TUGAS AKHIR – IS184853

PENJADWALAN MATA KULIAH OTOMATIS MENGGUNAKAN ALGORITMA *LATE ACCEPTANCE HILL CLIMBING HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

AUTOMATED COURSE TIMETABLING USING LATE ACCEPTANCE HILL CLIMBING HYPER-HEURISTICS ALGORITHM WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

CUT ALNA FADHILLA
NRP 05211540007002

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 *LATE ACCEPTANCE HILL CLIMBING HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

CUT ALNA FADHILLA
NRP 05211540007002

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 LATE ACCEPTANCE HILL CLIMBING HYPER-HEURISTICS ALGORITHM WITH ALGORITHM WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

CUT ALNA FADHILLA
NRP 05211540007002

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 *LATE ACCEPTANCE HILL CLIMBING 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:

CUT ALNA FADHILLA
NRP. 05211540007002

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 *LATE ACCEPTANCE HILL CLIMBING 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 :

CUT ALNA FADHILLA
NRP. 05211540007002

Disetujui Tim Penguji : Tanggal Ujian : 8 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 *LATE ACCEPTANCE HILL CLIMBING HYPER-HEURISTICS* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

Nama Mahasiswa : Cut Alna Fadhilla
NRP : 05211540007002
Departemen : Sistem Informasi
Dosen Pembimbing : Ahmad Muklason, S.Kom., M.Sc., Ph.D

ABSTRAK

Permasalahan penjadwalan mata kuliah merupakan topik yang sangat menarik untuk diselesaikan dikarenakan termasuk salah satu permasalahan NP-hard, dimana belum ada algoritma konvensional eksak yang mampu menyelesaikannya dalam waktu polinomial. International Timetabling Competition 2019 adalah kompetisi yang diadakan khusus untuk permasalahan penjadwalan mata kuliah. Kompetisi yang sudah diadakan keempat kalinya ini terus memberikan tantangan yang berbeda dalam penyelesaian masalah untuk mendapatkan solusinya. Tujuan permasalahan penjadwalan mata kuliah pada International Timetabling Competition 2019 adalah untuk meminimalkan biaya yang dikeluarkan pada semua konten permasalahan. Terdapat dua permasalahan mata kuliah secara umum yang akan diselesaikan. Pertama adalah menjadwalkan mata kuliah pada waktu dan ruang yang telah disediakan dan kedua adalah membagi mahasiswa kepada kelas-kelas yang telah terjadwal. Selain itu terdapat pula batasan yang harus dipenuhi akan menjadi tantangan dalam menyelesaikan masalah ini. Beberapa penelitian untuk

melakukan penyelesaian permasalahan penjadwalan mata kuliah telah dilakukan dengan menggunakan berbagai macam algoritma. Terdapat beberapa algoritma yang dapat digunakan untuk menyelesaikan permasalahan penjadwalan mata kuliah, salah satunya adalah algoritma Late acceptance hill climbing dengan menggunakan pendekatan Hyper-heuristics. Algoritma ini yang akan dipilih untuk menyelesaikan permasalahan penjadwalan mata kuliah menggunakan dataset dari International Timetabling Competition 2019. Hasil luaran yang diharapkan dari pengerjaan tugas akhir ini adalah daftar jadwal mata kuliah dan daftar mahasiswa yang mengambil mata kuliah tersebut dengan memenuhi batasan-batasan yang telah ditetapkan sehingga hasil dari luaran tersebut dapat menjadi solusi untuk penyelesaian permasalahan penjadwalan mata kuliah dari domain permasalahan International Timetabling Competition 2019 yang kompetitif dengan hasil dari algoritma benchmark.

Kata Kunci: Penjadwalan mata kuliah, International Timetabling Competition 2019, algoritma Late acceptance hill climbing, Hyper-heuristics

AUTOMATED COURSE TIMETABLING USING LATE ACCEPTANCE HILL CLIMBING HYPER-HEURISTICS ALGORITHM WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

Name : Cut Alna Fadhilla
NRP : 05211540007002
Department : Information Systems
Supervisior : Ahmad Muklason, S.Kom., M.Sc., Ph.D

ABSTRACT

Course timetabling problem is a very interesting topic to solve because it is one of the NP-hard problems, where there is no exact conventional algorithm that is able to solve it in polynomial time. International Timetabling Competition 2019 is a competition held specifically for subject scheduling problems. The competition that has been held for the fourth time continues to provide different challenges in solving problems to get a solution. The aim of the problem of scheduling courses at the 2019 International Timetabling Competition is to minimize the costs incurred on all content issues. There are two general subject matters that will be resolved. The first is to schedule the courses at the time and space provided and secondly to divide students into scheduled classes. Besides that there are also limits that must be met will be a challenge in solving this problem. Several studies to solve problems in scheduling courses have been carried out using various algorithms. There are several algorithms that can be used to solve the problem of scheduling courses, one of which

is the Late acceptance hill climbing algorithm using the Hyper-heuristics approach. This algorithm will be selected to solve the subject of scheduling problems using a dataset from the International Timetabling Competition 2019. The expected outcomes of this final assignment are a list of course schedules and a list of students taking these courses by meeting the prescribed limits so that the results of these outcomes can be a solution to solving the problem of scheduling courses from the problem domain of the International Timetabling Competition 2019 which is competitive with the results of the benchmark algorithm.

Keywords: Course Timetabling Problem, International Timetabling Competition 2019, algoritma Late acceptance hill climbing, Hyper-heuristics

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 *LATE ACCEPTANCE HILL CLIMBING HYPER HEURISTICS* DENGAN MENGGUNAKAN 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 Teuku Alamsyah dan Ibu Cut Marlina yang selalu menjadi penyemangat dan sumber kekuatan dalam menyelesaikan pendidikan sarjana ini.
2. Adik-adik penulis, Cut Syaffa Fazla TA dan Teuku Zawil Faiza TA yang selalu mendoakan kakaknya dan memberi dukungan moral untuk segera menyelesaikan tugas akhir dan segera pulang.
3. Bapak Ahmad Muklason, S.Kom., M.Sc., Ph.D., dosen pembimbing yang sangat teramat baik kepada penulis.

Selalu membantu dan membimbing serta menyemangati penulis dalam menyelesaikan 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. Teman-teman grup ITC 2019, Kharisma Diah P, Umar Rizki, Narenda Puspa A yang merupakan alasan terbesar saya dapat menyelesaikan tugas akhir ini. Tanpa mereka penulis tidak akan bisa sampai sejauh ini. Tidak bisa diungkapkan besarnya rasa terimakasih penulis dengan apa yang sudah mereka berikan kepada penulis.
6. Rekan-rekan pejuang Laboratorium RDIB, yang selalu menemani dan mendukung serta menjadi tempat diskusi penulis dalam pengerjaan tugas akhir hingga tak kenal waktu saat di Lab RDIB.
7. Teman-teman Lannister, surgawi, D15, blackpink yang selalu memberikan dukungan dan tempat berkeluh kesah bagi penulis.

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, Juli 2019

Penulis

DAFTAR ISI

LEMBAR PENGESAHAN v
LEMBAR PERSETUJUAN vii
ABSTRAK ix
ABSTRACT xi
KATA PENGANTAR xiii
DAFTAR ISI xv
DAFTAR GAMBAR xix
DAFTAR TABEL xxi
DAFTAR KODE xxiii
DAFTAR PERSAMAAN xxvii
BAB 1 PENDAHULUAN 1
1.1. Latar Belakang 1
1.2. Perumusan Masalah 4
1.3. Batasan Pengerjaan Tugas Akhir 4
1.4. Tujuan Tugas Akhir 5
1.5. Manfaat Tugas Akhir 5
1.6. Relevansi 5
BAB II TINJAUAN PUSTAKA 7
2.1. Studi Sebelumnya 7
2.2. Dasar Teori 15
2.2.1. Penjadwalan 15
2.2.2. *International Timetabling Competition 2019 Dataset* 16
2.2.3. *Hyper-Heuristic* 19
2.2.4. *Late acceptance Strategy* 22
BAB III METODOLOGI PENELITIAN 25
3.1. Tahapan Pelaksanaan Tugas Akhir 25

3.1.1. Identifikasi Permasalahan 26
3.1.2. Studi Literatur 26
3.1.3. Pembuatan Model Matematis 26
3.1.4. Implementasi Algoritma *LAHC dan Hyper - Heuristic* 29
3.1.5. Uji Coba 31
3.1.6. Analisis Hasil Performa Algoritma dan Kesimpulan 31
3.1.7. Penyusunan Buku Tugas Akhir 32
BAB IV PERANCANGAN 33
4.1. Pemahaman Data 33
4.1.1. Inisiasi Konten *Dataset* 33
4.1.1.1. Konten Times 34
4.1.1.2. Konten *Rooms* 36
4.1.1.3. Konten *Courses* 37
4.1.1.4. Konten *Students* 38
4.1.1.5. Konten Distribution *Constraint* 39
4.2. Formulasi Model Matematis 40
4.2.1. Konversi Fungsi Tujuan 40
4.2.2. Konversi Variabel Keputusan 43
4.2.3. Konversi Batasan Distribusi Masalah 44
4.3. Menentukan *Initial Solution* 57
4.4. Implementasi Algoritma *Late acceptance hill climbing Hyper Heuristics* 57
4.5. Perencanaan Alur Skenario Uji Coba 58
4.5.1 Skenario 1: Pembentukan Solusi Awal 59
4.5.2 Skenario 2: Jumlah Iterasi 60
4.5.3 Skenario 3: Perubahan Idle 60
4.5.4 Skenario 4: *History Length* 61
BAB V IMPLEMENTASI 63
5.1. Pembacaan *File* Masukan 63
5.1.1. Pembacaan Awal Data 63

5.1.2. Penyimpanan Konten *Rooms* dan Atributnya 64
5.1.3. Penyimpanan Konten *Courses* dan Atributnya 65
5.1.4. Penyimpanan Konten *Students* dan Atributnya 66
5.1.5. Penyimpanan Konten Batasan Distributions dan Atributnya 67
5.2. Pembentukan Solusi Awal 94
5.2.1 Metode Penetapan Jadwal Kelas 95
5.2.2 Perhitungan Nilai Penalti 98
5.3. Optimasi Solusi 100
5.3.1. Penyimpanan Solusi Awal dan Iterasi 101
5.3.2. Perhitungan Total Nilai Penalti Solusi Awal 102
5.3.3. Pembuatan *Low level heuristics* 102
5.3.4. Implementasi Algoritma *Late acceptance hill climbing* 108
BAB VI HASIL DAN PEMBAHASAN 111
6.1. Lingkungan Uji Coba 111
6.2. Hasil Implementasi Penjadwalan 112
6.2.1. Skenario 1: Pembentukan Solusi Awal 112
6.2.2 Skenario 2: Jumlah Iterasi 113
6.2.2 Skenario 3: Perubahan *Idle* 121
6.2.3 Skenario 4: Perubahan *History Length* 127
6.3. Perbandingan Parameter Algoritma Hasil Uji Coba 134
6.3.1 Perbandingan Hasil Uji Coba Iterasi 134
6.3.2 Perbandingan Hasil Uji Coba *Idle* 135
6.3.3 Perbandingan Hasil Uji Coba *History Length* 137
6.3.4 Perbandingan Algoritma *Hill climbing* 139
6.4. Kesimpulan Hasil Uji Parameter 142
BAB VII KESIMPULAN DAN SARAN 145
7.1. Kesimpulan 145
7.2. Saran 146

DAFTAR PUSTAKA.....149
BIODATA PENULIS.....151
LAMPIRAN A: Kebutuhan Data 153
LAMPIRAN B: Hasil Optimasi dalam Bentuk XML.....154

DAFTAR GAMBAR

Gambar 1.1 *Roadmap* Penelitian Laboratorium Rekayasa Data dan Intelegensi Bisnis 6
Gambar 2.1 Definisi Spesifik Permasalahan 19
Gambar 2.2 *Hyper-heuristic Framework* 21
Gambar 2.3 Ilustrasi Penerapan *Late acceptance Strategy* 22
Gambar 3.1 Alur Pengerjaan Tugas Akhir 25

Halaman ini sengaja dikosongkan

DAFTAR TABEL

Tabel 2.1 Penelitian Terdahulu..... 7
Tabel 2.2 Karakteristik *ITC 2019 Dataset*..... 16
Tabel 3.1 Daftar Batasan *Dataset*..... 28
Tabel 4.1 Daftar Parameter Uji Hasil 58
Tabel 6.1 Spesifikasi Perangkat Keras 111
Tabel 6.2 Spesifikasi Perangkat Lunak 111
Tabel 6.3 Hasil Implementasi Skenario 1 112
Tabel 6.4 Hasil Kesimpulan Nilai Penalti Uji Parameter *(LAHC)*..... 142

Halaman ini sengaja dikosongkan

DAFTAR KODE

Kode 2.1 *Pseudocode Late acceptance hill climbing* 23
Kode 4.1 Format Kode Garis Besar..... 34
Kode 4.2 Format Kode Konten Waktu 35
Kode 4.3 Format Kode Konten Ruangan 36
Kode 4.4 Format Kode Konten Mata Kuliah..... 38
Kode 4.5 Format Kode Konten Mahasiswa..... 39
Kode 4.6 Format Kode Konten Batasan-Batasan 39
Kode 5.1 Pembacaan Awal Masukan Data..... 64
Kode 5.2 Penyimpanan Konten *Rooms* dan Atributnya..... 65
Kode 5.3 Penyimpanan Konten *Course* dan Atributnya..... 66
Kode 5.4 Penyimpanan Konten *Student* dan Atributnya 67
Kode 5.5 Metode *Hard constraint* Same Start 68
Kode 5.6 Metode *Soft Constraint* Same Start..... 69
Kode 5.7 Metode *Hard constraint* Same Time..... 70
Kode 5.8 Metode *Soft Constraint* Same Time 71
Kode 5.9 Metode *Hard constraint* Different Time..... 72
Kode 5.10 Metode *Soft Constraint* Different Time 73
Kode 5.11 Metode *Hard constraint* Same Days..... 74
Kode 5.12 Metode *Soft Constraint* Same Days 75
Kode 5.13 Metode *Hard constraint* Different Days 76
Kode 5.14 Metode *Soft Constraint* Different Days..... 76
Kode 5.15 Metode *Hard constraint* Same Weeks 77
Kode 5.16 Metode *Soft Constraint* Same Weeks..... 77
Kode 5.17 Metode *Hard constraint* Different Weeks..... 78
Kode 5.18 Metode *Soft Constraint* Different Weeks..... 78
Kode 5.19 Metode *Hard constraint* Same *Room*..... 79
Kode 5.20 Metode *Soft Constraint* Same *Room* 79
Kode 5.21 Metode *Hard constraint* Different *Rooms*..... 80
Kode 5.22 Metode *Soft Constraint* Different *Rooms*..... 80
Kode 5.23 Metode *Hard constraint* Overlap 81

Kode 5.24 Metode *Soft Constraint* Overlap 81
Kode 5.25 Metode *Hard constraint* Not Overlap 82
Kode 5.26 Metode *Soft Constraint* Not Overlap..... 83
Kode 5.27 Metode *Hard constraint* Same Attendees 84
Kode 5.28 Metode *Soft Constraint* Same Attendees..... 85
Kode 5.29 Metode *Hard constraint* Precedence..... 86
Kode 5.30 Metode *Soft Constraint* Precedence 87
Kode 5.31 Metode *Hard constraint* Work Days..... 88
Kode 5.32 Metode *Soft Constraint* Work Days 88
Kode 5.33 Metode *Hard constraint Min gap*..... 89
Kode 5.34 Metode *Soft Constraint Min gap*..... 90
Kode 5.35 Metode *Hard constraint Max Days*..... 90
Kode 5.36 Metode *Soft Constraint Max Days*..... 91
Kode 5.37 Metode *Count non zero bits* 91
Kode 5.38 Metode *Hard constraint Max Day Load* 92
Kode 5.39 Metode *Soft Constraint Max Day Load*..... 92
Kode 5.40 Metode Pengecekan *Dayload*..... 92
Kode 5.41 Metode *Hard constraint Merge Blocks* 93
Kode 5.42 Metode *Soft Constraint Merge Blocks* 93
Kode 5.43 Metode *Hard constraint Max Block*..... 94
Kode 5.44 Metode *Soft Constraint Merge Blocks* 94
Kode 5.45 Kode Kelas *Initial Solution*..... 95
Kode 5.46 Metode Penetapan Jadwal Kelas 96
Kode 5.47 Metode Penghapusan Jadwal Kelas 96
Kode 5.48 Metode Pengecekan Jadwal Kelas 97
Kode 5.49 Metode Pengecekan Ketidaktersediaan Ruangan . 98
Kode 5.50 Metode Pengecekan Hasil Solusi 98
Kode 5.51 Metode Pengecekan Total Nilai Penalti Kelas 99
Kode 5.52 Metode Pengecekan Nilai Penalti Ruangan..... 99
Kode 5.53 Metode Pengecekan Nilai Penalti Jadwal Waktu Kelas 100
Kode 5.54 Metode Pengecekan Nilai Penalti Jadwal Ruang100
Kode 5.55 Metode Menyimpan Hasil Solusi Awal 101
Kode 5.56 Metode Pencetakan Hasil Solusi Awal..... 102

Kode 5.57 Metode Penghitungan Bobot Total Nilai Penalti 102
Kode 5.58 Metode *Move* 1 103
Kode 5.59 Metode *Move* 2 104
Kode 5.60 Metode *Move* 3 104
Kode 5.61 Metode *Move* 4 105
Kode 5.62 Metode *Move* 5 106
Kode 5.63 Metode *Move* 6 107
Kode 5.64 Metode *Move* 7 108
Kode 5.65 Metode Algoritma *Late acceptance hill climbing* 109

Halaman ini sengaja dikosongkan

DAFTAR PERSAMAAN

Persamaan (1)..... 40
Persamaan (2)..... 41
Persamaan (3)..... 42
Persamaan (4)..... 42
Persamaan (5)..... 43
Persamaan (6)..... 44
Persamaan (7)..... 45
Persamaan (8)..... 46
Persamaan (9)..... 46
Persamaan (10)..... 47
Persamaan (11)..... 47
Persamaan (12)..... 48
Persamaan (13)..... 48
Persamaan (14)..... 49
Persamaan (15)..... 49
Persamaan (16)..... 50
Persamaan (17)..... 50
Persamaan (18)..... 52
Persamaan (19)..... 52
Persamaan (20)..... 53
Persamaan (21)..... 54
Persamaan (22)..... 55
Persamaan (23)..... 55
Persamaan (24)..... 55
Persamaan (25)..... 56
Persamaan (26)..... 56

Halaman ini sengaja dikosongkan

BAB 1
PENDAHULUAN

Pada bab ini, akan dijelaskan tentang Latar Belakang Masalah, Perumusan Masalah, Batasan Masalah, Tujuan Tugas Akhir, Manfaat Kegiatan Tugas Akhir dan Relevansi dengan laboratorium RDIB.

1.1. Latar Belakang

Penjadwalan mata kuliah adalah salah satu permasalahan *NP-hard*, yang berarti sulit untuk diselesaikan menggunakan metode yang konvensional dan waktu komputasi yang dibutuhkan untuk mendapatkan solusi yang optimal dengan meningkatnya eksponensial seiring besarnya permasalahan [1]. Dengan kata lain, berdasarkan sudut pandang optimasi kombinatorial, penjadwalan merupakan permasalahan *NP-hard*, yang artinya belum ada algoritma konvensional eksak yang dapat menyelesaikan permasalahan tersebut dalam waktu polinomial [2]. Hal tersebut yang menjadi alasan penjadwalan menarik untuk menjadi topik penelitian.

Permasalahan penjadwalan yang disediakan oleh *International Timetabling Competition 2019* adalah kompetisi keempat kalinya yang diselenggarakan. Permasalahan penjadwalan yang disediakan adalah kategori penjadwalan mata kuliah pada perguruan tinggi. Mata kuliah harus menemukan waktu dan ruang yang tepat dengan mempertimbangkan kondisi *student* dan *teacher* untuk masing-masing mata kuliah tersebut. Tujuan utamanya adalah meminimalkan biaya yang harus dikeluarkan pada semua *resource* yang tersedia serta

memenuhi batasan yang telah ditentukan. Selain itu, hasil akhir juga harus meminimalkan adanya tumpang tindih dan kejanggalan yang terjadi pada *resources*. Dengan kata lain solusi yang dihasilkan harus *feasible* dengan memenuhi *hard constraint* dan memaksimalkan untuk memenuhi *soft constraint.*

Beberapa penelitian terkait permasalahan penjadwalan mata kuliah ini telah banyak dilakukan dengan menggunakan berbagai macam pendekatan yang berbeda-beda dan studi kasus yang beragam pula. Termasuk dengan penelitian yang dilakukan oleh para pemenang *ITC 2003*, *ITC 2007* dan *ITC 2011*. Pemenang *ITC 2003* menggunakan 3 fase *local search-based algorithm*, *ITC 2007* dimenangkan dengan menggunakan variasi dari algoritma *Simulated Annealing* dan yang terakhir kalinya *ITC 2011* dimenangkan dengan algoritma kombinasi antara *Simulated Annealing* dan *Iterated Local Search (SA-ILS)* [3]. Algoritma *Simulated Annealing* berhasil menempatkan pemenang *ITC 2007* dan *ITC 2011*. *Simulated Annealing* merupakan algoritma *meta-heuristics* yang menganalogikan proses pendinginan baja saat dipanaskan, kemudian didinginkan secara perlahan hingga mencapai suhu yang didinginkan. Pada saat baja dipanaskan, atom dalam baja akan bergerak bebas. Namun ketika didinginkan susunan atom yang lebih optimum [2].

Penelitian berikutnya [4] mengenalkan dan memberikan penjelasan secara lengkap mengenai algoritma *Late acceptance hill climbing (LAHC)*. Algoritma ini merupakan algoritma baru dan sangat sederhana dalam penerapannya. Permasalahan yang digunakan untuk percobaan implementasi

LAHC adalah *Exam Timetabling* dan *Travelling Salesman Problem (TSP)*. Sedangkan algoritma pembanding yang dipilih yaitu *Hill climbing*, *Simulated Annealing*, *Threshold Accepting*, dan *Great Deluge Algorithm*. Hasil penelitian menunjukkan bahwa *LAHC* menghasilkan performa rata – rata yang lebih baik dibandingkan algoritma lainnya. Pada *TSP*, *LAHC* memberikan rata – rata nilai *cost* yang paling minimum dibandingkan algoritma pembandingnya.

Selanjutnya penelitian lain [3] membahas penjadwalan dengan *ITC 2007 Dataset* menggunakan pendekatan algoritma *Late acceptance hill climbing*. Pada penelitian tersebut ditemukan bahwasannya performa dan hasil yang didapatkan untuk *dataset* yang sama mampu bersaing dan memberikan hasil setara dan terdapat beberapa yang lebih unggul dibandingkan menggunakan *Simulated Annealing* sebagaimana yang telah diselesaikan oleh pemenang. Namun pada penelitian ini metode yang digunakan adalah *meta heuristic*.

Berdasarkan beberapa penelitian yang telah dilakukan sebelumnya, maka *framework Hyper-heuristics* dan algoritma *Late acceptance hill climbing* akan digunakan untuk menyelesaikan permasalahan penjadwalan mata kuliah. *Hyper-Heuristic* memiliki keunggulan yaitu tidak diperlukannya parameter *tuning* dan tidak bergantung pada *problem domain* [5]. Sedangkan *LAHC* dipilih karena merupakan algoritma *Hyper-Heuristic* baru yang terbukti memiliki performa lebih baik dibandingkan beberapa algoritma *meta-Heuristics* [4]. Pemilihan algoritma tersebut dikarenakan *LAHC* berhasil menyelesaikan permasalahan

exam timetabling yang dilakukan sebagai uji coba dan menempati posisi kedua sebagai pemenang pada *ITC 2007*.

1.2. Perumusan Masalah

Berdasarkan latar belakang yang telah dijabarkan di atas, berikut adalah rumusan masalah yang dijadikan acuan dalam penelitian tugas ini adalah sebagai berikut:

1. Bagaimana model matematis untuk permasalahan penjadwalan mata kuliah pada studi kasus *ITC 2019*?
2. Bagaimana menerapkan algoritma *Late acceptance hill climbing* berbasis *Hyper-Heuristic* pada permasalahan penjadwalan mata kuliah *ITC 2019*?
3. Bagaimana performa algoritma *Late acceptance hill climbing berbasis Hyper-Heuristic* dalam menyelesaikan permasalahan penjadwalan mata kuliah *ITC 2019*?

1.3. Batasan Pengerjaan Tugas Akhir

Berdasarkan latar belakang yang telah dipaparkan di atas, adapun batasan masalah yang ditentukan dalam penelitian ini adalah sebagai berikut:

1. Data yang digunakan dalam penjadwalan mata kuliah adalah *ITC 2019 Dataset*.
2. *Dataset* yang digunakan hanya kumpulan data dengan tipe *Testing* dan *Early.*
3. Aplikasi dibangun dan dikembangkan menggunakan Bahasa Java
4. Penjadwalan hanya dilakukan untuk mata kuliah (kelas) tidak termasuk menjadwalkan mahasiswa.

1.4. Tujuan Tugas Akhir

Berdasarkan rumusan dan batasan masalah yang telah diuraikan pada bagian sebelumnya, maka tujuan yang ingin dicapai dari penelitian tugas akhir ini adalah:

a. Membuat model matematis untuk permasalahan penjadwalan mata kuliah pada studi kasus *ITC 2019.*
b. Menyelesaikan permasalahan penjadwalan mata kuliah *ITC 2019* dengan menggunakan algoritma *Late acceptance hill climbing* berbasis *Hyper-Heuristic.*
c. Menganalisis performa algoritma *Late acceptance hill climbing berbasis Hyper-Heuristic* dalam menyelesaikan permasalahan penjadwalan mata kuliah *ITC 2019*.

1.5. Manfaat Tugas Akhir

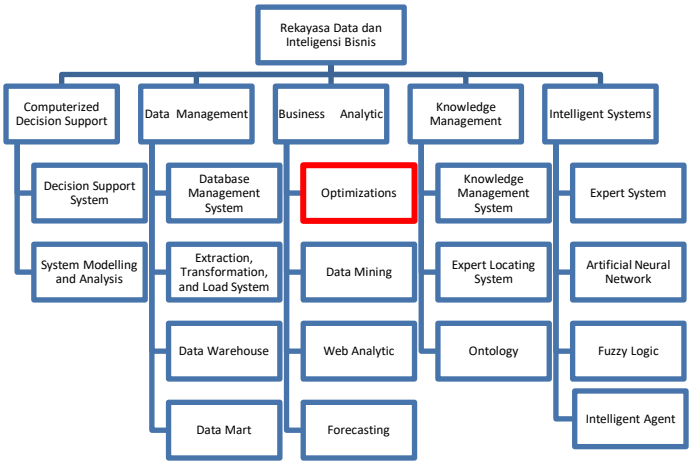
Beberapa manfaat yang diharapkan dapat tercapai pada penelitian ini adalah,

1. Menjadi bahan referensi atau rujukan tambahan dalam bidang *timetabling* khususnya penjadwalan mata kuliah.
2. Membantu menyelesaikan permasalahan penjadwalan mata kuliah dengan menggunakan algoritma *Late acceptance hill climbing Hyper-heuristics*.
3. Menghasilkan solusi yang kompetitif dengan solusi dari algoritma *benchmark*.

1.6. Relevansi

Penelitian tugas akhir ini berkaitan dengan *roadmap* penelitian Laboratorium Rekayasa Data dan Intelegensi Bisnis (RDIB) yang termasuk pada pokok penelitian *Business Analytics*, khususnya di bidang optimasi. Hal ini menunjukkan

bahwasannya penelitian ini akan berkaitan dan sejalan dengan tujuan Laboratorium RDIB yaitu menjadi pusat penelitian dan pusat rujukan terkait pemanfaatan data yang mendukung analisis bisnis dan organisasi untuk dapat ditransformasi menjadi informasi yang bermakna serta pengetahuan sehingga dapat mendukung proses pengambilan keputusan. Gambar 1.1 yang terdapat pada Laboratorium Rekayasa dan Intelegensi Bisnis.



Gambar 1.1 *Roadmap* Penelitian Laboratorium Rekayasa Data dan Intelegensi Bisnis

BAB II
TINJAUAN PUSTAKA

Dalam bab ini, akan dijelaskan mengenai penelitian terdahulu dan landasan teori yang digunakan sebagai acuan dalam pengerjaan tugas akhir. Penelitian terdahulu merupakan suatu penelitian yang pernah dilakukan oleh peneliti-peneliti sebelumnya yang digunakan sebagai acuan tugas akhir. Landasan teori merupakan teori-teori yang berhubungan dengan pengerjaan tugas akhir.

2.1. Studi Sebelumnya

Penelitian sebelumnya yang pernah dilakukan dapat dilihat pada Tabel 2.1.

Tabel 2.1 Penelitian Terdahulu

No	Penelitian Terdahulu	
1.	Judul Penelitian	*The Late acceptance Hill-Climbing Heuristic* [4]
	Tahun	2017
	Nama Peneliti	Edmund K.Burke, Yuri Bykov
	Permasalahan	*Examination Timetabling Problem; Travelling Salesman Problem.*
	Dataset	Socha *Dataset*
	Algoritma	*Late acceptance hill climbing, Simulated Annealing, Threshold Accepting, Great Deluge Algorithm ; etc*
	Deskripsi	Pada penelitian [4], peneliti

No	Penelitian Terdahulu	
		memperkenalkan mengenai algoritma baru yang dapat digunakan sebagai *local search algorithm* bernama *Late acceptance hill climbing (LAHC)*. Kemudian terhadap algoritma *LAHC* tersebut dilakukan pengujian dengan membandingkan performa terhadap algoritma serumpun seperti *Simulated Annealing, Threshold Accepting, Great Deluge Algorithm*. *LAHC* merupakan sebuah algoritma simpel dan mudah untuk diimplementasikan baik untuk *Examination Timetabling Problem* dan *Travelling Salesman Problem.* Hasil yang diperoleh dari penelitian tersebut adalah; a. Algoritma *LAHC* memiliki kesederhanaan seperti *Greedy HC*, akan tetapi jauh lebih kuat. Maka dari itu, dapat dengan mudah untuk diimplementasikan ke dalam sistem secara praktis. b. *LAHC* mengumpulkan informasi selama iterasi sebelumnya. c. Algoritma *LAHC* akan sangat bergantung pada *single algorithmic* parameter yang berarti akan berpengaruh terhadap pengaturan waktu *CPU*. d. *LAHC* mampu menghasilkan performa yang sebanding dengan

No	Penelitian Terdahulu	
		algoritma lainnya seperti *SA, TA* dan *GDA*. Bahkan dapat mengungguli pada sebagian masalah terutama yang memiliki ukuran besar. Namun *LAHC* masih memerlukan penelitian lebih lanjut lagi untuk terus dilakukan uji coba. e. *LAHC* tidak bergantung pada skala yang berbeda pada algoritma *scheduling*. Sehingga menunjukkan kehandalannya yang lebih kuat pada masalah *non-linear* dan dapat bermanfaat pada saat ingin menerapkan aplikasi baru. f. *Randomised Iterative Improvement Algorithm* (2007), menghasilkan rata-rata hasil akhir 5 untuk *Instance small*, 334 untuk *Instance medium*, dan 1068 untuk *Instance large.* g. Algoritma *LAHC* yang diteliti bersifat metaheuristik. Namun demikian dapat terus dilakukan pengembangan.
	Keterkaitan dengan Tugas Akhir	Algoritma *Late acceptance hill climbing* yang terdapat pada penelitian ini akan dijadikan rujukan utama dalam mengerjakan tugas akhir.
	Gap Penelitian	Penelitian [4] hanya menggunakan algoritma metaheuristik saja. Sementara penelitian ini, menggunakan *framework Hyper-Heuristic*. Selain itu penelitian tugas

No	Penelitian Terdahulu	
		akhir hanya berfokus pada permasalahan *Course Timetabling* dengan menggunakan *ITC 2019 Dataset*.
2.	Judul Penelitian	*Solver* Penjadwal Ujian Otomatis dengan Algoritma *Maximal Clique* dan *Hyper-heuristics* [5]
	Tahun	2017
	Nama Peneliti	Ahmad Muklason
	Permasalahan	*Examination Timetabling Problem*
	Dataset	*Carter Dataset*
	Algoritma	*Maximal Clique* dan *Hyper-heuristics*
	Deskripsi	Pada penelitian [5], peneliti membahas mengenai algoritma *Hyper-heuristics* yang digunakan sebagai metode pada penyelesaian masalah *Examining Timetabling*. Pendekatan metaheuristik yang biasanya digunakan diganti menjadi *Hyper-Heuristic*
	Keterkaitan dengan Tugas Akhir	Pada penelitian ini *framework* yang coba digunakan adalah *Hyper-Heuristic* sehingga akan mengambil referensi dari penelitian [5] untuk kemudian dapat diterapkan dalam pengerjaan tugas akhir ini.
	Gap Penelitian	Penelitian [5] menyelesaikan penjadwalan ujian menggunakan *Carter Dataset*. Sementara penelitian

No	Penelitian Terdahulu	
		ini menyelesaikan penjadwalan perkuliahan mata kuliah menggunakan *ITC 2019 Dataset.*
3.	Judul Penelitian	*Late acceptance hill climbing algorithm for solving patient admission scheduling problem* [6]
	Tahun	2018
	Nama Peneliti	Asaju La'aro Bolaji, Akeem Feemi Bamigbola, Peter Bamidele Shola
	Permasalahan	*Patient Admission Scheduling*
	Dataset	*Bilgin Dataset*
	Algoritma	*Late acceptance hill climbing*
	Deskripsi	Pada penelitian [6], peneliti memaparkan mengenai penerapan algoritma *LAHC* untuk menyelesaikan permasalahan penjadwalan penerimaan pasien. Data yang digunakan adalah data *benchmark* yang disediakan oleh Bilgin. Permasalahan yang diangkat juga merupakan salah satu permasalahan kompleks dalam hal optimasi dan merupakan *NP-problem*. Teknik yang digunakan adalah *one-point solution*. Hasil dari penelitian ini adalah algoritma *LAHC* mampu menghasilkan solusi yang setara dengan hasil menggunakan algoritma dan teknik lainnya.

No	Penelitian Terdahulu	
	Keterkaitan dengan Tugas Akhir	Pada penelitian [6] algoritma yang digunakan adalah *Late acceptance hill climbing* yang berarti sama dengan algoritma yang akan digunakan dalam penelitian ini.
	Gap Penelitian	Penelitian [6] menyelesaikan permasalahan *patient admission scheduling (PAS)* menggunakan Bilgin *Dataset* sedangkan penelitian ini *course timetabling* menggunakan *ITC 2019 Dataset.*
4.	Judul Penelitian	*Hyper-Heuristic*: *a survey of the state of the art*
	Tahun	2013
	Nama Peneliti	Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, etc
	Permasalahan	*Hyper-Heuristic problem*
	Dataset	-
	Algoritma	*Hyper-Heuristic*
	Deskripsi	Pada penelitian [7] merupakan rangkuman secara keseluruhan mengenai metode *Hyper-heuristics*. *Paper* tersebut bertujuan untuk menganalisis dan mendiskusikan literatur yang membahas mengenai *Hyper-heuristics* secara ringkas dan lengkap. *Paper* juga menyediakan sumber-sumber dan asal mula metode hingga berkembang sampai ini. Ide

No	Penelitian Terdahulu	
		muncul metode *Hyper-heuristics* tidaklah baru untuk dipelajari. Sejak tahun 1960-an sudah mulai dikembang dalam bidang ilmu *Operational Research*, *Computer Science* dan *Artificial Intelligence*. Pada *paper* ini dibahas secara ringkas dan jelas untuk perkembangan masing-masing bidang ilmu. Selain itu pada bagian akhir, diberikan masukan dan saran yang berpotensial akan terus berkembang dalam bidang riset yang berkaitan dengan *Heuristics.*
	Keterkaitan dengan Tugas Akhir	Pada penelitian [7] menjelaskan dan memberikan informasi mengenai metode *Hyper-heuristics* secara jelas dan ringkas. Metode ini adalah yang nan*tinya* akan digunakan pada pengerjaan penelitian ini.
	Gap Penelitian	Penelitian [7] hanya menjelaskan secara konsep mengenai metode *Hyper-heuristics* tanpa adanya implementasi dalam suatu masalah. Sedangkan pada penelitian ini bertujuan untuk mengimplementasikan metode untuk menyelesaikan sebuah *problem*.
5.	Judul Penelitian	*Late acceptance Hill-climbing for High School Timetabling* [10]
	Tahun	2015

No	Penelitian Terdahulu	
	Nama Peneliti	Georce H. G. Fonseca, Haroldo G. Santos, Eduardo G. Carrano
	Permasalahan	*High school timetabling problem*
	Dataset	*ITC 2011 Dataset*
	Algorithm	*Late acceptance hill climbing*
	Deskripsi	Pada penelitian [8], peneliti menyampaikan hasil penelitiannya menggunakan algoritma *Late acceptance hill climbing* untuk *ITC 2011 Dataset* yang merupakan kompetisi ke 3 *ITC*. Peneliti membuktikan bahwasannya algoritma tersebut mampu menghasilkan hasil yang lebih baik dari algoritma yang sudah diselesaikan oleh para pemenang *ITC 2011*.
	Keterkaitan dengan Tugas Akhir	Pada penelitian [3] menggunakan algoritma *LAHC* dan *ITC 2011 Dataset*. Tentunya keduanya akan saling berkaitan dengan yang akan dikerjakan pada penelitian ini. Nan*tiny*a penelitian [3] akan menjadi sumber dalam menyelesaikan penelitian ini.
	Gap Penelitian	Penelitian [3] menggunakan *ITC 2011 Dataset* sedangkan penelitian ini menggunakan *ITC 2019 Dataset*.

2.2. Dasar Teori

Pada subbab ini akan dijabarkan mengenai dasar teori yang digunakan untuk mendukung pengerjaan tugas akhir.

2.2.1. Penjadwalan

Penjadwalan adalah suatu kegiatan atau aktivitas pengalokasian, sumber daya dalam suatu ruang dan waktu, berdasarkan aturan dan batasan tertentu untuk mencapai tujuan optimal. Penjadwalan mata kuliah adalah kegiatan pengalokasian kumpulan mata kuliah terhadap ruang dan jangka waktu tertentu, pada saat yang sama, siswa dan guru juga ditugaskan ke mata kuliah sehingga pertemuan dapat berlangsung. Selain itu, permasalah penjadwalan juga memiliki batasan yang terbagi menjadi dua, yaitu *hard* dan *soft constraints*, berikut contoh dari masing-masing batasan. [9]

a. Hard constraints

- Masing-masing siswa dan guru tidak dapat berada dalam dua tempat secara bersamaan.
- Terdapat satu mata kuliah yang diizinkan untuk ditempatkan pada tiap slot waktu di setiap ruang kelas.
- Kapasitas kelas harus sama dengan atau lebih besar dari jumlah siswa yang menghadiri mata kuliah pada waktu tertentu.
- Kelas yang dialokasikan untuk mata kuliah harus dapat memenuhi fitur yang dibutuhkan oleh mata kuliah.

b. Soft Constraints

- Siswa tidak boleh memiliki hanya satu mata kuliah tunggal dalam sehari.

- Siswa tidak boleh menghadiri lebih dari dua mata kuliah secara berurutan dalam sehari.
- Siswa tidak boleh dijadwalkan menghadiri mata kuliah pada slot waktu akhir.

2.2.2. *International Timetabling Competition 2019 Dataset*

International Timetabling Competition 2019 Dataset yang selanjutnya akan disebut *ITC 2019 Dataset* adalah kumpulan data yang dikeluarkan oleh *Timetabling Competition* yang diselenggarakan keempat kalinya. *International Timetabling Competition* pertama diselenggarakan pada tahun 2002, kedua pada tahun 2007, ketiga pada tahun 2011 dan keempat pada tahun 2019. *Dataset* ini berisi tentang *ITC 2019 Dataset* yang terbagi menjadi tiga grup data *Instances* yang akan dikeluarkan yaitu *Early, middle,* dan *late Instances.* Pada pengerjaan penelitian ini, *dataset* yang digunakan adalah jenis *Testing* dan *Early Dataset* yang memiliki total 15 *Instances.* Pada Tabel 2.2 adalah penjelasan karakteristik secara umum untuk masing-masing *dataset.*

Tabel 2.2 Karakteristik *ITC 2019 Dataset*

Nama *Instances*	Tipe Data	Jumlah *Course*	Jumlah *Class*	Jumlah *Room*	Jumlah *Student*	Jumlah *Distribution*
Test Instance 2	*Testing*	19	20	62	0	2
Test Instance 3	*Testing*	48	127	46	0	144

Nama *Instances*	Tipe Data	Jumlah *Course*	Jumlah *Class*	Jumlah *Room*	Jumlah *Student*	Jumlah *Distribution*
Test Instance 4	*Testing*	44	174	13	2002	102
Test Instance 5	*Testing*	602	802	56	27881	329
Test Instance 6	*Testing*	1035	2417	207	2002	29514
Early Instance 1	*Early*	340	1239	80	1641	1220
Early Instance 2	*Early*	272	1852	44	2116	2690
Early Instance 3	*Early*	353	983	62	3018	1251
Early Instance 4	*Early*	1206	2641	214	0	2902
Early Instance 5	*Early*	544	882	90	3666	3947
Early Instance 6	*Early*	228	575	35	1543	740
Early Instance 7	*Early*	226	561	44	865	400

Nama *Instances*	Tipe Data	Jumlah *Course*	Jumlah *Class*	Jumlah *Room*	Jumlah *Student*	Jumlah *Distribution*
Early Instance 8	*Early*	1089	2526	70	2938	2026
Early Instance 9	*Early*	687	1001	75	27018	634
Early Instance 10	*Early*	36	711	15	0	501

Informasi yang diberikan pada Tabel 2.2 adalah nama untuk masing-masing *Instance*, tipe data dan jumlah konten untuk masing-masing *Instance* tersebut. Jumlah *course* adalah total jumlah mata kuliah yang harus tersedia dan diambil mahasiswa, jumlah *class* adalah total jumlah kelas yang harus dijadwalkan pada waktu dan ruang yang telah tersedia, jumlah *room* adalah total jumlah ruang yang dapat digunakan, jumlah *student* adalah total siswa yang mengambil mata kuliah dan jumlah *distribution* adalah total batasan yang harus dipenuhi dalam penyusunan jadwal mata kuliah tersebut pada masing-masing *Instance*.

ITC 2019 Dataset memiliki format ekstensi XML untuk setiap konten yang terdiri dari variabel dan batasan. Masing-masing konten memiliki ketentuan untuk membaca data tersebut. Berikut Gambar 2.1 adalah contoh penulisan *dataset* secara umum.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE solution PUBLIC "

        -//ITC 2019//DTD Problem Format/EN"
        "http://www.itc2019.org/competition-
        format.dtd">

<problem name="unique-Instance-name" nrDays="7"
nrWeeks="13" slotsPerDay="288">

        <optimization.../>
        <rooms>...</rooms>
        <courses>...</courses>
        <distributions>...</distributions>
        <students> ... </students>

</problem>
```

Gambar 2.1 Definisi Spesifik Permasalahan

2.2.3. *Hyper-Heuristic*

Pendekatan dalam penyelesaian permasalahan harus disesuaikan dengan jenis permasalahan yang ingin diselesaikan. Dalam permasalahan kombinatorial, terdapat dua jenis permasalahan yang digolongkan kepada Polinomial dan *Non*-Polinomial. Kebutuhan waktu algoritma yang efisien bervariasi, mulai dari O(1), O(log n), O(n), O(n^2), dan O(n^3). Solusi algoritma yang digolongkan baik dikenal sebagai solusi polinomial. Hal ini karena kebutuhan waktunya secara asimtotik dibatasi oleh fungsi polinomial. Misalnya log(n) < n untuk semua n > 1. Sebaliknya terdapat permasalahan dengan solusi waktu tidak polinomial. Permasalahan seperti ini digolongkan sebagai *hard problem*. Permasalahan seperti ini memiliki kompleksitas O(n!). Sehingga perlu strategi

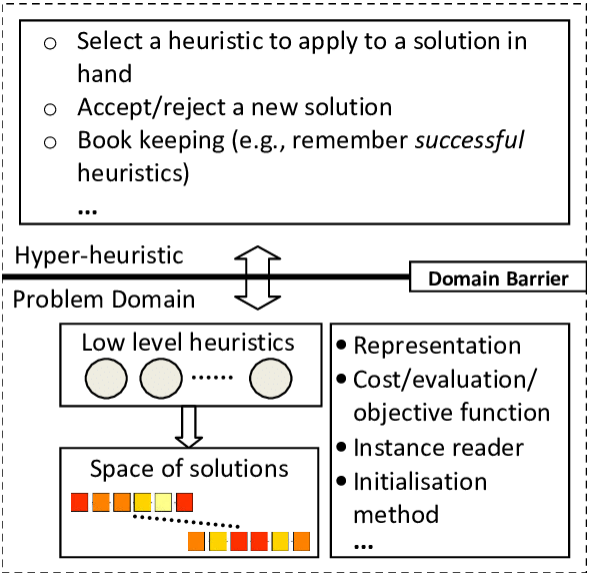
pendekatan dalam menyelesaikan permasalahan tersebut. Pendekatan tersebut adalah strategi heuristik. Kemudian strategi heuristik terbagi menjadi dua yaitu *meta- heuristics* dan *hyper-heuristics*. [7]

Hyper-heuristics dapat didefinisikan sebagai kumpulan pendekatan dengan tujuan untuk otomasi proses yang biasanya menggunakan *machine learning*. Terdapat dua tujuan yaitu untuk memilih dan mengkombinasikan *heuristics* yang lebih sederhana dan untuk menghasilkan *Heuristic* baru dengan komponen *Heuristic* yang sudah ada, untuk memecahkan permasalahan pencarian komputasi yang sangat sulit dilakukan secara manual (*hard computational search problem*). [5]

Dengan menggunakan *Hyper-Heuristic*, kita mencoba menemukan metode atau urutan heuristik yang tepat dalam situasi tertentu daripada kita mencoba memecahkan masalah secara langsung. *Hyper-Heuristic* dapat dianggap sebagai metode *'off-the-peg'* dibandingkan dengan teknik ' *made-to-measure'*. Oleh karena itu, tujuan utama nya adalah merancang metode generik atau umum yang harus menghasilkan solusi berkualitas yang dapat diterima. [7]

Hyper-Heuristic adalah pendekatan heuristik yang merupakan metode pencarian yang tidak memiliki jaminan keberhasilan untuk menemukan solusi optimal. Kadang-kadang, istilah *Hyper-Heuristic* dapat dipertukarkan dengan metaheuristik, sebuah proses berulang yang memandu heuristik bawahan dengan menggabungkan konsep yang berbeda secara cerdas untuk mengeksplorasi dan mengeksploitasi ruang pencarian dengan menggunakan strategi pembelajaran untuk menyusun

informasi agar dapat menemukan secara efisien mendekati optimal. [10]

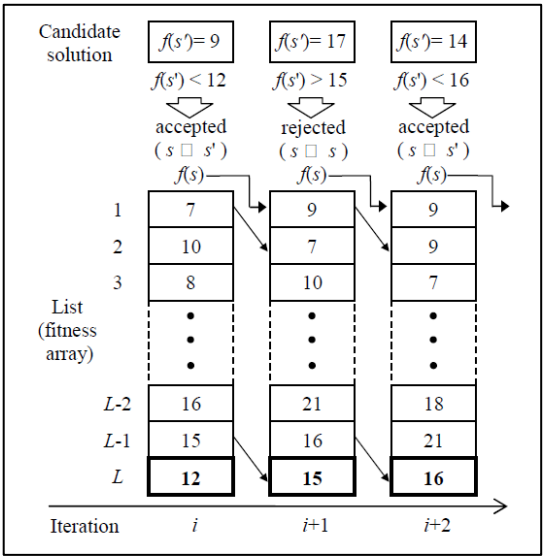


Gambar 2.2 *Hyper-heuristic Framework*

Hyper-Heuristic berbeda dengan *Meta-Heuristic*. Berdasarkan Gambar 2.2, pencarian solusi optimal *Hyper-Heuristic* dilakukan diatas *search space* berupa *lower-level Heuristic*. Sementara *meta-Heuristic* bekerja diatas *search space* berupa *solution space*. Hal ini dikarenakan *Hyper-heuristics* tidak bersinggungan langsung dengan *solution space*, namun dengan *lower-level Heuristic*. Hal tersebut menyebabkan *Hyper-Heuristic* tidak perlu parameter *tuning* manual untuk setiap *problem*, karena parameter *tuning* dilakukan secara otomatis. [6][10]

2.2.4. *Late acceptance Strategy*

Late acceptance Strategy adalah teknik baru dalam *meta-Heuristics* yang dikenalkan oleh Burke dan Bykov [8] yang membahas banyak mengenai permasalahan penjadwalan ujian. *Late acceptance* merupakan metode optimasi umum yang merupakan perluasan dari *Hill-Climbing*, pada *hill-climbing* solusi yang diterima adalah kualitasnya lebih baik dibandingkan sebelumnya, *Late acceptance Strategy* menerima solusi jika kualitasnya lebih baik daripada solusi dan iterasi yang terlihat. *Late acceptance* telah terbukti dapat bekerja sebaik metode lainnya seperti *Simulated Annealing* dan *Great Deluge*, namun tergantung pada pengaturan parameter, panjang memori. [10]
iaya sebelumnya pada urutan ke-*L*.



Gambar 2.3 Ilustrasi Penerapan *Late acceptance Strategy*

Pada Gambar 2.3 menunjukkan bahwasannya solusi pada iterasi ke-*(i)* menuju iterasi ke-*(i+1)* dapat diterima karena memiliki nilai biaya lebih baik daripada iterasi solusi pada urutan ke-*L* sebelumnya, sehingga kandidat solusi terus diterima dan masuk ke dalam daftar. Sedangkan iterasi ke-*(i+2),* kandidat solusi ditolak karena tidak memiliki hasil yang lebih baik daripada nilai b

2.2.5. *Late acceptance hill climbing*

Late acceptance hill climbing (*LAHC*) adalah prosedur pencarian *local* berulang yang terinspirasi oleh algoritma optimasi pendekatan *Hill climbing* [11].

```
Produce an initial solution s
Calculate initial cost function C (s)
Specify Lh
For all k € {0 . . . Lh -1} fk : = C (s)
First iteration I : = 0; Iidle : = 0
Do until (I > 100000) and (Iidle > I*0.02)
        Construct a candidate solution s*
        Calculate a candidate cost function C (s*)
        If C (s*) >= C(s)
                The increment the idle iteration number Iidle : =
Iidle+1
                Else reset the idle iterations number Iidle : = 0
        Calculate the virtual beginning v : = I mod Lh
        If C(s*) < fv or C (S*) <= C(s)
                Then accept the candidate s : = s*
                Else reject the candidate s : = s
        If C(s) < fv
                Then update the fitness array fv : = C(s)
        Increment the iteration number I : = I+1
```

Kode 2.1 ***Pseudocode Late acceptance hill climbing***

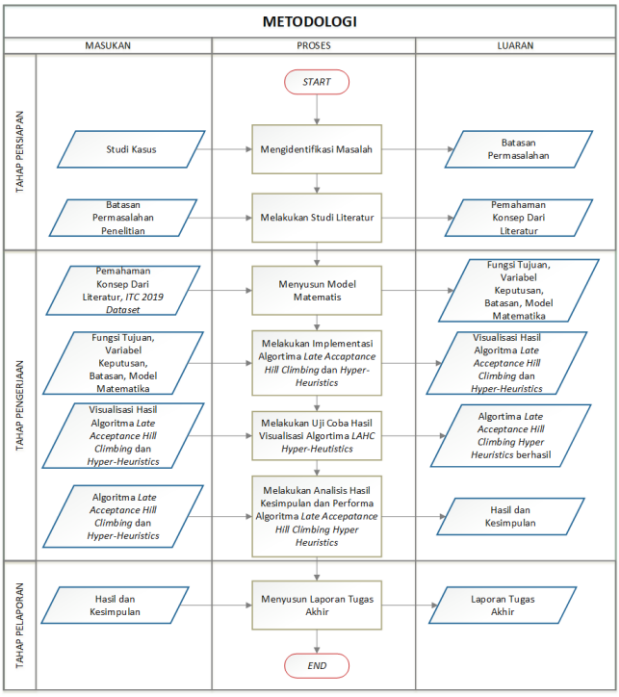
Algoritma ini merupakan strategi heuristik baru yang memanfaatkan berbagai pendekatan untuk keluar dari *local optimal* dengan hati-hati yang memiliki kemungkinan menghasilkan gerakan buruk selama proses pencarian untuk mencapai solusi kandidat yang lebih baik selanjutnya. Prosedur *LAHC* dimulai dari solusi awal dan secara *iterative* meningkatkan solusi individu untuk menghasilkan solusi baru yang diinginkan. Pada dasarnya, konsep klasik *Hill climbing* sudah memiliki kinerja bagus dan cepat, namun kekurangannya adalah terletak pada *local optimum* yang dapat dengan mudah terjebak disuatu titik. Hal inilah yang menjadi alasan untuk meningkatkan algoritma tersebut dengan menggunakan *LAHC*. [6]

Pada Kode 2.1 adalah *pseudocode* yang dimiliki oleh algoritma *Late acceptance hill climbing* dimana langkah awal didapatkan dari *initial solution*, kemudian menghasilkan biaya untuk solusi tersebut. Langkah selanjutnya adalah membangun solusi yang masuk kedalam daftar kandidat untuk kemudian dibandingkan. Apabila hasil solusi pada tahap berikutnya lebih baik, maka solusi tersebut akan diterima, namun jika tidak maka solusi tidak akan masuk ke dalam daftar solusi.

BAB III
METODOLOGI PENELITIAN

Pada bab ini akan dijelaskan mengenai alur metodologi yang akan dilakukan dalam tugas akhir ini. Metodologi ini juga digunakan sebagai pedoman untuk melaksanakan tugas akhir agar terarah dan sistematis.

3.1. Tahapan Pelaksanaan Tugas Akhir



Gambar 3.1 Alur Pengerjaan Tugas Akhir

Pada subbab ini akan menjelaskan mengenai metodologi pengerjaan dalam pelaksanaan tugas akhir. Metodologi ini digunakan sebagai panduan dan memberikan alur pengerjaan tugas akhir yang sistematis. Metodologi pengerjaan tugas akhir dapat dilihat pada Tabel 3.1. Berdasarkan bagan alur metodologi pengerjaan pada Gambar 3.1, dibawah ini merupakan penjelasan dari prosesnya.

3.1.1. Identifikasi Permasalahan

Tahapan ini merupakan langkah awal yang dilakukan dalam pengerjaan tugas akhir ini. Pada tahapan ini dilakukan identifikasi permasalahan yang ada pada *Course Timetabling Problem* menggunakan data yang disediakan oleh *International Timetabling Competition 2019*. Identifikasi masalah dilakukan dengan mengetahui proses bisnis, batasan, ruang lingkup dan hal terkait lainnya. Hasil yang akan ditemukan adalah permasalahan penjadwalan mata kuliah pada studi kasus tersebut.

3.1.2. Studi Literatur

Pada tahapan ini dilakukan pengumpulan berbagai sumber informasi yang memiliki keterkaitan dengan konsep yang akan digunakan dalam pengerjaan tugas akhir ini serta mengkaji pustaka yang terkait. Pustaka yang digunakan yaitu dapat berupa *paper*, jurnal, laporan penelitian, ataupun tugas akhir yang terkait dengan permasalahan yang diangkat pada penelitian ini.

3.1.3. Pembuatan Model Matematis

Model matematis diperlukan untuk mendapatkan menemukan penyelesaian permasalahan dari data-data yang sudah

diperoleh. Tujuan dari pemodelan ini adalah untuk menentukan fungsi tujuan, variabel keputusan dan juga batasan dari permasalahan yang didapatkan dan mengacu pada subbab 1.3. Berikut keterangan untuk masing-masing model:

1. Fungsi Tujuan
   Fungsi tujuan merupakan tujuan yang akan dicapai dari pengerjaan tugas akhir ini. Fungsi tujuan dari tugas akhir ini adalah meminimalkan semua biaya yang harus dikeluarkan untuk masing-masing konten yang terdapat pada *dataset*.
2. Variabel Keputusan
   Variabel keputusan merupakan seperangkat variabel yang belum diketahui nilainya yang kemudian akan dicari dalam permasalahan yang didapatkan. Dalam pengerjaan tugas akhir ini, didapatkan beberapa variabel yang akan dicari hasilnya. Permasalahan *ITC 2019* secara umum memiliki dua tujuan yaitu menjadwalkan kelas pada waktu dan ruang yang telah disediakan dan mahasiswa pada kelas yang telah terjadwal. Konten-konten yang kemudian akan menjadi variabel keputusan adalah jumlah *times*, jumlah *courses*, jumlah *rooms*, dan jumlah *students*.
3. Fungsi Batasan
   Batasan merupakan beberapa fungsi yang digunakan untuk memberi batasan terhadap pencapaian fungsi tujuan oleh variabel keputusan. Batasan permasalahan pada *ITC 2019 Dataset* terbagi menjadi 19 bagian dengan definisi dan ketentuan berbeda-beda untuk masing-masing batasan tersebut. Kemudian batasan tersebut dapat berlaku menjadi *Hard constraint*

dengan diikuti tanda penulisan *required* yang berarti batasan harus dipenuhi dan bersifat wajib dan menjadi *Soft Constraint* dengan diberi *penalty*. Berikut Tabel 3.1 adalah daftar semua batasan yang tersedia.

Tabel 3.1 Daftar Batasan *Dataset*

Batasan	Definisi
SameStrart	Kelas dimulai pada waktu yang sama
SameTime	Kelas berlangsung dengan rentang waktu yang sama
SameDays	Kelas dijadwalkan pada hari yang sama
SameWeeks	Kelas dijadwalkan pada minggu yang sama
SameRoom	Kelas dijadwalkan pada ruang yang sama
DifferentTime	Kelas dijadwalkan dengan durasi yang berbeda
DifferentDays	Kelas dijadwalkan pada hari yang berbeda
DifferentWeeks	Kelas dijadwalkan pada minggu yang berbeda
DifferentRoom	Kelas dijadwalkan pada ruang yang berbeda
Overlap	Kelas dijadwalkan saling tumpang tindih untuk durasi, hari dan minggu
NotOverlap	Kelas dijadwalkan tidak saling tumpang tindih untuk durasi, hari dan minggu. Sebagai berikut.

Batasan	Definisi
SameAttendees	Kelas yang dijadwalkan memenuhi kriteria dengan mempertimbangkan waktu perjalanan antara lokasi kelas
Precedence	Kelas harus dijadwalkan pada waktu tertentu yaitu diawal waktu pada setiap minggu, hari, dan *slot*
WorkDay	Kelas dijadwalkan tidak boleh lebih dari jam kerja
MinGap	Kelas dijadwalkan dengan memiliki minimal waktu selisih antara satu dengan yang lainnya
MaxDays	Kelas dijadwalkan tidak boleh lebih dari hari yang kerja telah ditentukan dalam seminggu
MaxDayload	Jumlah durasi kelas tidak boleh lebih dari maksimal *slot* yang telah ditentukan dalam sehari
MaxBreaks	Jumlah waktu istirahat tidak boleh lebih dari yang telah ditentukan
MaxBlock	Kelas dijadwalkan pada blok-blok tertentu sesuai dengan urutan yang ditentukan

3.1.4. Implementasi Algoritma *LAHC dan Hyper - Heuristic*

Pada tahap ini dilakukan pembuatan *Algoritma Late acceptance hill climbing* dan *Hyper-heuristics* menggunakan bahasa pemrograman Java. Sehingga dapat dijalankan dan diketahui performa dari algoritma tersebut. Dari hasil *generate*

code tersebut, akan muncul perhitungan komputasi yang menjawab permasalahan dalam tugas akhir ini. Proses implementasi diterapkan pada *ITC 2019 Dataset*. 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*. Dalam kelompok *dataset early* terdapat 10 *instances* yang tersimpan dalam 10 *files* yang berbeda. Kemudian algoritma akan mencari *initial solution* untuk masing-masing *instances* yang menjadi masukan dan selanjutnya akan disimpan pada *fitness array*. *Fitness array* adalah daftar dengan panjang tertentu yang berfungsi untuk menyimpan nilai dari solusi awal yang telah didapatkan. Kemudian dilakukan iterasi, untuk setiap iterasinya algoritma akan menghasilkan hasil yang berbeda-beda. Nilai hasil solusi tersebut yang kemudian akan dibandingkan. Terdapat perbedaan pada algoritma *Greedy Hill climbing* yaitu pada perbandingan kandidat solusi tepat pada solusi terakhir yang muncul, *LAHC* melakukan perbandingan kandidat solusi dengan solusi yang telah dilewati pada iterasi sebelumnya.

Tahap selanjutnya adalah menentukan *acceptance criteria* yang akan diterapkan untuk kemudian menentukan apakah solusi pada iterasi tertentu akan diterima atau ditolak. Pada *LAHC*, solusi yang akan diterima adalah solusi yang memiliki nilai lebih baik atau dibandingkan solusi pembanding sebelumnya. Kemudian juga diperlukan untuk menentukan *stopping criteria* untuk menentukan hasil terbaik untuk iterasi tersebut. Seperti jumlah iterasi ataupun lama waktu yang

dihabiskan. Hasil luaran yang didapatkan adalah model solusi untuk penjadwalan mata kuliah beserta mahasiswa yang mengambil mata kuliah tersebut.

3.1.5. Uji Coba

Pada tahapan ini adalah tahap dimana dilakukan uji coba algoritma yang telah diimplementasikan pada tahap sebelumnya untuk dilakukan validasi. Pengujian dilakukan pada *ITC 2019 Dataset* yang terdiri dari sepuluh *Instances*. Uji coba dilakukan untuk menunjukkan apakah pemodelan algoritma tersebut sudah dapat memenuhi *hard constraint* dan *soft constraint*, serta apakah hasil yang didapatkan sudah dapat memenuhi fungsi tujuan yang ingin dicapai. Maka untuk mencapai semua model matematis diperlukan uji coba untuk dapat dilakukan kajian ulang terhadap model yang didapat. Jika ternyata terdapat algoritma yang diimplementasikan tidak sesuai dengan batasan dan ketentuan yang berlaku, akan dilakukan evaluasi kesalahan dan perbaikan terhadap algoritma untuk kemudian dilakukan uji coba kembali sebagai validasi.

3.1.6. Analisis Hasil Performa Algoritma dan Kesimpulan

Pada tahap ini dilakukan analisis hasil performa algoritma. Analisis hasil performa dilakukan dengan mengamati dan membandingkan luaran dari setiap iterasi yang dilakukan untuk setiap *Instances* yang dijadikan sebagai masukan. Analisis juga dilakukan terhadap kecepatan algoritma dalam menyelesaikan permasalah dan kemampuan algoritma dalam mencapai fungsi tujuan yaitu mencari biaya terendah untuk setiap *resources* yang digunakan. Kemudian dilakukan

penarikan kesimpulan yang dilakukan dengan membandingkan hasil analisis dari perhitungan yang telah dilakukan antara data yang digunakan. Dari analisis yang dilakukan dapat diketahui kelebihan dan kekurangan dari algoritma yang digunakan untuk menyelesaikan permasalahan penjadwalan mata kuliah.

3.1.7. Penyusunan Buku Tugas Akhir

Penyusunan laporan merupakan tahap akhir dari proses-proses yang telah dilakukan sebelumnya. Pada tahap ini dilakukan dokumentasi terhadap proses-proses yang telah dilakukan dan kesimpulan dari permasalahan yang ingin diselesaikan. Seluruh pengerjaan dan pelaksanaan tugas akhir ini akan mengikuti format yang telah ditentukan oleh laboratorium Rekayasa Data dan Intelegensi Bisnis (RDIB) serta yang berlaku di Departemen Sistem Informasi ITS.

BAB IV
PERANCANGAN

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

4.1. Pemahaman Data

Pada penelitian tugas akhir ini, data yang digunakan adalah *ITC 2019 dataset*. Kumpulan data ini dapat diperoleh melalui situs resmi kompetisi berikut *https://www.itc2019.org/format* yang merupakan *dataset* milik Kompetisi Penjadwalan Internasional 2019 *(International Timetabling Competition 2019).*

ITC 2019 dataset memiliki tiga grup data yaitu *Early, Middle*, dan *Late.* Namun pada pengerjaan tugas akhir ini memiliki batasan hanya mengerjakan *dataset* dengan jenis grup *Early* yang memiliki 11 *instances*. Masing-masing *instances* memiliki ukuran yang berbeda-beda dan jumlah masing-masing konten berbeda pula. Adapun karakteristik masing-masing *dataset* dapat dilihat pada Tabel 2.1.

Pada subbab berikut akan dijelaskan secara lebih rinci mengenai masing-masing konten yang memiliki format inisiasi yang berbeda. Berikut inisiasi untuk variabel, batasan dan konten yang berlaku pada *ITC 2019 dataset*.

4.1.1. Inisiasi Konten *Dataset*

Permasalah yang didefinisikan pada masing-masing instance memiliki jumlah minggu (nrWeeks), jumlah hari (nrDays) dan

jumlah slot untuk setiap hari (slotsPerDay). Setiap parameter tersebut akan didefinisikan pada awal *file* data XML. Kode 4.1 menunjukkan inisiasi data yang disajikan secara umum.

```
<?xml      version="1.0"      encoding="UTF-8"?>
<!DOCTYPE solution PUBLIC "

        -//ITC  2019//DTD  Problem  Format/EN"
        "http://www.itc2019.org/competition-
        format.dtd">

<problem name="unique-instance-name" nrDays="7"
nrWeeks="13" slotsPerDay="288">

        <optimization.../>
        <rooms>...</rooms>
        <courses>...</courses>
        <distributions>...</distributions>
        <students>...</students>

</problem>
```

Kode 4.1 Format Kode Garis Besar

Setelah mendefinisikan permasalahan pada baris pertama, kemudian diikuti dengan *tag optimization* yang berisi pembobotan untuk masing-masing instance yang akan dihitung pada saat akhir setelah mendapatkan hasil solusi. Kemudian diikuti tag yang berisi konten-konten seperti *rooms*, *courses*, distributions, dan *students*.

4.1.1.1. Konten Times

Kelas-kelas dapat berlangsung dalam waktu yang telah didefinisikan. Pembacaan waktu terbagi menjadi 3 yaitu minggu, hari dan jam yang diinisiasi menjadi slot. Berikut masing-masing penjelasannya berdasarkan contoh pada Kode 4.2

```
<class id="1" limit="20">

        <!-- Monday-Wednesday-Friday 7:30 - 8:20 All Weeks -->
        <time      days="1010100"      start="90"      length="10"
        weeks="1111111111111"                       penalty="20"/>
        <!-- Tuesday-Thursday 9:00 - 10:15 All Weeks --> <time
        days="0101000"          start="108"          length="15"
        weeks="1111111111111"                       penalty="0"/>
        <!--Thursday   8:00   -   9:50   Even   Weeks-->   <time
        days="0001000"          start="96"          length="22"
        weeks="0101010101010" penalty="2"/>

</class>
```

Kode 4.2 Format Kode Konten Waktu

- Minggu (Weeks): Jumlah minggu telah didefinisikan di awal permasalahan yang kemudian didefinisikan menjadi biner 0 atau 1 sepanjang jumlah minggu yang berlangsung. Biner 0 berarti minggu tersebut tidak dijadwalkan sedangkan 1 memiliki arti minggu tersebut terjadwal. Contoh: Terdapat 13 minggu jadwal perkuliahan yang kemudian didefinisikan menjadi 0101010101010 yang menunjukkan jadwal tersebut berlangsung pada minggu ke 2, 4, 6, 8, 10 dan 12 dari total 13 minggu pertemuan.
- Hari (Days): Jumlah hari juga telah didefinisikan pada awal permasalahan. Pembacaan juga didefinisikan menjadi biner 0 atau 1 sebanyak jumlah hari. Sebagai contoh, hari 0100000 yang menunjukkan jadwal hanya berlangsung pada hari kedua. Hari kedua menunjukkan hari selasa dikarenakan hari dimulai senin, selasa sampai minggu secara urut biner.
- Slot (SlotPerDays): Jumlah slot juga telah didefinisikan pada awal permasalahan. Jumlah slot perharinya adalah 0-288 dimana masing-masing slot mengambil nilai 5 menit.

Sehingga slot ke-0 berarti pukul 00.05 dimulai pada malam hari sampai malam kembali.

4.1.1.2. Konten *Rooms*

Konten *room* memiliki beberapa elemen yang mengisi tag name pada *room*. Kode 4.3 menunjukkan contoh *file* xml untuk *room*.

```
<rooms>
        <room              id="1"              capacity="50"/>
        <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
        timare                    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 --> <unavailable days="000100" start="144"
        length="144"                    weeks="1010101010101"/>
        </room>
</rooms>
```

Kode 4.3 Format Kode Konten Ruangan

Elemen-elemen tersebut terdiri sebagai berikut:

- Nomor identitas ruangan (id) dan kapasitas ruangan (capacity). Nilai identitas menunjukkan nilai unik untuk masing-masing ruangan dan memiliki kapasitas tampung untuk jumlah mahasiswa yang nan*tinya* akan menggunakan ruangan tersebut.
- Waktu tempuh antar ruangan (*Travel Room*). Elemen ini menunjukkan waktu yang dihabiskan untuk jarak tempuh antara satu ruang dengan ruang lainnya. Nilai (value) masing-masing senilai dengan slot yang berarti jika value

memiliki nilai 1 maka jarak antara ruang tersebut ke ruang lainnya sebesar 1 slot (5 menit).
- Waktu yang tidak tersedia (Unavailable). Ruang tertentu memiliki kemungkinan tidak tersedia pada waktu-waktu tertentu yang diinisiasi dengan tag name unavailable dengan elemen sama dengan konten time yaitu minggu, hari, dan slot.

4.1.1.3. Konten *Courses*

Courses memiliki hirarki yang sangat kompleks untuk struktur kelas dan *event* yang akan dijadwalkan. Berikut adalah elemen yang terdapat dalam konten *courses*.
- Identitas mata kuliah (*course* id). Elemen ini menunjukkan kode unik untuk masing-masing mata kuliah.
- Identitas konfigurasi (*config* id). Elemen ini adalah turunan dari mata kuliah. Konfigurasi menunjukkan jenis mata kuliah tersebut. Dimana terdapat tiga jenis mata kuliah yaitu sesi perkuliahan, seminar dan laboratorium. Setiap konfigurasi mata kuliah dapat terdiri dari satu atau lebih jenis sesi.
- Identitas sub bagian (*subpart* id). Elemen ini adalah turunan konfigurasi dimana menunjukkan masing-masing sesi. *Subpart* dapat berupa sesi perkuliahan, seminar ataupun laboratorium.
- Identitas kelas (*class* id) dan batas kuota (*limit*). Elemen ini menunjukkan identitas masing-masing kelas dan batas kuota kelas yang dapat diikuti oleh mahasiswa. Kelas inilah yang kemudian akan dijadwalkan pada waktu tertentu dan ruang tertentu pula.

```
<course id="ME 263">
          <config id="1"><!-- Lec-Rec configuration, not
          linked (any Lec with any Lab) -->
          <subpart id="1_Lecture"> <!-- Lecture subpart --
          >
                    <classid="Lec1"limit="100"/>
                    <class id="Lec2"limit="100"/>
          </subpart>
          <subpart id="2_Recitation"> <!-- Recitation
          subpart -->
                    <classid="Rec1"limit="50"/>
                    <class id="Rec2"limit="50"/>
          </subpart>
          </config>
          <config id="2"> <!-- Lec-Rec-Lab configuration,
          linked                                        -->
          <subpart id="3_Recitation"> <!-- Lecture subpart
          -->
                    <classid="Rec5"parent="Lec3"
                    limit="50"/>
                    <classid="Rec6"parent="Lec3"
                    limit="50"/>
          </subpart>
          </config>
</course>
```

Kode 4.4 Format Kode Konten Mata Kuliah

Selain itu terdapat beberapa kelas yang akan memiliki hirarki *parent-child* yang akan saling terkait antara satu dan lainnya. Contohnya ketika mahasiswa mengambil kelas dengan kode 1, kemudian kelas kode 1 memiliki *parent* dengan kelas kode 10 maka mahasiswa tersebut juga akan mengambil kelas kode 10. Berikut Kode 4.4 adalah contoh kode yang dituliskan pada *file* XML.

4.1.1.4. Konten *Students*

Konten *students* memiliki dua elemen yang terdiri dari identitas mahasiswa (*student* id) berisi kode unik masing-masing mahasiswa dan (*course* id) yang merupakan daftar

mata kuliah yang harus diambil oleh mahasiswa tersebut. Berikut Kode 4.5 menunjukkan contoh kode yang dituliskan pada *file* XML untuk *student*.

```
<students>
        <student id="1">
                <courseid="1"/>
                <course id="5"/>
        </student>
        <student id="2">
                <courseid="1"/>
                <courseid="3"/>
                <course id="4"/>
        </student>
</students>
```

Kode 4.5 Format Kode Konten Mahasiswa

4.1.1.5. Konten Distribution *Constraint*

```
<distributions>
        <!-- classes 1 and 2 cannot overlap in time
        -->
        <distributiontype="NotOverlap"
        required="true">
                <classid="1"/>
                <class id="2"/>
        </distribution>
        <!-- class 1 should be before class 3,
        class 3 before class5-->
        <distribution type="Precedence"
        penalty="2">
                <classid="1"/>
                <classid="3"/>
                <class id="5"/>
        </distribution>
</distributions>
```

Kode 4.6 Format Kode Konten Batasan-Batasan

Distribution adalah batasan yang harus dipenuhi oleh konten lainnya dalam menjadwalkan masing-masing mata kuliah dan mahasiswa kedalam waktu, dan ruang yang telah disediakan. Terdapat dua jenis batasan yaitu *hard* dan *soft*. *Hard constraint* harus dipenuhi dan bersifat wajib untuk dipenuhi. Elemen ini diikuti dengan tanda required=”true” sedangkan *soft constraint* berupa kepuasan dengan nilai penalti yang akan mempengaruhi cost pada hasil akhir. Terdapat 19 jenis batasan yang berlaku dalam *dataset* dengan kriteria dan ketentuan masing-masing. Kode 4.6 adalah contoh penulisan batasan.

4.2. Formulasi Model Matematis

Pada tahap formulasi model matematis ini adalah tahap melakukan pendefinisian fungsi tujuan, variabel keputusan dan batasan menjadi model matematis dengan tujuan mempermudah penyelesaian permasalahan yang akan dikerjakan. Berikut masing-masing formulasi yang telah didefinisikan.

4.2.1. Konversi Fungsi Tujuan

$$\textbf{Minimize P} = \boldsymbol{P1} + \boldsymbol{P2} + \boldsymbol{P3}$$

Persamaan (1)

Fungsi tujuan digunakan bertujuan untuk memastikan apakah model yang dibentuk telah mencapai fungsi yang ingin dicapai. Pada model ini tujuan yang ingin dicapai adalah meminimalkan nilai penalti pada setiap batasan yang berlaku. Dengan artian biaya yang harus dikeluarkan dalam penyusunan jadwal harus seminimal mungkin sebagaimana pernyataan pada Persamaan (1) berikut.

Dengan:

P1 = Nilai penalti terhadap ruangan yang dipilih untuk digunakan dalam penjadwalan kelas suatu mata kuliah

P2 = Nilai penalti terhadap *timeslot* yang dipilih untuk digunakan dalam penjadwalan kelas suatu mata kuliah

P3 = Nilai penalti pada *soft* distribution *constraint* yang berhasil dipenuhi

Nilai penalti yang dihitung berasal dari tiga sumber. Yang pertama berasal dari penalti penggunaan ruangan yang dipilih untuk digunakan dalam penjadwalan kelas suatu mata kuliah yang dinyatakan sebagai *P1.* Persamaan dari *P1* dapat dinyatakan sebagai berikut pada Persamaan (2).

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

Persamaan (2)

Dengan:

c = {1,2, …, *C* } adalah urutan kelas dari masing-masing mata kuliah

PRc adalah nilai pinalti dari ruangan yang digunakan oleh kelas suatu mata kuliah.

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

P2 adalah menghitung nilai pinalti dari *timeslot* yang digunakan oleh kelas suatu mata kuliah. Persamaan dari *P2* dapat dinyatakan sebagai berikut pada Persamaan (3).

$$P2 = \min \sum_{c=1}^{C} PT_c$$

Persamaan (3)

Dengan:

c = {1,2, …,*C*} adalah urutan kelas dari masing-masing mata kuliah

PTc adalah nilai penalti dari *timeslot* yang digunakan oleh kelas suatu mata kuliah.

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

P3 adalah menghitung nilai penalti dari masing-masing batasan yang telah didefinisikan pada suatu *dataset*. Persamaan dari P3 dapat dinyatakan dalam bentuk matematis sebagai berikut pada Persamaan (4).

$$P3 = \min \sum_{i=1}^{I} PDC_i$$

Persamaan (4)

Dengan:

i = {1,2, …, I} adalah urutan kelas dari masing-masing mata kuliah

PDCi adalah vektor yang menunjukkan *soft constraint*

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

4.2.2. Konversi Variabel Keputusan

Variabel keputusan adalah variabel yang harus dapat memenuhi semua *hard constraint* yang telah ditentukan sebelumnya. Variabel keputusan ini digunakan untuk memastikan bahwa setiap kelas dan mahasiswa telah dijadwalkan. Variabel keputusan pada studi kasus ITC 2019 terbagi menjadi dua sebagai berikut.

(1) menempatkan *courses* pada *time* dan *rooms* yang tersedia

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

Persamaan (5)

dimana,

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

(2) membagi *student* dalam *classes* berdasarkan *course* yang mereka minta.

$$\sum_{s=1}^{S}\sum_{c=1}^{C}\sum_{o=1}^{O} Y_{sco} = 1$$

Persamaan (6)

dimana,

$$Ysco\begin{cases} 1, & jika\ mahasiswa\ ke-s\ dijadwalkan \\ & pada\ kelas\ ke-c\ berdasarkan \\ & mata\ kuliah\ ke-o \\ 0, & jika\ tidak \end{cases}$$

Dengan:

O = *course* = $\{o_1,\ldots,o_{|O|}\}$ adalah urutan mata kuliah yang diambil mahasiswa

C = *class* = $\{c_1,\ldots,c_{|C|}\}$ adalah urutan kelas yang akan dijadwalkan

T = *time* = $\{t_1,\ldots,t_{|T|}\}$ adalah *timeslot* yang tersedia

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

S = *student* = $\{s_1,\ldots,s_{|S|}\}$ adalah urutan mahasiswa

4.2.3. Konversi Batasan Distribusi Masalah

Batasan yang berlaku terbagi menjadi sembilan belas yang dapat berlaku sebagai batasan keras (*hard constraint*) dan batasan lunak (*soft constraint*). Variabel umum yang digunakan pada batasan distribusi.

D = *day* = $\{d_1,\ldots,d_{|7|}\}$ adalah urutan jumlah hari dalam seminggu

W = *week* = $\{1,...,w_{|W|}\}$ adalah urutan jumlah minggu yang berlangsung dalam satu semester

P = *slot* = $\{1,\ldots 288\}$ adalah urutan jumlah slot yang terdapat dalam satu hari

S = *start* = {1,…,288} adalah urutan slot dimulai

L = *length* = $\{0,\ldots,l_{|\mathbb{L}|}\}$ adalah urutan panjang slot berlangsung

E = *end* = {1,…,288} adalah urutan slot berakhir

Berikut adalah model matematis masing-masing batasan distribusi yang berlaku.

Distribution Constraint 1 (*Same Start*): Setiap kelas yang masuk dalam daftar batasan ini harus dimulai pada waktu yang sama. Pernyataan ini terdapat pada Persamaan (7) sebagai berikut.

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

Persamaan (7)

dimana,

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

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

Distribution Constraint 2 *(Same Time):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dalam rentang waktu yang sama. Pernyataan ini terdapat pada Persamaan (8) sebagai berikut.

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

Persamaan (8)

dimana,

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

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

e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu

Distribution Constraint 3 (*Different Time*): Setiap kelas yang masuk dalam daftar batasan ini tidak boleh berlangsung dalam rentang waktu yang sama. Pernyataan ini terdapat pada Persamaan (9) sebagai berikut.

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

Persamaan (9)

dimana,

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

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

e = {1,2, …, 288} adalah waktu akhir kelas yang terjadwal pada *timeslot* tertentu

Distribution Constraint 4 *(Same Days):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan dalam hari yang sama. Pernyataan ini terdapat pada Persamaan (10) sebagai berikut.

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

Persamaan (10)

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

Distribution Constraint 5 (*Different Days):* Setiap kelas yang masuk dalam daftar batasan ini tidak dapat berlangsung dan dijadwalkan dalam hari yang sama. Pernyataan ini terdapat pada Persamaan (11) sebagai berikut.

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

Persamaan (11)

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

Distribution Constraint 6 *(Same Weeks):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan

dalam hari yang sama. Pernyataan ini terdapat pada Persamaan (12) sebagai berikut.

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

Persamaan (12)

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

Distribution Constraint 7 *(Different Weeks):* Setiap kelas yang masuk dalam daftar batasan ini tidak dapat berlangsung dan dijadwalkan dalam minggu yang sama. Pernyataan ini terdapat pada Persamaan (13) sebagai berikut.

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

Persamaan (13)

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

Distribution Constraint 8 *(Same Rooms):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan dalam ruangan yang sama. Pernyataan ini terdapat pada Persamaan (14) sebagai berikut.

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

Persamaan (14)

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

Distribution Constraint 9 *(Different Rooms):* Setiap kelas yang masuk dalam daftar batasan ini tidak berlangsung dan dijadwalkan dalam ruangan yang sama. Pernyataan ini terdapat pada Persamaan (15) sebagai berikut.

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

Persamaan (15)

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

Distribution Constraint 10 *(Overlap):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan saling tumpah tindih pada durasi yang berlangsung, hari dan minggu. Pernyataan ini terdapat pada Persamaan (16) sebagai berikut.

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

Persamaan (16)

dimana,

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

Distribution Constraint 11 *(Not Overlap):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan tidak diperbolehkan saling tumpah tindih untuk durasi, hari dan minggu yang berlangsung. Pernyataan ini terdapat pada Persamaan (17) sebagai berikut.

$$DC11 = \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} e_i \leq s_j \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} e_j \leq s_i \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} d_{ij} = 0 \bigwedge \sum_{i=1}^{c-1}\sum_{j=i+1}^{c} w_{ij} = 0$$

Persamaan (17)

dimana,

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

Distribution Constraint 12 *(Same Attendees):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan berdasarkan kriteria dengan mempertimbangkan waktu *travel* antar kelas sehingga tidak terjadi konflik bagi mereka yang mengambil dan mengajar kelas tersebut. Pernyataan ini terdapat pada persamaan (2) sebagai berikut.

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

Persamaan (1)

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

dimana,

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

r = {1,2, …, R} adalah urutan ruangan yang dapat dijadwalkan pada kelas tertentu

Distribution Constraint 13 *(Precedence):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan diawal waktu untuk slot, hari dan minggu dari jadwal yang tersedia. Pernyataan ini terdapat pada Persamaan (18) sebagai berikut.

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

Persamaan (18)

dimana,

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

r = {1,2, …, R} adalah urutan ruangan yang dapat dijadwalkan pada kelas tertentu

Distribution Constraint 14 *(Work Day (S)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak boleh melebihi slot yang telah ditentukan setiap harinya. Dimana waktu antara mulainya kelas pertama dan berakhirnya kelas terakhir tidak melebihi S *timeslot*. Pernyataan ini terdapat pada Persamaan (19) sebagai berikut.

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

Persamaan (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

Distribution Constraint 15 *(Min gap (G)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan pada hari tertentu harus memiliki selisih setidaknya sebesar *G timeslot* antara berakhirnya suatu kelas dengan mulainya kelas berikutnya. Pernyataan ini terdapat pada Persamaan (20) sebagai berikut.

$$D15 = \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} d_{ij} = 0\right) \vee \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} w_{ij} = 0\right) \vee \left(\sum_{i=1}^{c-1} e_i + G \leq \sum_{j=i+1}^{c} s_j\right) \vee \left(\sum_{j=i+1}^{c} e_j + G \leq \sum_{i=1}^{c-1} s_i\right)$$

Persamaan (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

Distribution Constraint 16 *(Max Days (D)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak melebihi batas maksimal hari sebesar D yang telah ditentukan setiap minggunya pada hari kerja yang berbeda. Pernyataan ini terdapat pada Persamaan (21) sebagai berikut.

$$D16 = \text{countNonzeroBits} \sum_{i=1}^{c-1} d_i \leq D$$

Persamaan (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

Distribution Constraint 17 *(Max Day Load (S)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak boleh melebihi sebanyak S *timeslot* dalam sehari sesuai dengan yang telah ditentukan jumlahnya setiap minggunya dalam satu semester. Pernyataan ini terdapat pada Persamaan (22) sebagai berikut.

$$D17 = DayLoad(d, w) \leq S$$

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

Persamaan (22)

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

Distribution Constraint 18 *(Max Breaks (R, S)): Setiap* kelas yang masuk dalam daftar batasan ini dijadwalkan tidak boleh melebihi jumlah jeda sebanyak R antar kelas selama sehari dengan jumlah slot sebanyak S pula. Pernyataan ini terdapat pada Persamaan (23) sebagai berikut.

$$MaxBreaks(R,S) = \left| MergeBlocks\left\{(s,e) \left| (\sum_{i=1}^{c-1} d_i \neq 0 \ \wedge 2^d \neq 0) \ \wedge_{(\sum_{i=1}^{c-1} w_i \neq 0 \ \wedge 2^w \neq 0)} \right.\right\} \right| \leq R + 1$$

Persamaan (23)

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

Persamaan (24)

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

Distribution Constraint 19 *(Max Block (M, S)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak melebihi batas maksimal hari yang telah ditentukan setiap minggunya. Pernyataan ini terdapat pada Persamaan (25) sebagai berikut.

$$DC19 = \left(max\left\{\left(eb - es \middle| b \in MergeBlocks\left\{s, e \middle| (\sum_{i=1}^{c-1} d_i \neq 0 \wedge 2^d \neq 0) \wedge (\sum_{i=1}^{c-1} w_i \neq 0 \wedge 2^w \neq 0)\right\}\right\}\right) \leq M$$

Persamaan (25)

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

Persamaan (26)

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. Menentukan *Initial Solution*

Solusi inisial dibentuk secara otomatis menggunakan algoritma yang dibangun dengan konsep *greedy algorithm*, yang berarti dibentuk dengan segala kemungkinan dan secara berulang. Selain itu, cara lainnya yang juga digunakan adalah menggunakan *backtracking algorithm* sebagai antisipasi bila terdapat kelas yang tidak dapat dijadwalkan pada kelas tertentu. Jadwal dibentuk berdasarkan aturan-aturan yang telah disimpan dan juga batasan yang harus terpenuhi. Pembentukan solusi inisial dibentuk berdasarkan lowest degree dimana kelas yang memiliki pilihan *timeslot* dan ruang lebih sedikit akan dijadwalkan lebih dahulu. Kemudian *largest degree* untuk menjadwalkan kelas yang memiliki *hard constraint* paling banyak. Sehingga terbentuklah solusi awal yang memenuhi semua batasan yang berlaku.

4.4. Implementasi Algoritma *Late acceptance hill climbing Hyper Heuristics*

Penerapan algoritma ini digunakan untuk menghasilkan solusi yang optimal dengan menyesuaikan nilai fungsi tujuan seminimal mungkin sehingga biaya yang dihabiskan dapat

seminimal mungkin. Sehingga nan*tiny*a akan dihasilkan solusi baru berdasarkan solusi awal yang telah dibentuk. Selain itu juga akan diterapkan teknik *low level heuristic* (local search) yaitu *move* untuk memodifikasi solusi awal hingga menjadi solusi yang lebih maksimal. Kemudian akan kembali dilakukan perhitungan biaya total yang dikeluarkan untuk solusi baru. Tidak lupa pula solusi yang baru harus tetap memenuhi batasan-batasan yang berlaku. Selengkapnya akan dibahas lebih lanjut pada bab V berikut.

4.5. Perencanaan Alur Skenario Uji Coba

Uji parameter yang dilakukan mengikuti parameter yang dimiliki oleh algoritma terpilih. Dalam hal ini adalah algoritma *Late Acceptance Hill Climbing Hyper Heuristics*. Berbeda dengan algoritma pembanding yang akan dilakukan uji coba pula untuk melihat hasil performa yang dimiliki algoritma terpilih. Berikut tabel 4.1 adalah ringkasan dari nilai parameter yang akan dilakukan uji coba.

Tabel 4.1 Daftar Parameter Uji Hasil

Parameter	Keterangan
Solusi Awal	Pembentukan hasil inisial solusi
Jumlah Iterasi	Nilai maksimal iterasi dilakukan
Nilai *Idle*	*Stopping criteria*
History Length	Penyimpanan solusi terbaik

Algoritma *Late Acceptance Hill Climbin*g memiliki tiga parameter yaitu, jumlah batasan iterasi dan nilai idle yang menjadi *stopping criteria* dan nilai *history length* yang menjadi tempat penyimpanan solusi-soulusi baik selama iterasi dilakukan. Paramter akan dilakukan uji coba dengan

nilai yang berbeda-beda. Jumlah iterasi yang dilakukan adalah batasan iterasi yang dilakukan untuk satu kali *running*. Apabila iterasi telah mencapai nilai yang telah ditetapkan tersebut, maka akan menjadi *stopping criteria*. Nilai iterasi minimal adalah 100.000 mengacu pada *psedocode* yang disampaikan oleh Edmund k. Burke. Parameter kedua adalah nilai idle yang juga menjadi *stopping criteria*. Apabila nilai penolakan saat melakukan iterasi sudah mencapai 2% dari total iterasi yang dilakukan, maka proses running tidak dapat kembali dilanjutkan. Nilai *idle* adalah nilai jeda yang menjadi *stopping criteria* iterasi. Nilai parameter terakhir adalah *history length* yang menjadi tempat penyimpanan solusi-solusi baik selama iterasi masing dilakukan dan berlangsung.

Uji coba parameter yang akan dilakukan terbagi menjadi 4 skenario yang memiliki tujuan berbeda.

4.5.1 Skenario 1: Pembentukan Solusi Awal

Skenario satu adalah uji coba yang dilakukan untuk mengetahui hasil dari 15 dataset yang diambil untuk diuji apakah berhasil memenuhi batasan wajib yang berlaku pada masing-masing dataset. Tujuan dari uji coba skenario ini adalah untuk mengetahui apakah jadwal yang terpilih sudah berhasil memenuhi keseluruhan batasan yang berlaku. Jika berhasil memenuhi maka solusi awal akan didapatkan untuk kemudian dilanjutkan mencari solusi akhir yang sudah dilakukan optimasi sehingga menghasilkan nilai penalti lebih baik dari hasil solusi awal. Dilakukan sekali *running* untuk melihat apakah seluruh kelas berhasil terjadwal. Jika tidak maka kelas yang tidak berhasil terjadwal dihitung dan didokumentasikan.

4.5.2 Skenario 2: Jumlah Iterasi

Skenario kedua adalah mengubah jumlah iterasi yang dilakukan untuk 11 kali running yang dilakukan. Uji coba ini dilakukan untuk dataset yang berhasil memenuhi ketentuan solusi awal. Tujuan dilakukan uji coba skenario 2 adalah melihat apakah iterasi akan memberi pengaruh pada nilai penalti yang lebih baik. Parameter jumlah iterasi adalah salah satu *stopping criteria* dari algoritma terpilih. Uji coba ini dapat melihat apakah semakin banyak iterasi yang dilakukan yang juga berarti memberikan rentang berhenti iterasi lebih banyak akan menghasilkan jumlah penalti yang lebih baik. Kemudian pada saat melakukan uji coba ini parameter lainnya tetap dalam keadaaan standar hanya jumlah iterasi saja yang berubah. Nilai jumlah iterasi yang akan diuji coba adalah 10,000, 100,000, 500,000 dan 1,000,000. Nilai ini diuji coba untuk melihat hasil perbandingan pada skala perbedaan dengan nilai perbedaan sebanyak ratursan untuk uji yang dilakukan.

4.5.3 Skenario 3: Perubahan Idle

Skenario ketiga adalah mengubah nilai *idle* yang dilakukan untuk 11 kali *running* yang dilakukan. Uji coba ini dilakukan untuk *dataset* yang berhasil memenuhi ketentuan solusi awal. Tujuan dilakukan uji coba skenario 3 adalah melihat apakah nilai *idle* akan memberi pengaruh pada nilai penalti yang lebih baik. Parameter nilai *idle* adalah salah satu *stopping criteria* dari algoritma terpilih. Uji coba ini dapat melihat apakah semakin besar nilai idle yang berada dalam rentang nilai nol sampai 1 yang dilakukan akan memberikan rentang berhenti pada iterasi lebih sedikit akan menghasilkan jumlah penalti yang lebih baik. Kemudian pada saat melakukan uji coba ini

parameter lainnya tetap dalam keadaaan standar hanya jumlah iterasi saja yang berubah. Nilai parameter yang dilakukan adalah dalam rentang 0 samapi dengan 1 yaitu 0.02, 0.2 dan 0.8. Nilai ini dimunculkan untuk melihat hasil apakah semakin mendekati 0 atau semakin mendekati 1 nilai idle yang paling memberikan nilai penalti terbaik.

4.5.4 Skenario 4: *History Length*

Skenario ketiga adalah mengubah nilai *history length* yang dilakukan untuk 11 kali *running* yang dilakukan. Uji coba ini dilakukan untuk *dataset* yang berhasil memenuhi ketentuan solusi awal. Tujuan dilakukan uji coba skenario 4 adalah menyimpan nilai solusi-solusi terbaik yang dihasilkan dari iterasi yang dilakukan. Tempat penyimpanan ini yang berguna sebagai pembanding dari hasil solusi yang didapatkan. Sehingga tidak ada kemungkinan solusi berhenti pada satu local optimum karena solusi yang dibandingkan adalah tidak hanya solusi dengan nilai terbaik akan tetapi dapat menerima solusi yang tidak lebih baik dari iterasi sebelumnya. Tujuan dari seknario ini adalah untuk mendapatkan pembanding yang bervariasi dengan menyimpan solusi-solusi baik yang dihasilkan. Nilai uji parameter panjang yang akan dilakukan uji coba adalah 10, 100 dan 1000. Nilai ini diuji untuk melihat hasil dengan pajang yang paling sedikit atau banyak yang akan menghasilkan nilai penalti lebih baik.

Halam ini sengaja dikosongkan

BAB V
IMPLEMENTASI

Bab ini menjelaskan proses pelaksanaan penelitian tugas akhir dan proses implementasi algoritma *Late acceptance hill climbing Hyper-Heuristic* terhadap ITC 2019 *dataset* dengan menggunakan IDE Netbeans 8.0.2 dan bahasa pemrogaman Java.

5.1. Pembacaan *File* Masukan

Subbab ini menjelaskan mengenai proses awal dari membaca *file* yang menjadi masukan untuk program. Adapun *file* masubkan yang berupa *dataset* pada kasus ini memiliki ekstensi xml. Sehingga *file* terlebih dahulu dapat dibaca untuk kemudian disimpan masing-masing nilainya. Terdapat banyak informasi yang harus disimpan sehingga dalam menyimpan masukan harus memiliki struktur data yang baik untuk memudahkan saat akan membaca dan mengambil informasi tersebut. Pada subbab berikutnya akan dijelaskan struktur data untuk menyimpan masing-masing konten.

5.1.1. Pembacaan Awal Data

Jenis *file* masukan yang dibaca memiliki ekstensi XML sehingga perlu penyesuain dalam pembacaan menggunakan *java*. Dalam hal ini digunakan *Java Document Model Object Parser* untuk dapat membaca *file xml* tersebut. Pada Kode 5.1 merupakan kode yang berfungsi untuk membaca *file* dengan menyimpan elemen-elemen dari *tag xml* yang terdiri dari *root*

element dan *child element*. Kemudian untuk elemen-elemen selanjutnya dapat dipanggil dan disimpan dengan mudah.

```
  public Read File(String filename) throws
  Parser Configuration Exception,
        SAXException, IOException {
File fXml File = new File("src/Test/" + filename);
  DocumentBuilderFactory dbFactory =
  DocumentBuilderFactory.newInstance();
  DocumentBuilder dBuilder =
  dbFactory.newDocumentBuilder();
  Document doc = dBuilder.parse(fXml File);
  doc.getDocumentElement().normalize();

  Element rootElement = doc.getDocumentElement();
  Element problemChild = (Element)
  rootElement.getChildNodes();

  nama File = rootElement.getAttribute("name");
```

Kode 5.1 Pembacaan Awal Masukan Data

5.1.2. Penyimpanan Konten *Rooms* dan Atributnya

Konten *Rooms* memiliki beberapa nilai yang disimpan. Nilai yang harus disimpan tersebut adalah kode unik (*idRoom*) berupa *integer*, kapasitas ruangan (*room*cap) berupa *integer*, waktu yang tidak tersedia (*unRoom*) dalam bentuk array 4 dimensi dan waktu perjalanan antar ruang (*travelroom*) dalam bentuk *arraylist*. Semua nilai disimpan didalam sebuah kelas berbentuk objek dengan kelas bernama *Room*. Penyimpanan model seperti ini akan dapat membuat pemanggilan dan penggunaan model akan lebih mudah. Kode penyimpanan konten ini dapat dilihat pada Kode 5.2 berikut.

```
public class Room {
private int idRoom;
private int roomcap;
private ArrayList <Integer> travelroom = new ArrayList<>();
private int[][][][] unRoom;

Room(int idRoom, int roomcap) {
this.idRoom = idRoom;
this.roomcap = roomcap;
}
```

Kode 5.2 Penyimpanan Konten *Rooms* dan Atributnya

5.1.3. Penyimpanan Konten *Courses* dan Atributnya

Konten *Courses* memiliki hirarki yang sangat kompleks. Terdapat beberapa informasi yang harus disimpan. Yaitu kode unik (*courseID*) untuk nilai *course* dan hirearki yang dimiliki oleh *course* yang terdiri dari *config* (*configID*), *subpart* (*subpartID*) dan *class* (*classID*) semua kode unik tersebut disimpan dalam bentuk *integer*. Informasi ini disimpan dalam kelas-kelas baru berbentuk objek. Kemudian elemen yang menyimpan informasi lebih banyak adalah kelas, keterkaitan antar kelas (*parent*) dalam bentuk *integer*, kuota kelas (*limit*) dalam bentuk *integer*, kebutuhan ruangan (*hasRoom*) dalam bentuk *boolean* yang menyatakan benar atau salah, pilihan ruangan yang dapat digunakan beserta nilai penal*tiny*a (*availableroom*) berupa *arraylist* yang diambil dari kelas lain dan pilihan slot jadwal yang dapat digunakan beserta nilai penal*tiny*a (*availableTS*) dalam bentuk *arraylist* disimpan dalam kelas baru. Selain itu kelas juga menyimpan daftar batasan baik *hard* (*class*HC) maupun *soft* (*class*SC) yang dimiliki masing-masing kelas. Keduanya disimpan dalam

bentuk *arraylist* yang diambil dari kelas lain. Semua informasi yang dimiliki disimpan dalam bentuk objek dengan membuat kelas. Semua konten kelas beserta atributnya disimpan dalam sebuah kelas yaitu *class* Kelas. Berikut Kode 5.3 menunjukkan isi dari kode kelas.

```
public class Kelas {
private int courseID;
   private int configID;
   private int subpartID;
   private int classID;
   private int limit;
   private int parent;
   private int penalty;
   private boolean hasRoom;
private Arraylist<RoomAss> availableroom = new
Arraylist<>();
private Arraylist<TimeAss> availableTS = new
Arraylist<>();
private Arraylist<Constraint> clasHC = new Arraylist<>();
private Arraylist<Constraint> classC = new Arraylist<>();
public Kelas (int classID, int parent, int limit, boolean
hasRoom) {
this.classID = classID;
      this.parent = parent;
      this.limit = limit;
      this.hasRoom = hasRoom;
   }
```

Kode 5.3 Penyimpanan Konten *Course* dan Atributnya

5.1.4. Penyimpanan Konten *Students* dan Atributnya

Konten *student* memiliki informasi yang disimpan juga dalam bentuk objek. Informasi tersebut adalah kode unik (*studentID*) *student* dan daftar *courses* apa saja yang harus diambil oleh

masing-masing *student* dalam bentuk *arraylist student course*. Semua konten kelas beserta atributnya disimpan dalam sebuah kelas baru yaitu kelas *Student*. Kode 5.4 berikut adalah bentuk kode penyimpanannya.

```
public class Students {

private int studentID;
private Arraylist<Integer> stud;

Students(int cID, Arraylist<Integer> studentcourse) {

this.studentID = studentID;
this.studentcourse = studentcourse;
}
```

Kode 5.4 Penyimpanan Konten *Student* dan Atributnya

5.1.5. Penyimpanan Konten Batasan Distributions dan Atributnya

Konten batasan menyimpan informasi berupa metode untuk 19 batasan yang berlaku pada domain permasalahan ITC 2019. Batasan-batasan tersebut dapat berlaku sebagai *hard constraint* maupun *soft constraint*. Masing-masing batasan didefinisikan dalam bentuk metode sesuai dengan ketentuan. Dimana batasan yang bersifat *hard* memiliki tipe metode *boolean*. Sehingga harus dapat terpenuhi metode tersebut karena apabila tidak terpenuhi, pengembalian nilainya adalah benar atau salah. Sehingga tidak boleh ada metode yang menghasilkan pengembalian nilai salah untuk setiap batasan yang berlaku pada sebuah *dataset*. Sedangkan pengembalian nilai untuk metode dengan tipe *integer* adalah nilai penalti. Apabila tidak terpenuhi maka terhitung mendapatkan nilai

penalti. Seluruh metode disimpan dalam satu kelas yang sama. Berikut konversi masing-masing metode kedalam bentuk kode.

5.2.5.1. Distribution Constraint 1 *(Same Start):* Setiap kelas yang masuk dalam daftar batasan ini harus dimulai pada waktu yang sama. Pernyataan ini terdapat pada Kode 5.5 sebagai berikut untuk batasan yang bersifat *hard*. Metode ini memiliki parameter waktu yang telah terjadwal, objek kelas dan daftar kelas yang berada dalam satu batasan tersebut. Kemudian masing-masing daftar kelas yang sudah terjadwal pada suatu batasan akan dibandingkan apakah memiliki waktu mulai sama antara satu dengan yang lainnya. Jika benar makan pengembalian nilainya adalah benar, jika tidak sama maka pengembalian nilainya adalah salah.

```
boolean SameStart(int TS, Kelas kls, Arraylist<Integer>
listSS) {
      int startA = kls.getAvailableTS().get(TS).getStart();
boolean  smStart=false;
      for (int i = 0; i < listSS.size(); i++) {
Kelas klsB = SearchKls(listSS.get(i));
         int iTS = klsB.getTs();
         if (iTS != -1) {
         int startB = klsB.getAvailableTS().get(iTS).getStart();
            if (startA == startB) {
smStart=true;
```

Kode 5.5 Metode *Hard constraint* Same Start

Sedangkan Kode 5.6 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
int SameTime( Arraylist<Integer> listST, int penalty) {
int startA, lengthA, endA;
      int startB, lengthB, endB;
      int TSA, TSB;
      int jPenalty = 0;
if (listST.size() == 1) {
         return jPenalty;
      }
for (int i = 0; i < listST.size(); i++) {
Kelas klsA = SearchKls(listST.get(i));
TSA = klsA.getTs();
startA = klsA.getAvailableTS().get(TSA).getStart();
lengthA = klsA.getAvailableTS().getLength();
endA = startA + lengthA;
for (int j = i + 1; j < listST.size(); j++) {
Kelas klsB = SearchKls(listST.get(j));
TSB = klsB.getTs();
startB = klsB.getAvailableTS().get(TSB).getStart();
 lengthB = klsB.getAvailableTS().get(TSB).getLength();
endB = startB + lengthB;
if ((startA <= startB && endB <= endA)
|| (startB <= startA && endA <= endB)) {
jPenalty += 0;
```

Kode 5.6 Metode *Soft Constraint* Same Start

5.2.5.2. Distribution Constraint 2 *(Same Time):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dalam rentang waktu yang sama. Pernyataan ini terdapat pada Kode 5.7 sebagai berikut. Metode ini memiliki parameter waktu yang telah terjadwal, objek kelas dan daftar kelas yang berada dalam satu batasan tersebut. Kemudian masing-masing daftar kelas yang sudah terjadwal pada suatu batasan akan

dibandingkan apakah memiliki durasi waktu yang sama antara satu dengan yang lainnya. Jika benar makan pengembalian nilainya adalah benar, jika tidak sama maka pengembalian nilainya adalah salah.

```
boolean SameTime(int TS, Kelas kls, Arraylist<Integer>
listST) {
int startA = kls.getAvailableTS().get(TS).getStart();
int lengthA = kls.getAvailableTS().get(TS).getLength();
int endA = startA + lengthA;
boolean ST = false;
for (int i = 0; i < listST.size(); i++) {
Kelas klsB = SearchKls(listST.get(i));
int iTS = klsB.getTs();
if (iTS != -1) {
int startB = klsB.getAvailableTS().get(iTS).getStart();
int lengthB = klsB.getAvailableTS().get(iTS).getLength();
int endB = startB + lengthB;
if ((startA <= startB && endB <= endA)
|| (startB <= startA && endA <= endB)) {
ST = true;
} else {
                return false;
            }
```

Kode 5.7 Metode *Hard constraint* Same Time

Sedangkan Kode 5.8 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti. Penyimpanan nilai pengembalian yang berbeda-beda pada setiap batasan yang bersifat wajib dan tidak wajib ini dihasilkan dari nilai penalti yang ada pada masing-masing daftar batasan yang berlaku. Perbedaan ini yang nantinya akan memberikan pengaruh pada nilai hasil solusi akhir hasil.

```
int SameTime( Arraylist<Integer> listST, int penalty) {
int startA, lengthA, endA;
      int startB, lengthB, endB;
      int TSA, TSB;
      int jPenalty = 0;
if (listST.size() == 1) {
         return jPenalty;
      }
for (int i = 0; i < listST.size(); i++) {
Kelas klsA = SearchKls(listST.get(i));
TSA = klsA.getTs();
startA = klsA.getAvailableTS().get(TSA).getStart();
lengthA = klsA.getAvailableTS().getLength();
endA = startA + lengthA;
for (int j = i + 1; j < listST.size(); j++) {
Kelas klsB = SearchKls(listST.get(j));
TSB = klsB.getTs();
startB = klsB.getAvailableTS().get(TSB).getStart();
lengthB = klsB.getAvailableTS().get(TSB).getLength();
endB = startB + lengthB;
if ((startA <= startB && endB <= endA)
|| (startB <= startA && endA <= endB)) {
jPenalty += 0;
} else {
               jPenalty += penalty;
            }
```

Kode 5.8 Metode *Soft Constraint* Same Time

5.2.5.3 Distribution Constraint 3 *(Different Time):* Setiap kelas yang masuk dalam daftar batasan ini tidak boleh berlangsung dalam rentang waktu yang sama. Pernyataan ini terdapat pada Kode 5.9 sebagai berikut. Metode ini memiliki parameter waktu yang telah terjadwal, objek kelas dan daftar

kelas yang berada dalam satu batasan tersebut. Kemudian masing-masing daftar kelas yang sudah terjadwal pada suatu batasan akan dibandingkan apakah memiliki durasi waktu yang berbeda antara satu dengan yang lainnya. Jika benar makan pengembalian nilainya adalah benar, jika tidak sama maka pengembalian nilainya adalah salah.

```
boolean DifferentTime(int TS, Kelas kls, Arraylist<Integer>
listDT) {
int startA = kls.getAvailableTS().get(TS).getStart();
int lengthA = kls.getAvailableTS().get(TS).getLength();
 int endA = startA + lengthA;
boolean difTime=false;
for (int i = 0; i < listDT.size(); i++) {
Kelas klsB = SearchKls(listDT.get(i));
int iTS = klsB.getTs();
if (iTS != -1) {
int startB = klsB.getAvailableTS().get(iTS).getStart();
int lengthB = klsB.getAvailableTS().get(iTS).getLength();
int endB = startB + lengthB;
if (endA <= startB || endB <= startA) {
 difTime = true;
else {
                return false;
}
```

Kode 5.9 Metode *Hard constraint* Different Time

Sedangkan Kode 5.10 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti. Penyimpanan nilai pengembalian yang berbeda-beda pada setiap batasan yang bersifat wajib dan tidak wajib ini dihasilkan dari nilai penalti yang ada pada masing-masing

daftar batasan yang berlaku. Perbedaan ini yang nantinya akan memberikan pengaruh pada nilai hasil solusi akhir hasil.

```
int DifferentTime(Arraylist<Integer> listDT, int penalty) {
int startA, startB;
int lengthA, lengthB;
int endA, endB;
int TSA, TSB;
int jPenalty = 0;
if (listDT.size()==1){
        return jPenalty;
    }
for (int i = 0; i < listDT.size(); i++) {
Kelas klsA = SearchKls(listDT.get(i));
TSA = klsA.getTs();
startA = klsA.getAvailableTS().get(TSA).getStart();
lengthA = klsA.getAvailableTS().get(TSA).getLength();
endA = startA + lengthA;
for (int j = i + 1; j < listDT.size(); j++) {
Kelas klsB = SearchKls(listDT.get(j));
TSB = klsB.getTs();
startB = klsB.getAvailableTS().get(TSB).getStart();
lengthB = klsB.getAvailableTS().get(TSB).getLength();
endB = startB + lengthB;
if (endA <= startB || endB <= startA) {
jPenalty += 0;
} else {
jPenalty += penalty;
}
```

Kode 5.10 Metode *Soft Constraint* Different Time

5.2.5.4 Distribution Constraint 4 *(Same Days):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan dalam hari yang sama. Pernyataan ini terdapat

pada Kode 5.11 sebagai berikut. Metode ini memiliki parameter waktu yang telah terjadwal, objek kelas dan daftar kelas yang berada dalam satu batasan tersebut. Kemudian masing-masing daftar kelas yang sudah terjadwal pada suatu batasan akan dibandingkan apakah memiliki hari yang sama antara satu dengan yang lainnya. Jika benar makan pengembalian nilainya adalah benar, jika tidak sama maka pengembalian nilainya adalah salah.

```
boolean SameDays(int TS, Kelas kls, Arraylist<Integer> listSD) {
Arraylist<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
boolean SD = false;
for (int i = 0; i < listSD.size(); i++) {
Kelas klsB = SearchKls(listSD.get(i));
int iTS = klsB.getTs();
if (iTS != -1) {
Arraylist<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();
for (int j = 0; j < daysA.size(); j++) {
if (daysA.get(j) == 1 && daysB.get(j) == 1) {
SD = true;

if (SD==false) {
        return false;
    }
```

Kode 5.11 Metode *Hard constraint* Same Days

Sedangkan Kode 5.12 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
int SameDays( Arraylist<Integer> listSD, int penalty) {
int TSA, TSB;
      int jPenalty = 0;
      if (listSD.size()==1){
         return jPenalty;
      }
for (int i = 0; i < listSD.size(); i++) {
Kelas klsA = SearchKls(listSD.get(i));
TSA = klsA.getTs();
Arraylist<Integer> daysA =
klsA.getAvailableTS().get(TSA).getDays();
for (int j = i+1; j < listSD.size(); j++) {
Kelas klsB = SearchKls(listSD.get(i));
TSB = klsB.getTs();
Arraylist<Integer> daysB =
klsB.getAvailableTS().get(TSB).getDays();
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
                  jPenalty += 0;
                  break;
```

Kode 5.12 Metode *Soft Constraint* Same Days

5.2.5.5 Distribution Constraint 5 *(Different Days):* Setiap kelas yang masuk dalam daftar batasan ini tidak dapat berlangsung dan dijadwalkan dalam hari yang sama. Pernyataan ini terdapat pada Kode 4.2 sebagai berikut. Metode tersebut menjelaskan bahwasannya kelas-kelas yang sudah memiliki jadwal yang berada dalam daftar batasan ini harus memiliki hari yang berbeda, yakni tidak diperbolehkan memiliki jenis biner yang sama (0 dan 1). Pengembalian nilai

jika metode benar adalah benar dan pengembalian nilai salah jika tidak dapat memenuhi.

```
Arraylist<Integer> daysB =
klsB.getAvailableTS().get(iTS).getDays();
for (int j = 0; j < daysA.size(); j++) {
if ((daysA.get(j) == 0 && daysB.get(j) == 0)
|| (daysA.get(j) == 1 && daysB.get(j) == 0)
|| (daysA.get(j) == 0 && daysB.get(j) == 1)) {
DD = true;
}
```

Kode 5.13 Metode *Hard constraint* Different Days

Sedangkan Kode 5.14 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listDD.size(); i++) {
for (int j = i+1; j < listDD.size(); j++) {
for (int dy = 0; dy < daysA.size(); dy++) {
if ((daysA.get(j) == 0 && daysB.get(j) == 0)
|| (daysA.get(j) == 1 && daysB.get(j) == 0)
|| (daysA.get(j) == 0 && daysB.get(j) == 1)) {
jPenalty =0;
```

Kode 5.14 Metode *Soft Constraint* Different Days

5.2.5.6 Distribution Constraint 6 *(Same Weeks):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan dalam hari yang sama. Pernyataan ini terdapat pada Kode 5.15 sebagai berikut. Metode ini menunjukkan perulangan pada minggu dari jadwal kelas yang sudah terjadwal, lalu mengecek apakah antara minggu perulangan kelas satu dengan lainnya memiliki jenis biner yang sama. Dalam hal ini biner harus berupa angka satu yang

menunjukkan kelas terjadwal pada minggu tersebut. Jika metode terpenuhi maka pengembalian nilainya adalah benar, sebaliknya jika salah maka pengembaliannya salah.

```
for (int i = 0; i < listSW.size(); i++) {
for (int k = 0; k < weekA.size(); k++) {
if (weekA.get(k) == 1 && weekB.get(k) == 1) {
W = true;
break;
} else{
  SW = false;}
if (SW==false) {
                return false;
}
```

Kode 5.15 Metode *Hard constraint* Same Weeks

Sedangkan Kode 5.16 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listSW.size(); i++) {
for (int wk = 0; wk < weeksA.size(); wk++){
Arraylist<Integer> weeksB =
klsB.getAvailableTS().get(TSB).getWeeks();
if(weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
benar++;
break;
}else{
  jPenalty += penalty;
    }
 }
```

Kode 5.16 Metode *Soft Constraint* Same Weeks

***5.2.5.7 Distribution Constraint 7** (Different Weeks): Setiap* kelas yang masuk dalam daftar batasan ini tidak dapat

berlangsung dan dijadwalkan dalam minggu yang sama. Pernyataan ini terdapat pada Kode 5.17 sebagai berikut. Metode ini menunjukkan perulangan pada minggu dari jadwal kelas yang sudah terjadwal, lalu mengecek apakah antara minggu perulangan kelas satu dengan lainnya memiliki jenis biner yang sama. Dalam hal ini biner harus berupa angka satu dan nol yang menunjukkan kelas terjadwal pada minggu tersebut atau tidak. Jika metode terpenuhi maka pengembalian nilainya adalah benar, sebaliknya jika salah maka pengembaliannya salah.

```
for (int k = 0; k < weekA.size(); k++) {
if (weekA.get(k) == 0 && weekB.get(k) == 0
|| weekA.get(k) == 1 && weekB.get(k) == 0
|| weekA.get(k) == 0 && weekB.get(k) == 1) {
DW = true;
} else{
```

Kode 5.17 Metode *Hard constraint* Different Weeks

Sedangkan Kode 5.18 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listDW.size(); i++) {
for (int k = 0; k < weeksA.size(); k++) {
if (weeksA.get(k) == 0 && weeksB.get(k) == 0
|| weeksA.get(k) == 1 && weeksB.get(k) == 0
|| weeksA.get(k) == 0 && weeksB.get(k) == 1) {
benar++;
```

Kode 5.18 Metode *Soft Constraint* Different Weeks

5.2.5.8 Distribution Constraint 8 *(Same Rooms):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan dalam ruangan yang sama. Pernyataan ini

terdapat pada persamaan (2) sebagai berikut. Metode ini berfungsi untuk mengecek apakah kelas yang sudah terjadwal memiliki id ruangan yang sama antara daftar kelas yang terdapat didalam batasan tersebut, jika ruangannya sama maka pengembaliannya adalah benar, jika tidak sama maka salah.

```
for (int i = 0; i < listSR.size(); i++) {
if (idKLSA != idKLSB) {
int iRM = klsB.get Room();
            if (iRM != -1) {
if (roomA == roomB) {
SR = true;
 } else {
return false;
}
```

Kode 5.19 Metode *Hard constraint* Same *Room*

Kode 5.20 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listSR.size(); i++) {
for (int j = i+1; j < listSR.size(); j++) {
if (roomA == roomB) {
jPenalty +=0;
} else {
  jPenalty += penalty;
}
```

Kode 5.20 Metode *Soft Constraint* Same *Room*

5.2.5.9 Distribution Constraint 9 *(Different Rooms):* Setiap kelas yang masuk dalam daftar batasan ini tidak berlangsung dan dijadwalkan dalam ruangan yang sama. Pernyataan ini terdapat pada Kode 5.21 sebagai berikut. Metode ini berfungsi untuk mengecek apakah kelas yang sudah terjadwal memiliki

id ruangan yang sama antara daftar kelas yang terdapat didalam batasan tersebut, jika ruangannya sama maka pengembaliannya adalah salah, jika tidak sama maka benar.

```
for (int i = 0; i < listDR.size(); i++) {
if (roomA != roomB) {
                DR = true;
            } else {
                return false;
            }
```

Kode 5.21 Metode *Hard constraint* Different *Rooms*

Kode 5.20 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listDR.size(); i++) {
for (int j = i+1; j < listDR.size(); j++) {
if (roomA != roomB) {
                    jPenalty +=0;
                } else {
                    jPenalty += penalty;
            }
```

Kode 5.22 Metode *Soft Constraint* Different *Rooms*

5.2.5.10 Distribution Constraint 10 *(Overlap):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan saling tumpang tindih pada durasi yang berlangsung, hari dan minggu. Pernyataan ini terdapat pada Kode 5.23 sebagai berikut. Metode ini berfungsi melakukan perulangan pada daftar kelas yang telah terjadwal yang berada dalam daftar kelas batasan ini, lalu mengecek apakah waktu mulai dan berakhir antara kelas tersebut saling tumpang tindih dan kemudian mengecek dengan melakukan perulangan pada minggu dan hari dari jadwal perulangan tersebut.

```
for (int i = 0; i < listOv.size(); i++) {
if((startB < endA) && (startA < endB)){
for (int wk = 0; wk < weeksA.size(); wk++) {
if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) ==1) {
over = true;
break outerloop;
} else {
over = false;
}
```

Kode 5.23 Metode *Hard constraint* Overlap

Kode 5.24 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listOv.size(); i++) {
for (int j = i + 1; j < listOv.size(); j++) {
for (int wk = 0; wk < weeksA.size(); wk++) {
if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
jPenalty += 0;
break outerloop;
} else {
daySalah += 1;
if (daySalah == daysA.size()) {
jPenalty += penalty;
}
```

Kode 5.24 Metode *Soft Constraint* Overlap

5.2.5.11 Distribution Constraint 11 *(Not Overlap):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan tidak diperbolehkan saling tumpah tindih untuk

durasi, hari dan minggu yang berlangsung. Pernyataan ini terdapat pada Kode 5.25 sebagai berikut. Metode ini berfungsi melakukan perulangan pada daftar kelas yang telah terjadwal yang berada dalam daftar kelas batasan ini, lalu mengecek apakah waktu mulai dan berakhir antara kelas tersebut saling tumpang tindih dan kemudian mengecek dengan melakukan perulangan pada minggu dan hari dari jadwal perulangan tersebut. Jika ternyata kelas tersebut dijadwalkan saling tumpang tindih maka pernyataan tidak dapat terpenuhi, sebaliknya jika jadwal tidak saling tumpang tindih maka pernyataan benar.

```
for (int i = 0; i < listNotOv.size(); i++) {
for (int wk = 0; wk < weeksA.size(); wk++) {
if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) == 1)
{
if ((endA <= startB) || (endB <= startA)){
NO = true;
} else {
return false;
}
```

Kode 5.25 Metode *Hard constraint* Not Overlap

Kode 5.26 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti. Penyimpanan nilai pengembalian yang berbeda-beda pada setiap batasan yang bersifat wajib dan tidak wajib ini dihasilkan dari nilai penalti yang ada pada masing-masing daftar batasan yang berlaku. Perbedaan ini yang nantinya akan memberikan pengaruh pada nilai hasil solusi akhir hasil.

```
for (int i = 0; i < listNotOv.size(); i++) {
for (int j = i + 1; j < listNotOv.size(); j++) {
for (int wk = 0; wk < weeksA.size(); wk++) {
if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
if ((endA <= startB) || (endB <= startA)) {
jPenalty += 0;
} else {
jPenalty += penalty;
break outerloop;
}
```

Kode 5.26 Metode *Soft Constraint* Not Overlap

5.2.5.12 Distribution Constraint 12 *(Same Attendees):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan berdasarkan kriteria dengan mempertimbangkan waktu *travel* antar kelas sehingga tidak terjadi konflik bagi mereka yang mengambil dan mengajar kelas tersebut. Pernyataan ini terdapat pada Kode 5.27 sebagai berikut. Metode ini terlebih dahulu mengecek apakah *id* kelas yang akan dicek dalam daftar kelas batasan sama atau tidak, kemudian memastikan apakah kelas tersebut membutuhkan ruangan untuk jadwal yang ditentukan. Kemudian mempertimbangkan jarak tempuh yang diperlukan antara satu kelas dengan kelas lainnya. Dan kemudian dilakukan perulangan untuk mengecek apakah pasangan antar kelas tersebut dijadwalkan dalam waktu yang sama sehingga bagi para mahasiswa dan guru yang mengajar dalam kelas-kelas yang ada dalam daftar dapat mengestimasikan waktu yang dihabiskan. Jika pernyataan tersebut benar, maka pengembalian nilainya adalah benar, jika tidak maka salah.

```
for (int i = 0; i < listSA.size(); i++) {
if (idKLSA != idKLSB){
if (klsB.isHas<i>Room</i>() == false) {
                iRM = 999;
            }
if (iRM == 999) {
no<i>Room</i>B = true;
} else {
<i>room</i>B                                                    =
klsB.getAvailable<i>room</i>().get(iRM).get<i>Room</i>_id()-1;
}
if (no<i>Room</i>A == false && no<i>Room</i>B == false) {
trvl = value<i>Travel</i>[<i>room</i>A][<i>room</i>B];
}
for (int wk = 0; wk < weeksA.size(); wk++) {
if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1){
for (int dy = 0; dy < daysA.size(); dy++) {
if(daysA.get(dy) == 1 && daysB.get(dy) == 1){
if ((endA + trvl) <= startB || (endB + trvl) <= startA){
SA = true;
break outerloop;
} else {
return false;
}
} else {
SA =true;
}
```

Kode 5.27 Metode *Hard constraint* Same Attendees

Kode 5.26 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti. Penyimpanan nilai pengembalian yang berbeda-beda pada setiap batasan yang bersifat wajib dan tidak wajib ini dihasilkan dari nilai

penalti yang ada pada masing-masing daftar batasan yang berlaku. Perbedaan ini yang nantinya akan memberikan pengaruh pada nilai hasil solusi akhir hasil.

```
for (int i = 0; i < listSA.size(); i++) {
for (int j = i + 1; j < listSA.size(); j++) {
if (RMB < 0) {
noRoomB = true;
} else {
roomB =
klsB.getAvailableroom().get(RMB).getRoom_id() - 1;
}
if (noRoomA == false && noRoomB == false) {
trvl = valueTravel[roomA][roomB];
}
for (int wk = 0; wk < weeksA.size(); wk++) {
if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
if ((endA + trvl) <= startB || (endB + trvl) <= startA) {
jPenalty += 0;
break outerloop;
} else {
jPenalty += penalty;
}
```

Kode 5.28 Metode *Soft Constraint* Same Attendees

5.2.5.13 Distribution Constraint 13 *(Precendece):* Setiap kelas yang masuk dalam daftar batasan ini berlangsung dan dijadwalkan diawal waktu untuk slot, hari dan minggu dari jadwal yang tersedia. Pernyataan ini terdapat pada Kode 5.29 sebagai berikut. Metode ini melakukan pengecekan apakah kelas yang terjadwal pada waktu-waktu awal. Pertama

dilakukan perulangan pada minggu apakah minggu kelas pertama lebih awal dibandingkan kelas berikutnya. Kemudian mengecek dengan perulangan hari dan diikuti dengan waktu mulai kelas kedua lebih kecil atau sama dengan waktu mulai kelas pertama. Jika benar maka pengembaliannya adalah benar, jika tidak maka salah.

```
for (int i = 0; i < listP.size(); i++) {
if(i<iKlsA){
for (int wk = 0; wk < weeksA.size(); wk++) {
if(weeksA.get(wk) == 1 && weeksB.get(wk) ==1){
for (int dy = 0; dy < daysA.size(); dy++) {
if(daysA.get(dy) == 1 && daysB.get(dy)==1){
if (endB <= startA) {
P = true;
break outerloop;
} else {
return false;}
} else if(daysA.get(dy) < daysB.get(dy)){
P = true;
break outerloop;
} else if(daysA.get(dy) > daysB.get(dy)){
return false;
}
} else if(weeksA.get(wk) < weeksB.get(wk)){
  P = true;
break;
 } else if(weeksA.get(wk) > weeksB.get(wk)){
return false;}
```

Kode 5.29 Metode *Hard constraint* Precedence

Kode 5.26 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listP.size(); i++) {
for (int j = i+1; j < listP.size(); j++) {
for (int wk = 0; wk < weeksA.size(); wk++) {
if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
if (endB <= startA) {
benar++;
break outerloop;
} else {
jPenalty += penalty;
}
} else if (daysA.get(dy) < daysB.get(dy)) {
benar++;
break outerloop;
} else if (daysA.get(dy) > daysB.get(dy)) {
jPenalty += penalty;}}}
else if (weeksA.get(wk) < weeksB.get(wk)) {
benar++;
break;
} else if (weeksA.get(wk) > weeksB.get(wk)) {
jPenalty += penalty;
}
```

Kode 5.30 Metode *Soft Constraint* Precedence

5.2.5.14 Distribution Constraint 14 *(Work Days (S)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak boleh melebihi slot yang telah ditentukan. Pernyataan ini terdapat pada Kode 5.31 sebagai berikut. Metode ini melakukan perulangan pada minggu jadwal milik kelas untuk mengecek pada minggu berapa kelas dijadwalkan, lalu kemudian mengecek hari apa yang terjadwal dan kemudian menghitung jumlah slot waktu yang digunakan atau terjadwal.

Jika waktu terjadwal tersebut melebihi batas slot yang telah ditentukan maka pengembalian nilai bernilai salah, jika tidak maka benar.

```
for (int i = 0; i < listWD.size(); i++) {
for (int k = 0; k < weeksA.size(); k++) {
if(weeksA.get(k) == 1 && weeksB.get(k) == 1){
for (int l = 0; l < daysA.size(); l++) {
if((daysA.get(l) == 1 && daysB.get(l) == 1)){
if ((Math.max(endA, endB)) - (Math.min(startA,
startB)) <= S) {
WD = true;
break outerloop;
```

Kode 5.31 Metode *Hard constraint* Work Days

Kode 5.32 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listWD.size(); i++) {
for (int j = i+1; j < listWD.size(); j++) {
for (int wk = 0; wk < weeksA.size(); wk++) {
if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
for (int dy = 0; dy < daysA.size(); dy++) {
if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
if ((Math.max(endA, endB)) - (Math.min(startA,
startB)) <= S) {
benar++;
break outerloop;
} else {
jPenalty += penalty;}}
```

Kode 5.32 Metode *Soft Constraint* Work Days

5.2.5.15 Distribution Constraint 15 *(Min gap (G)*): Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan pada

hari tertentu harus memiliki selisih slot tertentu antara satu kelas dengan lainnya. Pernyataan ini terdapat pada Kode 5.35) sebagai berikut. Metode ini melakukan pengecekan terhadap jumlah slot yang dihabiskan untuk selisih jeda antara suatu kelas dengan kelas lainnya. Pertama dilakukan perulangan minggu kemudian perulangan hari dan kemudian dicek apakah waktu berakhir kelas pertama ditambah sejumlah minimal slot lebih kecil bila dibandingkan dengan waktu mulai kelas berikutnya. Jika pernyataan terpenuhi, maka pengembalian nilai benar begitu juga sebaliknya.

```
for (int i = 0; i < listMG.size(); i++) {
for (int k = 0; k < weeksA.size(); k++) {
if(weeksA.get(k) == 1 && weeksB.get(k) == 1){
for (int l = 0; l < daysA.size(); l++) {
if(daysA.get(l) == 1 && daysB.get(l) == 1){
if ((endA + G <= startB) || (endB + G <= startA)) {
MG = true;
break outerloop;
} else {
return false;
}
```

Kode 5.33 Metode *Hard constraint Min gap*

Kode 5.34 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti. Penyimpanan nilai pengembalian yang berbeda-beda pada setiap batasan yang bersifat wajib dan tidak wajib ini dihasilkan dari nilai penalti yang ada pada masing-masing daftar batasan yang berlaku. Perbedaan ini yang nantinya akan memberikan pengaruh pada nilai hasil solusi akhir hasil. Hasil solusi akhir adalah hasil yang didapatkan dari optimasi yang dilakukan.

```
for (int i = 0; i < listMG.size(); i++) {
for (int j = i+1; j < 10; j++) {
outerloop:
for (int k = 0; k < weeksA.size(); k++) {
if (weeksA.get(k) == 1 && weeksB.get(k) == 1) {
for (int l = 0; l < daysA.size(); l++) {
if (daysA.get(l) == 1 && daysB.get(l) == 1) {
if ((endA + G <= startB) || (endB + G <= startA)) {
benar++;
break outerloop;
} else {
jPenalty += penalty;
}
```

Kode 5.34 Metode *Soft Constraint Min gap*

5.2.5.16 Distribution Constraint 16 *(Max Days (D)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak melebihi batas maksimal hari yang telah ditentukan setiap minggunya. Pernyataan ini terdapat pada Kode 5.35 sebagai berikut. Metode ini melakukan pengecekan terhadap jumlah hari yang terjadwal dalam satu minggu. Jumlah tersebut tidak boleh melebihi jumlah maksimal hari yang telah ditentukan.

```
boolean Maxdays(int TS, int D, Kelas kls) {
Arraylist<Integer> daysA =
kls.getAvailableTS().get(TS).getDays();
      if (countNonzeroBits(daysA) <= D) {
         return true;  }
      return false;
   }
```

Kode 5.35 Metode *Hard constraint Max Days*

Kode 5.36 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listMD.size(); i++) {
if (countNonzeroBits(daysA) <= D) {
return jPenalty;
} else {
lebih = (countNonzeroBits(daysA) - D);
jPenalty += (lebih * penalty);
}
```

Kode 5.36 Metode *Soft Constraint Max Days*

Kode 5.37 adalah metode yang digunakan untuk mengecek nilai daftar hari yang bernilai biner (0 dan 1).

```
private int countNonzeroBits(Arraylist<Integer> daysA)
{
      int count = 0;
      for (int i = 0; i < daysA.size(); i++) {
         if (daysA.get(i) == 1) {
            count++;
         }
      }
      return count;
   }
```

Kode 5.37 Metode *Count non zero bits*

***5.2.5.17 Distribution Constraint 17** (Max Day Load (S)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak boleh melebihi slot waktu dalam sehari sesuai dengan yang telah ditentukan jumlahnya setiap minggunya. Pernyataan ini terdapat pada Kode 5.38 sebagai berikut. Metode ini melakukan pengecekan terhadap jumlah durasi waktu yang dihabiskan dalam sehari tidak boleh melebihi sejumlah S slot yang telah ditentukan. Slot yang ditentukan telah didefinisikan pada masing-masing batasan.

```
if (Dayload(S, length, days, weeks, listMDL) == 0) {
        return true;}
return false;}
```

Kode 5.38 Metode *Hard constraint Max Day Load*

Kode 5.39 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
for (int i = 0; i < listMDL.size(); i++) {
if (Dayload(S, length, daysA, weeksA) == 0) {
return jPenalty;
} else {
jPenalty = (penalty * (Dayload(S, length, daysA,
weeksA)) / weeksA.size());
}
```

Kode 5.39 Metode *Soft Constraint Max Day Load*

Kode 5.40 adalah metode yang digunakan untuk mengecek jumlah slot yang berlangsung untuk durasi tertentu.

```
private int Dayload(int S, int length, Arraylist<Integer>
days,
Arraylist<Integer> weeks) {
int lebih = 0;
for (int wk = 0; wk < weeks.size(); wk++) {
for (int dy = 0; dy < days.size(); dy++) {
if (weeks.get(wk) == 1 && days.get(dy) == 1) {
if (length > S) {
lebih += (length - S);
}}}}
return lebih;}
```

Kode 5.40 Metode Pengecekan *Dayload*

5.2.5.18 Distribution Constraint 18 *(Max Breaks (R, S)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan

tidak boleh melebihi jumlah jeda antar kelas selama sehari dengan jumlah slot tertentu pula. Pernyataan ini terdapat pada Kode 5.41 sebagai berikut. Metode ini membatasi jumlah jeda yang ada pada daftar kelas dalam batasan.

```
if (MergeBlocks(TS, S, kls, listMB) <= R + 1) {
        return true;
    }
    return false;
  }
```

Kode 5.41 Metode *Hard constraint Merge Blocks*

Kode 5.42 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
int Maxbreaks( int R, int S,
        Arraylist<Integer> listMB, int penalty) {
    int jPenalty = 0;
    if (MergeBlocks( S, listMB) <= (R + 1)) {
        return jPenalty;
    } else {
        jPenalty = (penalty * (MergeBlocks( S, listMB) -
R)) / weeksA.size();
    }
    return jPenalty;
  }
```

Kode 5.42 Metode *Soft Constraint Merge Blocks*

***5.2.5.19 Distribution Constraint 19** (Max Block (M, S)):* Setiap kelas yang masuk dalam daftar batasan ini dijadwalkan tidak melebihi batas maksimal hari yang telah ditentukan setiap minggunya. Pernyataan ini terdapat pada Kode 5.43 sebagai berikut. Metode ini mengecek apakah kelas-kelas yang berada dalam daftar batasan tidak melebihi M slot.

```
boolean Maxblock(int TS, int M, int S, Kelas kls,
 Arraylist<Integer> listMBL) {
      if (MergeBlockB(TS, M, S, kls, listMBL) <= M) {
         return true;
      }
      return false;
   }
```

Kode 5.43 Metode *Hard constraint Max Block*

Kode 5.44 adalah metode yang berlaku untuk batasan yang bersifat *soft* dengan pengembalian nilai penalti.

```
int Maxblock( int M, int S,  Arraylist<Integer> listMBL,
int penalty) {
      int jPenalty = 0;
      for (int i = 0; i < listMBL.size(); i++) {
         if (MergeBlockB(M, S,  listMBL) <= M) {
         return jPenalty;
         } else {
            jPenalty = (penalty * MergeBlockB( M, S,
 listMBL)) / weeksA.size();
         }
      }
      return jPenalty;
   }
```

Kode 5.44 Metode *Soft Constraint Merge Blocks*

5.2. Pembentukan Solusi Awal

Pada subbab ini menjelaskan langkah untuk menghasilkan solusi awal yang memenuhi semua ketentuan dan *hard constraint*. Pembentukan solusi awal menggunakan greedy *algorithm* dan backtracking *algorithm*. Keduanya digunakan untuk memastikan agar solusi dapat dibentuk dan dapat

digunakan oleh semua *dataset* yang akan diuji coba. Solusi awal dibentuk dengan membuat kelas berbentuk objek yang terdiri dari metode-metode. Pada subbab selanjutnya akan dijelaskan informasi yang dibutuhkan untuk menghasilkan solusi awal. Kode menunjukkan bentuk kelas *initial solution* yang memiliki atribut-atribut sebagai informasi dan memiliki parameter yang digunakan untuk menghasilkan solusi awal.

```
public final class InitSol {
Arraylist<Integer> weeks, days;
int start, length, penalty_ts,
idRoom, pen_rom;
   int[][][][] unRoom;
   Arraylist<Kelas> listKelas;
   Arraylist<Minggu> timeslot;
   Arraylist<Integer> klsNoTS;
InitSol (Arraylist<Kelas> listKls, Arraylist<Minggu>
timeslt, int[][][][] unR, int[][] travel){
this.listKelas = listKls;
this.timeslot = timeslt;
this.unRoom = unR;
```

Kode 5.45 Kode Kelas *Initial Solution*

5.2.1 Metode Penetapan Jadwal Kelas

Metode pada Kode 5.46 ini adalah metode yang digunakan untuk menjadwalkan kelas dengan dilakukan perulangan minggu untuk memilih minggu, kemudian dilakukan lagi perulangan hari untuk memilih hari dan memilih waktu mulai serta panjang slot pada jadwal. Penetapan jadwal kelas dilakukan menggunakan algoritma *Greedy*. Algoritma ini berarti memberikan jadwal kepada kelas dengan menemukan pilihan pertama yang mampu memenuhi batasan wajib.

```
void assign(int idKls, TimeAss time, RoomAss room){
      weeks = time.getWeeks();
      days = time.getDays();
      start = time.getStart();
      length = time.getLength();
      idRoom = room.getRoom_id();
      for (int wk = 0; wk < weeks.size(); wk++) {
         if (weeks.get(wk)==1) {
            for (int day = 0; day < days.size(); day++) {
               if (days.get(day)==1) {
                  for (int tm = start; tm <= (start+length); tm++) {
timeslot.get(wk).listHari.get(day).timeslot[tm][idRoom-1]        =
idKls;}}
```

Kode 5.46 Metode Penetapan Jadwal Kelas

Kemudian metode selanjutnya adalah Kode 5.47 yang berfungsi menghapus jadwal kelas jika ternyata kelas tidak cocok berada pada jadwal tersebut. Dilakukan penetapan kembali untuk kelas tersebut menjadi -1.

```
void removeKls( TimeAss time, RoomAss room){
      length = time.getLength();
      idRoom = room.getRoom_id();
      for (int wk = 0; wk < weeks.size(); wk++) {
         if (weeks.get(wk)==1) {
            for (int day = 0; day < days.size(); day++) {
               if (days.get(day)==1) {
                  for (int tm = start; tm <=
(start+length);timeslot.get(wk).listHari.get(day).timeslot[tm][id
Room-1]
```

Kode 5.47 Metode Penghapusan Jadwal Kelas

Metode pada Kode 5.48 berfungsi sebagai metode yang dapat melakukan pengecekan terhadap jadwal-jadwal kelas yang

telah ditetapkan. Apakah semua parameter sudah terisi dengan nilai yang tidak sama dengan nol.

```
boolean cekTS(TimeAss time, RoomAss room){
      weeks = time.getWeeks();
      days = time.getDays();
      start = time.getStart();
      length = time.getLength();
      idRoom = room.getRoom_id()
      for (int wk = 0; wk < weeks.size(); wk++) {
         if (weeks.get(wk)==1) {
            for (int day = 0; day < days.size(); day++) {
               if (days.get(day)==1) {
                  for (int tm = start; tm <=
(start+length); tm++) {
                     if
(timeslot.get(wk).listHari.get(day).timeslot[tm]
[idRoom-1]!=0) {
                        return false;
                     }
                  }
```

Kode 5.48 Metode Pengecekan Jadwal Kelas

Metode pada Kode 5.49 berfungsi untuk melakukan pengecekan apakah ketika menjadwalkan kelas pada *timeslot* tertentu dan ruang tertentu, ruangan yang terpilih tersebut tidak tersedia untuk digunakan pada waktu-waktu tertentu. Model dari metode disimpan dalam sebuah kelas objek yang dapat mempermudah jika akan melakukan pemanggilan. Selain itu metode ini sebelumnya sudah didefinisikan pada kelas objek ruangan. Ruangan memiliki waktu-waktu tertentu dimana tidak dapat digunakan atau dijadwalkan.

```
boolean cekUnvroom(TimeAss time, RoomAss room) {
      weeks = time.getWeeks();
      days = time.getDays();
      start = time.getStart();
      length = time.getLength();
      idRoom = room.getRoom_id();

      for (int wk = 0; wk < weeks.size(); wk++) {
         if (weeks.get(wk) == 1) {
            for (int dy = 0; dy < days.size(); dy++) {
               if (days.get(dy) == 1) {
                  for (int tm = start; tm <= (start + length);
tm++) {
     if ((unRoom[idRoom - 1][wk][dy][tm] == 1)) {
                        return false;
                     }
```

Kode 5.49 Metode Pengecekan Ketidaktersediaan Ruangan

Metode pada Kode 5.50 berfungsi untuk melakukan pengecekan untuk menampilkan semua solusi kelas yang sudah memiliki jadwal-jadwal yang telah ditetapkan.

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

Kode 5.50 Metode Pengecekan Hasil Solusi

5.2.2 Perhitungan Nilai Penalti

Metode pada Kode 5.51 berfungsi untuk melakukan pengecekan pada nilai total penalti yang dimiliki oleh masing-masing kelas dari jadwal yang telah ditentukan.

```
public static int countTSPenalty(Arraylist<Kelas>
listKelas){
        int totalPenaltiTS=0;
            for (int i = 0; i < listKelas.size(); i++) {
               Kelas kls = listKelas.get(i);
               int timeP = penaltyTS(kls);
               totalPenaltiTS += timeP;
            }
            return totalPenaltiTS;
  }
```

Kode 5.51 Metode Pengecekan Total Nilai Penalti Kelas

Metode Kode 5.52 berfungsi sebagai pengecekan total nilai penalti yang dimiliki oleh ruangan yang dipilih sebagai jadwal kelas. Sehingga nilai penalti akan masuk kedalam kelas-kelas yang sudah terjadwal.

```
public static int countRMPenalty(Arraylist<Kelas>
listKelas){
    int totalPenaltiRM=0;
    for (int i = 0; i < listKelas.size(); i++) {
       Kelas kls = listKelas.get(i);
       int roomP = penaltyRM(kls);
       totalPenaltiRM += roomP;
    }
    return totalPenaltiRM;
  }
```

Kode 5.52 Metode Pengecekan Nilai Penalti Ruangan

Metode Kode 5.53 berfungsi untuk mengambil nilai penalti pada jadwal-jadwal waktu yang menjadi pilihan bagi kelas-kelas tertentu. Setiap pilihan jadwal yang tersedia memiliki

nilai penalti masing-masing sehingga dibutuhkan metode untuk menyimpan nilai tersebut.

```
static int penaltyTS(Kelas a){
      int penalty=0;
      int ts = a.getTs();
      if (ts!=-1) {
         TimeAss timeA = a.getAvailableTS().get(ts);
         penalty = timeA.getPenalty();
      }
      return penalty;
   }
```

Kode 5.53 Metode Pengecekan Nilai Penalti Jadwal Waktu Kelas

Metode Kode 5.54 berfungsi untuk melakukan pengecekan terhadap nilai penalti masing-masing pilihan ruangan pada jadwal yang tersedia.

```
static int penaltyRM(Kelas a){
      int penalty=0;
      int rm = a.getRoom();
      if (rm!=-1) {
         RoomAss roomA = a.getAvailableroom().get(rm);
         penalty = roomA.getRoom_pen();
      }
      return penalty;
   }
```

Kode 5.54 Metode Pengecekan Nilai Penalti Jadwal Ruang

5.3. Optimasi Solusi

Pada subbab ini akan menjelaskan langkah untuk melakukan optimasi terhadap jadwal awal yang telah didapatkan dari hasil inisial solusi yang telah didapatkan. Optimasi akan dilakukan dengan menggunakan algoritma *late acceptance hill climbing*

hyper-heuristics. Dalam tahapan optimasi solusi, terdapat beberapa aktivitas yang dilakukan antara lain, menyimpan solusi awal, pembuatan dan pemilihan *low level heuristic*, implementasi algoritma dan penyimpanan solusi optimum.

5.3.1. Penyimpanan Solusi Awal dan Iterasi

Solusi awal terbentuk berdasarkan hasil dari inisial solusi yang didapatkan. Penyimpanan dilakukan dengan menghitung seluruh jumlah penalti yang yang didapatkan untuk sebuah *dataset* yang memenuhi *hard constraint* namun tidak memperdulikan *soft constraint*.

Kode 5.55 menunjukkan bahwa penyimpanan solusi awal dihasilkan dari kelas objek yang telah dibentuk. Keluaran yang dihasilkan dari metode ini adalah berupa daftar kelas beserta dengan pilihan jadwal waktu yang terpilih dan ruangan yang terpilih dengan indeks sebagai penunjuknya.

```
initialSol = new InitSol(listKelas, timeslot, unRoom,
travel);
initialSol.countNoTS();
klsWTS = initialSol.klsNoTS.size();
System.out.println(klsWTS);
listKelas = initialSol.listKelas;
timeslot = initialSol.timeslot;
```

Kode 5.55 Metode Menyimpan Hasil Solusi Awal

Kode 5.56 menunjukkan untuk mencetak hasil solusi awal yang telah terbentuk berdasarkan Kode 5.56 sebelumnya. Hasil cetak ini didapatkan dari kelas objek solusi inisial. Bentuk dari cetak solusi diubah menjadi bentuk file yang memiliki ektensi xml.

```
initialSol.cekSol();
ceks = initialSol.sol.size();
System.out.println();
```

Kode 5.56 Metode Pencetakan Hasil Solusi Awal

5.3.2. Perhitungan Total Nilai Penalti Solusi Awal

Metode kode berfungsi untuk menghitung hasil total dari nilai penalti yang didapatkan dari masing-masing nilai penalti yang berasal dari nilai penalti jadwal waktu, ruangan, dan *soft constraint*. Kemudian masing-masing nilai penalti dikalikan dengan nilai bobot masing-masing konten berdasarkan inisiasi *problem* masing-masing *dataset*.

```
bobotTS = bobot[0];
bobotRM = bobot[1];
bobotSC = bobot[2];
penaltyTS = countTSPenalty(listKelas) * bobotTS;
penaltyRM = countRMPenalty(listKelas) * bobotRM;
penaltySC = cekSC.totalPenalty(listKelas, travel, distrib) *
bobotSC;
totalPenalty = penaltyTS + penaltyRM + penaltySC;
```

Kode 5.57 Metode Penghitungan Bobot Total Nilai Penalti

5.3.3. Pembuatan *Low level heuristics*

Low level heuristics merupakan metode yang digunakan untuk memindah urutan solusi sedemikian hingga solusi menghasilkan nilai penalti baru. Pada penelitian tugas akhir ini, ditentukan *low level heuristic* berupa "*move*" yang dilakukan untuk memindahkan jadwal waktu terpilih dan jadwal ruang terpilih dengan pilihan berikutnya yang berada dalam satu daftar pilihan masing-masing kelas. Kombinasi-kombinasi *low level heuristics* (LLH) ini yang akan

menghasilkan nilai penalti baru dengan memenuhi ketentuan *hard constraint* dan *soft constraint*.

5.3.3.1 *Move* 1

Strategi *move* 1 digunakan untuk mengganti *timeslot* terpilih dengan yang baru. Pergantian tersebut dilakukan dengan menambah satu indeks dari indeks terpilih menjadi *timeslot* baru. Jika ternyata pilihan awal sudah berada pada pilihan terakhir maka indeks dikembalikan menjadi nilai awal yaitu indeks ke-0. Kemudian indeks baru yang terpilih ditetapkan menjadi *timeslot* baru. Kode 5.58 menunjukkan strategi ini.

```
void move1TS() {
        if ((ts + 1) < times.size()) {
                tsBaru = ts + 1;
        kelas.setTs(tsBaru);
        } else if (ts + 1 >= times.size()) {
                tsBaru = 0;
                kelas.setTs(tsBaru);
    }
```

Kode 5.58 Metode *Move* 1

5.3.3.2. *Move* 2

Strategi *move* 2 digunakan untuk mengganti ruangan terpilih dengan yang baru. Pergantian tersebut dilakukan dengan menambah satu indeks dari indeks terpilih menjadi ruangan baru. Jika ternyata pilihan awal sudah berada pada pilihan terakhir maka indeks dikembalikan menjadi nilai awal yaitu indeks ke-0. Kemudian indeks baru yang terpilih ditetapkan menjadi ruangan baru. Kode menunjukkan strategi ini. Metode ini digunakan untuk mendapatkan hasil yang lebih baik disaaat melakukan optimasi.

```
void moveRM() {
        if (rm != -1) {
                if ((rm + 1) < rooms.size()) {
                    rmBaru = rm + 1;
            kelas.setRoom(rmBaru);
        } else if (rm + 1 >= rooms.size()) {
                    rmBaru = 0;
                    kelas.setRoom(rmBaru);
          }
```

Kode 5.59 Metode *Move* 2

5.3.3.3. *Move* 3

Strategi ini menunjukkan pergantian *timeslot* terpilih sebanyak dua kali perpindahan. Sehingga pilihan *timeslot* akan diperbaharui sebanyak dua kali. Kode 5.60menunjukkan metode ini.

```
void move2TS() {
        if ((ts + 1) < times.size()) {
        kelas.setTs(tsBaru);
        else if (ts + 1 >= times.size()) {
                tsBaru = 0;
                kelas.setTs(tsBaru);   }
              if ((tsB + 1) < timesB.size()) {
                tsBaruB = tsB + 1;
        kelasB.setTs(tsBaruB);
        } else if (tsB + 1 >= timesB.size()) {
                tsBaruB = 0;
                kelasB.setTs(tsBaruB); }
```

Kode 5.60 Metode *Move* 3

5.3.3.4. *Move* 4

Strategi ini menunjukkan pergantian ruangan terpilih sebanyak dua kali perpindahan. Sehingga pilihan ruangan akan

diperbaharui sebanyak dua kali. Kode 5.61 menunjukkan metode ini.

```
void move2RM() {
        if (rm != -1 && rmB != -1) {
                if ((rm + 1) < rooms.size()) {
                    rmBaru = rm + 1;
            kelas.setRoom(rmBaru);
        } else if (rm + 1 >= rooms.size()) {
                    rmBaru = 0;
                    kelas.setRoom(rmBaru);
         }
        if ((rmB + 1) < roomsB.size()) {
                    rmBaruB = rmB + 1;
            kelasB.setRoom(rmBaruB);
        } else if (rmB + 1 >= roomsB.size()) {
                    rmBaruB = 0;
                    kelasB.setRoom(rmBaruB);
         }
```

Kode 5.61 Metode *Move* 4

5.3.3.5. *Move* 5

Strategi ini menunjukkan pergantian *timeslot* terpilih sebanyak tiga kali perpindahan. Sehingga pilihan *timeslot* akan diperbaharui sebanyak dua kali. Kode 5.62 menunjukkan metode ini. Metode ini digunakan agar mendapatkan hasil optimasi yang lebih baik dengan mencari segala kemungkinan yang bisa terjadi. Sehingga hasil yang didapatkan akan mampu mengoptimalkan hasil yang digunakan. Maka dari itu dibentuk l*ow level heuristics* yang lebih banyak untuk mencapai optimal yang lebih baik. Adapun *LLH* yang digunakan memiliki nilai *random* saat memilih yang akan digunakan.

```
void moveTS() {
        if ((ts + 1) < times.size()) {
                tsBaru = ts + 1;
        kelas.setTs(tsBaru);
        } else if (ts + 1 >= times.size()) {
                tsBaru = 0;
                kelas.setTs(tsBaru);
    }
        if ((tsB + 1) < timesB.size()) {
                tsBaruB = tsB + 1;
        kelasB.setTs(tsBaruB);
        } else if (tsC + 1 >= timesB.size()) {
                tsBaruB = 0;
                kelasB.setTs(tsBaruB);
    }
        if ((tsC + 1) < timesC.size()) {
                tsBaruC = tsC + 1;
        kelasC.setTs(tsBaruC);
        } else if (tsC + 1 >= timesC.size()) {
                tsBaruC = 0;
                kelasC.setTs(tsBaruC); }
```

Kode 5.62 Metode *Move* 5

5.3.3.6 *Move* 6

Strategi ini dilakukan dengan cara memindahkan ruangan dan *timeslot* dalam sekali iterasi sehingga menghasilkan ruangan dan *timeslot* baru. Kode 5.63 menunjukkan metode strategi ini. Maka dari itu dibentuk l*ow level heuristics* yang lebih banyak untuk mencapai optimal yang lebih baik. Adapun *LLH* yang digunakan memiliki nilai *random* saat memilih yang akan digunakan. Berikut adalah salah satu dari tujuh model yang akan digunakan dalam mencapai solusi optimal yang akan dihasilkan dari proses pembangunan algoritma yang akan dibangun.

```
void move1TSRM() {
        if ((ts + 1) < times.size()) {
                tsBaru = ts + 1;
                kelas.setTs(tsBaru);
                System.out.println("Index TS baru: " +
        tsBaru);
            } else if (ts + 1 >= times.size()) {
                tsBaru = 0;
                kelas.setTs(tsBaru);  }
            if (rm != -1) {
                if ((rm + 1) < rooms.size()) {
                    rmBaru = rm + 1;
                    kelas.setRoom(rmBaru);
                } else if (rm + 1 >= rooms.size()) {
                    rmBaru = 0;
                    kelas.setRoom(rmBaru);} }
```

Kode 5.63 Metode *Move* 6

5.3.3.7. *Move* 7

Strategi *move* ini dilakukan dengan cara pergantian *timeslot* awal sebanyak dua kali pergantian, kemudian melakukan pergantian ruangan sebanyak dua kali pula dalam satu iterasi. Sehingga menghasilkan solusi baru. Kode 5.64 menunjukkan hasil iterasi baru. Maka dari itu dibentuk l*ow level heuristics* yang lebih banyak untuk mencapai optimal yang lebih baik. Adapun *LLH* yang digunakan memiliki nilai *random* saat memilih yang akan digunakan. Berikut adalah salah satu dari tujuh model yang akan digunakan dalam mencapai solusi optimal yang akan dihasilkan dari proses pembangunan algoritma yang akan dibangun.

```
void move2TSRM() {
        if ((ts + 1) < times.size()) {
                tsBaru = ts + 1;
                kelas.setTs(tsBaru);
                System.out.println("Index TS baru: " +
        tsBaru);
            } else if (ts + 1 >= times.size()) {
                tsBaru = 0;
                kelas.setTs(tsBaru);  }
            if ((tsB + 1) < timesB.size()) {
                tsBaruB = tsB + 1;
                kelasB.setTs(tsBaruB);
                System.out.println("Index TS baru: " +
                } else if (rm + 1 >= rooms.size()) {   }
        if (rmB != -1) {
                if ((rmB + 1) < roomsB.size()) {
                   rmBaruB = rmB + 1;
```

Kode 5.64 Metode *Move* 7

5.3.4. Implementasi Algoritma *Late acceptance hill climbing*

Subbab ini akan membahas implementasi *late acceptance hill climbing hyper heuristic*s pada solusi awal yang telah dihasilkan sebelumnya. Algoritma ini diterapkan untuk memilih solusi yang paling optimum dari solusi yang telah dihasilkan di solusi awal. Algoritma ini merupakan pengembangan dari algoritma Greedy *Hill climbing*, perbedaan terdapat pada proses penerimaan hasil pada setiap iterasi. *LAHC* menerima solusi terbaik dan diletakkan di dalam sebuah tempat untuk menyimpan solusi terbaik selama iterasi.

Iterasi dilakukan berulang kali untuk mendapatkan hasil yang baik. Solusi terbaik didapatkan dengan memenuhi satu dari dua syarat yang berlaku, yaitu (1) nilai penalti yang dihasilkan lebih baik daripada nilai penalti solusi yang ada di dalam

History length, atau (2) nilai tidak sama dengan nilai solusi terbaik yang ada pada daftar *History length*. Kode 5.65 adalah pernyataan dari algoritma *LAHC*.

```
indexSolusi = iterasi%L;
if (newPenalty < penaltyList.get(indexSolusi) ||
 newPenalty <= currentPenalty ) {
currentPenalty = newPenalty;
for (int i = 0; i < listKelas.size(); i++) {
Kelas kelas = listKelas.get(i);
kelas.setTsBaru(kelas.getTs());
kelas.set<i>Room</i>Baru(kelas.get<i>Room</i>());}
        idle = 0;
        penaltyList.re<i>move</i>(0);
        penaltyList.add(newPenalty);
        } else {
        idle++;
        if (newPenalty != X) {}
        for (int i = 0; i < listKelas.size(); i++) {
        Kelas kelas = listKelas.get(i);
        kelas.setTs(kelas.getTsBaru());
        kelas.set<i>Room</i>(kelas.get<i>Room</i>Baru }
        if (idle >= maxIterasi* 0.2) {
                }iterasi++;
    } while (iterasi < maxIterasi && idle <
maxIterasi*0.2 );
```

Kode 5.65 Metode Algoritma *Late acceptance hill climbing*

5.3.5 Pengecekan Ulang *Hard constraint* dan Perhitungan Penalti

Subbab ini menjelaskan bagaimana solusi iterasi dari hasil optimasi yang terbentuk dilakukan pengecekan kembali untuk semua *hard constraint* yang berlaku. Kemudian selain melakukan pengecekan terhadap batasan, dilakukan pula penghitungan ulang nilai penalti

untuk jadwal terpilih dan ruangan terpilih. Untuk menghitung nilai penalti dari solusi optimum, beberapa aktivitas yang dilakukan adalah membaca solusi optimum, menguji solusi dengan *hard constraint* dan menghitung penalti soslusi terhadap *soft constraint*.

5.3.5.1 Pembacaan Solusi Optimum

Solusi akhir atau solusi optimum yang telah terbentuk dan telah dituliskan dalam sebuah *file* dalam bentuk dan ekstensi xml.

5.3.5.2 Uji *Hard constraint*

Uji *hard constraint* dilakukan untuk menentukan apakah solusi akhir yang dihasilkan telah memenuhi semua persyaratan *hard constraint*. Jika solusi akhir lolos semua syarat *hard constraint*, maka perhitungan nilai penalti akan dilakukan. Jika tidak lolos maka diberikan nilai penalti yang sangat besar sehingga otomatis akan tertolak.

5.3.5.3 Perhitungan Penalti Akhir

Skor penalti akhir merupakan total dari skor penalti yang dihasilkan pada masing-masing penalti. Skor penalti akhir akan dibandingkan pada setiap eksperimen.

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 Tabu-Simulated Annealing *Hyper-Heuristic*, termasuk eksperimen parameter yang digunakan.

6.1. Lingkungan Uji Coba

Pada subbab lingkungan uji coba ini akan dijelaskan terkait dengan lingkungan pengujian dalam melakukan implementasi penelitian tugas akhir terkait penjadwalan mata kuliah menggunakan domain ITC 2019 *Dataset*. Spesifikasi perangkat keras yang digunakan dalam implementasi ditunjukkan pada Tabel 6.1.

Tabel 6.1 Spesifikasi Perangkat Keras

Perangkat Keras	Spesifikasi
Jenis	Lenovo 80M7
Processor	AMD A8
RAM	4.00 GB
Hard Disk Drive	500 GB

Untuk spesifikasi perangkat lunak digunakan dalam implementasi metode ditunjukkan pada Tabel 6.2.

Tabel 6.2 Spesifikasi Perangkat Lunak

Perangkat Lunak	Fungsi
Windows 10 64 bit	Sistem operasi
Sublime Text 3	Menampilkan data

Perangkat Lunak	Fungsi
Netbeans 8.0.2	Implementasi algoritma
Micros*oft* Word 2016	Penulisan laporan

6.2. Hasil Implementasi Penjadwalan

6.2.1. Skenario 1: Pembentukan Solusi Awal

Pada percobaan pembentukan solusi awal, dilakukan eksperimen terhadap seluruh *dataset* yang telah telah disediakan oleh ITC 2019. Terdapat 15 jenis *dataset* yang dapat dilakukan uji coba. Ke-15 *dataset* tersebut dilakukan percobaan apakah dapat memenuhi ketentuan *hard constraints* yang berlaku menggunakan algoritma yang telah dibangun. Tabel 6.3 menunjukkan hasil eksperimen *dataset* yang telah dilakukan.

Tabel 6.3 Hasil Implementasi Skenario 1

Nama *Instances*	Tipe Data	Jumlah *Total Kelas*	Kelas Tidak Terjadwal
Test-Tiny	*Testing*	20	0
Test-Small 1	*Testing*	127	14
Test-Small 2	*Testing*	174	0
Test-Medium	*Testing*	802	10
Test-Large	*Testing*	2417	93
Early Instance 1	*Early*	1239	47
Early Instance 2	*Early*	1852	122

Nama *Instances*	Tipe Data	Jumlah *Total Kelas*	Kelas Tidak Terjadwal
Early Instance 3	*Early*	983	261
Early Instance 4	*Early*	2641	103
Early Instance 5	*Early*	882	12
Early Instance 6	*Early*	575	25
Early Instance 7	*Early*	561	41
Early Instance 8	*Early*	2526	106
Early Instance 9	*Early*	1001	13
Early Instance 10	*Early*	711	34

Berdasarkan hasil percobaan yang dilakukan pad 15 *dataset* tersebut, hanya dua *dataset* yang berhasil memenuhi semua ketentuan yang berlaku. Hal ini disebabkan keterbatasannya waktu dalam mengerjakan mengingat terlalu kompleksnya struktur *dataset* yang harus dijadwalkan. Untuk itu seharusnya diperlukan strategi algoritma baik dan bagus dalam penyelesaian masalah. Kemudian untuk selanjutnya, percobaan berdasarkan skenario-skenario yang akan dilakukan dan menjadi skenario akan dilakukan hanya pada dua *dataset* yang berhasil diselesaikan memenuhi *hard constraint* saja.

6.2.2 Skenario 2: Jumlah Iterasi

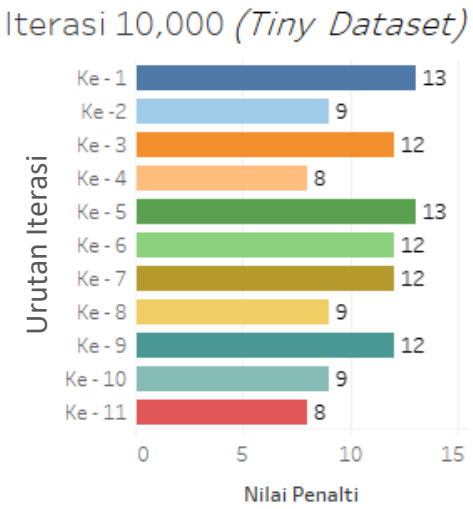
Pada skenario ini dilakukan eksperimen pada jumlah iterasi yang dilakukan untuk kedua *dataset*. Jumlah iterasi dimulai dari 10.000 sampai dengan 1.000.000 iterasi. Kemudian untuk

nilai parameter lainnya akan disesuaikan dengan nilai standar yaitu nilai *idle* sebesar 0.02 dan panjang *History* sebesar 10. Subbab berikutnya akan menjelaskan masing-masing hasil iterasi berikut.

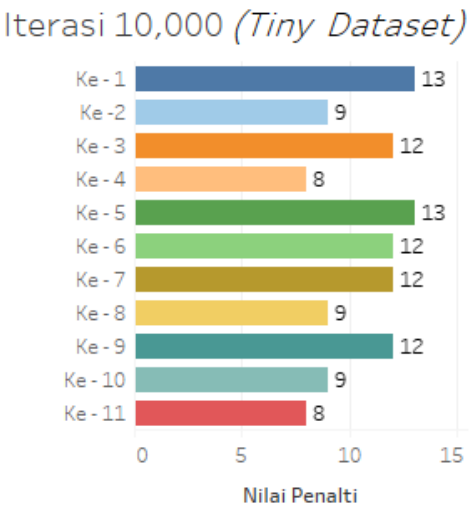
6.2.2.1 Iterasi 10.000

Pada skenario 2 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan iterasi total 10.000 iterasi. Iterasi menggunakan parameter standar pada *LAHC* hanya saja iterasi yang diubah. Parameter lainnya yaitu *idle* sebesar 0.02 dan panjang *History length* sebesar 10. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

A. *Dataset* 1 (*Tiny*)



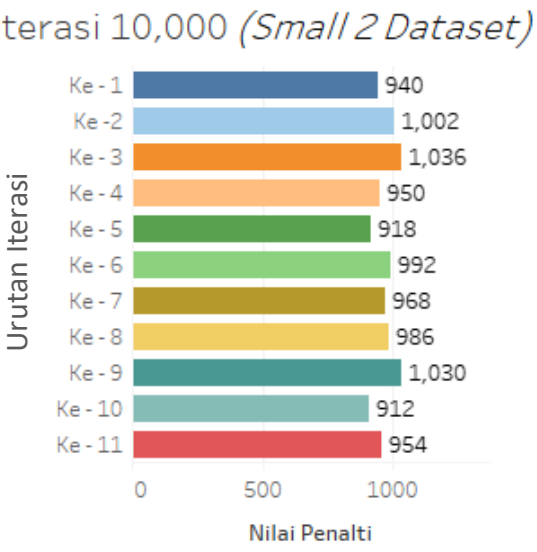
Grafik 6.1 Skenario 2 (10.000 Iterasi)



Grafik 6.1 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 10.000 iterasi. Nilai penalti rata-rata adalah 11 dari total 11 iterasi yang dilakukan. Nilai tersebut sudah sangat baik dari nilai penalti awal yang didapatkan sebesar 53.

B. *Dataset* 2 (*Small 2*)

Grafik **6.2** menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 10.000 iterasi. Nilai penalti rata-rata adalah 972 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 1048 dan terkecil adalah 921. Terdapat penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi.

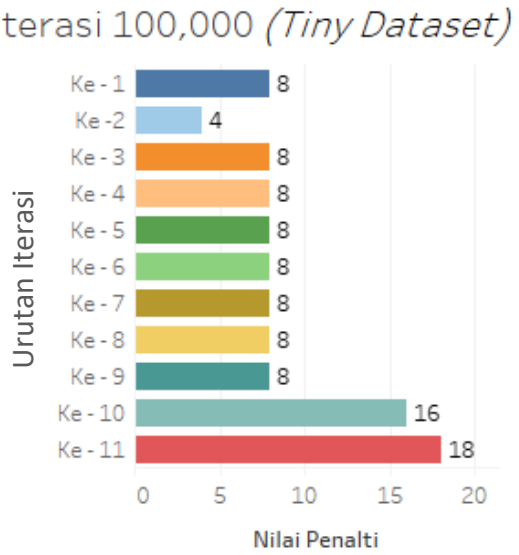


Grafik 6.2 Skenario 2 10.000 Iterasi

6.2.2.2 Iterasi 100.000

Pada skenario 2 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan iterasi total 100.000 iterasi. Iterasi menggunakan parameter standar pada *LAHC* hanya saja iterasi yang diubah. Parameter lainnya yaitu *idle* sebesar 0.02 dan panjang *History length* sebesar 10. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

A. *Dataset* 1 (*Tiny*)

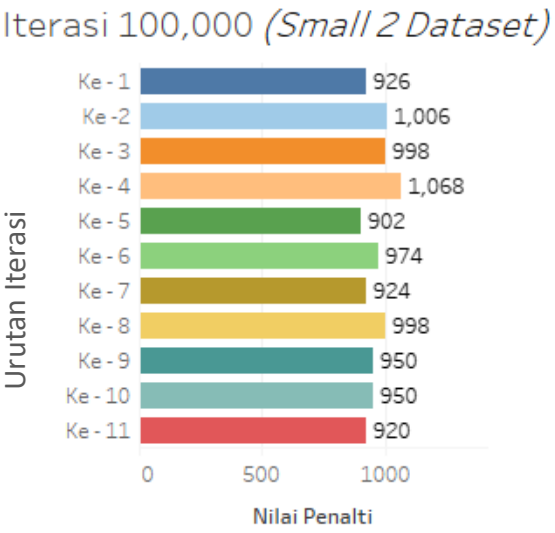


Grafik 6.3 Skenario 2 100.000 Iterasi

Grafik 6.3 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 100.000 iterasi. Nilai penalti rata-rata adalah 9 dari total 11 iterasi yang dilakukan. Terdapat penurunan nilai penalti dari hasil penalti inisial solusi.

B. *Dataset* 2 (*Small 2*)

Grafik 6.4 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 100.000 iterasi. Nilai penalti rata-rata adalah 965 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 1068 dan terkecil adalah 902. Terdapat penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi.

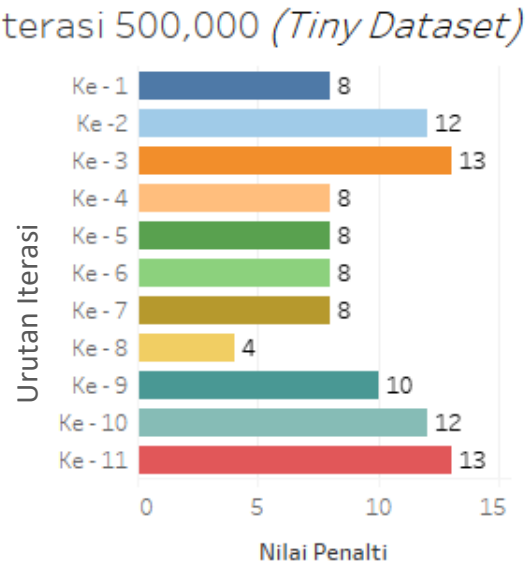


Grafik 6.4 Skenario 2 Iterasi 100.000

6.2.2.3 Iterasi 500.000

Pada skenario 2 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan iterasi total 500.000 iterasi. Iterasi menggunakan parameter standar pada *LAHC* hanya saja iterasi yang diubah. Parameter lainnya yaitu *idle* sebesar 0.02 dan panjang *History length* sebesar 10. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

A. *Dataset* 1 (*Tiny*)

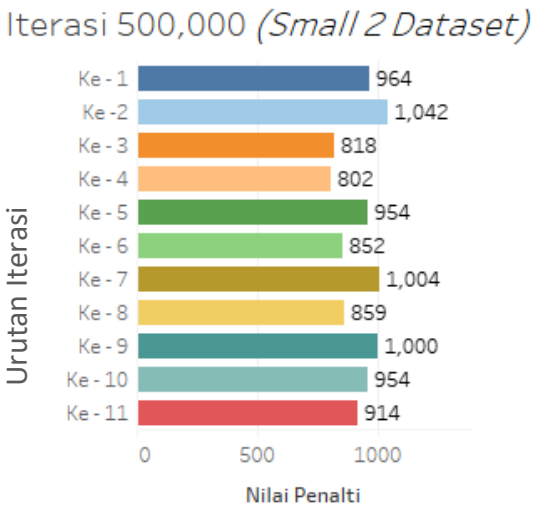


Grafik 6.5 Skenario 2 500.000 Iterasi

Gambar 6.5 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 500.000 iterasi. Nilai penalti rata-rata adalah 9 dari total 11 iterasi yang dilakukan. Terdapat peningkatan dengan menurunnya nilai penalti yang berarti nilai tersebut lebih bagus.

B. *Dataset* 2 (*Small 2*)

Grafik 6.6 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 500.000 iterasi. Nilai penalty rata-rata adalah 924 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 1042 dan terkecil adalah 802. Terdapat penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi.

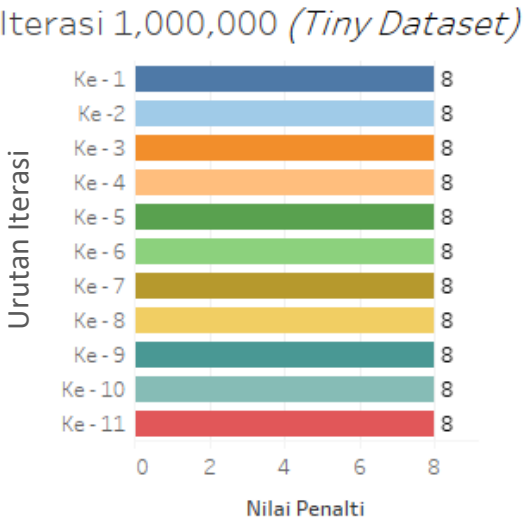


Grafik 6.6 Skenario 2 Iterasi 500.000

6.2.2.4 Iterasi 1.000.000

Pada skenario 2 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan iterasi total 1.000.000 iterasi. Iterasi menggunakan parameter standar pada *LAHC* hanya saja iterasi yang diubah. Parameter lainnya yaitu *idle* sebesar 0.02 dan panjang *History length* sebesar 10. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

A. *Dataset* 1 (*Tiny*)

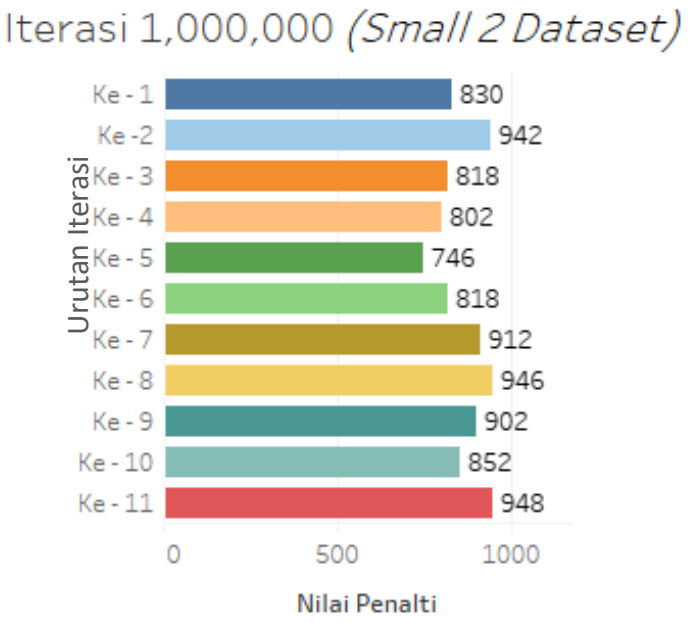


Grafik 6.7 Skenario 2 1.000.000 Iterasi

Grafik 6.8 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 1.000.000 iterasi. Nilai penalti rata-rata adalah 53 dari total 11 iterasi yang dilakukan. Tidak terdapat perubahan nilai dikarenakan data terlalu kecil sehingga tidak dapat terjadi peningkatan.

B. *Dataset* 2 (*Small 2*)

Grafik 6.9 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 1.000.000 iterasi. Nilai penalti rata-rata adalah 865 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 948 dan terkecil adalah 746. Terdapat penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi.



Grafik 6.8 Skenario 2 Iterasi 1.000.000

6.2.2 Skenario 3: Perubahan *Idle*

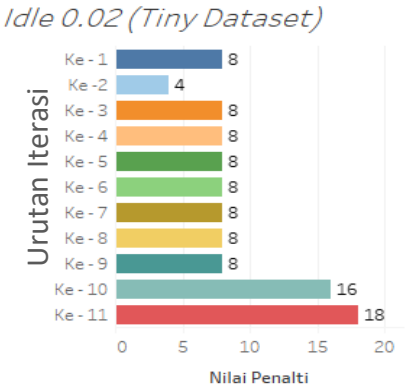
Pada skenario ini dilakukan eksperimen pada perubahan nilai *idle* yang menjadi *stopping criteria* iterasi yang dilakukan untuk kedua *dataset*. Nilai *idle* dimulai dari 0.02 sampai dengan 0.8. Kemudian untuk nilai parameter lainnya akan disesuaikan dengan nilai standar yaitu nilai iterasi sebesar 100.000 dan panjang *History* sebesar 10. Subbab berikutnya akan menjelaskan masing-masing hasil iterasi berikut.

6.2.2.1 *Idle* 2%

Pada skenario 3 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1)

tiny (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan nilai *idle* sebesar 0.02. Iterasi menggunakan parameter standar pada *LAHC* hanya saja iterasi yang diubah. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

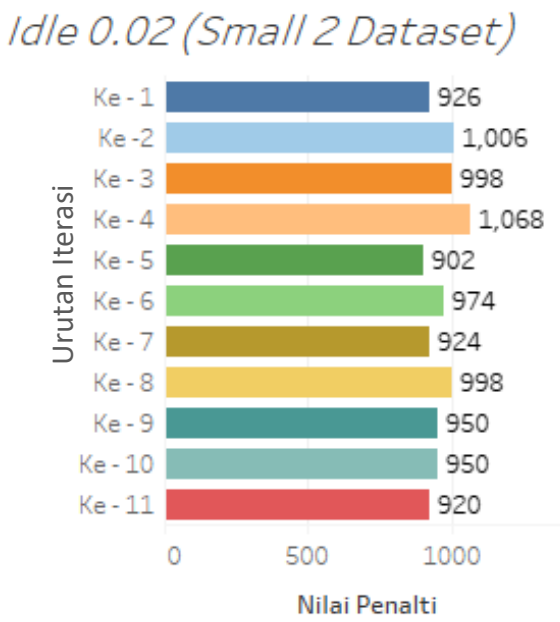
A. *Dataset* 1 (*Tiny*)



Grafik 6.9 Skenario 3 *Idle* 2%

Grafik 6.10 menunjukkan hasil percobaan yang masing-masing dilakukan dengan nilai *idle* 0.02. Nilai penalti rata-rata adalah 9. Nilai tertinggi adalah 18 dan nilai terkecil adalah 8. Dari hasil eksperimen ini menunjukkan bahwasannya terdapat peningkatan yang terjadi ketika menggunakan algoritma dengan parameter sesuai dibandingkan hasil nilai penalti yang didapatkan dari hasil inisial solusi. Grafik menunjukkan hasil nilai penalti yang didapatkan dari 11 kali eksperimen yang dilakukan.

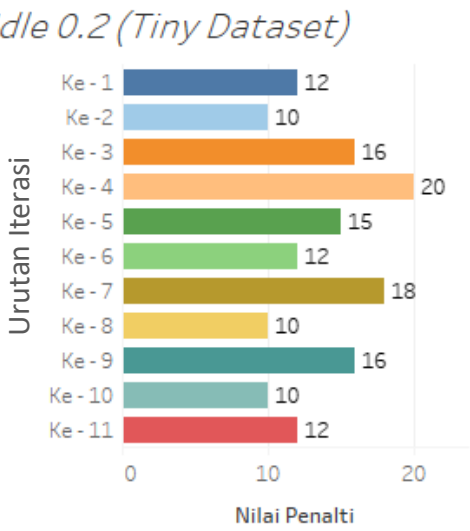
B. *Dataset* 2 (*Small*)



Grafik 6.10 Skenario 3 *Idle* 2%

Grafik 6.11 menunjukkan hasil percobaan yang masing-masing dilakukan sebanyak 100.000 iterasi. Nilai penalti rata-rata adalah 965 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 1068 dan terkecil adalah 902. Terdapat penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi. Dapat dilihat hasil perbedaan antara hasil penalti menggunakan insial solusi dan setelah diberi optimasi.

6.2.2.2 *Idle* 20%



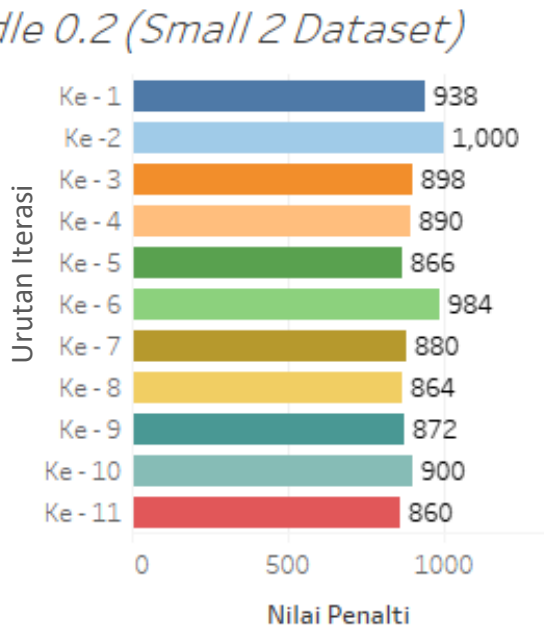
Grafik 6.11 Skenario 3 *Idle* 20%

Pada skenario 3 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan nilai *idle* sebesar 0.2. Iterasi menggunakan parameter standar pada *LAHC* hanya saja iterasi yang diubah. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

A. *Dataset* 1 (*Tiny*)

Gambar 6.12 menunjukkan hasil percobaan yang masing-masing dilakukan dengan nilai *idle* 0.2. Nilai penalty rata-rata adalah 14 dengan nilai tertinggi adalah 20 dan terkecil adalah 10.

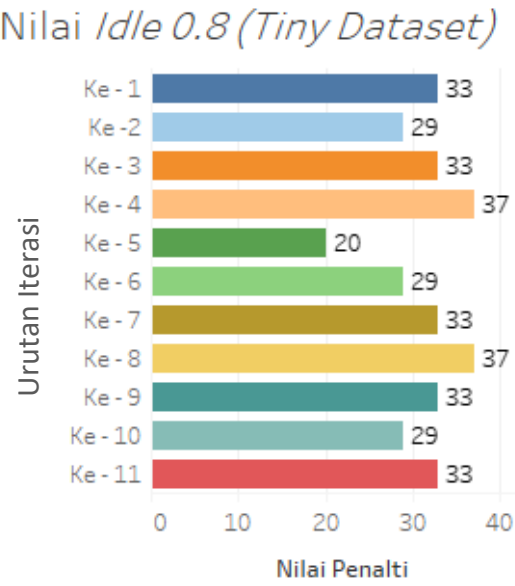
B. *Dataset* 2 (*Small 2*)



Grafik 6.12 Skenario 3 *Idle* 20%

Grafik 6.13 menunjukkan hasil percobaan yang masing-masing dilakukan dilakukan dengan nilai *idle* 0.2. Nilai penalty rata-rata adalah 905. Nilai tertinggi adalah 1000 dan terkecil adalah 860. Terdapat perbedaan nilai penalti yang dihasilkan bila dilakukan perbandingan dnegan nilai penalti yang dihasilkan oleh inisial solusi. Maka dari itu nilai penalti dengan parameter ini lebih baik dibandingkan dengan nilai penalti inisial solusi.

6.2.2.3 *Idle* 80%



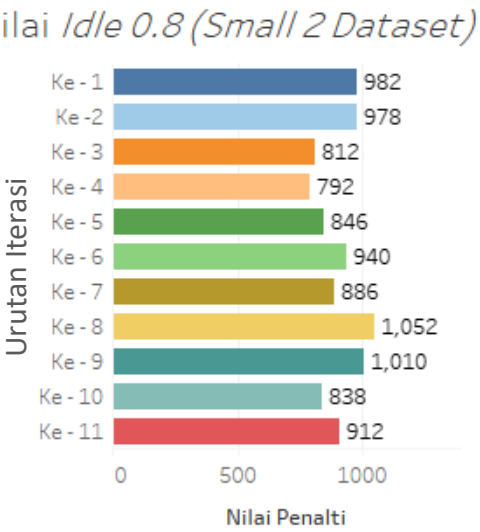
Grafik 6.13 Skenario 3 *Idle* 80%

Pada skenario 3 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan nilai *idle* sebesar 0.8. Iterasi menggunakan parameter standar pada *LAHC* hanya saja iterasi yang diubah. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

A. *Dataset* 1 (*Tiny*)

Grafik 6.14 menunjukkan hasil percobaan yang masing-masing dilakukan dengan nilai *idle* 0.8. Nilai penalti rata-rata adalah 31 dengan nilai terbesar 37 dan nilai terkecil 20.

Berdasarkan hasil inisial solusi, eksperimen dengan nilai penalti menggunakan parameter ini lebih baik dibandingkan.
B. *Dataset* 2 (*Small 2*)



Grafik 6.14 Skenario 3 *Idle* 80%

Grafik 6.15 menunjukkan hasil percobaan yang masing-masing dilakukan dengan nilai *idle* 0.8. Nilai penalti rata-rata adalah 913 dengan nilai terbesar 1052 dan nilai terkecil 794. Berdasarkan hasil inisial solusi, eksperimen dengan nilai penalti menggunakan parameter ini lebih baik dibandingkan nilai penalti inisisial solusi.

6.2.3 Skenario 4: Perubahan *History Length*

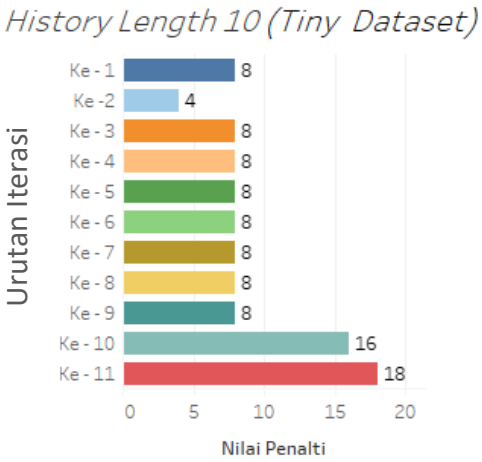
Pada skenario ini dilakukan eksperimen pada perubahan nilai nilai panjang penyimpanan *History* yang dilakukan untuk kedua *dataset*. Nilai *History length* dimulai dari 10 sampai

dengan 1000. Kemudian untuk nilai parameter lainnya akan disesuaikan dengan nilai standar yaitu nilai iterasi sebesar 100.000 dan nilai *idle* sebesar 0.02. Subbab berikutnya akan menjelaskan masing-masing hasil iterasi berikut

6.2.3.1 *History Length* 10

Pada skenario 4 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan nilai panjang *History* sebesar 10. Iterasi menggunakan parameter standar pada *LAHC* hanya saja nilai *History* yang diubah. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

A. *Dataset* 1 (*Tiny*)

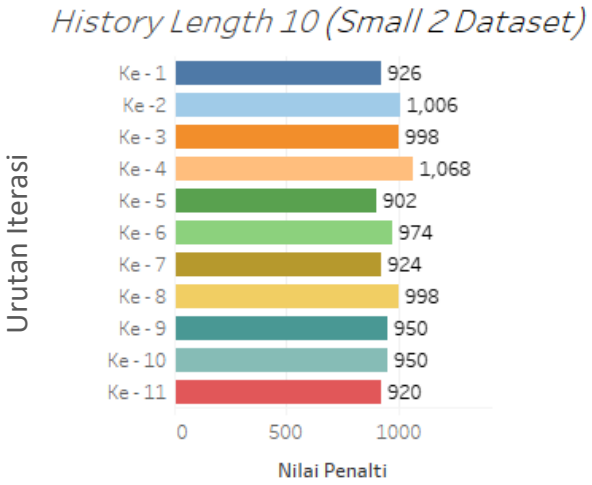


Grafik 6.15 Skenario 4 *History Length* 10

Grafik 6.16 menunjukkan hasil percobaan yang masing-masing dilakukan dengan *History length* adalah 10. Nilai

penalti rata-rata adalah 9. Nilai tertinggi adalah 18 dan nilai terkecil adalah 8. Dari hasil eksperimen ini menunjukkan bahwasannya terdapat peningkatan yang terjadi ketika menggunakan algoritma dengan parameter sesuai dibandingkan hasil nilai penalti yang didapatkan dari hasil inisial solusi. Grafik menunjukkan hasil nilai penalti yang didapatkan dari 11 kali eksperimen yang dilakukan.

B. *Dataset* 2 (*Small 2*)



Grafik 6.16 Skenario 4 *History Length* 10

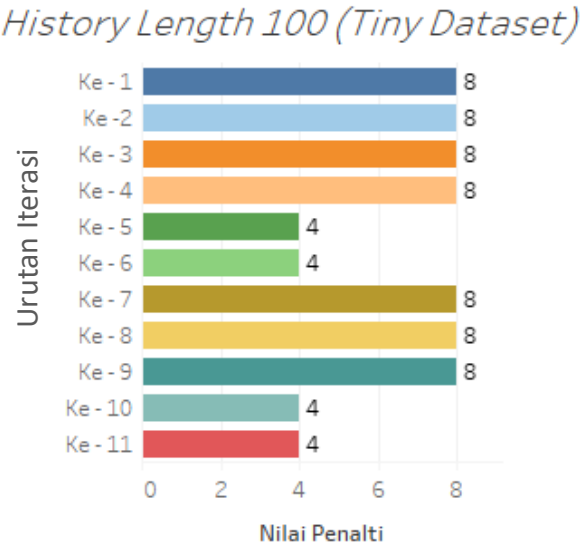
Grafik 6.16 meunjukkan hasil percobaan yang masing-masing dilakukan dilakukan dengan nilai *History* 10. Nilai penalti rata-rata adalah 965 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 1068 dan terkecil adalah 902. Terdapat

penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi. Dapat dilihat hasil perbedaan antara hasil penalti menggunakan insial solusi dan setelah diberi optimasi.

6.2.3.2 *History Length* 100

Pada skenario 4 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan nilai panjang *History* sebesar 100. Iterasi menggunakan parameter standar pada *LAHC* hanya saja nilai *History* yang diubah. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

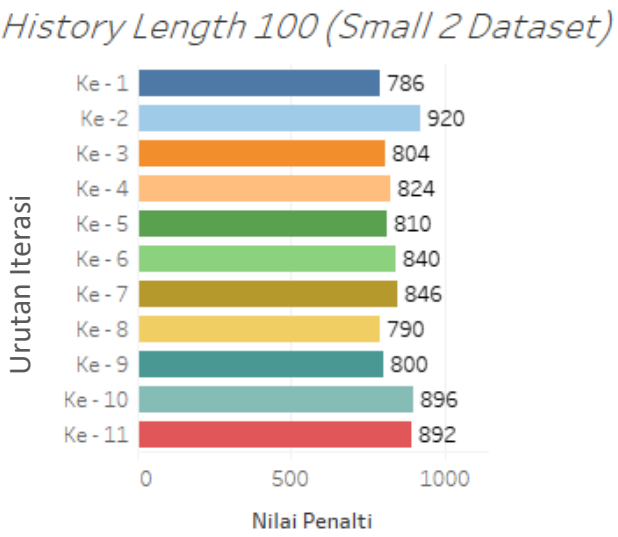
A. *Dataset* 1 (*Tiny*)



Grafik 6.17 Skenario 4 *HL* 100

Grafik 6.17 menunjukkan hasil percobaan yang masing-masing dilakukan dengan *History length* adalah 100. Nilai penalti rata-rata adalah 7. Nilai tertinggi adalah 8 dan nilai terkecil adalah 4. Dari hasil eksperimen ini menunjukkan bahwasannya terdapat peningkatan yang terjadi ketika menggunakan algoritma dengan parameter sesuai dibandingkan hasil nilai penalti yang didapatkan dari hasil inisial solusi. Grafik menunjukkan hasil nilai penalti yang didapatkan dari 11 kali eksperimen yang dilakukan.

B. *Dataset* 2 (*Small 2*)



Grafik 6.18 Skenario 4 *HL* 100

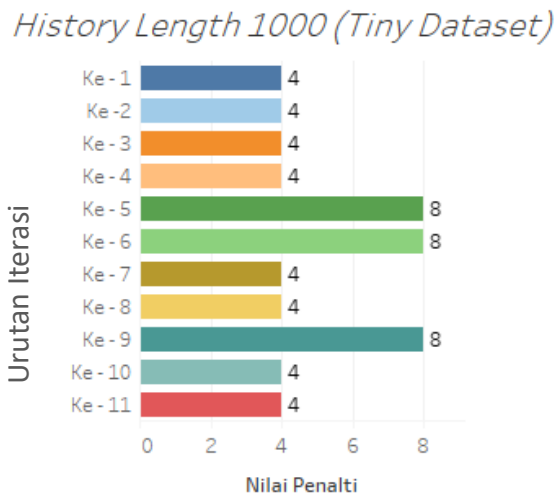
Grafik 6.18 menunjukkan hasil percobaan yang masing-masing dilakukan dilakukan dengan nilai *History* 100. Nilai

penalti rata-rata adalah 837 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 920 dan terkecil adalah 786. Terdapat penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi. Dapat dilihat hasil perbedaan antara hasil penalti menggunakan insial solusi dan setelah diberi optimasi.

6.2.3.3 *History Length* 1000

Pada skenario 4 untuk kedua *dataset* yang berhasil memenuhi *hard constraint* hingga menghasilkan solusi inisial yaitu, (1) *tiny* (lums-sum17) dan (2) *small* (pu-cs-fal07) dilakukan sebanyak 11 kali dengan nilai panjang *History* sebesar 10. Iterasi menggunakan parameter standar pada *LAHC* hanya saja nilai *History* yang diubah. Berikut hasil iterasi terbagi menjadi dua bagian pada poin a dan b.

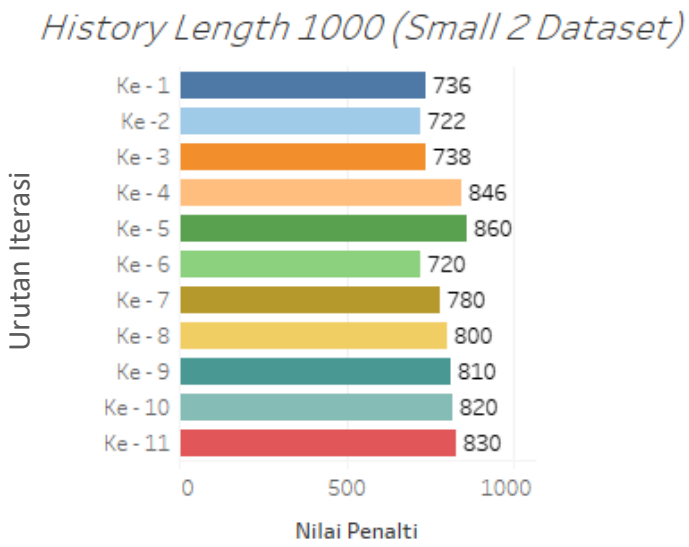
A. *Dataset* 1 (*Tiny*)



Grafik 6.19 Skenario 4 *HL* 1000

Grafik 6.19 menunjukkan hasil percobaan yang masing-masing dilakukan dengan *History length* adalah 1000. Nilai penalti rata-rata adalah 5. Nilai tertinggi adalah 8 dan nilai terkecil adalah 4. Dari hasil eksperimen ini menunjukkan bahwasannya terdapat peningkatan yang terjadi ketika menggunakan algoritma dengan parameter sesuai dibandingkan hasil nilai penalti yang didapatkan dari hasil inisial solusi. Grafik menunjukkan hasil nilai penalti yang didapatkan dari 11 kali eksperimen yang dilakukan.

B. *Dataset* 2 (*Small 2*)



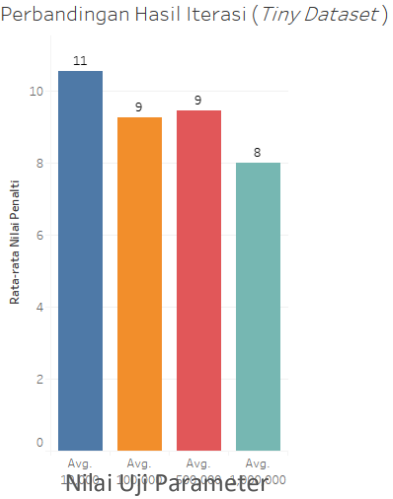
Grafik 6.20 Skenario 4 *HL* 1000

Grafik 6.20 menunjukkan hasil percobaan yang masing-masing dilakukan dilakukan dengan nilai *History* 100. Nilai penalti rata-rata adalah 787 dari total 11 iterasi yang dilakukan. Nilai penalti terbesar adalah 860 dan terkecil adalah 720. Terdapat penurunan nilai penalti yang dihasilkan bila dibandingkan dengan nilai penalti dari hasil inisial solusi. Dapat dilihat hasil perbedaan antara hasil penalti menggunakan insial solusi dan setelah diberi optimasi.

6.3. Perbandingan Parameter Algoritma Hasil Uji Coba

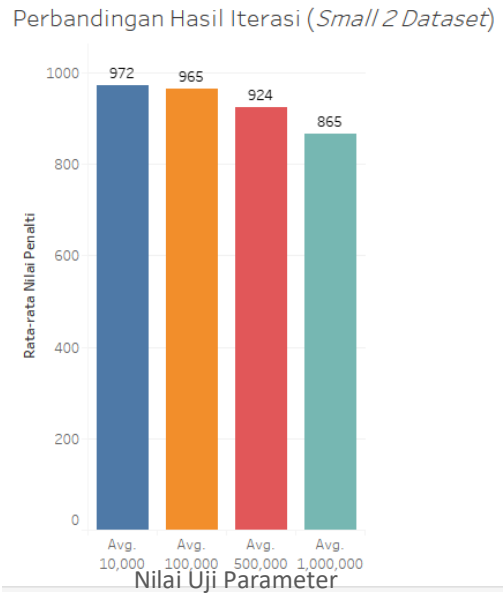
Pada subbab ini menjelaskan bagaimana perbandingan hasil yang didapatkan dari setiap skenario yang telah dihasilkan. Skenario yang dilakukan adalah mengubah parameter dari algoritma.

6.3.1 Perbandingan Hasil Uji Coba Iterasi



Grafik 6.21 Grafik Perbandingan Iterasi (*Tiny*)

Percobaan-percobaan yang telah dilakukan untuk parameter iterasi perlu dilakukan perbandingan hasil untuk dapat mengetahui perubahan yang terjadi pada setiap percobaan. Hal yang akan dibandingkan adalah berdasarkan skenario yang telah dilakukan. Berikut Grafik 6.21 dan Grafik 6.22 adalah perbandingan berdasarkan iterasi untuk *dataset tiny* dan *small*.



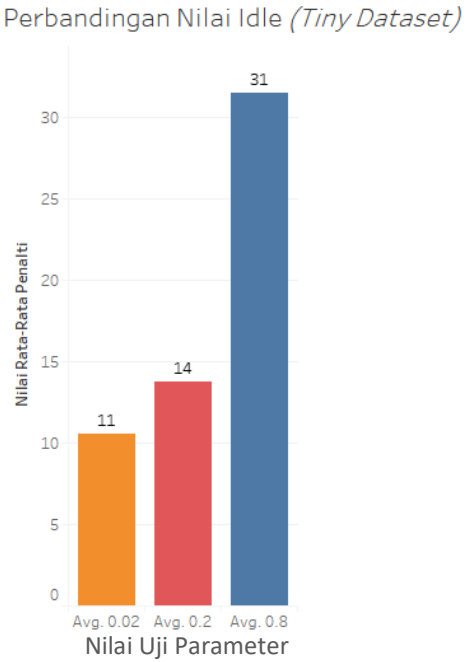
Grafik 6.22 Grafik Perbandingan Iterasi (*Small 2*)

Berdasarkan kedua grafik 6.21 dan 6.22 tersebut dapat dilihat terjadinya penurunan untuk nilai penalti dengan jumlah iterasi yang semakin besar. Penurunan terjadi untuk kedua *dataset* yang dilakukan uji coba. Maka semakin banyak iterasi yang dilakukan, hasil yang didapatkan juga akan semakin lebih baik.

6.3.2 Perbandingan Hasil Uji Coba *Idle*

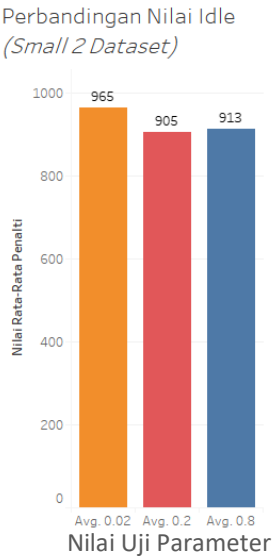
Percobaan-percobaan yang telah dilakukan untuk parameter *idle* perlu dilakukan perbandingan hasil untuk dapat mengetahui perubahan yang terjadi pada setiap percobaan. Hal yang akan dibandingkan adalah berdasarkan skenario yang

telah dilakukan. Berikut Grafik 6.23 adalah perbandingan berdasarkan perbandingan nilai *idle.*



Grafik 6.23 Grafik Perbandingan *Idle* (*Tiny*)

Berdasarkan grafik 6.23 dan 6.24 tersebut dapat dilihat terjadinya ketidakseimbangan antara uji coba yang dilakukan pada *dataset.* Nilai *idle* lebih kecil memiliki pengaruh pada *dataset tiny*, sedangkan tidak demikian pada *dataset small 2*. Kesimpulannya adalah dipelukan uji coba dengan melakukan eksperimen lebih banyak untuk mendapatkan hasil yang lebih meyakinkan untuk penentuan nilai *idle* yang digunakan dan yang paling berpengaruh untuk parameter algoritma.

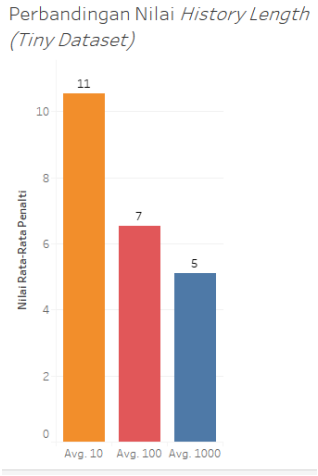


Grafik 6.24 Grafik Perbandingan *Idle* (*Small*)

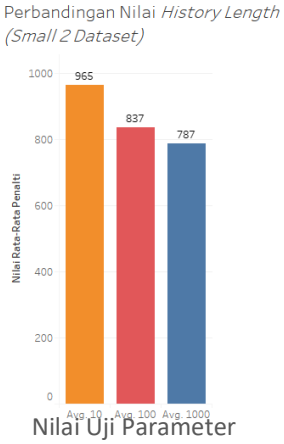
6.3.3 Perbandingan Hasil Uji Coba *History Length*

Percobaan-percobaan yang telah dilakukan untuk parameter *History length* perlu dilakukan perbandingan hasil untuk dapat mengetahui perubahan yang terjadi pada setiap percobaan. Hal yang akan dibandingkan adalah berdasarkan skenario yang telah dilakukan.

Grafik 6.24 dam 6.25 menunjukkan terjadinya penurunan pada nilai penalti dengan nilai *History length* yang lebih besar. Begitu pula yang ditunjukkan pada grafik yang menunjukkan penurunan pada nilai penalti dengan panjang yang lebih luas. Kesimpulannya adalah panjang *HL* menentukan nilai penalti yang dihasilkan dapat lebih baik dibandingkan jika panjang lebih dipersempit lagi.



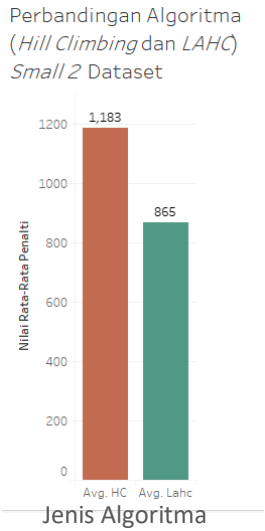
Grafik 6.25 Perbandingan *HL* (*Tiny*).



Grafik 6.26 Perbandingan *HL* (*Small*)

6.3.4 Perbandingan Algoritma *Hill climbing*

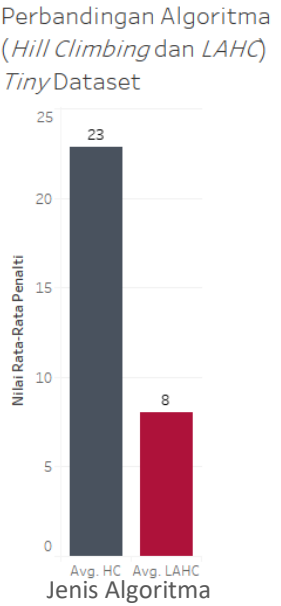
Perbandingan algoritma dilakukan untuk melihat performa yang dihasilkan oleh algoritma terpilih dalam hal ini adalah *LAHC*. Algoritma yang dipilih sebagai pembanding adalah *Hill climbing*. Algoritma tersebut dipilih karena memiliki kesamaan dengan *LAHC* dimana *LAHC* sendiri merupakan algoritma yang dikembangkan dari *Hill climbing*. Perbedaannya terletak pada acceptance dan *stopping criteria*. Uji coba menggunakan algoritma *Hill climbing* akan dilakukan pada kedua *dataset* dengan parameter pembandingnya adalah jumlah iterasi yang sama dengan jumlah iterasi yang digunakan pada *LAHC*.



Grafik 6.27 Perbandingan Nilai Penalti *Hill climbing* dan *LAHC*

Grafik 6.27 menunjukkan perbandingan hasil rata-rata dari nilai penalti menggunakan *Hill climbing* dan *LAHC* pada

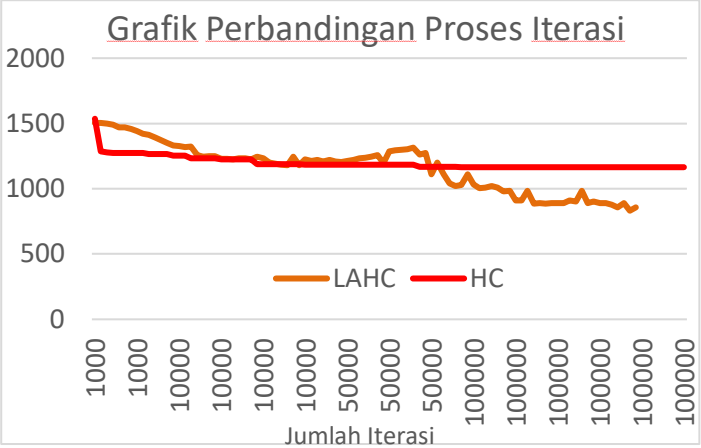
dataset jenis *small*. Berdasarkan grafik tersebut dapat dilihat bahwasannya nilai penalti dengan menggunakan *LAHC* jauh lebih rendah dibandingkan dengan nilai penalti yang dihasilkan dari algoritma *Hill climbing*. Hal ini menunjukkan bahwasanya *LAHC* mampu menghasilkan nilai penalti yang lebih bagus bila dibandingkan dengan nilai penalti *Hill climbing* dikarenakan fungsi tujuan dari model matematis pengerjaan tugas akhir ini adalah meminimalkan nilai penalti yang dikenakan pada masing-masing konten.



Grafik 6.28 Perbandingan Nilai Penalti *Hill climbing* dan *LAHC*

Grafik 6.28 juga menunjukkan perbandingan antara penggunaan algoritma *Hill climbing* pada *dataset* jenis *Tiny*.

Berdasarkan grafik tersebut dapat disimpulkan bahwasannnya nilai penalti dihasilkan oleh algoritma *LAHC* lebih baik untuk kedua jenis *dataset*. Gambar 6.29 juga menunjukkan hasil perbandingan proses iterasi algoritma *Hill Climbing* dan *LAHC* yang mana algoritma *HC* sangat cepat memiliki nilai *local optimum* dibandingkan *LAHC* yang mampu menerima nilai penalti lebih jelek.



Gambar 6.29 Performa Algoritma Hill Climbing dan LAHC

Maka dari itu kesimpulan yang dapat diambil adalah algoritma *LAHC* mampu menampilkan performa yang baik dalam menyelesaikan permasalahan timetabling. Dibuktikan dengan menurunnya nilai penalti dibandingkan dengan nilai penalti yang dihasilkan dari inisial solusi. Selain itu hasil perbandingan dengan algoritma *Hill climbing* juga menunjukkan bahwasannya nilai penalti yang dihasilkan oleh *LAHC* juga lebih baik.

6.4. Kesimpulan Hasil Uji Parameter

Hasil uji parameter untuk seluruh parameter yang telah dilakukan memiliki performa yang berbeda-beda untuk masing-masing nilai penalti yang dihasilkan. Tabel 6.4 menunjukkan hasil keseluruhan nilai rata-rata masing-masing parameter dari uji skenario yag telah dilakukan.

Tabel 6.4 Hasil Kesimpulan Nilai Penalti Uji Parameter *(LAHC)*

No	*Dataset*	Parameter			Nilai Rata-Rata Penalti
		Iterasi	*Idle*	*History Length*	
1	*Tiny*	10,000	0.02	10	11
2	*Tiny*	100,000	0.02	10	9
3	*Tiny*	500,000	0.02	10	9
4	*Tiny*	1,000,000	0.02	10	8
5	*Tiny*	100,000	0.2	10	14
6	*Tiny*	100,000	0.8	10	31
7	*Tiny*	100,000	0.02	100	7
8	*Tiny*	100,000	0.02	1000	5
9	*Small 2*	10,000	0.02	10	972
10	*Small 2*	100,000	0.02	10	965
11	*Small 2*	500,000	0.02	10	924
12	*Small 2*	1,000,000	0.02	10	865

No	*Dataset*	Parameter			Nilai Rata-Rata Penalti
		Iterasi	*Idle*	*History Length*	
13	*Small 2*	100,000	0.2	10	905
14	*Small 2*	100,000	0.8	10	913
15	*Small 2*	100,000	0.02	100	837
16	*Small 2*	100,000	0.02	1000	787

Berdasarkan tabel 6.4 dapat diketahui untuk dua *dataset* yang diuji coba, nilai paling minimum yang didapatkan dari hasil uji skenario adalah dengan menggunakan parameter apa saja. Untuk dataset jenis *Tiny*, ukuran parameter yang paling menghasilkan nilai penalti minimum adalah pada uji coba nomor 8 dengan ukuran parameter iterasi sebanyak 100,000, *idle* 0.02 dan *history length* sebesar 1000. Nilai penalti yang didapatkan adalah 5 poin. Sedangkan nilai paling minimum untuk dataset *small 2* didaptkan pada uji coba dengan ukuran parameter nomor 16 yaitu dengan banyak iterasi 100,000, nilai *idle* 0.02 dan *history length* 1000. Nilai penalti yang didapatkan adalah 787 poin. Namun demikian secara keseluruhan untuk parameter yang paling banyak memiliki pengaruh terhadap perubahan nilai paramter adalah jumlah iterasi yang dilakukan.

Berdasarkan uji parameter yang telah dilakukan dengan beberapa skenario tersebut, dapat disimpulkan beberapa poin sebagai berikut:

1. Jumlah Iterasi: parameter yang paling memiliki pengaruh terhadapt jumlah iterasi adalah uji coba yang dilakukan pada parameter jumlah iterasi. Terdapat perubahan yang signifikan terhadap nilai penalti yang diubah secara berkala dengan meningkatkan jumlah iterasinya. Hasil ini ditunjukkan dari skenario 2 yang telah dilakukan.
2. Idle: Nilai parameter idle tidak memberikan perubahan signifikan. Nilai idle yang paling ideal adalah nilai idle yang sesuai dengan *pseudocode* dari algoritma yaitu 0.02. Semakin mendektai 1 nilai *idle* akan meningkatkan penolakan semakin besar. Hasil ini ditunjukkan dari skenario 3 untuk masing-masing uji parameter yang telah dilakukan.
3. History Length: solusi terbaik dari uji parameter yang telah dilakukan didapatkan dari mengubah nilai history length dengan panjang 1000. Hal ini berlaku untuk kedua jenis dataset yang diuji. Hasil ini ditunjukkan dari skenario 4 untuk masing-masing uji parameter yang telah dilakukan.
4. Selain parameter jumlah iterasi, terdapat pula parameter history length yang juga memberikan pengaruh terhadapt perubahan nilai penalti. Hal ini terjadi dikarenakan nilai solusi yang tidak lebih bagus dapat diterima dan kemudian disimpan untuk memberikan nilai penalti yang lebih baik lagi.

BAB VII
KESIMPULAN DAN SARAN

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

7.1. Kesimpulan

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

1. Algoritma *Late acceptance Hill climbing* berbasis *Hyper-heuristics* mampu menyelesaikan permasalahan penjadwalan mata kuliah (*University Course Timetabling Problem*).
2. Struktur data yang efisien digunakan sebagai struktur data *hyper-heuristic* adalah struktur data berbasis *arraylist* dan pemrograman berbasis objek karena waktu komputasi yang relatif lebih cepat dan struktur data yang sangat kompleks.
3. Pada algoritma *Late acceptance Hill climbing Hyper-heuristics* tanpa pengembangan, jumlah *low level heuristics* dan jumlah iterasi berpengaruh pada solusi yang dihasilkan. Penggunaan dua jenis *low level heuristics* menghasilkan solusi yang semakin optimum daripada enam jenis *low level heuristic*. Sementara

semakin banyak iterasi, solusi yang dihasilkan semakin optimum.

4. Pada algoritma *Late acceptance Hill climbing Hyper-heuristics* dengan pengembangan, beberapa parameter mempengaruhi hasil.
   a. Jumlah iterasi yang menjadi kriteria batasan akan menghasilkan nilai penalti yang semakin baik dengan jumlah iterasi ditingkatkan lebih banyak.
   b. Perubahan kriteria yang menjadi batasan iterasi *idle* akan mempengaruhi nilai dengan meminimalkan nilai *idle* tersebut.
   c. Semakin panjang ukuran *History length* yang digunakan akan menghasilkan solusi yang lebih baik dibandingkan menggunakan yang lebih pendek.
5. Algoritma *LAHC* mampu memberikan performa yang baik dalam penyelesaian masalah timetabling bila dibandingkan dengan algoritma pembanding *Hill climbing*. Kemudian juga mampu melakukan optimasi terhadap nilai penalti yang dikenakan pada masing-masing konten.

7.2. Saran

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

1. Peneltian ini hanya dapat menjalankan 2 *dataset* dari total 15 *dataset* yang sudah dikeluarkan oleh pihak ITC 2019. Harapannya penelitian selanjutnya dapat

menyelesaikan lanjutan dari *dataset* yang belum berhasil diselesaikan hingga menemukan solusi inisial.

2. Penelitian hanya melakukan eksperimen sebanyak 11 kali iterasi untuk tiap-tiap skenario sehingga iterasi yang dilakukan tidak terlalu banyak. Harapannya dapat lebih banyak menghasilkan variasi.

3. Penelitian ini hanya menggunakan *low level heuristics* berupa *move*. Untuk mendapatkan hasil yang lebih variatif, *low level heuristics* dapat ditambah, sehingga memungkinan ditemukan solusi yang lebih optimum. Eksperimen total *low level heuristics* yang menghasilkan solusi lebih optimum juga dapat dilakukan pada penelitian selanjutnya.

4. Algoritma *Late acceptance Hill climbing* merupakan salah satu algoritma baru yang masih terus dikembangkan. Perlu banyak penelitian yang harus dilakukan untuk mendapatkan hasil bahwa algoritma ini mampu menghasilkan solusi sebaik algoritma simulated annealing.

Halam ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] A. Rochman, "Penjadwalan Kuliah Menggunakan Metode *Constraints* Programming Dan Simulated Annealing," Vol. 2012, No. Snati, Pp. 15–16, 2012.

[2] D. S. Informasi, "Optimasi Kuliah Otomatis Menggunakan Algoritma Tabu- Simulated Annealing Hyper-Heuristics Automated Course Timetabling Tabu-Simulated Annealing Hyper-Heuristics Algorithm."

[3] G. H. G. Fonseca, H. G. Santos, And E. G. Carrano, "*Late acceptance* Hill-Climbing For High School Timetabling," *J. Sched.*, Vol. 19, No. 4, Pp. 453–465, 2016.

[4] E. K. Burke And Y. Bykov, "The *Late acceptance* Hill-Climbing Heuristic," *Eur. J. Oper. Res.*, Vol. 258, No. 1, Pp. 70–78, 2017.

[5] A. Muklason, "Solver Penjadwal Ujian Otomatis Dengan Algoritma Maximal Clique Dan Hyper-Heuristics," Pp. 18–19, 2017.

[6] A. L. Aro Bolaji, A. F. Bamigbola, And P. B. Shola, "*Late acceptance Hill climbing* Algorithm For Solving Patient Admission Scheduling Problem," *Knowledge-Based Syst.*, Vol. 145, Pp. 1–14, 2018.

[7] 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.

[8] E. Özcan, Y. Bykov, M. Birben, And E. K. Burke, "Examination Timetabling Using *Late acceptance* Hyper-Heuristics," *2009 Ieee Congr. Evol. Comput. Cec 2009*, No. May 2014, Pp. 997–1004, 2009.

[9] F. Stefanello, "Heuristic Approaches For," Vol. 7, No. 1993, 2000.

[10] E. Riksakomara, *Tugas Akhir – Ks 141501*. 2017.
[11] J. Verstichel And G. Vanden Berghe, "Logistik Management," Pp. 457–458, 2009.

BIODATA PENULIS

Penulis yang memiliki nama lengkap Cut Alna Fadhilla ini, lahir di Bireuen pada tanggal 19 Februari 1998. Penulis merupakan anak pertama dari orang tua bernama Teuku Alamsyah dan Cut Marlina. Penulis menempuh pendidikan dasar di SD Negeri Suka Mulia Kec Rantau pada tahun 2003 hingga 2009. Setelah lulus dari sekolah dasar, penulis melanjutkan pendidikan formal di bangku sekolah menengah pertama di MTS Ulumul Qur'an Langsa YDBU dan lulus pada tahun 2012. Pada tahun yang sama, penulis melanjutkan pendidikan formal di MAS Ulumul Qur'an YDBU Langsa 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 kemitraan kementrian agama yaitu Program Beasiswa Santri Berprestasi (PBSB) tahun 2015.

Selama menjalani pendidikan tinggi di ITS, penulis aktif dalam berbagai kegiatan, baik organisasi maupun kepanitiaan dan juga kegiatan internasional. Penulis pernah mengikuti magang penelitian laboratorium di Univesiti Teknologi Malaysia pada tahun 2019 dan juga mengikuti *Winter School* di *National Chung Cheng University* Taiwan pada tahun 2018. Selain itu penulis juga aktif mengikuti kegiatan-kegiatan yang diselenggarakan oleh *International Office ITS*.

Penulis pernah menjadi staf CSSMoRA ITS selama 2 tahun menjabat yang merupakan komunitas penerima beasiswa PBSB. Pada tahun 2017, penulis pernah menjadi Anggota Selain itu, penulis juga aktif dalam organisasi mahasiswa tingkat departemen dan fakultas dengan menjadi staf Himpunan Mahasiswa Sistem Informasi dan Badan Eksekutif Mahasiswa Fakultas.

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.

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

LAMPIRAN A: Kebutuhan Data

Dataset Tiny (lums-sum17.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE problem PUBLIC "-//ITC 2019//DTD Problem Format/EN"
"http://www.itc2019.org/competition-format.dtd">
<problem name="lums-sum17" nrDays="7" slotsPerDay="288" nrWeeks="9">
  <optimization time="1" room="1" distribution="10" student="10"/>
  <rooms>
    <room id="1" capacity="58">
      <travel room="18" value="1"/>
      <travel room="19" value="1"/>
      <travel room="20" value="1"/>
      <travel room="21" value="1"/>
      <travel room="22" value="1"/>
      <travel room="61" value="1"/>
      <travel room="62" value="1"/>
0000100" start="96" length="120" weeks="000000100"/>
      <unavailable days="0000100" start="96" length="120" weeks="000000010"/>
      <unavailable days="0000010" start="96" length="120" weeks="000001000"/>
      <unavailable days="0000010" start="96" length="120" weeks="000000100"/>
      <unavailable days="0000010" start="96" length="120" weeks="000000010"/>
      <unavailable days="0000001" start="96" length="120" weeks="000001000"/>
      <unavailable days="0000001" start="96" length="120" weeks="000000100"/>
      <unavailable days="0000001" start="96" length="120" weeks="000000010"/>
    </room>
<courses>
    <course id="1">
      <config id="1">
        <subpart id="1">
          <class id="1" li mit="10">
            <room id="1" penalty="4"/>
            <room id="2" penalty="4"/>
            <room id="3" penalty="4"/>
            <room id="4" penalty="4"/>
            <room id="5" penalty="4"/>
            <time days="1111000" start="96" length="22" weeks="111111111"
penalty="0"/>
            <time days="1111000" start="108" length="22" weeks="111111111"
penalty="0"/>
penalty="12"/>
          </class>
        </subpart>
      </config>
    </course>
```

Halaman ini sengaja dikosongkan

LAMPIRAN B: Hasil Optimasi dalam Bentuk XML

1. Hasil Optimasi *Dataset Tiny* (lums-sum07.xml)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<solution author="Cut Alna" institution="ITS" name="Tiny Result" technique="Late acceptance Hill climbing">
<class days="1111000" id="1" length="22" room="45" start="120" weeks="111111111">
<student/>
</class>
<class days="1111000" id="2" length="15" room="23" start="96" weeks="111111111">
<student/>
</class>
<class days="1111000" id="3" length="15" room="8" start="96" weeks="111111111">
<student/>
</class>
<class days="1010000" id="4" length="34" room="62" start="132" weeks="111111111">
<student/>
</class>
<class days="1111000" id="5" length="15" room="12" start="96" weeks="111111111">
<student/>
</class>
<class days="1111000" id="6" length="22" room="41" start="138" weeks="111111111">
<student/>
</class>
<class days="1111000" id="7" length="22" room="28" start="120" weeks="111111111">
    <student/>
</class>
  <class days="1111000" id="8" length="22" room="23" start="120" weeks="111111111">
    <student/>
  </class>
```

```
   <class days="1111000" id="9" length="22" room="24"
start="132" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="10" length="22" room="25"
start="120" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="11" length="22" room="25"
start="96" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="12" length="15" room="26"
start="120" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="13" length="22" room="38"
start="144" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="14" length="15" room="8"
start="144" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="15" length="22" room="38"
start="168" weeks="111111111">
      <student/>
   </class>
   <class days="0101000" id="16" length="58" room="62"
start="96" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="17" length="22" room="38"
start="96" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="18" length="22" room="26"
start="96" weeks="111111111">
```

```
      <student/>
   </class>
   <class days="1111000" id="19" length="22" room="8"
start="114" weeks="111111111">
      <student/>
   </class>
   <class days="1111000" id="20" length="22" room="24"
start="96" weeks="111111111">
      <student/>
   </class>
</solution>
```

Hasil optimasi dalam bentuk tabel.

CourseID	**Weeks**	**Days**	**Start**	***Length***	**Room**
1	1-9	Senin Selasa Rabu Kamis	96	15	45
2	1-9	Senin Selasa Rabu Kamis	96	15	23
3	1-9	Senin Selasa Rabu Kamis	96	15	8
4	1-9	Senin Rabu	132	34	62
5	1-9	Senin Selasa Rabu Kamis	96	15	12
6	1-9	Senin	138	22	41

		Selasa Rabu Kamis			
7	1-9	Senin Selasa Rabu Kamis	120	22	28
8	1-9	Senin Selasa Rabu Kamis	120	22	23
9	1-9	Senin Selasa Rabu Kamis	132	22	24
10	1-9	Senin Selasa Rabu Kamis	120	22	25
11	1-9	Senin Selasa Rabu Kamis	96	22	25
12	1-9	Senin Selasa Rabu Kamis	120	15	25
13	1-9	Senin Selasa Rabu Kamis	144	22	26
14	1-9	Senin	144	15	8

		Selasa Rabu Kamis			
15	1-9	Senin Selasa Rabu Kamis	168	22	38
16	1-9	Selasa Kamis	96	58	62
17	1-9	Senin Selasa Rabu Kamis	96	22	62
18	1-9	Senin Selasa Rabu Kamis	96	22	26
19	1-9	Senin Selasa Rabu Kamis	114	22	8
20	1-9	Senin Selasa Rabu Kamis	96	22	24