sol;
4: Set estimated quality of final solution, estimated quality;
5: Set number of interations, NumOfIte;
6: Set initial level, level ← f(Sol);
7:    Set    decreasing    rate    β    =    ((f(Sol)-estimated
quality)/(NumOfIte));
8: Set iteration ← 0;
9: Set not_improving_counter ← 0;
10: do while (iteration < NumOfIte)
11: Define neighbourhood of Sol by randomly assigning course
to a valid timeslot to generate a new solution, Sol*;
12: Calculate f(Sol*);
13: if(f(Sol*)<f(Sol_best))
14:       Sol ← Sol*;
15:       Sol_best ← Sol*;
16:       not_improving_counter ← 0;
17:    else
18:       if(f(Sol*)≤ level)
19:         Sol ← Sol*;
20:         not_improving_counter ← 0;
21:       else
22:         Increase not_improving_counter by 1;
23:         if(not_improving_counter ==
            not_improving_length_GDA)
24:         exit;
25:    level = level - β;
26:    Increase iteration by 1;
27:  end do;
```

Kode Program 2.1 Pseudocode Algoritma *Great Deluge*

Halaman ini sengaja dikosongkan

BAB III
METODOLOGI PENELITIAN

Pada bab ini akan dijelaskan mengenai tahap-tahap dan rencana jadwal kegiatan yang dilakukan dalam pengerjaan tugas akhir beserta deskripsi dan penjelasan tiap tahapan tersebut.

3.1. Metodologi Penelitian

Tahapan pelaksanaan tugas akhir dapat dilihat pada Bagan 3.1.

Bagan 3.1 Tahapan Pelaksanaan Tugas Akhir

3.2. Tahapan Pelaksanaan Tugas Akhir

Tahapan pelaksanaan tugas akhir akan menjelaskan segala sesuatu yang akan dikerjakan oleh penulis atau merupakan langkah-langkah pengerjaan tugas akhir.

3.2.1. Identifikasi Masalah

Pada tahap ini dilakukan identifikasi masalah secara menyeluruh dengan cara menganalisis studi kasus guna mendapatkan wawasan rinci terkait topik permasalahan yang

akan digunakan dalam penelitian. Hasil dari tahap ini yaitu berupa topik permasalahan penjadwalan pada ITC 2019 yang kemudian akan dijadikan masukan pada tahap berikutnya.

3.2.2. Studi Literatur

Dari topik permasalahan yang didapat pada tahap pertama, selanjutnya dilakukan studi literatur berdasarkan topik permasalahan tersebut sebagai dasar dari penelitian. Studi literatur dilakukan dengan cara mencari, mengumpulkan dan mempelajari berbagai pustaka yang memuat penelitian-penelitian yang telah dilakukan sebelumnya terkait dengan topik permasalahan dan metode penyelesaiannya. Tahapan studi literatur berguna untuk memperkuat pemahaman peneliti mengenai topik permasalahan yang akan diteliti, mengetahui dasar-dasar teori yang mendukung konsep dan model untuk penyelesaian permasalahan yang tentunya berkaitan dengan penelitian ini.

3.2.3. Pengambilan dan Analisis Dataset

Setelah data didapatkan, tahap selanjutnya yaitu memahami dan menganalisis data termasuk pemahaman terdapat format *dataset* dan batasan yang digunakan serta model matematis dari ITC 2019. Pemahaman dan analisis juga dilakukan melalui berbagai studi literatur terkait dengan dataset ITC 2019.

3.2.4. Pemodelan Matematis Permasalahan

Setelah mendapatkan seluruh data yang diperlukan untuk menyelesaikan permasalahan, maka perlu dilakukan pembuatan model matematis dari data-data tersebut. Pembuatan model matematis ini dilakukan dengan tujuan untuk menentukan variabel keputusan, batasan dan fungsi tujuan dari permasalahan yang diangkat. Dari penjelasan pada subbab 2.2.4

dapat diketahui bahwa *datasets* ITC 2019 memiliki variabel keputusan, batasan dan fungsi tujuan dengan penjelasan untuk masing-masingnya sebagai berikut.

a. Variabel keputusan
   Variabel keputusan yang biasa disimbolkan dengan huruf *x* adalah variabel yang menentukan nilai dari fungsi tujuan [3]. Variabel keputusan pada penelitian tugas akhir ini berdasarkan studi kasus ITC 2019 yaitu menempatkan *courses* pada *time* dan *rooms* yang tersedia.

b. *Distribution constraints*
   Batasan merupakan ketetapan yang harus dipenuhi (*hard constraints*) dan tidak harus dipenuhi (*soft constraints*) dan menjadi acuan dalam proses penyusunan jadwal [1]. Pada studi kasus ITC 2019 batasan yang ditetapkan berupa *distribution constraints* yang mana dapat berlaku sebagai *hard costraints* atau *soft constraints* bergantung pada setiap *problem domain*. Tabel 3.1 menampilkan *distribution constraints* pada *datasets* ITC 2019.

c. Fungsi Tujuan
   Fungsi tujuan biasa dinotasikan dengan *f(x)* merupakan fungsi yang akan dioptimasi, dalam hal ini dicari nilai penalti minimumnya [3]. Pada studi kasus ITC 2019 fungsi tujuan yang akan dicapai dalam pembuatan model yaitu meminimalkan biaya untuk segala sumber daya yang digunakan pada permasalahan penjadwalan mata kuliah dengan memenuhi berbagai batasan yang telah ditetapkan.

Tabel 3.1 Daftar Tipe *Distribution Constraints*

Batasan	Definisi
SameStart	Semua kelas dalam batasan ini harus dimulai pada waktu yang bersamaan
SameTime	Semua kelas dalam batasan ini harus dijadwalkan dalam suatu rentang waktu yang bersamaan
DifferentTime	Semua kelas dalam batasan ini tidak boleh dijadwalkan dalam suatu rentang waktu yang bersamaan
SameDays	Semua kelas dalam batasan ini harus dijadwalkan pada hari yang sama
DifferentDays	Semua kelas dalam batasan ini tidak dapat dijadwalkan pada hari yang sama
SameWeeks	Semua kelas dalam batasan ini harus ditawarkan dalam minggu yang sama pada satu semester
DifferentWeeks	Semua kelas dalam batasan ini tidak boleh mengadakan kelas di minggu yang sama pada satu semester
SameRoom	Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang sama
DifferentRoom	Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang berbeda
Overlap	Dua kelas dalam batasan ini harus memiliki waktu yang saling tumpang tindih.
NotOverlap	Tidak boleh ada dua kelas dalam batasan ini yang saling tumpang tindih.
SameAttendees	Semua kelas dalam batasan ini harus bertemu pada waktu dan lokasi tertentu sehingga siswa atau dosen dapat menghadirinya
Precedence	Batasan ini menetapkan daftar urutan, dimana kelas yang terdaftar harus

Batasan	Definisi
	dijadwalkan pada minggu, hari, atau *timeslot* yang lebih awal
WorkDay (S)	Batasan ini mencegah atau memberikan penalti ketika terdapat pasangan kelas yang ada dalam daftar, dimana waktu antara mulainya kelas pertama dan berakhirnya kelas terakhir, melebihi S *timeslot*
MinGap (G)	Dua kelas yang dijadwalkan pada hari yang bersamaan harus memiliki setidaknya *timeslot* sebesar G yang terpisah (antara berakhirnya kelas pertama dan mulainya kelas kedua)
MaxDays (D)	Kelas-kelas pada batasan ini tidak boleh dijadwalkan ke lebih dari D hari dalam satu minggu
MaxDayLoad (S)	Batasan ini membatasi jumlah total waktu yang ditetapkan untuk sekumpulan kelas tidak lebih dari S *timeslot* tiap hari selama satu semester
MaxBreaks (R, S)	Batasan ini membatasi jumlah jeda antar kelas selama sehari yang melebihi S *timeslot*, tidak boleh lebih dari R per hari
MaxBlock (M, S)	Batasan ini membatasi jumlah waktu (M slot) dimana sekumpulan kelas mungkin dijadwalkan secara berurutan, yang mana masing-masingnya dipisahkan oleh waktu yang tidak melebihi S slot

3.2.5. Pembentukan Solusi Awal

Implementasi dilakukan dengan menggunakan masukan berupa *benchmark dataset* pada kompetisi ITC 2019. Program akan membaca masukan berupa *file dataset* dengan ekstensi .xml yang kemudian diproses menggunakan algoritma *sequential heuristics* untuk menentukan solusi awal. Kemudian solusi awal

tersebut akan dioptimasi dengan menggunakan algoritma *hyper-heuristics*.

3.2.6. Implementasi Algoritma *Great Deluge Hyper-Heuristics*

Dalam kelompok *test instance* terdapat 5 *instance* sementara itu pada *early data instance* terdapat 10 *instance* yang tersimpan dalam *file* yang berbeda. Sementara itu metode optimasi dilakukan dengan menerapkan algoritma *Great Deluge* dengan bahasa pemrograman Java. Proses penerapan algoritma *Great Deluge* diawali dengan menetapkan solusi awal, menghitung biaya awal dari solusi awal tersebut, menetapkan *initial level* dan *decreasing rate* untuk kemudian dilakukan iterasi. Pada iterasi dilakukan penetapan *neighborhood* dengan cara menempatkan mata kuliah secara acak ke dalam *timeslot* guna mendapatkan solusi baru untuk selanjutnya menghitung biaya baru dari solusi baru tersebut, dan dilakukan perbandingan dengan biaya awal untuk menetapkan diterima atau tidaknya solusi baru tersebut. Apabila solusi baru diterima maka solusi awal akan digantikan dengan solusi baru, sementara apabila tidak diterima maka akan dilakukan penuruan *initial level* dengan *decreasing rate*, untuk selanjutnya dilakukan iterasi kembali sampai terpenuhi *stopping criteria* berupa jumlah iterasi yang telah ditetapkan sebelumnya. Luaran dari tahapan ini adalah berupa model solusi untuk selanjutnya dilakukan uji coba menggunakan *benchmark dataset* dari ITC 2019.

3.2.7. Uji Coba Implementasi

Uji coba dilakukan dengan memasukkan *dataset* ke dalam pemodelan yang dihasilkan dan mengevaluasi hasilnya apakah sudah memenuhi *hard constraints* dan *soft constraints* yang

ditetapkan. Apabila hasilnya belum optimal, maka dilakukan peninjauan kembali terhadap model yang didapatkan. Tujuan yang ingin dicapai pada tahap ini adalah untuk mendapatkan penjadwalan yang optimal dengan mempertimbangkan biaya yang dikeluarkan untuk seluruh sumber daya yang diperlukan. Nilai penalti digunakan untuk mengukur kualitas jadwal yang dihasilkan. Semakin rendah nilai penalti maka dapat dikatakan bahwa semakin baik pula performa algoritma yang digunakan.

3.2.8. Analisis Hasil dan Kesimpulan

Setelah uji coba, pada tahap ini dilakukan analisis terhadap hasil performa algoritma *Great Deluge* dengan cara mengamati dan menghitung keluaran dari setiap iterasi maupun data yang menjadi masukan. Kemudian dilakukan penarikan kesimpulan dengan membandingkan hasil analisis dari percobaan yang telah dilakukan.

3.2.9. Penyusunan Laporan Tugas Akhir

Penyusunan laporan tugas akhir merupakan tahapan terakhir dari pelaksanaan tugas akhir ini. Pada tahap ini dilakukan pengumpulan semua dokumentasi mulai dari masukan, proses dan keluaran dari penelitian untuk selanjutnya dibuat dalam bentuk laporan tugas akhir. Pendokumentasian ini dilakukan dengan mengikuti format yang telah ditetapkan oleh laboratorium Rekayasa Data dan Inteligensia Bisnis (RDIB) serta aturan yang berlaku di Departemen Sistem Informasi ITS.

Halaman ini sengaja dikosongkan

BAB IV
PERANCANGAN

Dalam bab ini akan dijelaskan persiapan perancangan terkait dengan data yang digunakan dan konversi model matematis ke struktur data sebagai tahap persiapan implementasi algoritma.

4.1. Pemahaman Data

Data yang digunakan pada penelitian ini adalah *benchmark dataset* yang disediakan dalam ITC 2019. Dataset tersebut dapat diperoleh dari situs resmi ITC 2019 yaitu *www.itc2019.org*.

Sebagaimana telah dijelaskan pada subbab 2.2.4, dataset ITC 2019 terdiri dari 4 kelompok *data instances* yaitu *test, early, middle*, dan *late*. Sementara itu, ruang lingkup penelitian tugas akhir ini hanya mengacu pada kelompok *test instance* yang terdiri atas 5 *instances* yaitu *test instance 2 (tiny)*, *test instance 3 (small 1)*, *test instance 4 (small 2)*, *test instance 5 (medium)*, *test instance 6 (large)*, dan *early data instances*, yang mana terdiri atas 10 *instances* yaitu *early instance 1*, *early instance 2*, *early instance 3*, *early instance 4*, *early instance 5*, *early instance 6*, *early instance 7*, *early instance 8*, *early instance 9*, *early instance 10*, dengan ekstensi file berupa .xml. Setiap *instance* memiliki jumlah mata kuliah (*course*), kelas (*class*), ruangan (*room*), siswa (*student*), dan batasan (*distribution constraint*) yang berbeda-beda. Adapun statistika dan karakteristik telah dijelaskan pada Tabel 2.2.

4.1.1. Inisiasi Konten Dataset

Pada kelompok *early data instances* ITC 2019 setiap *instance* memiliki jumlah minggu (*nrWeeks*) dalam satu semester yang

berbeda-beda, yang mana hal tersebut didefinisikan pada baris awal (bagian *problem*) di setiap *file* XML seperti yang tersaji pada Gambar 4.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<problem    name="unique-instance-name"    nrDays="7"    nrWeeks="13"
slotsPerDay="288">
</problem>
```

Gambar 4.1 Inisiasi Jumlah *Days*, *Weeks*, dan *Timeslot*

Sementara itu pada baris selanjutnya diikuti dengan tag *optimization* yang mendefinisikan pembobotan untuk masing-masing penalti yang terkait dengan *time*, *room*, *distribution constraint*, dan *student* seperti yang terlihat pada Gambar 4.2.

```
<optimization time="2" room="1" distribution="1" student="2"/>
```

Gambar 4.2 Inisiasi Pembobotan Untuk Penalti

Di setiap *instance* terdapat 4 konten utama yaitu *rooms*, *courses*, *distribution constraint*, dan *student*. Namun dalam penelitian tugas akhir ini hanya digunakan konten *rooms*, *courses*, dan *distribution constraint*.

4.1.1.1. Konten *Rooms*

Rooms berisi nomor ID beserta kapasitas ruangan. Untuk setiap ruangan terdapat daftar *travel room* disertai *value* yang merepresentasikan waktu (dalam bentuk *timeslot*) yang diperlukan untuk melakukan perpindahan dari *room* ke *travel room*. Waktu perpindahan tersebut bersifat simetris dan hanya terdaftar pada salah satu ruangan. Selain itu terdapat pula *tag unavailable* yang merepresentasikan waktu saat ruangan tersebut tidak dapat digunakan yang mana mencakup atribut *weeks*, *days*, *start* dan *length* seperti yang tertera pada Gambar 4.3.

```
<rooms>
  <room id="2" capacity="100">
    <travel room="1" value="2"/> <!-- travel time is in the
    number of slots -->
  </room>
  <room id="3" capacity="80">
    <travel room="2" value="3"/> <!-- only non-zero travel
    times are present -->
    <!-- not available on Mondays and Tuesdays betwen 8:30
    - 10:30, all weeks -->
    <unavailable days="110000" start="102" length="24"
    weeks="1111111111111"/> <!-- not available on Fridays
    betwen 12:00 - 24:00, odd weeks only -->
</rooms>
```

Gambar 4.3 Contoh Spesifikasi XML Ketersediaan *Rooms*

4.1.1.2. Konten *Courses*

Courses berisi nomor ID mata kuliah, konfigurasi tiap mata kuliah (*config*), *subpart* untuk setiap konfigurasi, pilihan *class* untuk masing-masing *subpart* beserta pilihan *room* dan waktu (*time*) pelaksanaan mata kuliah seperti yang tampak pada Gambar 4.4. Pada setiap *class* terdapat batas kuota untuk mahasiswa yang dapat mengambil kelas tersebut, selain itu juga termuat atribut *parent* yang mendeskripsikan tentang relasi *parent-child* dan atribut *room* yang akan bernilai *false* apabila *class* tersebut tidak memerlukan ruangan untuk pelaksanaannya. Apabila terdapat suatu *class* yang memiliki atribut *parent*, maka siswa yang mengambil *class* tersebut juga diharuskan untuk mengambil *parent class* dari *class* tersebut. Sementara di setiap *room* dan *time* terdapat nilai penalti untuk menghitung *cost* yang akan dikeluarkan ketika menggunakan *room* atau *time* tersebut.

```
<courses>
    <course id="1">
      <config id="1">
        <subpart id="1">
          <class id="1" limit="10">
            <room id="4" penalty="0"/>
            <time days="1000000" start="164" length="27"
            weeks="00001000000000" penalty="0"/>
          </class>
        </subpart>
      </config>
    </course>
</courses>
```

Gambar 4.4 Contoh Spesifikasi XML Dari *Courses*

Courses memiliki struktur hierarki sangat kompleks (Gambar 4.5) yang mana setiap *course* terdiri atas satu atau lebih konfigurasi (*config*) dimana terdapat tiga jenis konfigurasi yaitu perkuliahan, seminar, dan laboratorium. Setiap konfigurasi terdiri atas satu atau lebih sesi perkuliahan (*subpart*). Masing-masing siswa diharuskan untuk menghadiri satu *class* dari setiap *subpart* dari suatu *config*.

```
<course id="ME 263">
  <config id="1"> <!-- Lec-Rec configuration, not linked
  (any Lec with any Lab) -->
    <subpart id="1_Lecture"> <!-- Lecture subpart -->
      <class id="Lec1" limit="100"/>
      <class id="Lec2" limit="100"/>
    </subpart>
    <subpart id="2_Recitation"> <!-- Recitation subpart -->
      <class id="Rec1" limit="50"/>
      <class id="Rec2" limit="50"/>
      <class id="Rec3" limit="50"/>
      <class id="Rec4" limit="50"/>
    </subpart>
  </config>
</course>
```

Gambar 4.5 Struktur Hierarki *Course*

4.1.1.3. Konten *Students*

Setiap *students* memiliki ID unik dan daftar *course* yang harus mereka ambil. Setiap *course* tersebut dispesifikasi oleh ID *course*. Gambar 4.6 menunjukkan contoh kode untuk *student* beserta *course* yang harus mereka ambil.

```
<students>
  <student id="1">
    <course id="1"/>
    <course id="5"/>
  </student>
</students>
```

Gambar 4.6 Spesifikasi XML dari *students* dan *courses*

4.1.1.4. Konten Distribution Constraint

Distributions berisi *type* dan kategori dari *distribution* tersebut apakah termasuk dalam *hard constraint* atau *soft constraint*. *Hard constraint* ditandai dengan atribut "*required=true"*, sementara itu *soft constraint* ditandai dengan atribut *penalty* beserta nilai penaltinya yang akan diberlakukan untuk setiap pelanggaran. Terdapat 19 jenis *distribution constraint* yang ada pada ITC 2019. Dalam setiap *distribution constraint* terdapat daftar *class* yang dikenai batasan-batasan tersebut. Adapun contoh spesifikasi xml dari konten *distribution constraints* ditunjukkan pada Gambar 4.7.

```
<distributions>
  <!-- classes 1 and 2 cannot overlap in time -->
  <distribution type="NotOverlap" required="true">
    <class id="1"/>
    <class id="2"/>
  </distribution>
</distributions>
```

Gambar 4.7 Spesifikasi XML Dari *Distribution Constraints*

4.2. Formulasi Model Matematis

Pada tahap ini dilakukan pendefinisian fungsi tujuan, variabel keputusan, dan batasan (*hard constraint* dan *soft constraint*) yang telah dijelaskan pada subbab 3.2.4 menjadi model matematis untuk mempermudah pemahaman konsep guna menyelesaikan permasalahan yang ada.

4.2.1. Fungsi Tujuan

Pada studi kasus ITC 2019 fungsi tujuan yang akan dicapai dalam pembuatan model yaitu meminimalkan biaya (nilai penalti) untuk segala sumber daya yang digunakan pada permasalahan penjadwalan mata kuliah dengan memenuhi berbagai batasan yang telah ditetapkan. Sehingga persamaan dari fungsi tujuan tersebut akan tampak sebagaimana pada persamaan (4.1) berikut.

$$\text{Minimize P} = P1 + P2 + \text{P3} \tag{4.1}$$

Dengan:
P1 = Nilai penalti bagi ruangan yang dipilih untuk digunakan dalam menjadwalkan kelas dari suatu mata kuliah
P2 = Nilai penalti bagi *timeslot* yang dipilih untuk digunakan dalam menjadwalkan kelas dari suatu mata kuliah
P3 = Nilai penalti bagi *soft constraint* yang berhasil dipenuhi

Nilai penalti yang digunakan berasal dari tiga sumber, yang mana penalti pertama berasal dari penggunaan ruangan untuk menjadwalkan kelas dari suatu mata kuliah yang dinyatakan dalam bentuk P1 dengan persamaan (4.2) sebagai berikut.

$$P1 = \min \sum_{c=1}^{C} PR_c \tag{4.2}$$

Dengan:
c = {1,2,3,…,C} = urutan kelas dari masing-masing mata kuliah
PRc = nilai penalti dari ruangan yang digunakan oleh kelas suatu mata kuliah

$$PRc = \begin{cases} 1, jika\ ruangan\ digunakan\ oleh\ kelas \\ 0, jika\ tidak \end{cases}$$

Sementara itu nilai penalti kedua berasal dari penggunaan *timeslot* suatu kelas mata kuliah yang dinyatakan dalam bentuk P2 dengan persamaan (4.3) seperti berikut.

$$P2 = \min \sum_{c=1}^{C} PT_c \qquad (4.3)$$

Dengan:
c = {1,2,3,…,C} = urutan kelas dari masing-masing mata kuliah
PTc = nilai penalti dari *timeslot* yang digunakan oleh kelas suatu mata kuliah

$$PTc = \begin{cases} 1, jika\ timeslot\ digunakan\ oleh\ kelas \\ 0, jika\ tidak \end{cases}$$

Nilai penalti ketiga berasal dari masing-masing *soft constraint* yang terdapat dalam tiap dataset, yang mana dinyatakan dalam bentuk P3 dengan persamaan (4.4) sebagai berikut.

$$P3 = \min \sum_{i=1}^{I} PDC_i \qquad (4.4)$$

Dengan:
i = {1,2,3,…,I} = urutan kelas dari masing-masing mata kuliah
PDCi = vector yang menunjukkan *soft constraint*

$$PDCi = \begin{cases} 1, jika\ soft\ constraint\ digunakan \\ 0, jika\ tidak \end{cases}$$

4.2.2. Variabel Keputusan

Variabel keputusan pada studi kasus ITC 2019 yaitu menempatkan *courses* pada *time* dan *rooms* yang tersedia.

$$\sum_{c=1}^{C}\sum_{t=1}^{T}\sum_{r=1}^{R} X_{ctr} = 1 \qquad (4.5)$$

Dengan:

C = *class* = $\{c_1,\ldots,c_{|C|}\}$

T = *time* = $\{t_1,\ldots,t_{|T|}\}$

R = *room* = $\{r_1,\ldots,r_{|R|}\}$

$$X_{ctr} = \begin{cases} 1, jika\ kelas\ ke-c\ dijadwalkan\ pada \\ timeslot\ ke-t\ dan\ ruang\ ke-r \\ 0, jika\ tidak \end{cases}$$

4.2.3. Model Matematis *Distribution Constraints*

Dalam studi kasus ITC 2019 terdapat 19 *distribution constraint* yang mana dapat berlaku sebagai *hard constraint* maupun *soft constraint* bergantung dari aturan yang telah ditetapkan pada setiap *dataset*.

Same Start : Semua kelas dalam batasan ini harus dimulai pada waktu yang bersamaan.

$$DC1 = \sum_{i=1}^{C-1}\sum_{j=i+1}^{C} s_i = s_j \qquad (4.6)$$

Dengan:

i, j = {1, 2, 3, …, C} = daftar kelas yang terdaftar pada batasan

s = {1,2, …, 288} adalah waktu mulai kelas pada *timeslot* tertentu

Same Time : Semua kelas dalam batasan ini harus berlangsung dalam suatu rentang waktu yang sama.

$$DC2 = \left(\sum_{i=1}^{C-1}\sum_{j=i+1}^{C} s_i \leq s_j \bigwedge \sum_{i=1}^{C-1}\sum_{j=i+1}^{C} e_j \leq e_i\right) \bigvee \left(\sum_{i=1}^{C-1}\sum_{j=i+1}^{C} s_j \leq s_i \bigwedge \sum_{i=1}^{C-1}\sum_{j=i+1}^{C} e_i \leq e_j\right) \quad (4.7)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
s = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu

Different Time : Semua kelas dalam batasan ini tidak boleh berlangsung dalam rentang waktu yang sama.

$$DC3 = \left(\sum_{i=1}^{C-1}\sum_{j=i+1}^{C} e_i \leq s_j\right) \bigvee \left(\sum_{i=1}^{C-1}\sum_{j=i+1}^{C} e_j \leq s_i\right) \quad (4.8)$$

Dengan:
i, j = {1, 2, …, C} = daftar kelas yang terdaftar pada batasan
s = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu

Same Days : Semua kelas dalam batasan ini harus dijadwalkan pada hari yang sama.

$$DC4 = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} d_i = d_j \qquad (4.9)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu

Different Days : Semua kelas dalam batasan ini tidak dapat mengadakan pertemuan pada hari yang sama.

$$DC5 = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} d_i \neq d_j \qquad (4.10)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu

Same Weeks : Semua kelas dalam batasan ini harus ditawarkan dalam minggu yang sama pada satu semester.

$$DC6 = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} w_i = w_j \qquad (4.11)$$

Dengan:
i, j = {1, 2, …, C} = daftar kelas yang terdaftar pada batasan

w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

Different Weeks : Semua kelas dalam batasan ini tidak boleh dijadwalkan di minggu yang sama pada satu semester.

$$DC7 = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} w_i \neq w_j \qquad (4.12)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

Same Room : Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang sama.

$$DC8 = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} r_i = r_j \qquad (4.13)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
r = {1,2, …, R} adalah urutan ruangan yang dapat dijadwalkan pada kelas tertentu

Different Room : Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang berbeda.

$$DC9 = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} r_i \neq r_j \qquad (4.14)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
r = {1,2, …, R} adalah urutan ruangan yang dapat dijadwalkan pada kelas tertentu

Overlap : Dua kelas dalam batasan ini harus memiliki waktu yang saling tumpang tindih.

$$DC10 = \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} s_j < e_i \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} s_i < e_j \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} d_{ij} \neq 0 \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} w_{ij} \neq 0 \quad (4.15)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
s = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

Not Overlap : Tidak boleh ada dua kelas dalam batasan ini yang saling tumpang tindih.

$$DC11 = \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} e_i \le s_j \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} e_j$$
$$\le s_i \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} d_{ij} \qquad (4.16)$$
$$= 0 \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} w_{ij} = 0$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
s = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

Same Attendees : Semua kelas dalam batasan ini harus bertemu pada waktu dan lokasi tertentu sehingga siswa atau dosen dapat menghadirinya.

$$DC12 = \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} c_{ij}.v_{ij} = 0 \qquad (4.17)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut

$$v_{ij}\begin{cases}1, \text{jika } t_i = t_j \\ 0\end{cases}$$

c_{ij} = jumlah mahasiswa yang mengambil mata kuliah *i* dan *j*
v_{ij} = vector yang menunjukkan apakah mata kuliah *i* dan mata kuliah *j* bentrok

Precedence : Batasan ini menetapkan daftar urutan, dimana kelas yang terdaftar harus dijadwalkan pada minggu, hari, atau *timeslot* yang lebih awal.

$$DC13 = \sum_{i=1}^{c-1}\sum_{j=i+1}^{c}\left(w_i \le w_j\left(\sum_{d=1}^{D} d_i \le d_j \sum_{e}^{E}\sum_{s}^{S} e_i \le s_j\right)\right) \quad (4.18)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
s = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

Work Day (S) : Batasan ini mencegah atau memberikan penalti ketika terdapat pasangan kelas yang ada dalam daftar, dimana waktu antara mulainya kelas pertama dan berakhirnya kelas terakhir, melebihi S *timeslot*.

$$DC14 = \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} d_{ij} = 0\right) \bigvee \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} w_{ij} = 0\right) \bigvee \left(\left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} max(e_i, e_j)\right) - \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} min(s_i, s_j)\right) \leq S\right) \quad (4.19)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
s = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu
S = jumlah *timeslot* maksimal

Min Gap (G) : Dua kelas yang dijadwalkan pada hari yang bersamaan harus memiliki setidaknya *timeslot* sebesar G yang terpisah (antara berakhirnya kelas pertama dan mulainya kelas kedua).

$$DC15 = \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} d_{ij} = 0\right) \bigvee \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} w_{ij} = 0\right) \bigvee \left(\sum_{i=1}^{c-1} e_i + G \leq \sum_{j=i+1}^{c} s_j\right) \bigvee \left(\sum_{j=i+1}^{c} e_j + G \leq \sum_{i=1}^{c-1} s_i\right) \quad (4.20)$$

Dengan:
i, j = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut

s = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu
G = jumlah *timeslot* minimal

Max Days (D) : Kelas-kelas pada batasan ini tidak boleh dijadwalkan ke lebih dari D hari kerja yang berbeda dalam satu minggu.

$$DC16 = \text{countNonzeroBits} \sum_{i=1}^{c-1} d_i \leq D \qquad (4.21)$$

Dengan:
i = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
D = jumlah maksimal hari kerja

Max Day Load (S) : Batasan ini membatasi jumlah total waktu yang ditetapkan untuk sekumpulan kelas tidak lebih dari S *timeslot* tiap hari selama satu semester.

$$DC17 = DayLoad(d,w) \leq S \qquad (4.22)$$

$$Dayload(d,w) = \sum i \left\{ \sum_{i=1}^{c-1} l_i \left| \begin{matrix} \left(\sum_{i=1}^{c-1} d_i \neq 0 \wedge 2^d \neq 0\right) \\ \wedge \left(\sum_{i=1}^{c-1} w_i \neq 0 \wedge 2^w \neq 0\right) \end{matrix} \right. \right\}$$

Dengan:
i = {1, 2, …, C} adalah daftar kelas yang terdaftar pada batasan tersebut
l = {1,2, …, 288} adalah waktu durasi berlangsungnya kelas yang terjadwal pada *timeslot* tertentu
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

Max Breaks (R, S) : Batasan ini membatasi jumlah jeda antar kelas selama sehari yang melebihi S *timeslot*, yang mana tidak boleh lebih dari R per hari.

$$D18 = \left|MergeBlocks\left\{(s,e)\middle|\left(\sum_{i=1}^{c-1} d_i \neq 0 \wedge 2^d \neq 0\right) \wedge \left(\sum_{i=1}^{c-1} w_i \neq 0 \wedge 2^w \neq 0\right)\right\}\right| \leq R + 1 \quad (4.23)$$

$$MergeBlocks = \left(\sum_{i=1}^{c-1} eb_i + S \geq \sum_{j=i+1}^{c} sb_j\right) \wedge \left(\sum_{j=i+1}^{c} eb_j + S \geq \sum_{i=1}^{c-1} sb_i\right) \Rightarrow \left(sb = min\left(\sum_{i=1}^{c-1} sb_i, \sum_{j=i+1}^{c} sb_j\right)\right) \wedge \left(eb = max\left(\sum_{i=1}^{c-1} eb_i, \sum_{j=i+1}^{c} eb_j\right)\right)$$

Dengan:
S = maksimal *timeslot*
R = jumlah *break* maksimal
sb = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu dalam suatu blok
e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu dalam suatu blok
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

Max Block (M, S) : Batasan ini membatasi jumlah waktu (diukur sebagai M slot) yang mana satu set kelas dapat dijadwalkan secara berurutan dan masing-masingnya dipisahkan oleh tidak lebih dari S slot.

$$D19 =$$

$$\left(max\left\{\left(eb - es \middle| b \in MergeBlocks\left\{s, e \middle| \left(\sum_{i=1}^{c-1} d_i \neq 0 \wedge 2^d \neq 0\right) \wedge \left(\sum_{i=1}^{c-1} w_i \neq 0 \wedge 2^w \neq 0\right)\right\}\right\}\right) \leq M \tag{4.24}$$

$$MergeBlocks = \left(\sum_{i=1}^{c-1} eb_i + S \geq \sum_{j=i+1}^{c} sb_j\right) \bigwedge \left(\sum_{j=i+1}^{c} eb_j + S \geq \sum_{i=1}^{c-1} sb_i\right) \Rightarrow \left(sb = min\left(\sum_{i=1}^{c-1} sb_i, \sum_{j=i+1}^{c} sb_j\right)\right) \bigwedge \left(eb = max\left(\sum_{i=1}^{c-1} eb_i, \sum_{j=i+1}^{c} eb_j\right)\right)$$

Dengan:
S = {1,2, …, 288} adalah maksimal *timeslot*
M = {1,2, …, 288} adalah batasan panjang blok berlangsung
sb = {1,2, …, 288} adalah waktu mulai kelas yang terjadwal pada *timeslot* tertentu dalam suatu blok

e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu dalam suatu blok
d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu
w = {1,2, …, W} adalah urutan minggu yang dijadwalkan oleh kelas tertentu

4.3. Implementasi Algoritma *Great Deluge*

Implementasi algoritma *Great Deluge* dilakukan dengan menggunakan kerangka *hyper-heuristic* untuk kemudian akan dilakukan pencarian solusi optimal. Implementasi dilakukan ketika tahap perancangan telah menghasilkan solusi awal

(*initial solution*). Adapun *low level heuristic* yang akan digunakan adalah *move* dengan pertimbangan untuk menyesuaikan kondisi *dataset*. Solusi awal yang telah melalui *low level heuristic,* kemudian akan dioptimasi menggunakan algoritma *Great Deluge*.

4.4. Perencanaan Alur Skenario Uji Coba

Algoritma *Great Deluge* memiliki beberapa parameter yaitu, *level*, *estimate penalty*, *decay rate,* jumlah iterasi, dan *low level heuristics*, namun dalam penelitian tugas akhir ini nilai parameter *level* diinisiasi sama dengan nilai penalti saat ini, sedangkan nilai *estimate penalty* ditetapkan 100, sementara terhadap nilai *decay rate,* jumlah iterasi dan *low level heuristics* dilakukan uji coba menggunakan berbagai skenario. Nilai *decay rate* yang diujikan yaitu 0, 1, 3, 5, dan 10, sementara itu minimal jumlah iterasi yang ditetapkan adalah 1.000 dan paling banyak 1.000.000, sedangkan *low level heuristic* yang digunakan merupakan kombinasi dari *move 1 timeslot*, *move 1 room*, *move 1 timeslot-room*, *move 2 timeslot*, *move 2 room*, dan *move 2 timeslot-room*. Untuk masing-masing skenario, uji coba dilakukan sebanyak sebelas kali.

4.4.1. Skenario A : Jumlah Iterasi

Pada skenario A, percobaan dilakukan menggunakan parameter jumlah iterasi untuk mengetahui apakah jumlah iterasi mempengaruhi nilai fungsi tujuan akhir. Dari uji coba skenario ini dapat diketahui apakah semakin banyak iterasi yang dilakukan akan memberikan hasil nilai fungsi tujuan akhir yang lebih baik.

Dalam penelitian tugas akhir ini dilakukan uji coba dengan mengubah jumlah iterasi mulai dari 1.000, 10.000, 100.000,

hingga 1.000.000, dengan kondisi lingkungan dimana parameter *decay rate* ditetapkan dengan rumus (*current penalty-estimate penalty*) / jumlah iterasi, sementara *low level heuristics* yang digunakan merupakan kombinasi dari *move 1 timeslot, move 1 room, move 1 timeslot-room, move 2 timeslot, move 2 room*, dan *move 2 timeslot-room*. Pada setiap jumlah iterasi yang berbeda dilakukan uji coba sebanyak sebelas kali untuk mengetahui rata-rata nilai fungsi tujuan akhir dari setiap jumlah iterasi yang diujikan.

4.4.2. Skenario B : *Low Level Heuristics*

Pada skenario B, percobaan dilakukan menggunakan jenis *low level heuristics move* dengan berbagai kombinasi atribut untuk mengetahui apakah kombinasi *low level heuristics* mempengaruhi nilai fungsi tujuan akhir. Dari uji coba skenario ini dapat diketahui apakah semakin banyak kombinasi atribut yang digunakan akan memberikan hasil nilai fungsi tujuan akhir yang lebih baik.

Dalam penelitian tugas akhir ini dilakukan 3 skenario uji coba *low level heuristics*, yaitu (1) *move* 1 *timeslot*, 1 *room*, 1 *timeslot-room*, (2) *move 2 timeslot, 2 room, 2 timeslot-room*, (3) kombinasi skenario (1) dan (2). Sementara itu kondisi lingkungan yang ditetapkan yaitu parameter *decay rate* sesuai dengan rumus (*current penalty-estimate penalty*) / jumlah iterasi, sedangkan jumlah iterasi adalah 1.000. Pada setiap skenario yang berbeda dilakukan uji coba sebanyak sebelas kali untuk mengetahui rata-rata nilai fungsi tujuan akhir dari setiap skenario yang diujikan.

4.4.3. Skenario C : *Decay Rate*

Skenario C dilakukan dengan menerapkan uji coba perhitungan *decay rate* untuk mengetahui apakah *decay rate* mempengaruhi nilai fungsi tujuan akhir. Dari uji coba skenario ini dapat diketahui apakah semakin kecil nilai *decay rate* yang digunakan akan memberikan hasil nilai fungsi tujuan akhir yang lebih baik.

Dalam penelitian tugas akhir ini dilakukan uji coba dengan mengubah nilai *decay rate,* yaitu dengan nilai 0, 1, 3, 5, dan 10. Skenario ini dilakukan dengan kondisi lingkungan parameter yaitu jumlah iterasi 1.000 dan *low level heuristics* yang digunakan merupakan kombinasi dari *move 1 timeslot, move 1 room, move 1 timeslot-room, move 2 timeslot, move 2 room,* dan *move 2 timeslot-room*. Masing-masing iterasi dilakukan sebanyak 11 kali percobaan untuk mengetahui rata-rata nilai fungsi tujuan akhir dari setiap *decay rate* yang diujikan.

Halaman ini sengaja dikosongkan

BAB V
IMPLEMENTASI

Bab ini menjelaskan proses pelaksanaan penelitian tugas akhir dan proses implementasi algoritma *Great Deluge hyper-heuristic* terhadap dataset ITC 2019 dengan menggunakan bahasa pemrogaman Java versi 8.

5.1. Membaca File Masukan

Proses awal implementasi dimulai dengan membaca file yang menjadi masukan untuk proses selanjutnya. Seluruh masukan merupakan file dengan ekstensi .xml yang selanjutnya akan dibaca menggunakan bahasa pemrograman java.

5.2. Pembentukan *Initial Solution*

Initial solution merupakan jadwal baru yang dibuat secara otomatis dengan memperhatikan batasan-batasan yang telah ditetapkan. Setiap solusi dideskripsikan menggunakan format data XML seperti pada Gambar 5.1 yang mana berisi daftar kelas beserta jadwal mata kuliah. Masing-masing solusi terdiri atas kumpulan kelas yang dilengkapi dengan ID, *start*, *days*, *weeks*, dan *room*.

Setiap permasalahan memiliki beberapa kriteria optimasi seperti pada Gambar 5.1 yang mana untuk setiap *time*, *room*, *distribution constraint* yang dipilih, dan *student* yang mengalami konflik dikaitkan dengan *weight* yang didefinisikan pada tag *optimization*.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE solution PUBLIC "-//ITC 2019//DTD Problem Format/EN"
"http://www.itc2019.org/competition-format.dtd">
<solution name="unique-instance-name" runtime="12.3" cores="4" technique="Local
Search" author="Pavel Novak" institution="Masaryk University" country="Czech
Republic">
<class id="1" days="1010100" start="90" weeks="1111111111111" room="1">
  <student id="1"/>
  <student id="3"/>
</class>
<class id="2" days="0100000" start="86" weeks="0101010101010" room="4"> <student
id="2"/>
  <student id="4"/>
</class>
<class id="3" days="0010000" start="108" weeks="0100000000000">
  <student id="1"/>
</class>
</solution>
```

Gambar 5.1 Spesifikasi XML Untuk Solusi Permasalahan

5.2.1. Membaca File .xml

File yang pertama merupakan file input yang berisi dataset ITC 2019. Kode Program 5.1 menunjukkan kode program untuk membaca file input.

```
1.  public ReadFile(String filename) throws Parser
    ConfigurationException,SAXException,IOException{
2.  File fXmlFile = new File("src/dataset/" + filename);
3.  DocumentBuilderFactory dbFactory = Document
    BuilderFactory.newInstance();
4.  DocumentBuilder dBuilder = dbFactory.new
    DocumentBuilder();
5.  Document doc = dBuilder.parse(fXmlFile);
6.  doc.getDocumentElement().normalize();
7.  Element rootElement = doc.getDocumentElement();
8.  Element problemChild = (Element) rootElement.
    getChildNodes();
9.  jWeek = Integer.parseInt(rootElement.getAttribute
    ("nrWeeks"));
10. jDays = Integer.parseInt(rootElement.getAttribute
```

```
("nrDays"));
11. jSlot = Integer.parseInt(rootElement.getAttribute
("slotsPerDay")) + 1;
12. ReadRoom(problemChild);
13. ReadCourse(problemChild);
14. ReadCons(problemChild);
15. makeTS(jWeek, jDays, idRomBesar);}
```

Kode Program 5.1 Membaca File .xml

5.2.2. Pembacaan dan Penyimpanan Konten *Rooms*

Konten *Rooms* memiliki beberapa atribut dan nilai yang harus dibaca dan disimpan untuk selanjutnya digunakan dalam proses pembentukan solusi awal yaitu *room id* beserta *capacity* untuk masing-masing ruangan (Kode Program 5.2), *travel room id* beserta *value* yang menunjukkan waktu perpindahan dari ruangan satu ke ruangan lainnya (Kode Program 5.3), *unavailable days* beserta *start*, *length* dan *weeks* yang menunjukkan waktu dimana ruangan tersebut tidak dapat digunakan (Kode Program 5.4). Semua nilai tersebut dibaca dalam sebuah kelas dan disimpan dalam bentuk objek.

```
1.  void ReadRoom(Element problemChild){
2.  NodeList rL = problemChild.getElementsByTag
    Name("rooms");
3.  Node rN = rL.item(0);
4.  Element roomsElement = (Element) rN.getChild
    Nodes();
5.  NodeList childRoomsL = (NodeList) rooms
    Element.getElementsByTagName("room");
6.  setjRoom(childRoomsL.getLength());
7.  ArrayList<Integer> idRom = new ArrayList<>();
```

```
8.  for (int rm = 0; rm < jRoom; rm++) {
9.  Node nRoom = childRoomsL.item(rm);
10. Element roomElement = (Element) nRoom;
11. int roomID, capRoom;
12. roomID = Integer.parseInt(roomElement.
    getAttribute("id"));
13. capRoom = Integer.parseInt(roomElement.
    getAttribute("capacity"));
14. Room room = new Room(roomID, capRoom);
15. listRoom.add(room);
16. idRom.add(roomID);}
18. idRomBesar = Collections.max(idRom);
```

Kode Program 5.2 Membaca Konten *Rooms* Atribut *Room ID* dan *Capacity*

```
1.  roomcap = new int[idRomBesar];
2.  unroom = new int[idRomBesar][getjWeek()]
    [getjDays()][getjSlot()];
3.  travelroom = new int[idRomBesar][idRomBesar];
4.  for (int room = 0; room < childRoomsL.getLength();
    room++) {
5.  Node nRoom = childRoomsL.item(room);
6.  Element roomElement = (Element) nRoom;
7.  Room roomA = listRoom.get(room);
8.  int roomID = Integer.parseInt(roomElement.
    getAttribute("id"))-1;
9.  NodeList travelList = (NodeList) roomElement.
    getElementsByTagName("travel");
10. NodeList unRoomList = (NodeList) roomElement.
    getElementsByTagName("unavailable");
```

```
11.  for (int tvrm = 0; tvrm < travelList.getLength();
     tvrm++) {
12.  Node travelNode = travelList.item(tvrm);
13.  Element travelElement = (Element) travelNode;
14.  int trvl = Integer.parseInt(travelElement.
     getAttribute("room"))-1;
15.  travelroom[room][trvl] = Integer.parseInt(
     travelElement.getAttribute("value"));
16.  travelroom[trvl][roomID] = travelroom[roomID]
     [trvl];}
```

Kode Program 5.3 Membaca Konten *Rooms* Atribut *Travel Room* dan *Value*

```
1.   for (int unrm = 0; unrm < unRoomList.getLength();
     unrm++) {
2.   Node unavNode = unRoomList.item(unrm);
3.   Element unRoomElement = (Element) unavNode;
4.   String[] wk = unRoomElement.getAttribute("weeks").
     split("");
5.   String[] ds = unRoomElement.getAttribute("days").
     split("");
6.   int[] a = new int[wk.length];
7.   int[] b = new int[ds.length];
8.   int ts = Integer.parseInt(unRoomElement.
     getAttribute("start"));
9.   int te = Integer.parseInt(unRoomElement.getAttribute
     ("start")) + Integer.parseInt(unRoomElement.
     getAttribute("length"));
10.  for (int j = 0; j < wk.length; j++) {
11.  a[j] = Integer.parseInt(wk[j]);
12.  if (a[j] == 1) {
```

```
13.  for (int k = 0; k < ds.length; k++) {
14.  b[k] = Integer.parseInt(ds[k]);
15.  if (b[k]==1) {
16.  for (int l = ts; l <= te; l++) {
17.  unroom[room][j][k][l]=1;}
```

Kode Program 5.4 Membaca Konten *Rooms* Atribut *Unavailable Days*

5.2.3. Pembacaan dan Penyimpanan Konten *Courses*

Konten *Courses* memiliki beberapa atribut dan nilai yang harus dibaca dan disimpan untuk selanjutnya digunakan dalam proses pembentukan solusi awal yaitu *course* (Kode Program 5.5), *config* (Kode Program 5.6), *subpart* (Kode Program 5.7), *class* beserta *limit* mahasiswa yang dapat bergabung di kelas tersebut dan *room* yang menunjukkan apakah kelas tersebut membutuhkan ruangan atau tidak, *room id* dan *penalty*, *available time* beserta *days*, *start*, *length*, *weeks* dan *penalty* yang menunjukkan pilihan waktu bagi kelas tersebut dapat dilaksanakan. Semua nilai tersebut dibaca dalam suatu kelas dan disimpan dalam bentuk objek.

```
1.  void ReadCourse(Element problemChild){
2.  NodeList cL = problemChild.getElementsByTag
    Name("courses");
3.  Node cN = cL.item(0);
4.  Element coursesElement = (Element) cN.getChild
    Nodes();
5.  NodeList courseL = coursesElement.getElements
    ByTagName("course");
6.  jCourse = courseL.getLength();
7.  for (int i = 0; i < courseL.getLength(); i++) {
8.  Node courseN = courseL.item(i);
```

```
9.  Element courseE = (Element) courseN;
10. courseID = Integer.parseInt(courseE.getAttribute
    ("id"));
11. Course course = new Course(courseID);
```

Kode Program 5.5 Membaca Konten *Courses* Atribut *Course*

```
1.  NodeList configL = courseE.getElementsByTag
    Name("config");
2.  for (int j = 0; j < configL.getLength(); j++) {
3.  Node configN = configL.item(j);
4.  Element configE = (Element) configN;
5.  configID = Integer.parseInt(configE.getAttribute
    ("id"));
6.  Config config = new Config(configID);
```

Kode Program 5.6 Membaca Konten *Courses* Atribut *Config*

```
1.  NodeList subpartL = configE.getElementsByTag
    Name("subpart");
2.  for (int k = 0; k < subpartL.getLength(); k++) {
3.  Node subpartN = subpartL.item(k);
4.  Element subpartE = (Element) subpartN;
5.  subpartID = Integer.parseInt(subpartE.getAttribute
    ("id"));
6.  Subpart subpart = new Subpart(subpartID);
```

Kode Program 5.7 Membaca Konten *Courses* Atribut *Subpart*

5.2.4. Pembacaan dan Penyimpanan Konten *Distribution Constraints*

Konten *distribution constraints* memuat berbagai batasan baik berupa *hard constraint* maupun *soft constraint*, yang mana *hard constraint* ditandai dengan adanya atribut *required* yang

bernilai *true*, sementara *soft constraint* ditandai dengan adanya atribut *penalty*. Adapun atribut dan nilai yang harus dibaca dan disimpan untuk selanjutnya digunakan dalam pembentukan solusi awal antara lain *distribution type*, *required* atau *penalty*, dan daftar *class id* yang memiliki batasan-batasan tersebut. Semua nilai tersebut dibaca dalam suatu kelas dan disimpan dalam bentuk objek. Kode Program 5.8 menunjukkan kode program untuk membaca konten *distribution constraint*.

```
1.  void ReadCons(Element problemChild){
2.  String[] typeS;
3.  NodeList dL = problemChild.getElementsByTag
    Name("distributions");
4.  Node dN = dL.item(0);
5.  Element distribsElement = (Element) dN.getChild
    Nodes();
6.  NodeList distriL = (NodeList) distribsElement.get
    ElementsByTagName("distribution");
7.  distrib = new Distributions();
8.  for (int distri=0; distri<distriL.getLength(); distri++) {
9.  required = 0;
10. penalty = 1000;
11. int angka1 = -1, angka2 = -1;
12. ArrayList<Integer> temp  = new ArrayList<>();
13. type = disElement.getAttribute("type");
14. if (disElement.hasAttribute("required")) {
15. required = 1;
16. } else {
17. penalty = Integer.parseInt(disElement.getAttribute
    ("penalty"));}
```

```
18.  if (required == 1) {  //HARD CONSTRAINT
19.  distrib.addHard(cns);
20.  for (int i = 0; i < temp.size(); i++) {
21.  Kelas kelas = listKelas.get(temp.get(i)-1);
22.  kelas.addclasHC(cns);}
23.  } else {  //SOFT CONSTRAINT
24.  distrib.addSoft(cns);
25.  for (int i = 0; i < temp.size(); i++) {
26.  Kelas kelas = listKelas.get(temp.get(i)-1);
27.  kelas.addclasSC(cns);}
```

Kode Program 5.8 Membaca Konten *Distribution Constraint*

5.2.5. Pembuatan *Initial Solution*

Tujuan utama dari proses ini adalah untuk menghasilkan solusi awal yang memenuhi *hard constraint* menggunakan *Greedy Algorithm*. *Initial Solution* merupakan solusi awal yang digunakan sebagai jadwal awal dalam optimasi. Solusi awal ini berisi jadwal awal sebelum optimasi dilakukan. Dalam pembuatan solusi awal, algoritma yang diterapkan adalah *Greedy Algorithm*, dimana urutan pertama dalam sebuah daftar mata kuliah diletakkan pada slot tersedia pertama, sedemikian hingga seluruh mata kuliah terjadwal. Sehingga penghitungan penalti atau skor *soft constraint* diabaikan dan hanya memperhatikan *hard constraint*. Jika mata kuliah telah semua terjadwal pada matriks *timeslot-rooms* dan telah lolos *hard constraint*, maka solusi awal dianggap telah terbentuk.

Hal pertama yang harus dilakukan adalah membuat objek waktu, ruangan, mata kuliah, dan batasan yang berisi *setter* dan *getter* sebagai objek penyimpanan sementara. Setelah itu dilakukan pembuatan fungsi dan metode yang berfungsi

mencari *timeslot* dan ruangan yang sesuai dengan mempertimbangkan kebutuhan kelas akan ruangan, pengecekan ruangan yang tidak tersedia, dan pengecekan *hard constraint* seperti yang tampak pada Kode Program 5.9.

```
1.  outerloop: for(int ts=0; ts < availableTS.size(); ts++) {
2.  TimeAss Tims = availableTS.get(ts);
3.  for (int rm = 0; rm < availableroom.size(); rm++) {
4.  RoomAss roms = availableroom.get(rm);
5.  boolean cek = cekTS(Tims, roms);
6.  if (!cek) {
7.  boolean cekUnv = cekUnvroom(Tims, roms);
8.  if (!cekUnv) {
9.  boolean hardC = cekHC.HardConstraint(ts, rm, kelas,
    listKls, travel);
10. if (!hardC) {
11. if (cek && cekUnv && hardC) {
12. kelas.setTs(ts);
13. kelas.setTsBaru(ts);
14. TS = ts;
15. kelas.setRoom(rm);
16. kelas.setRoomBaru(rm);
17. assign(idKelas, Tims, roms);
18. break outerloop;
```

Kode Program 5.9 Mengecek *Timeslot* dan *Room* Kelas

Apabila telah dilakukan pengecekan terhadap *timeslot* dan ruangan, maka dibuat fungsi untuk menjadwalkan kelas ke dalam *timeslot* dan ruangan yang tersedia, seperti tampak pada Kode Program 5.10.

```
1.  void assign(int idKls, TimeAss time, RoomAss
    room){
2.  weeks = time.getWeeks();
3.  days = time.getDays();
4.  start = time.getStart();
5.  length = time.getLength();
6.  idRoom = room.getRoom_id();
7.  for (int wk = 0; wk < weeks.size(); wk++) {
8.  if (weeks.get(wk) == 1) {
9.  for (int day = 0; day < days.size(); day++) {
10. if (days.get(day) == 1) {
11. for (int tm = start; tm <= (start+length); tm++) {
12. timeslot.get(wk).listHari.get(day).timeslot[tm]
    [idRoom-1] = idKls;}}}}}}
```

Kode Program 5.10 Menjadwalkan Kelas Dalam *Timeslot* Dan Ruangan

Untuk memastikan bahwa semua mata kuliah telah terjadwal, dibuat *method* seperti yang tampak pada Kode Program 5.11.

```
1.  void cekSol(){
2.  sol = new ArrayList<>();
3.  for (int i = 0; i < listKelas.size(); i++) {
4.  if(listKelas.get(i).getTs() == -1){
5.  sol.add(i);}}}
```

Kode Program 5.11 Memastikan Semua Kelas Terjadwal

5.2.6. Pembuatan Kode Program *Hard Constraint*

Hard constraint digunakan untuk menunjukkan fisibilitas dari solusi yang dihasilkan. Apabila suatu solusi melanggar *hard constraint*, maka solusi tersebut dinyatakan tidak *feasible*. Sebaliknya apabila memenuhi *hard constraint* maka dinyatakan sebagai solusi yang *feasible*. Dalam studi kasus ITC 2019

terdapat 19 batasan yang mana masing-masingnya dapat berlaku sebagai *hard constraint* atau *soft constraint* bergantung pada aturan yang ada di setiap *dataset*. Kode Program 5.12 menunjukkan kode untuk mengecek *hard constraint*.

```
1.  boolean HardConstraint(int TS, int Rom, Kelas kls,
    ArrayList<Kelas> listKls, int[][] travel){
2.  ArrayList<Constraint> HardCons = kls.getClasHC();
3.  if (HardCons.size()<1) {
4.  return true;}
5.  for (int i = 0; i < HardCons.size(); i++) {
6.  Constraint Hard = HardCons.get(i);
7.  ArrayList<Integer> KlsCons = Hard.getConsClass();
8.  String type = Hard.getType();
9.  int angka1 = Hard.getAngka1();
10. int angka2 = Hard.getAngka2();
11. if (Cons1(type, angka1, angka2, TS, Rom, kls,
    KlsCons) || Cons2(type, TS, Rom, kls, KlsCons,
    travel)) {
12. hardKls = true;}
13. } else{ return false;}
```

Kode Program 5.12 Mengecek *Hard Constraint*

5.2.6.1. Same Start

Batasan yang pertama adalah *same start* dimana semua kelas dalam batasan ini harus dimulai pada waktu yang bersamaan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap waktu mulai (*start*) kelas. Fungsi ini akan bernilai benar apabila waktu

mulai (*start*) antar kelas sama, begitu pun sebaliknya. Kode Program 5.13 menunjukkan kode untuk fungsi batasan *same start* sebagai *hard constraint*.

```
1.  if(iTS != -1){
2.  if(startA == startB){
3.  SS = true;
4.  } else{
5.  return false;}
6.  } else if(iTS == -1){
7.  SS = true;
```

Kode Program 5.13 Batasan *Same Start* Sebagai *Hard Constraint*

5.2.6.2. *Same Time*

Semua kelas yang ada dalam batasan *same time* harus dijadwalkan dalam suatu rentang waktu yang bersamaan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap waktu mulai (*start*) dan waktu berakhir (*end*) kelas.

```
1.  if(iTS != -1){
2.  if ((startA <= startB && endB <= endA) || (startB <=
    startA && endA <= endB)) {
3.  ST = true;
4.  } else{
5.  return false;}
6.  } else if(iTS == -1){
7.  ST = true;
```

Kode Program 5.14 Batasan *Same Time* Sebagai *Hard Constraint*

Fungsi ini akan bernilai benar apabila kelas A dimulai lebih dahulu dan berakhir setelah kelas B berakhir, atau kelas B dimulai lebih dahulu dan dan berakhir setelah kelas A berakhir. Kode Program 5.14 menunjukkan kode untuk fungsi batasan *same time*.

5.2.6.3. *Different Time*

Semua kelas dalam batasan ini tidak boleh dijadwalkan dalam suatu rentang waktu yang bersamaan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap waktu berakhir (*end*) dan waktu dimulainya (*start*) kelas. Fungsi ini akan bernilai benar apabila waktu berakhirnya kelas A lebih dahulu dibanding waktu mulainya kelas B, atau waktu berakhirnya kelas B lebih dahulu dibanding waktu mulainya kelas A. Kode Program 5.15 menunjukkan kode untuk fungsi batasan *different time*.

```
1.  if(iTS != -1){
2.  if(endA <= startB || endB <= startA){
3.  DT = true;
4.  } else{
5.  return false;}
6.  } else if(iTS == -1){
7.  DT = true;
```

Kode Program 5.15 Batasan *Different Time* Sebagai *Hard Constraint*

5.2.6.4. *Same Days*

Semua kelas dalam batasan ini harus dijadwalkan pada hari yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist*

apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada hari yang sama. Kode Program 5.16 menunjukkan kode untuk fungsi batasan *same days*.

```
1.  if(iTS != -1){
2.  for (int dy = 0; dy < daysA.size(); dy++) {
3.  if(daysA.get(dy) == 1 && daysB.get(dy) == 1){
4.  SD = true;
5.  break;
6.  } else{
7.  SD =  false;}
8.  } if(SD == false){
9.  return false;}
10. } else if(iTS == -1){
11. SD = true;
```

Kode Program 5.16 Batasan *Same Days* Sebagai *Hard Constraint*

5.2.6.5. *Different Days*

Semua kelas dalam batasan ini tidak dapat dijadwalkan pada hari yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada hari yang berbeda. Kode Program 5.17 menunjukkan kode untuk fungsi batasan *different days*.

```
1.  if(iTS != -1){
2.  for (int dy = 0; dy < daysA.size(); dy++) {
3.  if((daysA.get(dy) == 0 || daysB.get(dy) == 0) ||
    (daysA.get(dy) == 1 && daysB.get(dy) == 0) ||
    (daysA.get(dy) == 0 && daysB.get(dy) == 1)){
4.  DD = true;
5.  } else{ DD =  false;
6.  break;}
7.  } if(DD == false){
8.  return false;}
9.  } else if(iTS == -1){ DD = true;}
```

Kode Program 5.17 Batasan *Different Days* Sebagai *Hard Constraint*

5.2.6.6. *Same Weeks*

Semua kelas dalam batasan ini harus dijadwalkan pada minggu yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada minggu yang sama. Kode Program 5.18 menunjukkan kode untuk fungsi batasan *same weeks*.

```
1.  if(weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
2.  SW = true; break;}
3.  } else{ SW =  false;}
4.  } if(SW == false){ return false;}
5.  } else if(iTS == -1){ SW = true;}
```

Kode Program 5.18 Batasan *Same Weeks* Sebagai *Hard Constraint*

5.2.6.7. *Different Weeks*

Semua kelas dalam batasan ini tidak dapat dijadwalkan pada minggu yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada minggu yang berbeda. Kode Program 5.19 menunjukkan kode untuk fungsi batasan *different weeks*.

```
1.  if(iTS != -1){
2.  for (int wk = 0; wk < weeksA.size(); wk++){
3.  if(weeksA.get(wk) == 0 && weeksB.get(wk) == 0 ||
    weeksA.get(wk) == 1 && weeksB.get(wk) == 0 ||
    weeksA.get(wk) == 0 && weeksB.get(wk) == 1){
4.  DW = true;
5.  } else{
6.  DW =  false;
7.  break;}}}
8.  } else if(iTS == -1){
9.  DW = true;}
```

Kode Program 5.19 Batasan *Different Weeks* Sebagai *Hard Constraint*

5.2.6.8. *Same Room*

Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap *id* kelas untuk mengetahui apakah *id* kelas yang sedang dicek sama. Kemudian masing-masing kelas dalam *arraylist* dicek apakah sudah dijadwalkan dalam ruangan. Apabila sudah dijadwalkan, maka dilakukan

pengecekan dan pembandingan terhadap ruangan untuk pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada ruangan yang sama. Kode Program 5.20 menunjukkan kode untuk fungsi batasan *same room*.

```
1.  if(idKLSA != idKLSB){
2.  if(iRM != -1){
3.  if(roomA == roomB){
4.  SR = true;
5.  } else{ return  false;
6.  } else if(iTS == -1){
7.  SR = true;}
8.  } else{ SR = true;}
```

Kode Program 5.20 Batasan *Same Room* Sebagai *Hard Constraint*

5.2.6.9. *Different Room*

Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang berbeda. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam ruangan. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap ruangan untuk pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada ruangan yang berbeda. Kode Program 5.21 menunjukkan kode untuk fungsi batasan *different room*.

```
1.  if(roomA != roomB){ DR = true;
2.  } else{ return  false;
3.  } else if(iRM == -1){ DR = true;}
```

Kode Program 5.21 Batasan *Different Room* Sebagai *Hard Constraint*

5.2.6.10. *Overlap*

Semua kelas dalam batasan ini harus dijadwalkan pada waktu yang saling tumpang tindih. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan minggu, hari, dan waktu dimulai serta berakhirnya kelas. Fungsi ini akan bernilai benar apabila waktu dimulainya kelas B lebih dahulu dibandingkan waktu berakhirnya kelas A dan waktu dimulainya kelas A lebih dahulu dibandingkan waktu berakhirnya kelas B. Apabila memenuhi kondisi tersebut maka dilakukan pengecekan terhadap minggu dan hari pelaksanaan kelas A dan B, yang mana keduanya harus dijadwalkan di minggu dan hari yang sama. Kode Program 5.22 menunjukkan kode untuk fungsi batasan *overlap*.

```
1.  if(iTS != -1){
2.  if((startB < endA) && (startA < endB)){
3.  outerloop:
4.  for (int wk = 0; wk < weeksA.size(); wk++) {
5.  if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
6.  for (int dy = 0; dy < daysA.size(); dy++) {
7.  if(daysA.get(dy) == 1 && daysB.get(dy) == 1){
8.  O = true; break outerloop;
9.  } else{ O = false;}}}
10. if (O == false) { return false;}
11. } else{ return false;}
12. } else if(iTS == -1){ O = true;}
```

Kode Program 5.22 Batasan *Overlap* Sebagai *Hard Constraint*

5.2.6.11. *Not Overlap*

Semua kelas dalam batasan ini harus dijadwalkan pada waktu yang tidak saling tumpang tindih. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan minggu, hari, dan waktu dimulai serta berakhirnya kelas. Terdapat fungsi *for* untuk mengecek apakah kelas A dan B dijadwalkan pada minggu dan hari yang sama. Apabila memenuhi kondisi tersebut maka dilakukan pengecekan terhadap waktu dimulai dan berakhirnya kelas A dan kelas B. Fungsi ini bernilai benar apabila waktu berakhirnya kelas A lebih dahulu atau sama dengan waktu dimulainya kelas B, atau waktu berakhirnya kelas B lebih dahulu atau sama dengan waktu dimulainya kelas A. Kode Program 5.23 menunjukkan kode untuk fungsi batasan *not overlap*.

```
1.  for (int wk = 0; wk < weeksA.size(); wk++) {
2.  if(weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
3.  for (int dy = 0; dy < daysA.size(); dy++) {
4.  if(daysA.get(dy) == 1 && daysB.get(dy) == 1){
5.  if((endA <= startB) || (endB <= startA)){ NO = true;
6.  } else{ return false;}
7.  } else{ NO = true;}}
8.  } else{ NO = true;}}
9.  } else if(iTS == -1){
10. NO = true;}
```

Kode Program 5.23 Batasan *Not Overlap* Sebagai *Hard Constraint*

5.2.6.12. *Same Attendees*

Semua kelas dalam batasan ini harus bertemu pada waktu dan ruangan tertentu sehingga mahasiswa atau dosen dapat menghadirinya. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah memiliki ruangan dan telah dijadwalkan dalam *timeslot*. Kemudian dilakukan pengecekan terhadap *id* kelas untuk mengetahui apakah *id* kelas yang sedang dicek sama. Apabila kedua kelas yang sedang dicek memiliki ruangan, dilakukan pengambilan waktu *travel* antara kelas satu dengan lainnya. Selanjutnya digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila waktu berakhirnya kelas A ditambah dengan waktu *travel* kurang dari atau sama dengan waktu dimulainya kelas B, atau ketika waktu berakhirnya kelas B ditambah dengan waktu *travel* kurang dari atau sama dengan waktu dimulainya kelas A. Kode Program 5.24 menunjukkan kode untuk fungsi batasan *same attendees*.

```
1.  if (Rom < 0) {
2.  noRoomA = true;
3.  } else {
4.  roomA = kls.getAvailableroom().get(Rom).get
    Room_id() - 1;}
5.  if (idKLSA != idKLSB){
6.  if (klsB.isHasRoom() == false) {
7.  iRM = 999;}
8.  if(iTS != -1 && iRM != -1){
9.  if (iRM == 999) {
10. noRoomB = true;
```

```
11.  } else {
12.  roomB = klsB.getAvailableroom().get(iRM).get
     Room_id()-1;}
13.  if (noRoomA == false && noRoomB == false) {
14.  trvl = valueTravel[roomA][roomB];}
15.  outerloop:for (int wk = 0; wk < weeksA.size(); wk++){
16.  if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
17.  for (int dy = 0; dy < daysA.size(); dy++){
18.  if (daysA.get(dy) == 1 && daysB.get(dy) == 1){
19.  if ((endA + trvl) <= startB || (endB + trvl) <= startA) {
20.  SA = true;
21.  break outerloop;
```

Kode Program 5.24 Batasan *Same Attendees* Sebagai *Hard Constraint*

5.2.6.13. *Precedence*

Batasan ini menetapkan daftar urutan, dimana kelas yang terdaftar harus dijadwalkan pada minggu, hari, atau timeslot yang lebih awal. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila pada minggu dan hari yang sama, waktu berakhirnya kelas B tidak melebihi waktu dimulainya kelas A, atau pada saat hari atau minggu pelaksanaan kelas A lebih dahulu dibanding hari atau minggu pelaksanaan kelas B, begitu pun ketika pelaksanaan kelas B lebih dahulu dibanding kelas A. Kode Program 5.25 menunjukkan kode untuk fungsi batasan *precedence*.

```
1.  if(iTS != -1){
2.  outerloop: for (int wk=0; wk<weeksA.size(); wk++) {
3.  if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
4.  for (int dy = 0; dy < daysA.size(); dy++) {
5.  if(daysA.get(dy) == 1 && daysB.get(dy)==1){
6.  if (endB <= startA) {
7.  P = true;
8.  break outerloop;
9.  } else { return false;}
10. } else if(daysA.get(dy) < daysB.get(dy)){
11. P = true;
12. break outerloop;
13. } else if(daysA.get(dy) > daysB.get(dy)){
14. return false;}}
15. } else if(weeksA.get(wk) < weeksB.get(wk)){
16. P = true;
17. break;
18. } else if(weeksA.get(wk) > weeksB.get(wk)){
19. return false;}}
20. } else if(iTS == -1){
21. P = true;}
```

Kode Program 5.25 Batasan *Precedence* Sebagai *Hard Constraint*

5.2.6.14. *WorkDay*

Batasan ini mencegah atau memberikan penalti ketika terdapat pasangan kelas yang ada dalam daftar, dimana waktu antara mulainya kelas pertama dan berakhirnya kelas terakhir, melebihi S timeslot. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek

dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila antara waktu dimulainya kelas pertama sampai dengan waktu berakhirnya kelas terakhir pada suatu hari tidak melebihi S *timeslot*. Kode Program 5.26 menunjukkan kode untuk fungsi batasan *workday*.

```
1.  if(iTS != -1){
2.  outerloop:for (int wk=0; wk < weeksA.size(); wk++) {
3.  if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
4.  for (int dy = 0; dy < daysA.size(); dy++) {
5.  if(daysA.get(dy) == 1 && daysB.get(dy)==1){
6.  if((Math.max(endA, endB)) - (Math.min(startA,
    startB)) <= S){
7.  WD = true;
8.  break outerloop;
9.  } else{
10. return false;}
11. } else{
12. WD = true;}}
13. } else{
14. WD = true;}}
15. } else if(iTS == -1){
16. WD = true;}
```

Kode Program 5.26 Batasan *WorkDay* Sebagai *Hard Constraint*

5.2.6.15. *MinGap*

Dua kelas yang dijadwalkan pada hari yang bersamaan harus memiliki setidaknya timeslot sebesar G yang terpisah (antara berakhirnya kelas pertama dan mulainya kelas kedua). Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah

dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila waktu berakhirnya kelas A ditambah G *timeslot* tidak melebihi waktu dimulainya kelas B, atau ketika waktu berakhirnya kelas B ditambah G *timeslot* tidak melebihi waktu dimulainya kelas A. Kode Program 5.27 menunjukkan kode untuk fungsi batasan *mingap*.

```
1.    if(iTS != -1){
2.    outerloop:for (int wk=0; wk < weeksA.size(); wk++) {
3.    if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
4.    for (int dy = 0; dy < daysA.size(); dy++) {
5.    if(daysA.get(dy) == 1 && daysB.get(dy)==1){
6.    if((endA + G <= startB) || (endB + G <= startA)){
7.    MG = true;
8.    break outerloop;
9.    } else{ return false;}
10.   } else{ MG = true;}}
11.   } else{ MG = true;}}
12.   } else if(iTS == -1){
13.   MG = true;}
```

Kode Program 5.27 Batasan *MinGap* Sebagai *Hard Constraint*

5.2.6.16. *MaxDays*

Kelas-kelas pada batasan ini tidak boleh dijadwalkan ke lebih dari D hari dalam satu minggu. Dalam fungsi batasan ini, terlebih dahulu digunakan fungsi *for* untuk mengecek hari pelaksanaan kelas yang kemudian dihitung berapa jumlah hari pelaksanaan kelas dalam satu minggu. Fungsi ini akan bernilai benar apabila jumlah hari pelaksanaan kelas dalam satu minggu

tidak melebihi D. Kode Program 5.28 menunjukkan kode untuk fungsi batasan *maxdays*.

```
1.  if (countNonzeroBits(daysA) <= D) {
2.  return true;}
3.  private int countNonzeroBits(ArrayList<Integer>
    daysA) { int count = 0;
4.  for (int i = 0; i < daysA.size(); i++) {
5.  if (daysA.get(i) == 1) {
6.  count++;}}
```

Kode Program 5.28 Batasan *MaxDays* Sebagai *Hard Constraint*

5.2.6.17. *MaxDayLoad*

Batasan ini membatasi jumlah total waktu yang ditetapkan untuk sekumpulan kelas tidak lebih dari S timeslot tiap hari selama satu semester. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila beban kerja pada suatu hari tidak melebihi S *timeslot*. Kode Program 5.29 menunjukkan kode untuk fungsi batasan *maxdayload*.

```
1.  if (DayLoad(S, length, days, weeks, listMDL) == 0) {
2.  return true;}
3.  private int DayLoad(int S, int lengthA,
    ArrayList<Integer> daysA, ArrayList<Integer>
    weeksA, ArrayList<Integer> listMDL) {
4.  int load = 0;
5.  if (iTS != -1) {
6.  outerloop:
```

```
7.  for (int wk = 0; wk < weeksA.size(); wk++) {
8.  if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
9.  for (int dy = 0; dy < daysA.size(); dy++) {
10. if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
11. load += (lengthA - S);}}}}
```

Kode Program 5.29 Batasan *MaxDayLoad* Sebagai *Hard Constraint*

5.2.6.18. *MaxBreaks*

Batasan ini membatasi jumlah jeda antar kelas selama sehari yang melebihi S *timeslot*, tidak boleh lebih dari R per hari. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Jika waktu berakhirnya kelas A ditambah S *timeslot* tidak kurang dari waktu dimulainya kelas B dan ketika waktu berakhirnya kelas B ditambah S *timeslot* tidak kurang dari waktu dimulainya kelas A, maka dianggap benar. Fungsi ini akan bernilai benar apabila jumlah blok tidak melebihi jumlah R+1. Kode Program 5.30 menunjukkan kode untuk fungsi batasan *maxbreaks*.

```
1.  boolean Maxbreaks(int TS, int R, int S, Kelas kls,
    ArrayList<Integer> listMB){
2.  if(MergeBlocks(TS, S, kls, listMB) <= (R+1)){
3.  return true;}
4.  private int MergeBlocks(int TS, int S, Kelas kls,
    ArrayList<Integer> listMB) {
5.  int initialBlock = 1;
6.  if(iTS != -1){
7.  outerloop:for (int wk=0; wk < weeksA.size(); wk++) {
```

```
8.  for (int dy = 0; dy < daysA.size(); dy++) {
9.  if(weeksA.get(wk) == 1 && daysA.get(dy) == 1 &&
    weeksB.get(wk) == 1 && daysB.get(dy) == 1){
10. if(endA + S >= startB && endB + S >= startA){
11. startA = Math.min(startA, startB);
12. endA = Math.max(endA, endB);
13. } else{
14. initialBlock++;
15. startA = startB;
16. endA = endB;
17. daysA = daysB;
18. weeksA = weeksB;
19. break outerloop;}}}}
20. } else if(iTS == -1) {
21. return initialBlock;}
```

Kode Program 5.30 Batasan *MaxBreaks* Sebagai *Hard Constraint*

5.2.6.19. *MaxBlock*

Semua kelas dalam batasan ini harus dijadwalkan tidak melebihi batas maksimal blok yang ditetapkan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu pelaksanaan kelas. Fungsi ini akan bernilai benar apabila penjadwalan tidak melebihi M jumlah blok. Kode Program 5.31 menunjukkan kode untuk fungsi batasan *max block*.

```
1.  boolean Maxblock(int TS, int M, int S, Kelas kls,
    ArrayList<Integer> listMBL){
2.  if(MergeBlockB(TS, M, S, kls, listMBL) <= M){
```

```
3.  return true;}
4.  private int MergeBlockB(int TS,int M, int S, Kelas kls,
    ArrayList<Integer> listMBL) {
5.  int lengthBlock = 0;
6.  int lebih = 0;
7.  boolean blockBaru = true;
8.  if (iTS != -1) {
9.  outerloop:for (int wk=0; wk < weeksA.size(); wk++) {
10. for (int dy = 0; dy < daysA.size(); dy++) {
11. if(weeksA.get(wk) == 1 && daysA.get(dy) == 1 &&
    weeksB.get(wk) == 1 && daysB.get(dy) == 1){
12. if(endA+S >= startB && endB+S >= startA){
13. startA = Math.min(startA, startB);
    endA = Math.max(endA, endB);
    lengthBlock = endA - startA;
14. if(lengthBlock > M && blockBaru == true){
    lebih += 1; blockBaru = false; }else{ lebih += 0;}
15. } else{
16. lengthBlock = endA - startA;
17. if(lengthBlock > M && blockBaru == true){ lebih +=
    1; }else{ lebih += 0;}
18. blockBaru = true; startA = startB; endA = endB;
    daysA = daysB; weeksA = weeksB; break
    outerloop;}}}}
19. } else if (iTS == -1) { return lengthBlock;}}
20. return lengthBlock;}
```

Kode Program 5.31 Batasan *MaxBlock* Sebagai *Hard Constraint*

5.2.7. Pembuatan Kode Program *Soft Constraint*

Kode program *soft constraint* merupakan kode program yang akan menghitung nilai penalti yang akan menjadi hasil

perbandingan dari proses optimasi. Total nilai penalti dari *soft constraint* akan dihitung berdasarkan penjadwalan mata kuliah ke *timeslot* dan *rooms* yang tersedia, dan batasan yang dilanggar. Semua nilai penalti dari *soft constraint* kemudian dijumlahkan untuk menghasilkan nilai penalti total yang digunakan sebagai pembanding setiap solusi yang dihasilkan. Kode Program 5.32 menunjukkan kode untuk mengecek *soft constraint*.

```
1.  int totalPenalty(ArrayList<Kelas> listKls, int[][]
    travel, Distributions distri){
2.  ArrayList<Constraint> SoftCons = distrib.getSoftC();
3.  for (int i = 0; i < SoftCons.size(); i++) {
4.  Constraint Soft = SoftCons.get(i);
5.  ArrayList<Integer> KlsCons = Soft.getConsClass();
6.  String type = Soft.getType();
7.  int angka1 = Soft.getAngka1();
8.  int angka2 = Soft.getAngka2();
9.  penalty = Soft.getPenalty();
10. if (type.contains("WorkDay") || type.contains
    ("MinGap") || type.contains("MaxDays") || type.
    contains("MaxDayLoad") || type.contains
    ("MaxBreaks") || type.contains("MaxBlock")) {
11. jmlPenalty += Cons1(type, angka1, angka2, KlsCons,
    penalty); } else {
12. jmlPenalty += Cons2(type, KlsCons, travel, penalty);}
13. } return jmlPenalty;}
```

Kode Program 5.32 Mengecek *Soft Constraint*

5.2.7.1. *Same Start*

Batasan *same start* mengharuskan semua kelas dalam batasan ini dimulai pada waktu yang bersamaan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap waktu mulai (*start*) kelas. Fungsi ini akan bernilai benar apabila waktu mulai (*start*) antar kelas sama, namun apabila batasan ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.33.

```
1.    if(startA == startB){
2.    jPenalty += 0;
3.    } else{
4.    jPenalty += penalty;}
5.    return jPenalty;
```

Kode Program 5.33 Batasan *Same Start* Sebagai *Soft Constraint*

5.2.7.2. *Same Time*

Semua kelas yang ada dalam batasan *same start* harus dijadwalkan dalam suatu rentang waktu yang bersamaan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap waktu mulai (*start*) dan waktu berakhir (*end*) kelas. Fungsi ini akan bernilai benar apabila kelas A dimulai lebih dahulu dan berakhir setelah kelas B berakhir, atau kelas B dimulai lebih dahulu dan dan berakhir setelah kelas A berakhir, namun apabila batasan

ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.34.

```
1.  if ((startA <= startB && endB <= endA) || (startB <=
    startA && endA <= endB)) {
2.  jPenalty += 0;
3.  } else {
4.  jPenalty += penalty;}
5.  return jPenalty;
```

Kode Program 5.34 Batasan *Same Time* Sebagai *Soft Constraint*

5.2.7.3. *Different Time*

Semua kelas dalam batasan ini tidak boleh dijadwalkan dalam suatu rentang waktu yang bersamaan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap waktu berakhir (*end*) dan waktu dimulainya (*start*) kelas. Fungsi ini akan bernilai benar apabila waktu berakhirnya kelas A lebih dahulu dibanding waktu mulainya kelas B, atau waktu berakhirnya kelas B lebih dahulu dibanding waktu mulainya kelas A, namun apabila batasan ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.35.

```
1.  if (endA <= startB || endB <= startA) {
2.  jPenalty += 0;
3.  } else { jPenalty += penalty;}
4.  return jPenalty;
```

Kode Program 5.35 Batasan *Different Time* Sebagai *Soft Constraint*

5.2.7.4. *Same Days*

Semua kelas dalam batasan ini harus dijadwalkan pada hari yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada hari yang sama, namun apabila batasan ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.36.

```
1.  if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
2.  jPenalty += 0;
3.  break;
4.  } else {
5.  salah += 1;}
6.  if (salah == daysA.size()) {
7.  jPenalty += penalty;}
8.  return jPenalty;
```

Kode Program 5.36 Batasan *Same Days* Sebagai *Soft Constraint*

5.2.7.5. *Different Days*

Semua kelas dalam batasan ini tidak dapat dijadwalkan pada hari yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada hari yang berbeda, namun apabila batasan ini dilanggar akan

dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.37.

```
1.  if ((daysA.get(dy) == 0 && daysB.get(dy) == 0) ||
    (daysA.get(dy) == 1 && daysB.get(dy) == 0) ||
    (daysA.get(dy) == 0 && daysB.get(dy) == 1)) {
2.  jPenalty += 0;
3.  } else {
4.  jPenalty += penalty;
5.  break;
6.  return jPenalty;
```

Kode Program 5.37 Batasan *Different Days* Sebagai *Soft Costraint*

5.2.7.6. *Same Weeks*

Semua kelas dalam batasan ini harus dijadwalkan pada minggu yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada minggu yang sama, namun apabila batasan ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.38.

```
1.  if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
2.  jPenalty += 0; break;
3.  } else { salah += 1;}
4.  if (salah == weeksA.size()) { jPenalty += penalty;}
```

Kode Program 5.38 Batasan *Same Weeks* Sebagai *Soft Constraint*

5.2.7.7. *Different Weeks*

Semua kelas dalam batasan ini tidak dapat dijadwalkan pada minggu yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada minggu yang berbeda, namun apabila batasan ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.39.

```
1.  if ((weeksA.get(wk) == 0 && weeksB.get(wk) == 0)
    || (weeksA.get(wk) == 1 && weeksB.get(wk) == 0) ||
    (weeksA.get(wk) == 0 && weeksB.get(wk) == 1)) {
2.  jPenalty += 0;
3.  } else {
4.  jPenalty += penalty;
5.  break;}
6.  return jPenalty;
```

Kode Program 5.39 Batasan *Different Weeks* Sebagai *Soft Constraint*

5.2.7.8. *Same Room*

Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang sama. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap *id* kelas untuk mengetahui apakah *id* kelas yang sedang dicek sama. Kemudian masing-masing kelas dalam *arraylist* dicek apakah sudah dijadwalkan dalam ruangan. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap ruangan untuk pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada ruangan yang sama, namun

apabila batasan ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.40.

```
1.  if(roomA == roomB){
2.  jPenalty += 0;
3.  } else{
4.  jPenalty += penalty;}
5.  return jPenalty;
```

Kode Program 5.40 Batasan *Same Room* Sebagai *Soft Constraint*

5.2.7.9. *Different Room*

Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang berbeda. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam ruangan. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan terhadap ruangan untuk pelaksanaan kelas. Fungsi ini akan bernilai benar apabila kelas A dan kelas B dijadwalkan pada ruangan yang berbeda, namun apabila batasan ini dilanggar akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.41.

```
1.  if(roomA != roomB){
2.  jPenalty += 0;
3.  } else{
4.  jPenalty += penalty;}
5.  return jPenalty;
```

Kode Program 5.41 Batasan *Different Room* Sebagai *Soft Constraint*

5.2.7.10. *Overlap*

Semua kelas dalam batasan ini harus dijadwalkan pada waktu yang saling tumpang tindih. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas

dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan minggu, hari, dan waktu dimulai serta berakhirnya kelas. Fungsi ini akan bernilai benar apabila waktu dimulainya kelas B lebih dahulu dibandingkan waktu berakhirnya kelas A dan waktu dimulainya kelas A lebih dahulu dibandingkan waktu berakhirnya kelas B. Apabila memenuhi kondisi tersebut maka dilakukan pengecekan terhadap minggu dan hari pelaksanaan kelas A dan B, yang mana keduanya harus dijadwalkan di minggu dan hari yang sama. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.42.

```
1.    if((startB < endA) && (startA < endB)){
2.    outerloop:
3.    for (int wk = 0; wk < weeksA.size(); wk++) {
4.    if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
5.    for (int dy = 0; dy < daysA.size(); dy++) {
6.    if(daysA.get(dy) == 1 && daysB.get(dy) == 1){
7.    jPenalty += 0;
8.    break outerloop;
9.    } else{ daySalah += 1;
10.   if (daySalah == daysA.size()) {
11.   jPenalty += penalty;}}}
12.   } else { weekSalah += 1;
13.   if (weekSalah == weeksA.size()) {
14.   jPenalty += penalty;}}}
15.   } else { jPenalty += penalty;}}}
16.   return jPenalty;
```

Kode Program 5.42 Batasan *Overlap* Sebagai *Soft Constraint*

5.2.7.11. *Not Overlap*

Semua kelas dalam batasan ini harus dijadwalkan pada waktu yang tidak saling tumpang tindih. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka dilakukan pengecekan dan pembandingan minggu, hari, dan waktu dimulai serta berakhirnya kelas. Terdapat fungsi *for* untuk mengecek apakah kelas A dan B dijadwalkan pada minggu dan hari yang sama. Apabila memenuhi kondisi tersebut maka dilakukan pengecekan terhadap waktu dimulai dan berakhirnya kelas A dan kelas B. Fungsi ini bernilai benar apabila waktu berakhirnya kelas A lebih dahulu atau sama dengan waktu dimulainya kelas B, atau waktu berakhirnya kelas B lebih dahulu atau sama dengan waktu dimulainya kelas A. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.43.

```
1.  outerloop:
2.  for (int wk = 0; wk < weeksA.size(); wk++) {
3.  if(weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
4.  for (int dy = 0; dy < daysA.size(); dy++) {
5.  if(daysA.get(dy) == 1 && daysB.get(dy) == 1){
6.  if((endA <= startB) || (endB <= startA)){
7.  jPenalty += 0;
8.  } else { jPenalty += penalty;
9.  break outerloop;}
10. return jPenalty;
```

Kode Program 5.43 Batasan *Not Overlap* Sebagai *Soft Constraint*

5.2.7.12. *Same Attendees*

Semua kelas dalam batasan ini harus bertemu pada waktu dan ruangan tertentu sehingga mahasiswa atau dosen dapat menghadirinya. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah memiliki ruangan dan telah dijadwalkan dalam *timeslot*. Kemudian dilakukan pengecekan terhadap *id* kelas untuk mengetahui apakah *id* kelas yang sedang dicek sama. Apabila kedua kelas yang sedang dicek memiliki ruangan, dilakukan pengambilan waktu *travel* antara kelas satu dengan lainnya. Selanjutnya digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila waktu berakhirnya kelas A ditambah dengan waktu *travel* kurang dari atau sama dengan waktu dimulainya kelas B, atau ketika waktu berakhirnya kelas B ditambah dengan waktu *travel* kurang dari atau sama dengan waktu dimulainya kelas A. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.44.

```
1.  outerloop: for (int wk=0; wk < weeksA.size(); wk++){
2.  if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
3.  for (int dy = 0; dy < daysA.size(); dy++){
4.  if (daysA.get(dy) == 1 && daysB.get(dy) == 1){
5.  if ((endA + trvl) <= startB || (endB + trvl) <= startA) {
6.  jPenalty += 0;
7.  break outerloop;
8.  } else { jPenalty += penalty;}
```

Kode Program 5.44 Batasan *Same Attendees* Sebagai *Soft Constraint*

5.2.7.13. *Precedence*

Batasan ini menetapkan daftar urutan, dimana kelas yang terdaftar harus dijadwalkan pada minggu, hari, atau timeslot yang lebih awal. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas.

```
1.  outerloop:
2.  for (int wk = 0; wk < weeksA.size(); wk++) {
3.  if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
    break outerloop;
4.  } else if (weeksA.get(wk) == 1 && weeksB.get(wk)
    == 1) {
5.  for (int dy = 0; dy < daysA.size(); dy++) {
6.  if (daysA.get(dy) > daysB.get(dy)) {
7.  break outerloop;
8.  }else if (daysA.get(dy) ==1 && daysB.get(dy) == 1) {
9.  if (endA <= startB) {
10. break outerloop;
11. } else{ jPenalty += penalty;
12. break outerloop;}
13. } else if (daysA.get(dy) < daysB.get(dy)) {
14. jPenalty += penalty; break outerloop;}}
15. } else if (weeksA.get(wk) < weeksB.get(wk)) {
16. jPenalty += penalty;
17. break outerloop;}
```

Kode Program 5.45 Batasan *Precedence* Sebagai *Soft Constraint*

Fungsi ini akan bernilai benar apabila pada minggu dan hari yang sama, waktu berakhirnya kelas B kurang dari atau sama dengan waktu dimulainya kelas A, atau pada saat hari atau minggu pelaksanaan kelas A lebih dahulu dibanding hari atau minggu pelaksanaan kelas B, begitu pun ketika pelaksanaan kelas B lebih dahulu dibanding kelas A. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.45.

5.2.7.14. *WorkDay*

Batasan ini mencegah atau memberikan penalti ketika terdapat pasangan kelas yang ada dalam daftar, dimana waktu antara mulainya kelas pertama dan berakhirnya kelas terakhir, melebihi S timeslot. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas.

```
1.  outerloop: for (int wk=0; wk<weeksA.size(); wk++) {
2.  if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
3.  for (int dy = 0; dy < daysA.size(); dy++) {
4.  if(daysA.get(dy) == 1 && daysB.get(dy)==1){
5.  if((Math.max(endA, endB)) - (Math.min(startA,
    startB)) <= S){
6.  break outerloop;
7.  } else{ jPenalty += penalty;
8.  return jPenalty;
```

Kode Program 5.46 Batasan *WorkDay* Sebagai *Soft Constraint*

Fungsi ini akan bernilai benar apabila antara waktu dimulainya kelas pertama sampai dengan waktu berakhirnya kelas terakhir

pada suatu hari tidak melebihi S *timeslot*. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.46.

5.2.7.15. *MinGap*

Dua kelas yang dijadwalkan pada hari yang bersamaan harus memiliki setidaknya timeslot sebesar G yang terpisah (antara berakhirnya kelas pertama dan mulainya kelas kedua). Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila waktu berakhirnya kelas A ditambah G *timeslot* tidak melebihi waktu dimulainya kelas B, atau ketika waktu berakhirnya kelas B ditambah G *timeslot* tidak melebihi waktu dimulainya kelas A. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.47.

```
1.  outerloop: for (int wk=0; wk<weeksA.size(); wk++) {
2.  if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
3.  for (int dy = 0; dy < daysA.size(); dy++) {
4.  if(daysA.get(dy) == 1 && daysB.get(dy)==1){
5.  if((endA + G <= startB) || (endB + G <= startA)){
6.  break outerloop;
7.  } else{ jPenalty += penalty;}
8.  return jPenalty;
```

Kode Program 5.47 Batasan *MinGap* Sebagai *Soft Constraint*

5.2.7.16. *MaxDays*

Kelas-kelas pada batasan ini tidak boleh dijadwalkan ke lebih dari D hari dalam satu minggu. Dalam fungsi batasan ini, terlebih dahulu digunakan fungsi *for* untuk mengecek hari pelaksanaan kelas yang kemudian dihitung berapa jumlah hari pelaksanaan kelas dalam satu minggu. Fungsi ini akan bernilai benar apabila jumlah hari pelaksanaan kelas dalam satu minggu tidak melebihi D. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.48.

```
1.  if (countNonzeroBits(daysA) <= D) {
2.  jPenalty += 0;
3.  } else {
4.  lebih = (countNonzeroBits(daysA) - D);
5.  jPenalty += (lebih * penalty);}
6.  return jPenalty;
7.  private int countNonzeroBits(ArrayList<Integer>
    daysA) { int count = 0;
8.  for (int i = 0; i < daysA.size(); i++) {
9.  if (daysA.get(i) == 1) {
10. count++;}}
11. return count;}
```

Kode Program 5.48 Batasan *MaxDays* Sebagai *Soft Constraint*

5.2.7.17. *MaxDayLoad*

Batasan ini membatasi jumlah total waktu yang ditetapkan untuk sekumpulan kelas tidak lebih dari S timeslot tiap hari selama satu semester. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila

sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu dan hari pelaksanaan kelas. Fungsi ini akan bernilai benar apabila beban kerja pada suatu hari tidak melebihi S *timeslot*. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.49.

```
1.  if (DayLoad(S, length, days, weeks, listMDL) == 0) {
2.  return jPenalty;
3.  } else {
4.  jPenalty = (penalty * (DayLoad(S, lengthA, daysA,
    weeksA)) / weeksA.size());}
5.  return jPenalty;}
6.  private int DayLoad(int S,int length,ArrayList
    <Integer> days, ArrayList<Integer> weeks) {
7.  int lebih = 0;
8.  for (int wk = 0; wk < weeksA.size(); wk++) {
9.  for (int dy = 0; dy < daysA.size(); dy++) {
10. if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
11. if(length > S){
12. lebih += (length-S);}
13. return lebih;
```

Kode Program 5.49 Batasan *MaxDayLoad* Sebagai *Soft Constraint*

5.2.7.18. *MaxBreaks*

Batasan ini membatasi jumlah jeda antar kelas selama sehari yang melebihi S *timeslot*, tidak boleh lebih dari R per hari. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan

minggu dan hari pelaksanaan kelas. Jika waktu berakhirnya kelas A ditambah S *timeslot* tidak kurang dari waktu dimulainya kelas B dan ketika waktu berakhirnya kelas B ditambah S *timeslot* tidak kurang dari waktu dimulainya kelas A, maka dianggap benar . Fungsi ini akan bernilai benar apabila jumlah blok tidak melebihi jumlah R+1. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.50.

```
1.  boolean Maxbreaks(int R, int S, ArrayList<Integer>
    listMB, int penalty){
2.  if(MergeBlocks(S, listMB) <= (R+1)){
3.  return jPenalty;
4.  } else { jPenalty = (penalty * (MergeBlocks(S, listMB)
    - R)) / weeksA.size();}
5.  return jPenalty;}
6.  private int MergeBlocks(int S, ArrayList<Integer>
    listMB) { int initialBlock = 1;
7.  if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
8.  if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
9.  if (endA + S >= startB && endB + S >= startA) {
10. startA = Math.min(startA, startB);
11. endA = Math.max(endA, endB);
12. } else { initialBlock++;
13. startA = startB;
14. endA = endB;
15. daysA = daysB;
16. weeksA = weeksB;}
17. return initialBlock;}
```

Kode Program 5.50 Batasan *MaxBreaks* Sebagai *Soft Constraint*

5.2.7.19. MaxBlock

Semua kelas dalam batasan ini harus dijadwalkan tidak melebihi batas maksimal blok yang ditetapkan. Dalam fungsi batasan ini, terlebih dahulu dilakukan pengecekan terhadap masing-masing kelas dalam *arraylist* apakah sudah dijadwalkan dalam *timeslot*. Apabila sudah dijadwalkan, maka digunakan fungsi *for* untuk mengecek dan membandingkan minggu pelaksanaan kelas. Fungsi ini akan bernilai benar apabila penjadwalan tidak melebihi M jumlah blok. Jika batasan ini dilanggar maka akan dikenakan penalti sebagaimana yang ditunjukkan pada Kode Program 5.51.

```
1.  boolean Maxblock(int M, int S, ArrayList<Integer>
    listMBL, int penalty){
2.  if(MergeBlockB(M, S, listMBL) <= M){
3.  return jPenalty;}
4.  } else {
5.  jPenalty = (penalty * MergeBlockB(M, S, listMBL)) /
    weeksA.size();}
6.  return jPenalty;}
7.  private int MergeBlockB(int M, int S,
    ArrayList<Integer> listMBL) {
8.  int lengthBlock = 0;
9.  int lebih = 0;
10. boolean blockBaru = true;
11. Kelas kls = SearchKls(listMBL.get(0));
12. outerloop:
13. for (int wk=0; wk < weeksA.size(); wk++) {
14. for (int dy = 0; dy < daysA.size(); dy++) {
15. if(weeksA.get(wk) == 1 && daysA.get(dy) == 1 &&
    weeksB.get(wk) == 1 && daysB.get(dy) == 1){
```

```
16.  if(endA+S >= startB && endB+S >= startA){
17.  startA = Math.min(startA, startB);
18.  endA = Math.max(endA, endB);
19.  lengthBlock = endA - startA;
20.  if(lengthBlock > M && blockBaru == true){
21.  lebih += 1;
22.  blockBaru = false;
23.  }else{
24.  lebih += 0;}
25.  } else{
26.  lengthBlock = endA - startA;
27.  if(lengthBlock > M && blockBaru == true){
28.  lebih += 1;
29.  }else{
30.  lebih += 0;}
31.  blockBaru = true;
32.  startA = startB;
33.  endA = endB;
34.  daysA = daysB;
35.  weeksA = weeksB;
36.  } else if (iTS == -1) { return lengthBlock;}}
37.  return lengthBlock;}
```

Kode Program 5.51 Batasan *MaxBlock* Sebagai *Soft Constraint*

5.3. Optimasi Solusi

Bagian ini akan menjelaskan bagaimana cara untuk mengoptimasi jadwal yang sudah terbentuk pada solusi awal. Optimasi akan dilakukan dengan melakukan implementasi algoritma *Great Deluge Hyper-Heuristics*. Dalam tahapan optimasi solusi, beberapa aktivitas yang dilakukan antara lain, menyimpan solusi awal dalam struktur data tertentu, membaca

solusi awal, pembuatan dan pemilihan *low level heuristic*, implementasi algoritma, dan penyimpanan solusi optimum.

5.3.1. Penyimpanan Solusi Awal

Solusi awal yang telah terbentuk, kemudian disimpan sebagai tahap awal untuk proses optimasi. Penyimpanan dilakukan pada setiap objek kelas. Penyimpanan solusi ini dibangun dengan tujuan untuk menyimpan solusi setiap iterasi optimasi, sehingga riwayat perubahan solusi dapat diketahui. Adapun kode program penyimpanan solusi awal berbasis file dapat dilihat pada Kode Program 5.56.

5.3.2. Pembuatan *Low Level Heuristics*

Low level heuristic merupakan metode yang digunakan untuk memindah urutan solusi sehingga menghasilkan nilai penalti baru. Pada penelitian tugas akhir ini, ditentukan *low level heuristic* berupa "*move*". *Move* dilakukan dengan memindahkan *timeslot* atau *room* milik satu atau beberapa mata kuliah ke *timeslot* atau *room* acak yang terpilih. Adapun contoh *low level heuristic* ditunjukkan oleh Kode Program 5.52, Kode Program 5.53, dan Kode Program 5.54.

```
1.  int random = rand.nextInt(5);
2.  if (random == 0) {
3.  move1TSRM();
4.  move1TS();
5.  move1RM();}
6.  outerloop:for (int i = 0; i < listKelas.size(); i++) {
7.  Kelas kelas = listKelas.get(i);
8.  int TS = kelas.getTs();
9.  int RM = kelas.getRoom();
10. int idC = kelas.getClassID();
```

```
11.  if (kelas.isHasRoom() == true) {
12.  ArrayList<TimeAss> availableTS = kelas.get
     AvailableTS();
13.  ArrayList<RoomAss> availableroom = kelas.get
     Availableroom();
14.  for (int tm = 0; tm < availableTS.size(); tm++) {
15.  TimeAss tims = availableTS.get(tm);
16.  for (int rm = 0; rm < availableroom.size(); rm++) {
17.  RoomAss rooms = availableroom.get(rm);
18.  unvR = cekUnroom(tims, rooms);
19.  if (unvR == false) {
20.  System.out.println(idC);
21.  System.out.println("UNAVAILABLE ROOM!");
22.  break outerloop;}}}
23.  } else { unvR = true;}
24.  hardC = hard.HardConstraint(TS, RM, kelas,
     listKelas, travel);
25.  if (!hardC) {
26.  break;}}
27.  if (hardC && unvR) {
28.  penaltyTSBaru=countTSPenalty(listKelas)*bobot[0];
29.  penaltyRMBaru = countRMPenalty(listKelas) * bobot
     [1];
30.  penaltySCBaru = soft.totalPenalty(listKelas, travel,
     distrib) * bobot[2];
31.  newPenalty = penaltyTSBaru + penaltyRMBaru +
     penaltySCBaru;
32.  } else { newPenalty = X;}
```

Kode Program 5.52 ***Low Level Heuristics*** **Umum**

```
1.  void move1TS() {
2.  int kls, ts, tsBaru = 0, rm, rmBaru, idKls;
3.  ArrayList<TimeAss> times;
4.  ArrayList<RoomAss> rooms;
5.  do { kls = rand.nextInt(listKelas.size() - 1);
6.  Kelas kelas = listKelas.get(kls);
7.  int id = kelas.getClassID();
8.  times = kelas.getAvailableTS();
9.  ts = kelas.getTs();
10. if ((ts + 1) < times.size()) {
11. tsBaru = ts + 1;
12. kelas.setTs(tsBaru);
13. } else if (ts + 1 >= times.size()) {
14. tsBaru = 0;
15. kelas.setTs(tsBaru);}
16. } while (ts == tsBaru);}
```

Kode Program 5.53 ***Low Level Heuristics - Move Timeslot***

```
1.  void move1RM() {
2.  int kls, ts, tsBaru = 0, rm, rmBaru, idKls;
3.  ArrayList<TimeAss> times;
4.  ArrayList<RoomAss> rooms;
5.  do { kls = rand.nextInt(listKelas.size() - 1);
6.  Kelas kelas = listKelas.get(kls);
7.  int id = kelas.getClassID();
8.  times = kelas.getAvailableTS();
9.  rooms = kelas.getAvailableroom();
10. ts = kelas.getTs();
11. rm = kelas.getRoom();
12. if (rm != -1) {
```

```
13.  if ((rm + 1) < rooms.size()) {
14.  rmBaru = rm + 1;
15.  kelas.setRoom(rmBaru);
16.  } else if (rm + 1 >= rooms.size()) {
17.  rmBaru = 0;
18.  kelas.setRoom(rmBaru);}}
19.  } while (rm == -1);}
```

Kode Program 5.54 ***Low Level Heuristics – Move Room***

Selain *low level heuristics* yang ditunjukkan pada Kode Program 5.52, Kode Program 5.53, dan Kode Program 5.54 dalam penelitian tugas akhir ini juga dilakukan eksperimen dengan menggunakan sejumlah mata kuliah yang dilakukan *move* terhadap 2 *timeslot*, 2 *room*, gabungan 1 *timeslot* dan 1 *room*, dan gabungan 2 *timeslot* dan 2 *room*.

5.3.3. Implementasi Algoritma

Implementasi algoritma *Great Deluge* dilakukan untuk memilih solusi yang optimum. Algoritma *Great Deluge* diimplementasikan dengan membandingkan nilai penalti solusi sebagai *acceptance criteria*. Berdasarkan penjelasan tersebut, solusi baru dapat diterima menjadi solusi terbaik jika memenuhi salah satu dari 2 syarat, yaitu (1) nilai penalti dari solusi baru lebih rendah daripada nilai penalti solusi saat ini, atau (2) nilai penalti dari solusi baru tidak melebihi nilai batas (*level*). Sehingga dapat dituliskan melalui kode program seperti pada Kode Program 5.55. Optimasi menggunakan algoritma *Great Deluge* terus dijalankan hingga *stopping criteria*, yaitu berdasarkan jumlah iterasi yang ditentukan.

```
1.  Do{
2.  if(newPenalty < currentPenalty){
3.  currentPenalty = newPenalty;
4.  penaltyTS = penaltyTSBaru;
5.  penaltyRM = penaltyRMBaru;
6.  penaltySC = penaltySCBaru;
7.  penaltyTS = penaltyTSBaru;
8.  for (int i = 0; i < listKelas.size(); i++) {
9.  Kelas kelas = listKelas.get(i);
10. kelas.setTsBaru(kelas.getTs());
11. kelas.setRoomBaru(kelas.getRoom());}
12. notImproveCounter = 0;
13. } else{ currentPenalty = newPenalty;
14. penaltyTS = penaltyTSBaru;
15. penaltyRM = penaltyRMBaru;
16. penaltySC = penaltySCBaru;
17. notImproveCounter = 0;
18. } else{ notImproveCounter++;
19. if(notImproveCounter == notImproveCounterGDA){
20. break;}}
21. for (int i = 0; i < listKelas.size(); i++) {
22. Kelas kelas = listKelas.get(i);
23. kelas.setTs(kelas.getTsBaru());
24. kelas.setRoom(kelas.getRoomBaru());}}
25. level = level - decayrate;
26. iterasi++;
27. } while(iterasi < numOfIte);
```

Kode Program 5.55 Algoritma *Great Deluge*

5.3.4. Penyimpanan Solusi Optimum Akhir

Solusi optimum akhir yang telah terbentuk dari hasil optimasi algoritma kemudian disimpan dalam file berekstensi .xml. Adapun kode program yang digunakan untuk penyimpanan solusi akhir ditunjukkan oleh Kode Program 5.56.

```
1.  public class WriteSol {
2.  WriteSol(String instance, ArrayList<Kelas> listKelas,
    int penalty) {
3.  String name = "C:\\Users\\user\\Documents\\"
    + "NetBeansProjects\\ITC2019\\src\\itcoop\\
    solution.xml";
4.  try{ Collections.sort(listKelas, new sortIdKelas());
5.  DocumentBuilderFactory documentFactory =
    DocumentBuilderFactory.newInstance();
6.  DocumentBuilder documentBuilder = document
    Factory.newDocumentBuilder();
7.  Document document = documentBuilder.new
    Document();
8.  Element root = document.createElement("solution");
9.  root.setAttribute("name", instance+"("+penalty + ")");
10. root.setAttribute("technique", "Great Deluge");
11. root.setAttribute("author", "Kharisma");
12. root.setAttribute("institution", "ITS");
13. for (int i = 0; i < listKelas.size(); i++) {
14. Element kelas = document.createElement("class");
15. int idKls = kls.getClassID();
16. int idRM = room.get(RM).getRoom_id();
17. int start = time.get(TS).getStart();
18. int length = time.get(TS).getLength();
```

```
19.  String wkwk = time.get(TS).getWkwk();
20.  String dydy = time.get(TS).getDydy();
21.  kelas.setAttribute("id", Integer.toString(idKls));
22.  kelas.setAttribute("days", dydy);
23.  kelas.setAttribute("start", Integer.toString(start));
24.  kelas.setAttribute("length", Integer.toString(length));
25.  kelas.setAttribute("weeks", wkwk);
26.  if (RM != -1) {
27.  kelas.setAttribute("room", Integer.toString(idRM));}
28.  Element test = document.createElement("student");
29.  kelas.appendChild(test);
30.  root.appendChild(kelas);}
31.  document.appendChild(root);
32.  TransformerFactory transformerFactory =
     TransformerFactory.newInstance();
33.  Transformer transformer =
     transformerFactory.newTransformer();
34.  transformer.setOutputProperty(OutputKeys.INDENT,
     "yes");
35.  transformer.setOutputProperty(OutputKeys.
     STANDALONE, "yes");
36.  transformer.setOutputProperty("{http://xml.
     apache.org/xslt}indent-amount", "4");
37.  DOMSource domSource = new DOMSource
     (document);
38.  StreamResult streamResult = new StreamResult(new
     File(name));
39.  transformer.transform(domSource, streamResult);
```

Kode Program 5.56 Menyimpan Solusi Akhir

5.4. Perhitungan Penalti

Bagian ini akan menjelaskan penghitungan skor penalti yang dihasilkan dari proses optimasi. Skor penalti akan menentukan solusi yang paling optimum dalam beberapa eksperimen yang dilakukan. Skor penalti akhir merupakan total dari skor penalti yang dihasilkan pada masing-masing penalti pada Kode Program 5.12, Kode Program 5.32, Kode Program 5.52, dan Kode Program 5.55. Skor penalti akhir akan dibandingkan pada setiap eksperimen.

Halaman ini sengaja dikosongkan

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini akan menjelaskan mengenai proses dan hasil uji coba serta analisis terhadap hasil yang diperoleh dari proses implementasi Algoritma *Great Deluge hyper-heuristic*, termasuk eksperimen parameter yang digunakan.

6.1. Data Uji Coba

Data yang digunakan sebagai uji coba adalah dataset ITC 2019 yang digunakan langsung dalam penelitian tugas akhir ini.

6.2. Lingkungan Uji Coba

Pada subbab Lingkungan Uji Coba ini akan menjelaskan terkait lingkungan pengujian dalam melakukan implementasi penelitian tugas akhir terkait optimasi penjadwalan mata kuliah. Spesifikasi perangkat keras yang digunakan dalam implementasi ditunjukkan pada Tabel 6.1.

Tabel 6.1 Spesifikasi Perangkat Keras

Perangkat Keras	Spesifikasi
Jenis	Laptop HP Pavilion 14-v043tx Notebook PC
Processor	Intel(R) Core(TM) i7-4510U 2.00GHz
RAM	4 GB
Hard Disk Drive	1 TB 5400 rpm SATA

Untuk spesifikasi perangkat lunak yang digunakan dalam pengerjaan penelitian tugas akhir ditunjukkan pada Tabel 6.2.

Tabel 6.2 Spesifikasi Perangkat Lunak

Perangkat Lunak	Fungsi
Windows 10 64 bit	Sistem Operasi
Netbeans 8.0	Implementasi Algoritma
Sublime Text	Pengolahan Data dan Hasil
Microsoft Word 365 Pro Plus	Penulisan Laporan

6.3. Penentuan Urutan Objek Pembentukan Solusi Awal

Subbab ini menjelaskan susunan dan urutan indeks mata kuliah yang digunakan dalam pembentukan solusi awal. Dalam pembentukan solusi awal, urutan indeks obyek mata kuliah harus diperhatikan untuk mendapatkan solusi yang lolos *hard constraint*. Indikator yang digunakan dalam mengurutkan indeks objek mata kuliah antara lain jumlah daftar waktu tersedia (*available times*), jumlah daftar ruang tersedia (*available rooms*), jumlah *hard constraint* per kelas, dan jumlah kelas yang sudah di urutkan (*Sorted Class*).

6.3.1. Skenario Urutan Indikator 1 Kombinasi

Pada skenario 1 pembentukan solusi awal, tidak dilakukan kombinasi indikator. Urutan objek hanya berdasarkan 1 indikator terkait. Hasil dari eksperimen skenario 1 ditunjukkan pada Tabel 6.3 yang mana angka di dalam tabel merupakan jumlah kelas yang belum terjadwal.

Tabel 6.3 Hasil Skenario 1 Pembentukan Solusi Awal

Instances	Indikator		
	Timeslot	***Rooms***	***Hard Constraint***
Test-Tiny	**0**	**0**	**0**
Test-Small 1	12	9	19
Test-Small 2	**0**	**0**	**0**
Test-Medium	20	14	28
Test-Large	158	114	174
Early 1	39	63	66
Early 2	101	127	154
Early 3	268	280	285
Early 4	120	139	158
Early 5	11	18	16
Early 6	28	35	37
Early 7	34	49	45
Early 8	83	131	125
Early 9	29	16	29
Early 10	28	62	76
Rata-rata	**62,07**	**70,47**	**80,8**

6.3.2. Skenario Urutan Indikator 2 Kombinasi

Berdasarkan Tabel 6.3, diketahui bahwa indikator *timeslot* dan *rooms* memiliki hasil yang lebih baik daripada *hard constraint*. Sehingga pada tahap eksperimen ini dilakukan kombinasi 2 indikator dengan menggunakan indikator *timeslot* dan *rooms*. Hasil eksperimen tersebut ditunjukkan pada Tabel 6.4.

Tabel 6.4 Hasil Skenario 2 Pembentukan Solusi Awal

Instances	Indikator			
	Time - Room	***Time - HC***	***Room - Time***	***Room - HC***
Test-Tiny	**0**	**0**	**0**	**0**
Test-Small 1	11	12	9	9
Test-Small 2	**0**	**0**	**0**	**0**
Test-Medium	17	20	10	14
Test-Large	148	158	92	114
Early 1	46	39	59	63
Early 2	116	101	116	127
Early 3	268	268	266	280
Early 4	114	120	128	139
Early 5	13	11	14	18
Early 6	25	28	33	35
Early 7	39	34	42	49
Early 8	91	83	124	131
Early 9	28	29	14	16
Early 10	26	28	43	62
Rata-rata	**62,8**	**62,07**	**63,33**	**70,47**

6.3.3. Skenario Urutan Indikator 3 Kombinasi

Hasil eksperimen sebelumnya yang ditunjukkan oleh Tabel 6.4 menunjukkan bahwa kombinasi *Rooms-Times* dan *Times-HC* memiliki hasil yang paling baik daripada kombinasi lain. Pada skenario 3 dengan 3 kombinasi indikator, dilakukan eksperimen menggunakan kombinasi *Rooms-Times* dan *Times-HC* pada susunan pertama dan kedua. Adapun hasil dari skenario 3 menggunakan 3 kombinasi indikator ditunjukkan oleh Tabel 6.5.

Tabel 6.5 Hasil Skenario 3 Pembentukan Solusi Awal

Instances	Indikator		
	Room – Time – HC	***Time – HC - Room***	***Sorted Class***
Test-Tiny	**0**	**0**	**0**
Test-Small 1	9	12	14
Test-Small 2	**0**	**0**	**0**
Test-Medium	10	20	10
Test-Large	92	158	93
Early 1	59	39	47
Early 2	116	101	122
Early 3	266	268	261
Early 4	128	120	103
Early 5	14	11	12
Early 6	33	28	25
Early 7	42	34	41
Early 8	124	83	106
Early 9	14	29	13
Early 10	43	28	34
Rata-rata	**63,33**	**62,07**	**58,73**

Berdasarkan Tabel 6.5, *Sorted Class* atau kombinasi kelas yang sudah diurutkan menunjukkan hasil rata-rata yang paling baik. Hasil eksperimen yang ditunjukkan dalam Tabel 6.3, Tabel 6.4, Tabel 6.5 belum ada kombinasi indikator yang mampu menjadwalkan semua kelas pada semua *dataset*, hanya pada *dataset Test-Tiny* dan *Test-Small 2* semua kelas dapat terjadwal. Uji coba menggunakan metode *backtrack*, mengulang terus iterasi sehingga menyebabkan iterasi tidak bisa berhenti jika belum menemukan solusi yang tepat, juga sudah dilakukan untuk menemukan solusi dari data yang belum terjadwalkan. Namun proses dari metode *backtrack* tidak menghasilkan hasil

yang memuaskan. Sehingga penelitian tugas akhir ini dilanjutkan dengan menggunakan 2 *dataset* yaitu *Test-Tiny* dan *Test-Small 2*. Sementara itu kombinasi yang digunakan sebagai pembentuk solusi awal sebelum dilakukan proses optimasi adalah *sorted class* karena mampu menghasilkan rata-rata nilai fungsi tujuan akhir yang paling baik, selain itu *sorted class* juga mampu menjadwalkan semua kelas pada *dataset Test-Tiny* dan *Test-Small2*.

6.4. Hasil Eksperimen Implementasi Solusi Awal

Subbab ini menjelaskan mengenai hasil implementasi solusi awal terhadap *dataset* ITC 2019. Dari implementasi yang dilakukan, didapatkan nilai penalti masing-masing *instance* sebagaimana tertera pada Tabel 6.6.

Tabel 6.6 Hasil Implementasi Solusi Awal

Instance	Initial Solution
Test-Tiny	53
Test-Small 2	1536

Dari hasil implementasi diatas, maka untuk selanjutnya penelitian tugas akhir ini hanya berfokus pada *instance* yang memiliki nilai penalti dari solusi awal yaitu *test instance 2 (tiny)* dan *test instance 4 (small 2)*. Nilai fungsi tujuan dari solusi awal inilah yang digunakan sebagai acuan untuk implementasi algoritma, dimana apabila hasil nilai fungsi tujuan akhir dari implementasi algoritma *Great Deluge* mengalami penurunan, maka hal tersebut mengindikasikan bahwa penerapan algoritma *Great Deluge* telah mampu mengoptimasi solusi awal yang dihasilkan.

6.5. Hasil Eksperimen Implementasi Algoritma

Subbab ini menjelaskan mengenai hasil optimasi *dataset* ITC 2019 menggunakan algoritma *Great Deluge hyper-heuristic*. Setiap eksperimen dijalankan sebanyak 11 kali, mengacu pada penelitian sebelumnya. Tabel 6.7 menunjukkan daftar parameter yang digunakan. Nilai parameter didasarkan pada percobaan penulis secara acak, namun dalam rentang nilai yang masih memungkinkan berdasarkan beberapa penelitian sebelumnya.

Tabel 6.7 Daftar Parameter Percobaan

Parameter	Keterangan
Level	Nilai batas
Estimate Penalty	Nilai perkiraan penalti = 100
Iterasi	Jumlah iterasi yang dilakukan
Decay rate	Tingkat peluruhan
LLH	*Low level heuristics* yang digunakan

Nilai parameter *level* diinisiasi sama dengan nilai penalti saat ini, sedangkan nilai *estimate penalty* ditetapkan 100, sementara terhadap nilai *decayrate*, jumlah iterasi dan *low level heuristics* dilakukan uji coba menggunakan berbagai skenario. Untuk masing-masing scenario, uji coba dilakukan sebanyak sebelas kali.

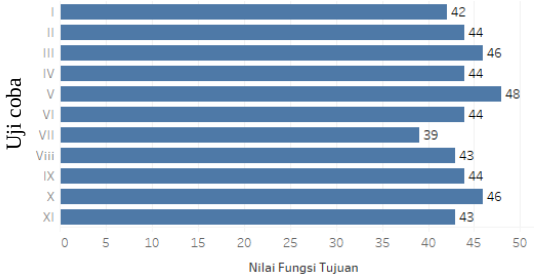
6.5.1. Skenario A

Pada skenario A, percobaan dilakukan menggunakan parameter jumlah iterasi untuk mengetahui apakah jumlah iterasi mempengaruhi nilai fungsi tujuan akhir. Dalam penelitian tugas akhir ini dilakukan uji coba dengan mengubah jumlah iterasi mulai dari 1.000, 10.000, 100.000, hingga 1.000.000, dengan kondisi lingkungan dimana parameter *decay rate* ditetapkan

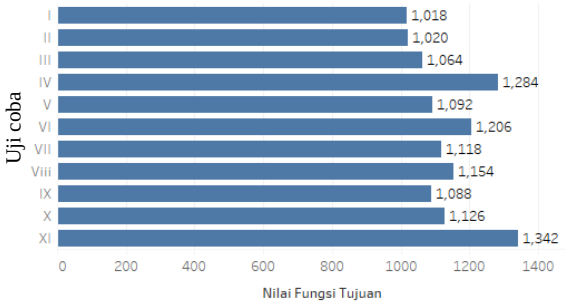
dengan rumus (*current penalty-estimate penalty*) / jumlah iterasi, sementara *low level heuristics* yang digunakan merupakan kombinasi dari *move 1 timeslot*, *move 1 room*, *move 1 timeslot-room*, *move 2 timeslot*, *move 2 room*, dan *move 2 timeslot-room*. Pada setiap jumlah iterasi yang berbeda dilakukan uji coba sebanyak sebelas kali.

6.5.1.1. Iterasi 1.000

Pada uji coba ini nilai parameter iterasi diubah menjadi 1.000 yang berarti algoritma melakukan iterasi sebanyak 1.000 kali. Adapun hasil uji coba ini ditampilkan pada Gambar 6.1 dan Gambar 6.2.



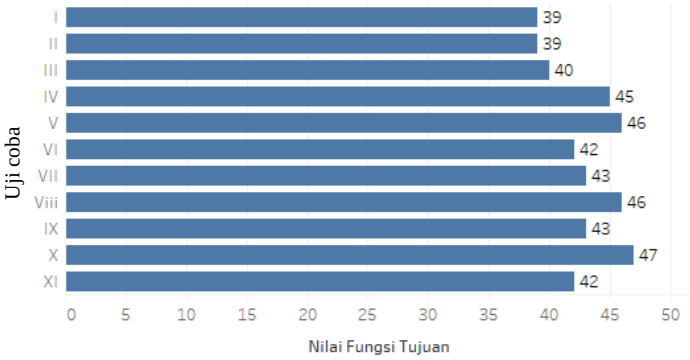
Gambar 6.1 Hasil Skenario Jumlah Iterasi 1.000 Data *Test-Tiny*



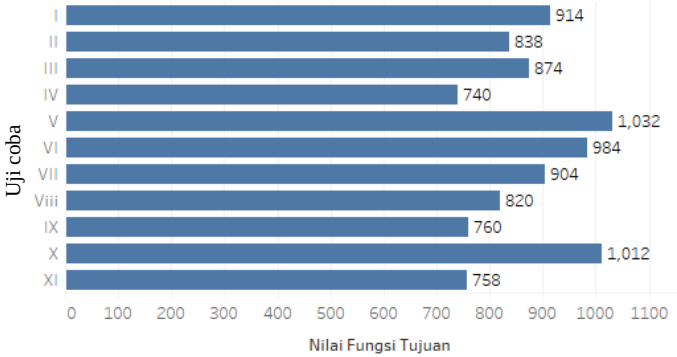
Gambar 6.2 Hasil Skenario Jumlah Iterasi 1.000 Data *Test-Small2*

6.5.1.2. Iterasi 10.000

Pada uji coba ini nilai parameter iterasi diubah menjadi 10.000 yang berarti algoritma melakukan iterasi sebanyak 10.000 kali. Adapun hasil uji coba ini ditampilkan pada Gambar 6.3 dan Gambar 6.4.



Gambar 6.3 Hasil Skenario Jumlah Iterasi 10.000 Data *Test-Tiny*

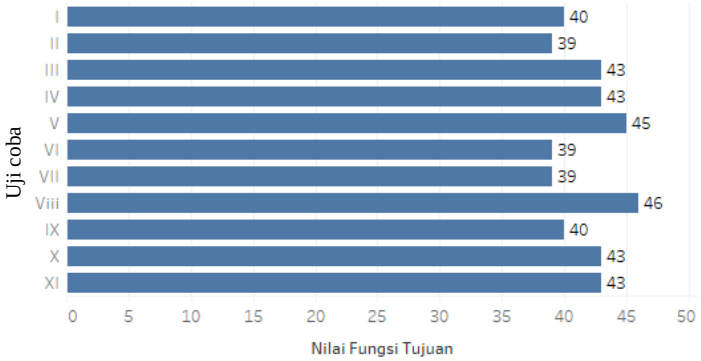


Gambar 6.4 Hasil Skenario Jumlah Iterasi 10.000 Data *Test-Small2*

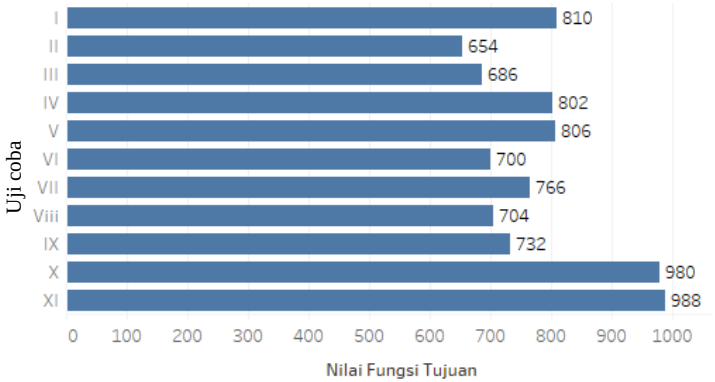
6.5.1.3. Iterasi 100.000

Pada uji coba ini nilai parameter iterasi diubah menjadi 100.000 yang berarti algoritma melakukan iterasi sebanyak 100.000 kali.

Adapun hasil uji coba ini ditampilkan pada Gambar 6.5 dan Gambar 6.6.



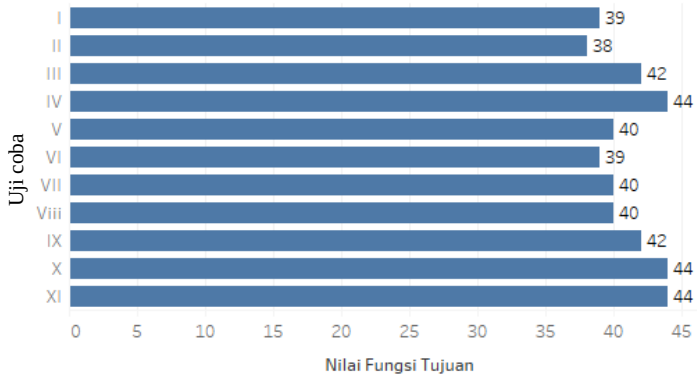
Gambar 6.5 Hasil Skenario Jumlah Iterasi 100.000 Data *Test-Tiny*



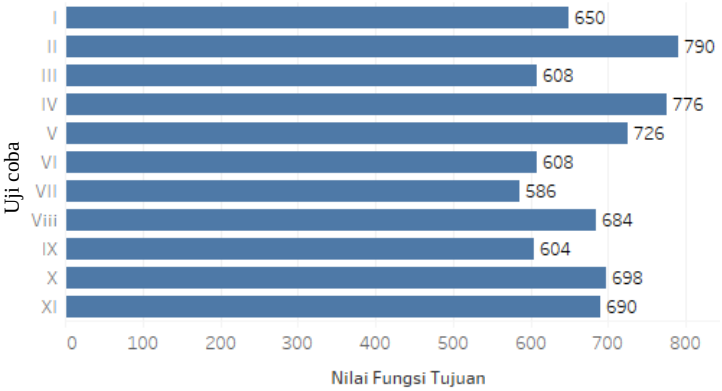
Gambar 6.6 Hasil Skenario Jumlah Iterasi 100.000 Data *Test-Small2*

6.5.1.4. Iterasi 1.000.000

Pada uji coba ini nilai parameter iterasi diubah menjadi 1.000.000 yang berarti algoritma melakukan iterasi sebanyak 1.000.000 kali. Adapun hasil uji coba ini ditampilkan pada Gambar 6.7 dan Gambar 6.8.



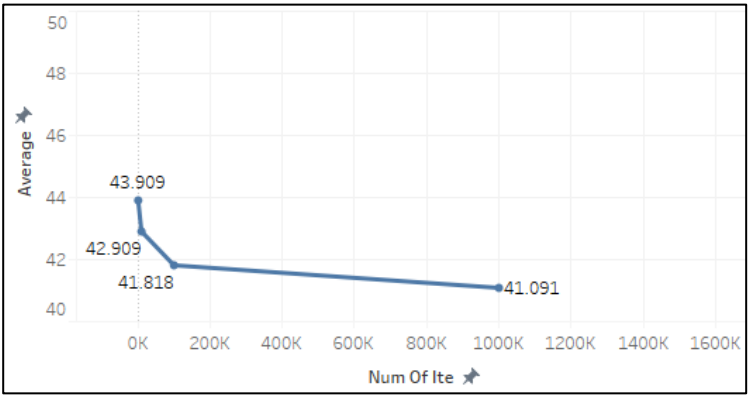
Gambar 6.7 Hasil Skenario Jumlah Iterasi 1.000.000 Data *Test-Tiny*



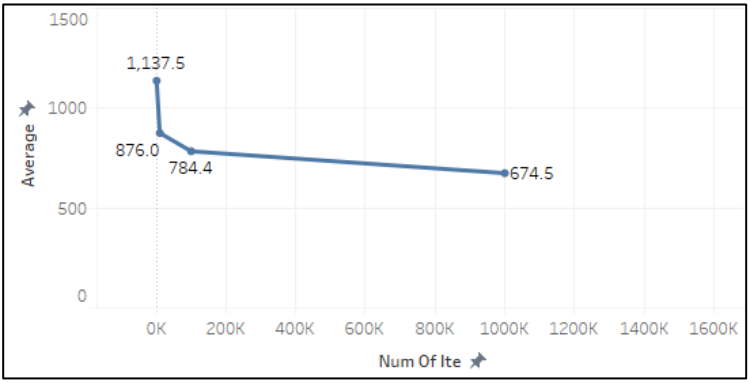
Gambar 6.8 Hasil Skenario Jumlah Iterasi 1.000.000 Data *Test-Small2*

6.5.1.5. Perbandingan Hasil Skenario A

Rata-rata hasil uji coba skenario A dengan melakukan pengubahan jumlah iterasi 1.000, 10.000, 100.000, 1.000.000 dapat dilihat pada Gambar 6.9 dan Gambar 6.10.



Gambar 6.9 Rata-rata Hasil Skenario A, Data *Test-Tiny*



Gambar 6.10 Rata-rata Hasil Skenario A, Data *Test-Small2*

Berdasarkan rata-rata pada Gambar 6.9 dan Gambar 6.10 dapat diketahui bahwa nilai penalti cenderung menurun seiring bertambahnya jumlah iterasi yang dilakukan.

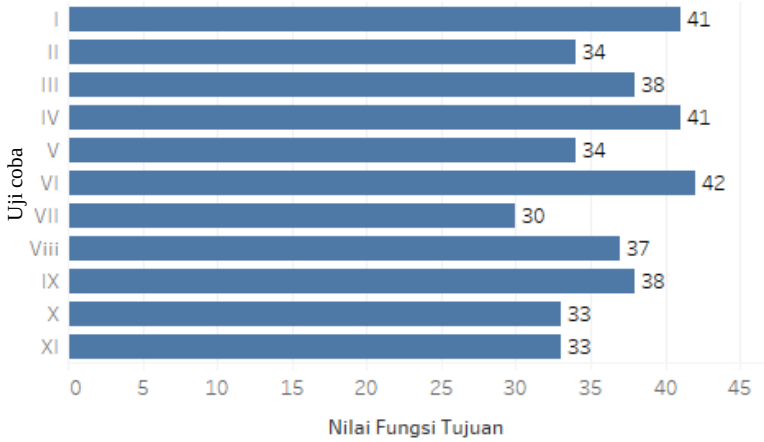
6.5.2. Skenario B

Pada skenario B, percobaan dilakukan menggunakan jenis LLH *move* dengan 3 skenario, yaitu (1) *move* 1 *timeslot*, 1 *room*, 1

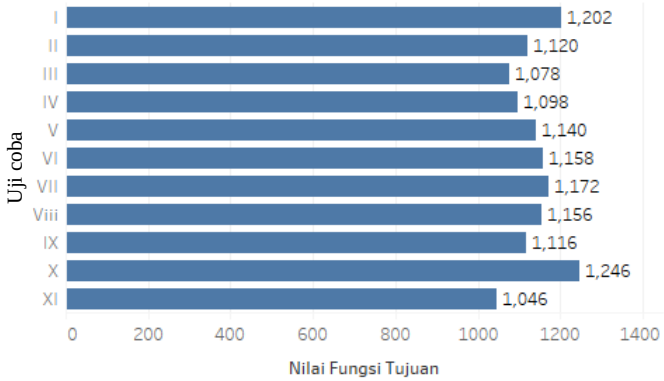
timeslot-room, (2) *move 2 timeslot, 2 room, 2 timeslot-room*, (3) kombinasi skenario (1) dan (2). Sementara itu kondisi lingkungan yang ditetapkan yaitu parameter *decay rate* sesuai dengan rumus (*current penalty-estimate penalty*) / jumlah iterasi, sedangkan jumlah iterasi adalah 1.000.

6.5.2.1. *Move 1 timeslot, 1 room, 1 timeslot-room*

Pada uji coba ini dilakukan percobaan menggunakan parameter LLH dengan skenario *move 1 timeslot, 1 room,* dan *1 rimeslot-room* yang berarti algoritma melakukan iterasi dengan memindahkan jadwal kelas dari satu *timeslot* dan satu *room* ke *timeslot* dan *room* lain. Adapun hasil uji coba ini ditampilkan pada Gambar 6.11 dan Gambar 6.12.



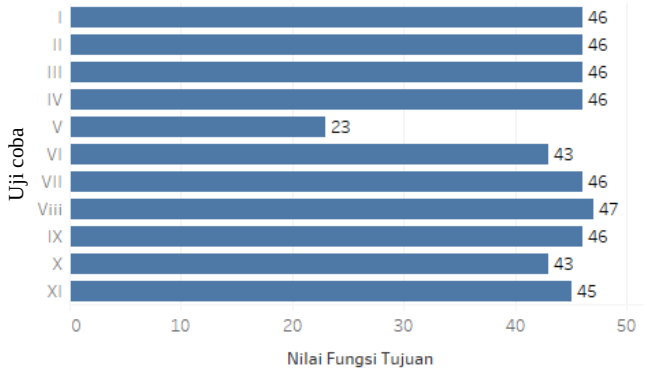
Gambar 6.11 Hasil Skenario LLH 1, Data *Test-Tiny*



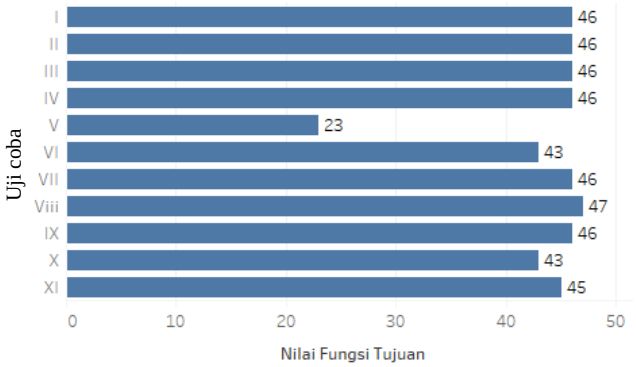
Gambar 6.12 Hasil Skenario LLH 1, Data *Test-Small2*

6.5.2.2. *Move 2 timeslot, 2 room, 2 timeslot-room*

Pada uji coba ini dilakukan percobaan menggunakan parameter LLH dengan skenario *move 2 timeslot*, *2 room*, dan *2 timeslot-room* yang berarti algoritma melakukan iterasi dengan memindahkan jadwal kelas dari dua ruang dan dua *timeslot* ke ruang dan *timeslot* lain. Adapun hasil uji coba ini ditampilkan pada Gambar 6.13 dan Gambar 6.14.



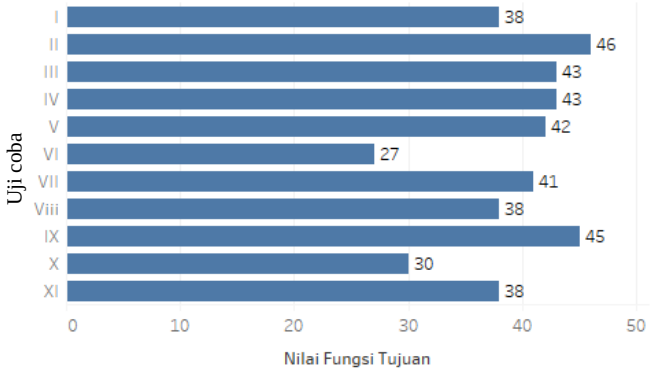
Gambar 6.13 Hasil Skenario LLH 2, Data *Test-Tiny*



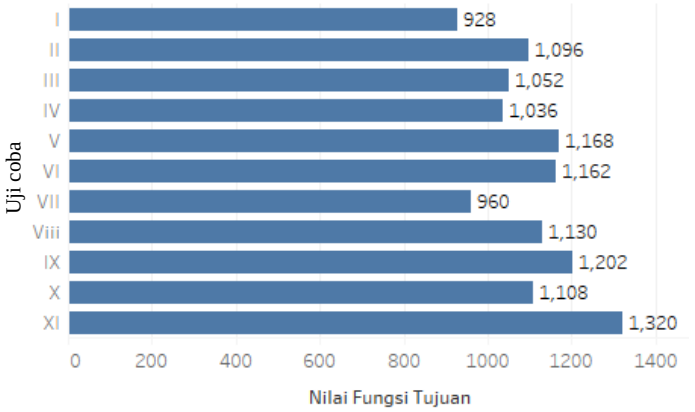
Gambar 6.14 Hasil Skenario LLH 2, Data *Test-Small2*

6.5.2.3. *Move* Kombinasi *Timeslot* dan *Room*

Pada uji coba ini dilakukan percobaan menggunakan parameter LLH dengan kombinasi skenario 6.5.2.1 dan 6.5.2.2 yang berarti algoritma melakukan iterasi dengan memindahkan jadwal kelas dari suatu waktu dan ruang ke waktu dan ruang yang lain. Adapun hasil uji coba ini ditampilkan pada Gambar 6.15 dan Gambar 6.16.



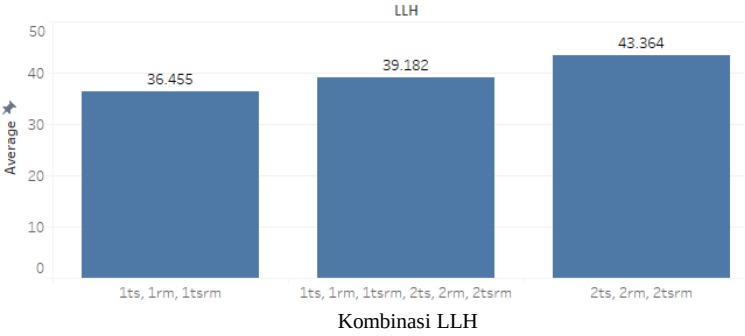
Gambar 6.15 Hasil Skenario LLH Kombinasi, Data *Test-Tiny*



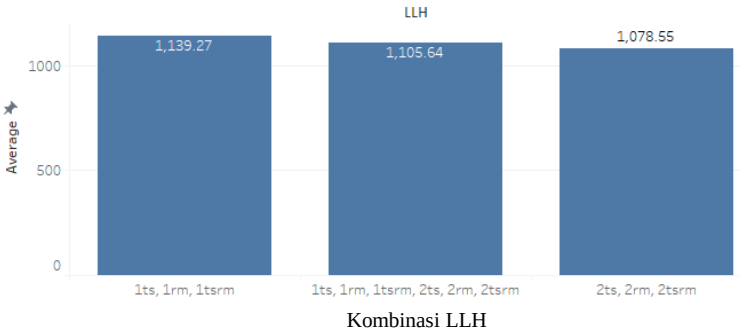
Gambar 6.16 Hasil Skenario LLH Kombinasi, Data *Test-Small2*

6.5.2.4. Perbandingan Hasil Skenario B

Rata-rata hasil uji coba skenario B dengan melakukan pengubahan *low level heuristics* dapat dilihat pada Gambar 6.17 dan Gambar 6.18.



Gambar 6.17 Rata-rata Hasil Skenario B, Data *Test-Tiny*



Gambar 6.18 Rata-rata Hasil Skenario B, Data *Test-Small2*

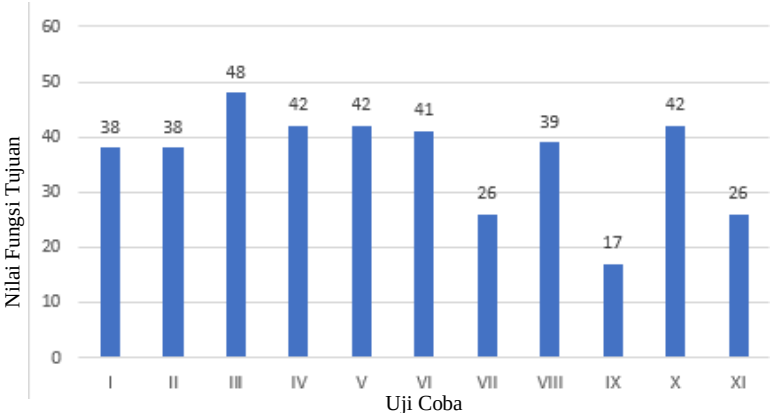
Berdasarkan data pada Gambar 6.17 dan Gambar 6.18 dapat diketahui bahwa nilai penalti cenderung menurun seiring dengan semakin beragamnya atribut yang dipindah (*move*).

6.5.3. Skenario C

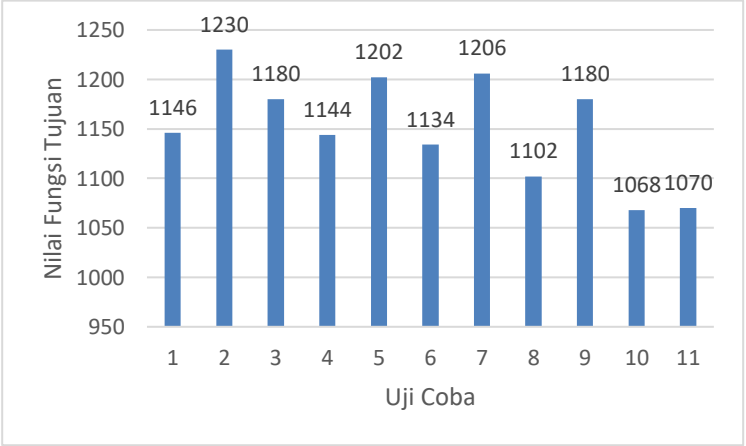
Pada skenario C, dilakukan dengan menerapkan uji coba perhitungan *decay rate*, yaitu dengan nilai 0, 0,02, 0,2, 0,5, dan 0,8. Skenario ini dilakukan dengan kondisi lingkungan parameter yaitu jumlah iterasi 1.000 dan *low level heuristics* yang digunakan merupakan kombinasi dari *move 1 timeslot*, *move 1 room*, *move 1 timeslot-room*, *move 2 timeslot*, *move 2 room*, dan *move 2 timeslot-room*. Masing-masing iterasi dilakukan sebanyak 11 kali percobaan untuk mengetahui rata-rata nilai fungsi tujuan akhir dari setiap *decay rate* yang diujikan.

6.5.3.1. *Decay Rate* 0

Pada uji coba ini nilai parameter *decay rate* diubah menjadi 0. Adapun hasil uji coba terhadap data *Test-Tiny* ditampilkan pada Gambar 6.19, sementara hasil terhadap data *Test-Small2* ditampilkan pada Gambar 6.20.



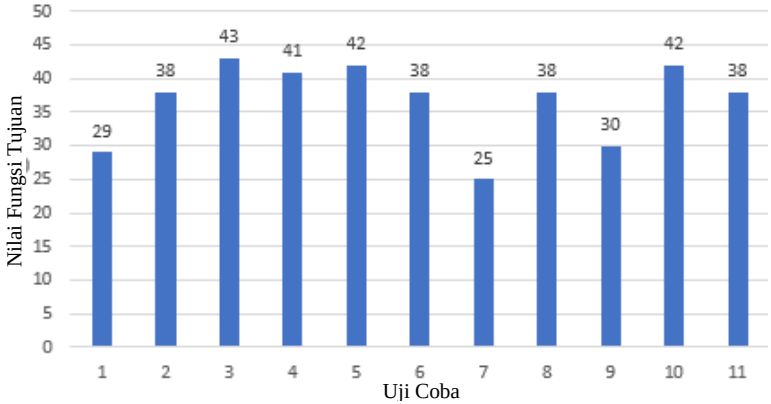
Gambar 6.19 Hasil Skenario *Decay Rate* 0, Data *Test-Tiny*



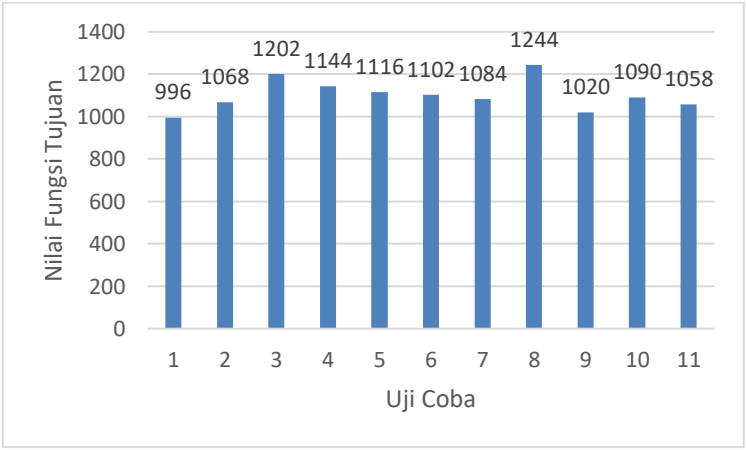
Gambar 6.20 Hasil Skenario *Decay Rate* 0, Data *Test-Small2*

6.5.3.2. *Decay Rate* 0,02

Pada uji coba ini nilai parameter *decay rate* diubah menjadi 0,02. Adapun hasil uji coba terhadap data *Test-Tiny* ditampilkan pada Gambar 6.21, sementara hasil terhadap data *Test-Small2* ditampilkan pada Gambar 6.22.



Gambar 6.21 Hasil Skenario *Decay Rate* 0,02, Data *Test-Tiny*

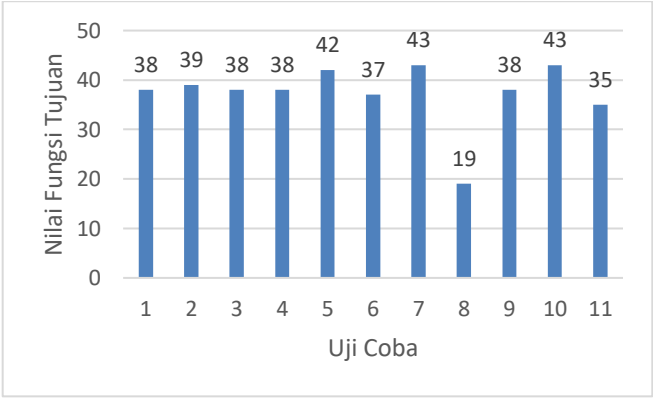


Gambar 6.22 Hasil Skenario *Decay Rate* 0,02, Data *Test-Small2*

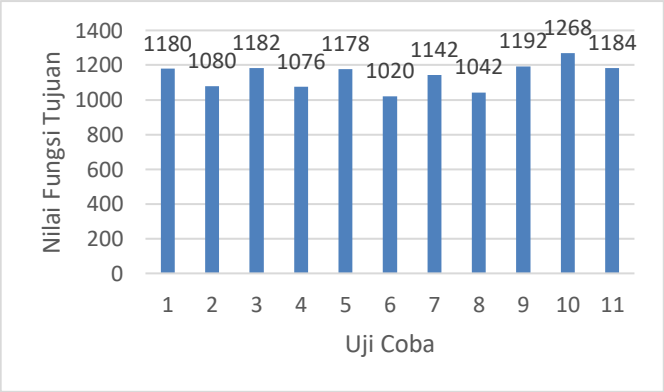
6.5.3.3. *Decay Rate* 0,2

Pada uji coba ini nilai parameter *decay rate* diubah menjadi 0,2. Adapun hasil uji coba terhadap data *Test-Tiny* ditampilkan pada

Gambar 6.23, sementara hasil terhadap data *Test-Small2* ditampilkan pada Gambar 6.24.



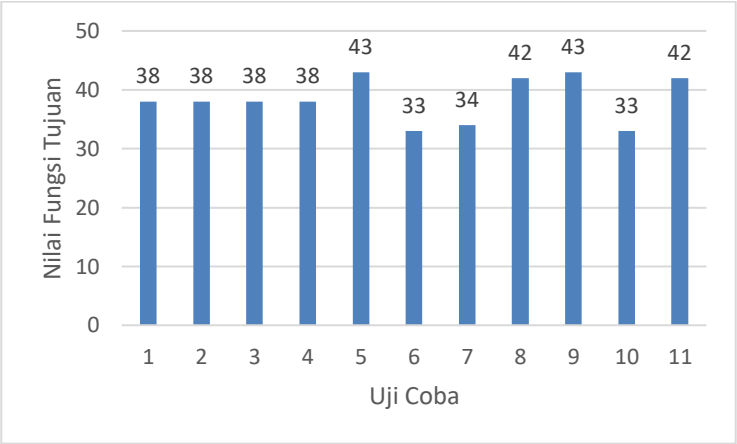
Gambar 6.23 Hasil Skenario *Decay Rate* 0,2, Data *Test-Tiny*



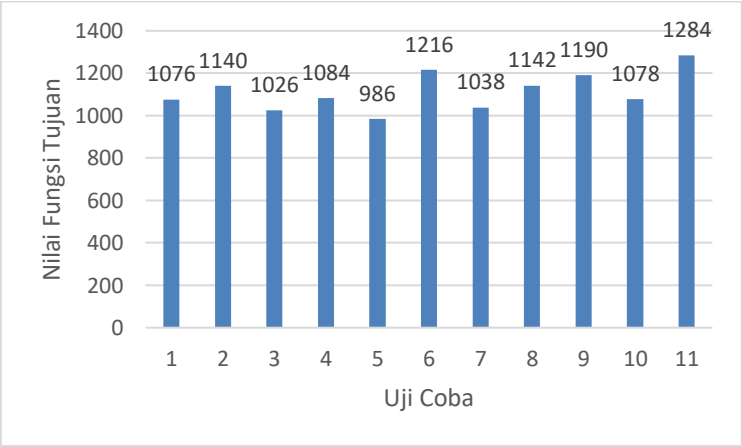
Gambar 6.24 Hasil Skenario *Decay Rate* 0,2, Data *Test-Small2*

6.5.3.4. *Decay Rate* 0,5

Pada uji coba ini nilai parameter *decay rate* diubah menjadi 0,5. Adapun hasil uji coba terhadap data *Test-Tiny* ditampilkan pada Gambar 6.25, sementara hasil terhadap data *Test-Small2* ditampilkan pada Gambar 6.26.



Gambar 6.25 Hasil Skenario *Decay Rate* 0,5, Data *Test-Tiny*

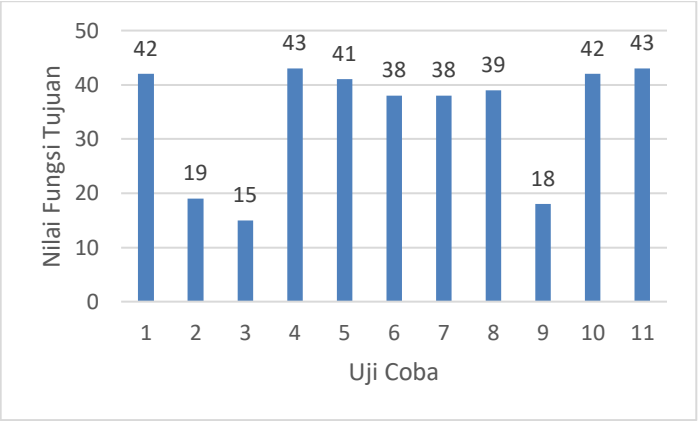


Gambar 6.26 Hasil Skenario *Decay Rate* 0,5, Data *Test-Small2*

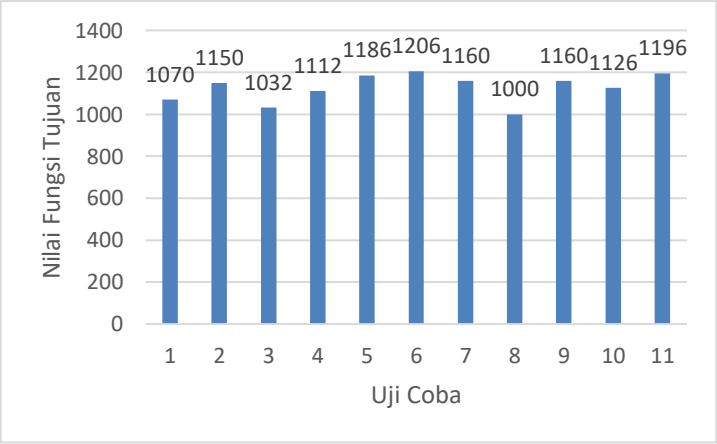
6.5.3.5. *Decay Rate* 0,8

Pada uji coba ini nilai parameter *decay rate* diubah menjadi 0,8. Adapun hasil uji coba terhadap data *Test-Tiny* ditampilkan pada

Gambar 6.27, sementara hasil terhadap data *Test-Small2* ditampilkan pada Gambar 6.28.



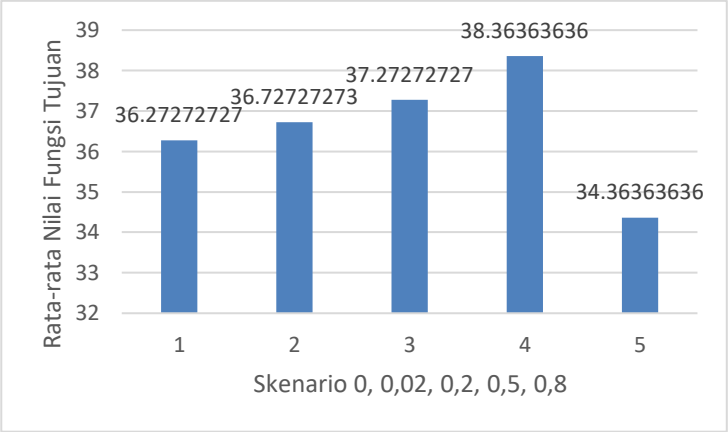
Gambar 6.27 Hasil Skenario *Decay Rate* 0,8, Data *Test-Tiny*



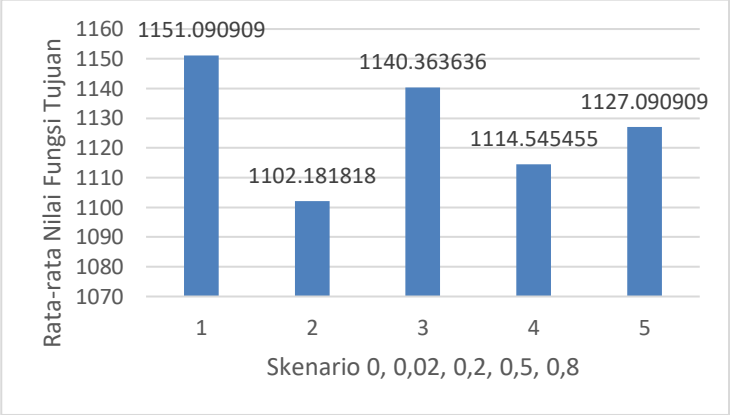
Gambar 6.28 Hasil Skenario *Decay Rate* 0,8, Data *Test-Small2*

6.5.3.6. Perbandingan Hasil Skenario C

Rata-rata hasil uji coba skenario C dengan melakukan pengubahan nilai *decay rate* dapat dilihat pada Gambar 6.29 dan Gambar 6.30.



Gambar 6.29 Rata-rata Hasil Skenario C, Data *Test-Tiny*



Gambar 6.30 Rata-rata Hasil Skenario C, Data *Test-Small2*

Berdasarkan data pada Gambar 6.29 dan Gambar 6.30 dapat diketahui bahwa nilai penalti naik turun seiring dengan perubahan nilai *decay rate*.

6.5.4. Pembahasan Hasil Skenario Eksperimen Algoritma

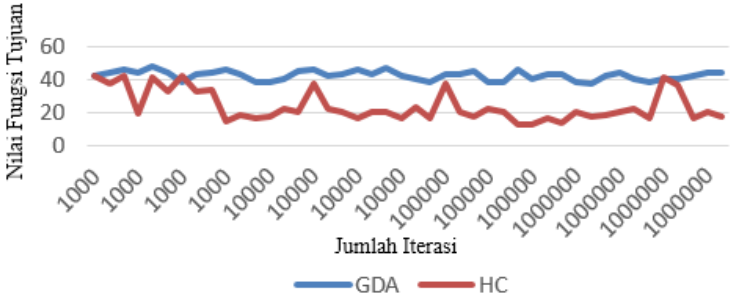
Hasil Skenario A, Skenario B, dan Skenario C menunjukkan bahwa uji coba yang dilakukan terhadap parameter jumlah iterasi, *decay rate*, dan *low level heuristics* mempengaruhi nilai penalti yang dihasilkan. Dari hasil yang telah didapatkan pada proses sebelumnya, diperoleh beberapa bahasan sebagai berikut.

1. Pemilihan jumlah iterasi pada skenario A memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana semakin banyak jumlah iterasi yang dilakukan maka nilai fungsi tujuan akhir yang didapatkan semakin baik.
2. Pemilihan *low level heuristics* pada skenario B memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana meskipun metode yang dipakai hanya *move* namun penerapan metode *move* dengan kombinasi atribut yang dipindah, menghasilkan nilai fungsi tujuan akhir yang lebih baik.
3. Pemilihan *decay rate* pada skenario C memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana perubahan nilai *decay rate* menghasilkan nilai fungsi tujuan akhir yang naik turun.

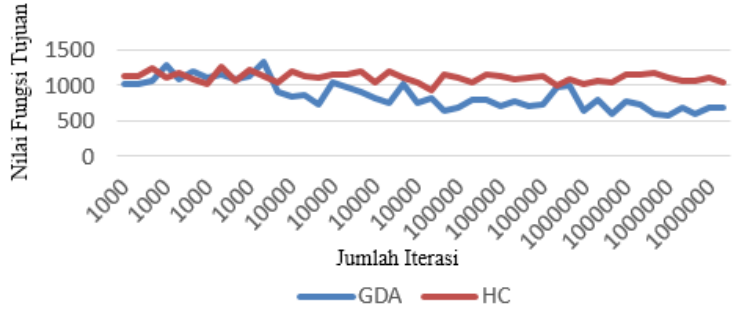
6.6. Perbandingan Hasil Eksperimen dengan Algoritma Lain

Pada subbab ini dilakukan perbandingan hasil uji coba menggunakan algoritma *great deluge* dengan hasil uji coba

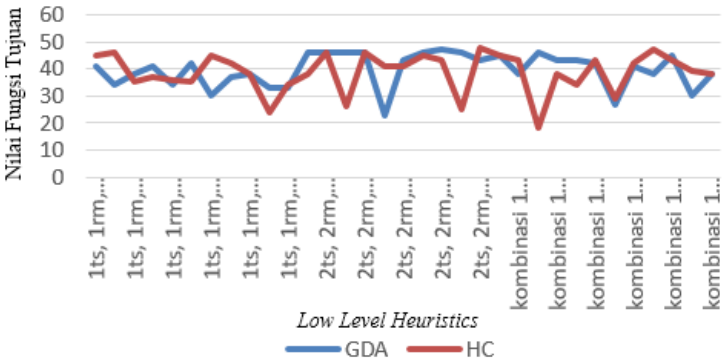
algoritma *hill climbing*. Uji coba ini dilakukan untuk mengetahui perbandingan kinerja algoritma dalam menentukan solusi terbaik. Percobaan dilakukan sebanyak 11 kali menggunakan skenario jumlah iterasi (Gambar 6.31 dan Gambar 6.32) dan *low level heuristics* (Gambar 6.33 dan Gambar 6.34).



Gambar 6.31 Perbandingan Hasil Skenario Jumlah Iterasi *Great Deluge* dan *Hill Climbing*, Data *Test-Tiny*



Gambar 6.32 Perbandingan Hasil Skenario Jumlah Iterasi *Great Deluge* dan *Hill Climbing*, Data *Test-Small2*



Gambar 6.33 Perbandingan Hasil Skenario LLH *Great Deluge* dan *Hill Climbing*, Data *Test-Tiny*

Gambar 6.34 Perbandingan Hasil Skenario LLH *Great Deluge* dan *Hill Climbing*, Data *Test-Small2*

Berdasarkan hasil percobaan menggunakan berbagai skenario yang dilakukan, didapatkan hasil terbaik diperoleh ketika menggunakan parameter jumlah iterasi. Dengan parameter jumlah iterasi, algoritma *hill climbing* menghasilkan rata-rata nilai fungsi tujuan akhir terbaik sebesar 1085,3, sementara itu algoritma *great deluge* menghasilkan rata-rata nilai fungsi tujuan akhir yang lebih baik yaitu sebesar 674,55.

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini akan dijelaskan rangkuman singkat yang dapat disimpulkan dari penelitian tugas akhir ini. Terdapat saran dari penulis yang diharapkan dapat membantu dalam meningkatkan hasil penelitian selanjutnya.

7.1. Kesimpulan

Berdasarkan hasil yang telah diuraikan pada bagian sebelumnya, kesimpulan yang dapat diambil adalah,

1. Penerapan algoritma *great deluge* untuk optimasi penjadwalan mata kuliah terbukti mampu menghasilkan nilai fungsi tujuan akhir yang lebih baik. Sebagai bukti penelitian tugas akhir ini telah menerapkan algoritma *great deluge* untuk optimasi solusi awal dan hasilnya mendukung pernyataan tersebut.
2. Perubahan jumlah iterasi memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana semakin banyak jumlah iterasi yang dilakukan maka hasil fungsi tujuan akhir yang didapatkan semakin baik. Sebagai bukti penelitian tugas akhir ini telah melakukan paling banyak 1.000.000 kali iterasi dan hasilnya mendukung pernyataan tersebut.
3. Perubahan metode *low level heuristics* memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana semakin beragam kombinasi atribut yang digunakan

maka hasil fungsi tujuan akhir yang didapatkan semakin baik. Sebagai bukti penelitian tugas akhir ini telah menggunakan tiga kombinasi atribut dan hasilnya mendukung pernyataan tersebut.

4. Perubahan nilai *decay rate* memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana perubahan nilai *decay rate* akan menghasilkan nilai fungsi tujuan akhir yang naik turun. Sebagai bukti penelitian tugas akhir ini telah menetapkan nilai *decay rate* paling banyak 10 dan hasilnya mendukung pernyataan tersebut.
5. Dalam menyelesaikan permasalahan penjadwalan mata kuliah terbukti performa algoritma *great deluge* lebih baik dibandingkan algoritma *hill climbing*. Sebagai bukti penelitian tugas akhir ini telah melakukan perbandingan hasil uji coba penerapan algoritma *great deluge* dengan algoritma *hill climbing*, yang mana hasilnya mendukung pernyataan tersebut.

7.2. Saran

Berdasarkan hasil dan kesimpulan tersebut, saran yang dapat diberikan untuk penelitian selanjutnya adalah,

1. Penelitian tugas akhir hanya mampu menghasilkan solusi untuk 2 dataset dari 15 dataset yang ada dalam studi kasus ITC 2019, sehingga diharapkan untuk penelitian selanjutnya dapat menyediakan solusi untuk seluruh dataset yang tersedia.
2. Dalam penelitian tugas akhir ini hanya dilakukan 3 skenario uji coba parameter, sehingga diharapkan untuk penelitian selanjutnya dapat melakukan lebih banyak

skenario uji coba parameter agar dapat lebih banyak menghasilkan variasi solusi.

3. Implementasi algoritma *Great Deluge* dalam penelitian tugas akhir ini belum disertai dengan pengembangan algoritma, sehingga harapannya untuk penelitian selanjutnya dapat dilakukan pengembangan terhadap algoritma *Great Deluge.*

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] K. Setemen, "IMPLEMENTASI ALGORITMA GENETIKA DALAM PENGEMBANGAN SISTEM APLIKASI PENJADWALAN KULIAH," no. 2005, pp. 56–68, 2008.

[2] D. Ariani, A. Fahriza, and I. Prasetyaningrum, "Optimasi Penjadwalan Mata Kuliah di Jurusan Teknik Informatika PENS Dengan Menggunakan Algoritma Particle Swarm Optimization (PSO)," pp. 1–11, 2011.

[3] D. A. R. Wati and Y. A. Rochman, "Model Penjadwalan Matakuliah Secara Otomatis Berbasis Algoritma Particle Swarm Optimization (PSO)," 2012.

[4] W. A. Puspaningrum, A. Djunaidy, and R. A. Vinarti, "Penjadwalan Mata Kuliah Menggunakan Algoritma Genetika di Jurusan Sistem Informasi ITS," vol. 2, no. 1, pp. 127–131, 2013.

[5] H. Saragih, G. Hoendarto, B. Reza, and D. Setiyadi, "APLIKASI SISTEM PERANGKAT LUNAK MENGGUNAKAN ALGORITMA ANT UNTUK MENGATUR PENJADWALAN KULIAH," pp. 241–256, 2012.

[6] T. B. Cooper, J. H. Kingston, T. B. Cooper, and J. H. Kingston, "The Complexity of Timetable Construction Problems Basser Department of Computer Science University of Sydney NSW 2006 The Complexity of Timetable Construction Problems," no. 495, pp. 0–12, 2006.

[7] A. Muklason, "Solver Penjadwal Ujian Otomatis Dengan Algoritma Maximal Clique dan Hyper-heuristics," pp. 18–19, 2017.

[8] A. Rochman, "PENJADWALAN KULIAH MENGGUNAKAN METODE CONSTRAINTS

PROGRAMMING DAN SIMULATED ANNEALING," vol. 2012, no. Snati, pp. 15–16, 2012.

[9] T. Muller, H. Rudova, and Z. Mullerova, "University Course Timetabling and International Timetabling Competition 2019," 2019.

[10] E. Burke, Y. Bykov, J. Newall, and S. Petrovi, "A TIME-PREDEFINED APPROACH TO COURSE TIMETABLING," vol. 13, no. 2, pp. 139–151, 2003.

[11] G. Dueck, "New Optimization Heuristics The Great Deluge Algorithm and The Record-to-Record Travel." 1991.

[12] E. Burke, Y. Bykov, J. Newall, and S. Petrovic, "A time-predefined local search approach to exam timetabling problems," *IIE Trans. (Institute Ind. Eng.*, 2004.

[13] P. Mcmullan, "An Extended Implementation of the Great Deluge Algorithm for Course Timetabling," *Lncs*, vol. 4487, pp. 538–545, 2007.

[14] G. Kendall, E. K. Burke, M. Hyde, G. Ochoa, E. Ozcan, and R. Qu, "A Survey of Hyper-heuristics," no. May 2014, 2009.

[15] a Jain, D. Jain, and D. Chande, "Formulation of Genetic Algorithm to Generate Good Quality Course Timetable," *Int. J. Innov. …*, vol. 1, no. 3, pp. 248–251, 2010.

[16] E. L. Lawler, "Combinatorial Optimization : Networks and Matroids," *Comb. Optim. networks matroids*, 1976.

[17] R. Qu, E. K. Burke, B. McCollum, L. T. G. Merlot, and S. Y. Lee, "A survey of search methodologies and automated system development for examination timetabling," *J. Sched.*, vol. 12, no. 1, pp. 55–89, 2009.

[18] M. Dimopoulou and P. Miliotis, "Implementation of a university course and examination timetabling system," vol. 130, pp. 202–213, 2001.

[19] B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke,

and R. Qu, “A new model for automated examination timetabling,” *Ann. Oper. Res.*, vol. 194, no. 1, pp. 291–315, 2012.

[20] Itc2019, “ITC 2019: International Timetabling Competition,” 2019. [Online]. Available: https://www.itc2019.org/home. [Accessed: 11-Feb-2019].

[21] E. K. Burke *et al.*, “Hyper-heuristics: A survey of the state of the art,” *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, 2013.

[22] G. Dueck and T. Scheuer, “Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing,” *J. Comput. Phys.*, 1990.

[23] B. McCollum, P. McMullan, A. J. Parkes, E. K. Burke, and S. Abdullah, “An Extended Great Deluge Approach to the Examination Timetabling Problem,” *Multidiscip. Int. Conf. Sched. Theory Appl. (MISTA 2009) 10-12 August 2009, Dublin, Irel.*, 2009.

[24] S. Abdullah, “Heuristic Approaches for University Timetabling Problems,” no. June, p. 226, 2006.

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Penulis memiliki nama lengkap Kharisma Diah Puspitasari. Lahir di Blitar pada tanggal 30 November 1996. Penulis merupakan anak terakhir dari orang tua bernama Setyo Satuhu Putro dan Widji Utami. Penulis menempuh pendidikan dasar di SD Negeri Sananwetan 3 Blitar pada tahun 2003 hingga 2009. Setelah lulus dari sekolah dasar, penulis melanjutkan pendidikan formal di bangku sekolah menengah pertama di SMP Negeri 1 Blitar dan lulus pada tahun 2012. Pada tahun yang sama, penulis melanjutkan pendidikan formal di SMA Negeri 1 Blitar dan lulus pada tahun 2015. Pada tahun 2015, penulis melanjutkan pendidikan tinggi di Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Insitut Teknologi Sepuluh Nopember (ITS) Surabaya, melalui jalur Seleksi Nasional Masuk Perguruan Tinggi (SNMPTN) tahun 2015.

Selama menjalani pendidikan tinggi di ITS, penulis aktif dalam berbagai kegiatan, baik organisasi maupun kepanitiaan. Penulis pernah menjadi staf Divisi *Information Media* Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi dan Komunikasi periode 2016/2017, staf Administrasi *Information Systems Expo* (*ISE*), dan staf Kesekretariatan Diklatmen Koperasi Mahasiswa dr Angka ITS pada tahun 2016. Pada tahun 2017, penulis pernah menjabat menjadi Koordinator Divisi Administrasi FTIF *Festival*, staf ahli Divisi Administrasi

Information Systems Expo (*ISE*), dan *Volunteer* dalam *Conference on Innovation and Industrial Application* (*CINIA*). Sementara itu mulai tahun 2018, penulis lebih aktif dalam kepanitian, antara lain menjadi staf Administrasi Bursa Karir ITS Ke-35 dan panitia babak penyisihan GEMASTIK 11.

Selain aktif dalam organisasi dan kepanitiaan, penulis juga mengikuti beberapa pelatihan diantaranya *SAP University Alliance Course* yang diadakan Departemen Sistem Informasi. Penulis juga telah mendapatkan sertifikasi *IC3* (*Internet Core Competency Certification*) pada tahun 2017, dan sertifikasi *Google Cloud* pada tahun 2018.

Untuk mengetahui informasi lebih lanjut mengenai penelitian ini maupun terkait dengan penulis, dapat menghubungi melalui email kharismadiah@gmail.com.

LAMPIRAN A: Hasil Optimasi Penjadwalan

Tabel A.1 Hasil Optimasi Penjadwalan Instance *Test-Tiny*

Class ID	Week	Day	Start	Length	Room
1	1-9	Senin, Selasa, Rabu, Kamis	114	22	8
2	1-9	Senin, Selasa, Rabu, Kamis	108	15	45
3	1-9	Senin, Selasa, Rabu, Kamis	96	15	12
4	1-9	Selasa, Kamis	114	34	62
5	1-9	Senin, Selasa, Rabu, Kamis	96	15	8
6	1-9	Senin, Selasa, Rabu, Kamis	144	22	28
7	1-9	Senin, Selasa, Rabu, Kamis	144	22	25
8	1-9	Senin, Selasa,	168	22	23

		Rabu, Kamis			
9	1-9	Senin, Selasa, Rabu, Kamis	138	22	23
10	1-9	Senin, Selasa, Rabu, Kamis	120	22	25
11	1-9	Senin, Selasa, Rabu, Kamis	108	22	28
12	1-9	Senin, Selasa, Rabu, Kamis	96	15	27
13	1-9	Senin, Selasa, Rabu, Kamis	150	22	22
14	1-9	Senin, Selasa, Rabu, Kamis	144	15	26
15	1-9	Senin, Selasa, Rabu, Kamis	108	22	38
16	1-9	Selasa, Kamis	96	58	58
17	1-9	Senin, Selasa,	120	22	22

		Rabu, Kamis			
18	1-9	Senin, Selasa, Rabu, Kamis	162	22	8
19	1-9	Senin, Selasa, Rabu, Kamis	96	22	24
20	1-9	Senin, Selasa, Rabu, Kamis	174	22	24