Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - IS184853

PENYELESAIAN PERMASALAHAN OTOMASI DAN OPTIMASI PENJADWALAN MATA KULIAH MENGGUNAKAN ALGORITMA *ITERATED LOCAL SEARCH HYPER-HEURISTIC* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION* 2019

AUTOMATED AND OPTIMIZATION COURSE TIMETABLING USING ITERATED LOCAL SEARCH HYPER-HEURISTIC ALGORITHM WITH THE PROBLEM DOMAIN OF INTERNATIONAL TIMETABLING COMPETITION 2019

UMAR RIZKI KUSUMO WIDAYU
NRP 05211540000096

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

PENYELESAIAN PERMASALAHAN OTOMASI DAN OPTIMASI PENJADWALAN MATA KULIAH MENGGUNAKAN ALGORITMA *ITERATED LOCAL SEARCH HYPER-HEURISTIC* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION* 2019

UMAR RIZKI KUSUMO WIDAYU
05211540000096

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

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

UNDERGRADUATE THESIS - IS184853

AUTOMATED AND OPTIMIZATION COURSE TIMETABLING USING ITERATED LOCAL SEARCH HYPER-HEURISTIC ALGORITHM WITH THE PROBLEM DOMAIN OF INTERNATIONAL TIMETABLING COMPETITION 2019

UMAR RIZKI KUSUMO WIDAYU
05211540000096

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

INFORMATION SYSTEM DEPARTMENT
Information Technology and Communication Faculty
Sepuluh Nopember Institute of Technology
Surabaya 2019

LEMBAR PENGESAHAN

PENYELESAIAN PERMASALAHAN OTOMASI DAN OPTIMASI PENJADWALAN MATA KULIAH MENGGUNAKAN ALGORITMA ITERATED LOCAL SEARCH HYPER-HEURISTIC 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:

UMAR RIZKI KUSUMO WIDAYU
NRP. 05211540000096

Surabaya, Juli 2019

KEPALA
DEPARTEMEN SISTEM INFORMASI

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

LEMBAR PERSETUJUAN

PENYELESAIAN PERMASALAHAN OTOMASI DAN OPTIMASI PENJADWALAN MATA KULIAH MENGGUNAKAN ALGORITMA *ITERATED LOCAL SEARCH HYPER-HEURISTIC* 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 :

UMAR RIZKI KUSUMO WIDAYU
NRP. 05211540000096

Disetujui Tim Penguji : Tanggal Ujian : 09 Juli 2019
Periode Wisuda : September 2019

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

Edwin Riksakomara, S.Kom., M.T **(Penguji I)**

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

DEPARTEMEN SISTEM INFORMASI

PENYELESAIAN PERMASALAHAN OTOMASI DAN OPTIMASI PENJADWALAN MATA KULIAH MENGGUNAKAN ALGORITMA *ITERATED LOCAL SEARCH HYPER-HEURISTIC* DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION* 2019

Nama Mahasiswa	**:**	**Umar Rizki Kusumo Widayu**
NRP	**:**	**05211540000096**
Departemen	**:**	**Sistem Informasi**
Dosen Pembimbing	**:**	**Ahmad Muklason, S.Kom., M.Sc., Ph.D.**

ABSTRAK

Penjadwalan mata kuliah pada perguruan tinggi merupakan permasalahan yang NP-Hard dan menjadi topik yang menarik untuk dikaji karena belum ada algoritma eksak yang bisa menyelesaikan dalam waktu polynomial. Hal ini menjadi perhatian hingga kalangan Internasional hingga diadakannya kompetisi penjadwalan dengan skala Internasional. International Timetabling Competition adalah kompetisi terkait topik penjadwalan bertaraf Internasional dan dalam penelitian ini khususnya adalah International Timetabling Competition 2019 yang mana merupakan kompetisi keempat yang telah diselenggarakan. Tujuan dari tugas akhir ini adalah untuk menyelesaikan permasalahan penjadwalan mata kuliah pada kompetisi International Timetabling Competition (ITC) 2019. Algoritma yang akan digunakan untuk menyelesaikan permasalahan ini adalah algoritma Iterated Local Search dalam kerangka kerja Hyperheuristic. Solusi awal yang dihasilkan pada dataset tiny adalah 53, setelah itu dilakukan optimasi menggunakan ILS-HC hingga hasil rata-ratanya menjadi 18,3 dan juga menggunakan ILS-SA dengan rata-rata hasilnya 8,6. Solusi awal yang dihasilkan dataset small adalah 1790, ketika dilakukan optimasi menggunakan ILS-HC nilai rata-ratanya menjadi 1069,8 dan ketika menggunakan ILS-SA nilai rata-ratanya adalah 778. Algoritma ILS memiliki

performa yang lebih baik ditunjukkan dengan nilai fungsi tujuan pada tiap iterasi adalah lebih baik jika dibandingkan dengan Hill Climbing.

Kata Kunci: Penjadwalan mata kuliah, International Timetabling Competition 2019, Iterated Local Search, Hyperheuristic

AUTOMATED AND OPTIMIZATION COURSE TIMETABLING USING ITERATED LOCAL SEARCH HYPER-HEURISTIC ALGORITHM WITH THE PROBLEM DOMAIN OF INTERNATIONAL TIMETABLING COMPETITION 2019

Name	**:**	**Umar Rizki Kusumo Widayu**
NRP	**:**	**05211540000096**
Department	**:**	**Information Systems**
Supervisior	**:**	**Ahmad Muklason, S.Kom., M.Sc., Ph.D.**

ABSTRACT

Timetabling in higher education is a problem that NP-Hard is an interesting topic to study because there is no exact algorithm that can solve in polynomial time. This is a concern for the international community until the scheduling competition with an international scale. The International Timetabling Competition is a competition related to international level scheduling topics and in this research in particular is the International Timetabling Competition 2019 which is the fourth competition that has been held. The purpose of this final project is to solve the problem of scheduling courses at the 2019 International Timetabling Competition (ITC) competition. The algorithm that will be used to solve this problem is the Iterated Local Search algorithm within the Hyperheuristic framework. The initial solution generated in the tiny dataset is 53, after optimization is done with ILS-HC until the average value is 18,3 then with ILS-SA until the average value is 8,6. The initial solution produced by the small dataset is 1790, after optimization is done with. ILS-HC until the average value is 1069,8 then with ILS-SA until the average value is 778. Comparing to Hill Climbing, ILS has better objective function value. Therefore, ILS method is way better than Hill Climbing.

Keywords: University Timetabling, International Timetabling Competition 2019, Iterated Local Search, Hyperheuristic

KATA PENGANTAR

Dengan mengucapkan rasa syukur kepada Tuhan Yang Maha Pengasih dan Maha Penyayang atas izin-Nya penulis dapat menyelesaikan buku yang sederhana ini dengan judul Penyelesaian Permasalahan Otomasi Dan Optimasi Penjadwalan Mata Kuliah Menggunakan Algoritma Iterated Local Search Hyper-Heuristic Dengan Domain Permasalahan Dari International Timetabling Competition 2019. Dalam penyelesaian Tugas Akhir ini, penulis diiringi oleh pihak-pihak yang selalu memberi dukungan, saran, dan doa sehingga penelitian berlangsung dengan lancar. Secara khusus penulis mengucapkan terima kasih dari lubuk hati terdalam kepada:

1. Tuhan, yang selalu menemani dan membimbing penulis dalam segala aspek kehidupan.
2. Ibu Mahendrawathi ER,. S.T., M.Sc., Ph.D. selaku Ketua Departemen Sistem Informasi ITS Surabaya.
3. Bapak Ahmad Muklason, S.Kom., M.Sc., Ph.D. selaku dosen pembimbing yang telah mencurahkan segenap tenaga, waktu dan pikiran dalam penelitian ini, serta memberikan motivasi yang membangun.
4. Bapak Edwin Riksakomara, S.Kom., M.T dan Ibu Retno Aulia Vinarti, S.Kom., M.Kom., Ph.D. selaku dosen penguji yang telah memberikan kritik dan saran yang membuat kualitas penelitian ini lebih baik lagi.
5. Segenap dosen dan karyawan Departemen Sistem Informasi.
6. Orang tua penulis, yang tiada hentinya mendoakan dan memberikan dukungan kepada penulis.
7. Narendra, Kharisma, dan Cut Alna selaku tim ITC yang telah berjuang Bersama untuk menerjang penelitian ini hingga selesai.
8. Teman-teman Lannister, grup kocheng, teman-teman RDIB, dan teman-teman MSE yang senantiasa memberi semangat dan dukungan.

9. Dan seluruh pihak yang telah membantu penulis dalam mengerjakan tugas akhir ini yang tidak mungkin disebutkan satu per satu.

Penyusunan tugas akhir ini masih jauh dari kata sempurna, untuk itu penulis menerima segala kritik dan saran yang membangun sebagai upaya menjadi lebih baik lagi ke depannya. Semoga buku tugas akhir ini dapat memberikan manfaat untuk pembaca.

Surabaya, 09 Juli 2019

Penulis

DAFTAR ISI

LEMBAR PENGESAHAN.... **Error! Bookmark not defined.**
LEMBAR PERSETUJUAN... **Error! Bookmark not defined.**
ABSTRAK v
ABSTRACT vii
KATA PENGANTAR ix
DAFTAR ISI xi
DAFTAR GAMBAR xv
DAFTAR TABEL xvii
DAFTAR KODE PROGRAM xix
1 BAB I PENDAHULUAN 1
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Permasalahan 3
1.4 Tujuan 3
1.5 Manfaat 3
1.6 Relevansi 4
2 BAB II TINJAUAN PUSTAKA 5
2.1 Penelitian Sebelumnya 5
2.2 Dasar Teori 7
2.2.1 Penjadwalan 7
2.2.2 *International Timetabling Competition 2019 Dataset* 7
2.2.3 *Hyperheuristic* 11
2.2.4 *Iterated Local Search* 13
3 BAB III METODOLOGI PENELITIAN 15
3.1 Pemahaman Permasalahan ITC2019 15
3.2 Studi Literatur 15
3.3 Pemahaman Dataset ITC2019 16
3.4 Permodelan 16
3.5 Implementasi Algoritma Iterated Local Search - Hyperheuristic 16
3.6 Uji Coba Algoritma 17
3.7 Analisis Hasil dan Kesimpulan 17
3.8 Pengerjaan Laporan Tugas Akhir 17

4 BAB IV PERANCANGAN 19
4.1 Pemahaman Data 19
4.2 Formulasi Model Matematis 19
4.2.1 Fungsi Tujuan 19
4.2.2 Variabel Keputusan 21
4.2.3 Distribution Constraints 22
4.3 Pembentukan Initial Solution 32
4.3.1 Pembacaan *File* 33
4.3.2 Penyimpanan Konten *Rooms* 33
4.3.3 Penyimpanan Konten *Courses* 33
4.3.4 Penyimpanan Konten *Distribution Constraint* 33
4.3.5 Pembuatan Kode Program *Hard Constraint* 34
4.3.6 Pembentukan *Initial Solution* 34
4.3.7 Pembuatan Kode Program *Soft Constraint* 34
4.4 Pembentukan *Low Level Heuristic* 34
4.5 Implementasi Algoritma *Iterated Local Search* 35
5 BAB V IMPLEMENTASI 37
5.1 Pemahaman Data 37
5.1.1 Inisiasi Konten Dataset 37
5.1.2 Inisiasi Konten *Rooms* 38
5.1.3 Inisiasi Konten *Courses* 39
5.1.4 Inisiasi Konten Student 40
5.1.5 Inisiasi Konten Distribution Constraint 40
5.2 Pembacaan File Input 41
5.3 Penyimpanan Konten Rooms 41
5.4 Penyimpanan Konten Courses 42
5.5 Pembuatan Kode Program Hard Constraint 43
5.6 Pembuatan Initial Solution 58
5.7 Pembuatan Kode Program Soft Constraint 60
5.8 Pembuatan Low Level Heuristic 74
5.9 Implementasi Algoritma Iterated Local Search 76
5.9.1 Algoritma Local Search 76
5.9.2 Perturbation 77
5.9.3 Perhitungan Penalti Akhir 79
6 BAB VI HASIL DAN PEMBAHASAN 81
6.1 Data Uji Coba 81
6.2 Pembuatan Initial Solution 81

6.2.1 Skenario 1 Indikator 81
6.2.2 Skenario 2 Indikator 82
6.2.3 Skenario 3 Indikator 83
6.3 Hasil Initial Solution 84
6.4 Hasil Eksperimen Algoritma Iterated Local Search 84
6.4.1 Iterated Local Search dengan Hill Climbing 85
6.4.1.1 Skenario 1 85
6.4.1.2 Skenario 2 91
6.4.1.3 Skenario 3 95
6.4.1.4 Skenario 4 100
6.4.1.5 Skenario 5 112
6.4.1.6 Skenario 6 117
6.4.1.7 Kesimpulan Skenario ILS-HC..... 121
6.4.2 Iterated Local Search dengan Simmulated Annealing..... 122
6.4.2.1 Skenario 1 122
6.4.2.2 Skenario 2 123
6.4.2.3 Skenario 3 124
6.4.2.4 Kesimpulan Skenario ILS-SA 125
6.4.3 Pembahasan Hasil Skenario 126
6.5 Perbandingan Hasil Eksperimen dengan Algoritma Lain 127
7 BAB VII KESIMPULAN DAN SARAN..... 131
7.1 Kesimpulan 131
7.2 Saran 132
8 DAFTAR PUSTAKA 135
9 BIODATA PENULIS 137

DAFTAR GAMBAR

Gambar 1.1 Bidang Keilmuan Laboratorium RDIB 4
Gambar 2.1 Spesifikasi dari Distribution Constraints..... 10
Gambar 2.2 Framework Hyper-heuristics 13
Gambar 2.3 Pseudocode Iterated Local Search 14
Gambar 3.1 Metodologi Pengerjaan Tugas Akhir 15
Gambar 6.1 Grafik Iterasi ILS Small..... 88
Gambar 6.2 Grafik Iterasi ILS Tiny 89
Gambar 6.3 Box Plot Iterasi ILS Small 90
Gambar 6.4 Box Plot Iterasi ILS Tiny..... 90
Gambar 6.5 Box Plot Iterasi LS Tiny 93
Gambar 6.6 Box Plot Iterasi LS Small 93
Gambar 6.7 Grafik Iterasi LS Tiny..... 94
Gambar 6.8 Grafik Iterasi ILS Small..... 94
Gambar 6.9 Perbandingan Algoritma (Dataset Tiny) 128
Gambar 6.10 Perbandingan Algoritma (Dataset Small) 129

DAFTAR TABEL

Tabel 2.1 Penelitian Sebelumnya 1 5
Tabel 2.2 Penelitian Sebelumnya 2 6
Tabel 2.3 Penelitian Sebelumnya 3 6
Tabel 2.4 Karakteristik Test Dataset dan Early Dataset 8
Tabel 2.5 Macam-macam constraint pada ITC2019..... 10
Tabel 4.1 Macam-Macam Constraint pada Dataset Small2... 22
Tabel 6.1 Hasil Skenario 1 Pembuatan Initial Solution 81
Tabel 6.2 Hasil Skenario 2 Pembuatan Initial Solution 82
Tabel 6.3 Hasil Skenario 3 Pembuatan Initial Solution 83
Tabel 6.4 Hasil Initial Solution Awal..... 84
Tabel 6.5 Parameter Percobaan 85
Tabel 6.6 Hasil Skenario 1 (Iterasi 10, 50, dan 100) 86
Tabel 6.7 Hasil Skenario 1 (Iterasi 500, 1.000, dan 1.500).... 86
Tabel 6.8 Hasil Skenario 1 (Iterasi 2.000, 5.000, 10.000) 87
Tabel 6.9 Hasil Skenario 2 (Iterasi 2, 5, dan 7) 91
Tabel 6.10 Hasil Skenario 2 (Iterasi 10, 20, dan 50) 92
Tabel 6.11 LLH yang digunakan..... 95
Tabel 6.12 Kombinasi 5 LLH 1..... 96
Tabel 6.13 Kombinasi 5 LLH 2..... 96
Tabel 6.14 Kombinasi 5 LLH 3..... 97
Tabel 6.15 Kombinasi 5 LLH 4..... 98
Tabel 6.16 Kombinasi 5 LLH 5..... 99
Tabel 6.17 Kombinasi 5 LLH 6..... 99
Tabel 6.18 Kombinasi 4 LLH 1..... 101
Tabel 6.19 Kombinasi 4 LLH 2..... 101
Tabel 6.20 Kombinasi 4 LLH 3..... 102
Tabel 6.21 Kombinasi 4 LLH 4..... 103
Tabel 6.22 Kombinasi 4 LLH 5..... 103
Tabel 6.23 Kombinasi 4 LLH 6..... 104
Tabel 6.24 Kombinasi 4 LLH 7..... 105
Tabel 6.25 Kombinasi 4 LLH 8..... 106
Tabel 6.26 Kombinasi 4 LLH 9..... 106
Tabel 6.27 Kombinasi 4 LLH 10..... 107
Tabel 6.28 Kombinasi 4 LLH 11..... 108
Tabel 6.29 Kombinasi 4 LLH 12..... 109

Tabel 6.30 Kombinasi 4 LLH 13..... 109
Tabel 6.31 Kombinasi 4 LLH 14..... 110
Tabel 6.32 Kombinasi 4 LLH 15..... 111
Tabel 6.33 Kombinasi 3 LLH 1..... 112
Tabel 6.34 Kombinasi 3 LLH 2..... 113
Tabel 6.35 Kombinasi 3 LLH 3..... 113
Tabel 6.36 Kombinasi 3 LLH 4..... 114
Tabel 6.37 Kombinasi 3 LLH 5..... 115
Tabel 6.38 Kombinasi 3 LLH 6..... 116
Tabel 6.39 Kombinasi 3 LLH 7..... 116
Tabel 6.40 Kombinasi 2 LLH 1..... 118
Tabel 6.41 Kombinasi 2 LLH 2..... 118
Tabel 6.42 Kombinasi 2 LLH 3..... 119
Tabel 6.43 Kombinasi 2 LLH 4..... 120
Tabel 6.44 Kombinasi 2 LLH 5..... 121
Tabel 6.45 Kesimpulan Skenario 121
Tabel 6.46 Hasil Skenario 1 ILS-SA 122
Tabel 6.47 Hasil Skenario 2 ILS-SA 123
Tabel 6.48 Hasil Skenario 3 ILS-SA 125
Tabel 6.49 Kesimpulan ILS-SA 125

DAFTAR KODE PROGRAM

Kode Program 5.1 Inisiasi Jumlah *Days, Weeks, d*an *Timeslots* 38
Kode Program 5.2 Inisiasi pembobotan *penalty* 38
Kode Program 5.3 Kode Kontan Ruangan 39
Kode Program 5.4 Kode konten Courses 39
Kode Program 5.5 Kode konten student 40
Kode Program 5.6 Contoh XML dari Distribution Constraint 40
Kode Program 5.7 Script Pembacaan File Input 41
Kode Program 5.8 Script Penyimpanan Konten Room 42
Kode Program 5.9 Script Penyimpanan Konten Course 43
Kode Program 5.10 Script Hard Constraint 1 44
Kode Program 5.11 Script Hard Constraint 2 45
Kode Program 5.12 Script Hard Constraint 3 46
Kode Program 5.13 Script Hard Constraint 4 47
Kode Program 5.14 Script Hard Constraint 5 48
Kode Program 5.15 Script Hard Constraint 6 48
Kode Program 5.16 Script Hard Constraint 7 49
Kode Program 5.17 Script Hard Constraint 8 49
Kode Program 5.18 Script Hard Constraint 9 50
Kode Program 5.19 Script Hard Constraint 10 51
Kode Program 5.20 Script Hard Constraint 11 51
Kode Program 5.21 Script Hard Constraint 12 52
Kode Program 5.22 Script Hard Constraint 13 53
Kode Program 5.23 Script Hard Constraint 14 54
Kode Program 5.24 Script Hard Constraint 15 55
Kode Program 5.25 Script Hard Constraint 16 56
Kode Program 5.26 Script Count Non Zero Bits 56
Kode Program 5.27 Script Hard Constraint 17 56
Kode Program 5.28 Script Pengecekan Dayload 57
Kode Program 5.29 Script Hard Constraint 18 57
Kode Program 5.30 Script Hard Constraint 19 58
Kode Program 5.31 Script Initial Solution 58
Kode Program 5.32 Script Cek Available Timeslot 59
Kode Program 5.33 Script Menjadwalkan Kelas 59
Kode Program 5.34 Script Menghapus Jadwal Kelas 60

Kode Program 5.35 Menghitung Total Penalty Soft Constraint 61
Kode Program 5.36 Script Soft Constraint 1 62
Kode Program 5.37 Script Soft Constraint 2 63
Kode Program 5.38 Script Soft Constraint 3 64
Kode Program 5.39 Script Soft Constraint 4 65
Kode Program 5.40 Script Soft Constraint 5 65
Kode Program 5.41 Script Soft Constraint 6 66
Kode Program 5.42 Script Soft Constraint 7 66
Kode Program 5.43 Script Soft Constraint 8 67
Kode Program 5.44 Script Soft Constraint 9 67
Kode Program 5.45 Script Soft Constraint 10 68
Kode Program 5.46 Script Soft Constraint 11 68
Kode Program 5.47 Script Soft Constraint 12 69
Kode Program 5.48 Script Soft Constraint 13 70
Kode Program 5.49 Script Soft Constraint 14 70
Kode Program 5.50 Script Soft Constraint 15 71
Kode Program 5.51 Script Soft Constraint 16 71
Kode Program 5.52 Script Count Non Zero Bits 72
Kode Program 5.53 Script Soft Constraint 17 72
Kode Program 5.54 Script Pengecekan Dayload 72
Kode Program 5.55 Script Soft Constraint 18 73
Kode Program 5.56 Script Soft Constraint 19 74
Kode Program 5.57 Script Low Level Heuristic moveTS 76
Kode Program 5.58 LLH Local Search 77
Kode Program 5.59 Algoritma penerimaan solusi 79
Kode Program 5.60 Menghitung Penalti dari Solusi Baru 79
Kode Program 5.61 Menerima Penalti Terbaru 79

BAB I
PENDAHULUAN

Pada bagian pendahuluan ini terdapat penjelasan mengenai latar belakang masalah, perumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, serta relevansi tugas akhir dengan laboratorium RDIB. Berdasarkan uraian pada bab ini diharapkan gambaran umum mengenai permasalahan dan pemecahan masalah pada tugas akhir dapat dipahami.

1.1 Latar Belakang

Proses belajar mengajar pada instansi pendidikan pastinya tidak lepas dari yang namanya penjadwalan terkhususkan pada perguruan tinggi yang membutuhkan penjadwalan mata kuliah, dimana penjadwalan ini merupakan hal yang sangat vital. Penjadwalan yang baik merupakan penjadwalan yang dapat dilakukan atau dipenuhi oleh pihak yang terkait dalam kegiatan belajar mengajar, baik dosen yang mengajar maupun mahasiswa yang mengambil mata kuliah tersebut. Banyak kendala yang dihadapi ketika menyusun sebuah penjadwalan yang baik. Penjadwalan mata kuliah di sebuah perguruan tinggi merupakan masalah yang sulit untuk dipecahkan[1].

Permasalahan penjadwalan merupakan permasalahan kombinatorial yang rumit karena memiliki pilihan alternatif solusi yang luas dan banyak dijumpai lokal optimal. Permasalahan tersebut menjadi salah satu permasalahan kombinatorial yang mendapatkan banyak perhatian dari para peneliti. Beberapa diantaranya membuktikan bahwa permasalahan tersebut bertipe *NP-hard* (*non deterministic polynomial -time hard*) atau tipe permasalahan yang sulit untuk diselesaikan untuk ukuran yang besar[2] dan hal ini menarik untuk dikaji.

Salah satu kompetisi terkait permasalahan penjadwalan berbasis Internasional adalah *International Timetabling Competition* (ITC). ITC pernah diselenggarakan selama 4 kali, yaitu

ITC2002, ITC2007, ITC2011, dan ITC2019. Pada kompetisi pertama (ITC2002) berfokus pada versi sederhana dari permasalahan penjadwalan mata kuliah di universitas. Kompetisi kedua (ITC2007) mampu mengenalkan 3 trek yaitu *curriculum-based timetabling, post-enrollment timetabling,* dan *examination timetabling*. Untuk kompetisi *timetabling* ketiga (ITC2011) memiliki fokusan pada permasalahan penjadwalan sekolah menengah. Sedangkan untuk kompetisi keempat (ITC2019) berfokus pada permasalahan penjadwalan mata kuliah[3] dan juga yang nantinya akan menjadi pokok bahasan pada penelitian ini.

Dalam penyelesaian permasalahan penjadwalan sendiri terdapat berbagai macam metode atau pendekatan untuk menyelesaikannya, namun yang akan digunakan adalah pendekatan *Iterated Local Search* yang dikembangkan dalam kerangka kerja *Hyperheuristic*. Pada penelitian sebelumnya [4][5], penggunaan metode *Iterated Local Search* telah dibuktikan lebih optimal dibanding dengan metode lainnya dan hasil pada penelitian [4] menunjukkan bahwa solusi yang dihasilkan lebih baik dari pada penelitian-penelitian sebelumnya.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, permasalahan yang akan diangkat pada penelitian tugas akhir ini adalah sebagai berikut:

1. Bagaimana model matematis terkait permasalahan penjadwalan mata kuliah dengan studi kasus *International Timetabling Competition 2019*?
2. Bagaimana menerapkan algoritma *Iterated Local Search* untuk menyelesaikan permasalahan penjadwalan mata kuliah dalam kerangka kerja *hyper-heuristics*?
3. Bagaimana performa algoritma *Iterated Local Search* dalam menyelesaikan permasalahan penjadwalan mata kuliah?

1.3 Batasan Permasalahan

Berdasarkan deskripsi permasalahan diatas, adapun batasan masalah dari penelitian tugas akhir ini adalah sebagai berikut:

1. Data yang digunakan dalam penelitian tugas akhir ini adalah *International Timetabling Competition 2019 dataset*.
2. Pengerjaan tugas akhir hanya sampai menjadwalkan mata kuliah, tanpa menjadwalkan *student* atau mahasiswa ke dalam kelas setiap mata kuliah.
3. Aplikasi dibangun dan dikembangkan menggunakan Bahasa Java.

1.4 Tujuan

Tujuan dari penelitian tugas akhir ini adalah sebagai berikut:

1. Membuat model matematis terkait permasalahan penjadwalan mata kuliah dengan menggunakan *dataset* dari *International Timetabling Competition 2019*.
2. Menerapkan *Iterated Local Search – Hyperheuristic* untuk menyelesaikan permasalahan penjadwalan mata kuliah pada *International Timetabling Competition 2019*.
3. Menganalisa hasil performa *Iterated Local Search – Hyperheuristic* dalam penyelesaian permasalahan penjadwalan mata kuliah pada *International Timetabling Competition 2019.*

1.5 Manfaat

Manfaat yang dapat diperoleh dari tugas akhir ini adalah sebagai berikut:

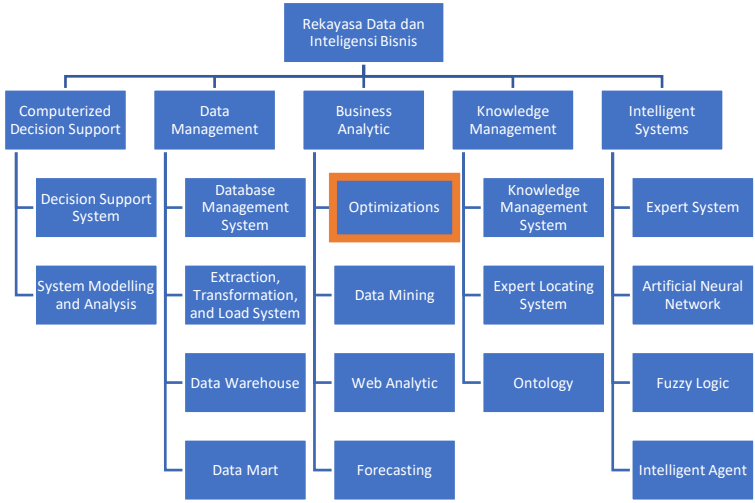
1. Manfaat Akademis
   Memberikan referensi atau rujukan penelitian yang mengimplementasikan metode optimasi khususnya metode *Iterated Local Search - Hyperheuristic*

sehingga dapat dijadikan sebagai acuan dalam penelitian berikutnya.

2. Bagi Penulis
   Menambah wawasan dan pengalaman dalam bidang keprofesian Sistem Informasi khususnya terkait optimasi penjadwalan mata kuliah yang menjadi topik pada penelitian tugas akhir ini.

1.6 Relevansi

Penelitian yang dilakukan pada tugas akhir memiliki topik tentang Optimasi dan berkaitan dengan mata kuliah Optimasi Kombinatorik dan Heuristik dari Departemen Sistem Informasi ITS yang tercakup dalam laboratorium Rekayasa Data dan Intelegensi Bisnis (RDIB). Bidang keilmuan pada penelitian tugas akhir ini sesuai dengan bidang keilmuan yang terdapat pada laboratorium RDIB. Bidang keilmuan tersebut adalah *Business Analytic* dengan topik *Optimizations*, sehingga dapat disimpulkan bahwa penelitian tugas akhir ini memiliki relevansi yang sesuai dengan bidang keilmuan laboratorium RDIB.



Gambar 1.1 Bidang Keilmuan Laboratorium RDIB

BAB II
TINJAUAN PUSTAKA

Pada bagian ini akan dijelaskan mengenai studi literatur yang digunakan sebagai landasan dan referensi objek penelitian tugas akhir serta teori-teori yang mendukung.

2.1 Penelitian Sebelumnya

Pada bagian ini terdapat penelitian sebelumnya yang berhubungan dengan tugas akhir ini dimana akan dijelaskan melalui Tabel 2.1 hingga Tabel 2.3 sebagai berikut.

Tabel 2.1 Penelitian Sebelumnya 1

Judul	Iterated local search using an add and delete hyper-heuristic for university course timetabling[4]
Penulis / tahun	Jorge A. Soria-Alcaraz, Ender Özcan, Jerry Swan, Graham Kendall, Martin Carpio/ 2015
Deskripsi Umum Penelitian	Penelitian ini membahas terkait penyelesaian permasalahan *timetabling*, menggunakan dua domain yang berbeda yaitu *post-enrollment university course timetabling* dan *curriculum-based university course timetabling* dengan data dari *International Timetabling Competition* 2007 (ITC2007) track 2 dan 3. Penelitian ini menggunakan algoritma *Iterated Local Search* (ILS) yang dikombinasikan dengan *hyperheuristic* berdasarkan operasi *add-delete*.
Hasil Penelitian	Hasil penelitian ini menunjukkan bahwa penggunaan algoritma *Iterated Local Search* dengan pendekatan *hyperheuristic* berdasarkan operasi *add-delete mencapai hasil yang kompetitif bila dibandingkan dengan penelitian lainnya dan pada beberapa kasus menunjukkan hasil yang lebih baik pada Track 3.*
Relevansi Penelitian	Penelitian ini memiliki pembuktian bahwa metode *Iterated Local Search* lebih unggul dan dapat dijadikan sebagai metode untuk penelitian ini.

Tabel 2.2 Penelitian Sebelumnya 2

Judul	Using Iterated Local Search to Solve the Course Timetabling Problem at Engineering Faculty of Necmettin Erbakan University[6]
Penulis / tahun	Kemal Alaykiran, Mehmet Hacibeyoglu/ 2016
Deskripsi Umum Penelitian	Penelitian ini membahas mengenai permasalahan *timetabling* yang diselesaikan menggunakan metode *Iterated Local Search*. Studi kasus yang digunakan adalah permasalahan pada Fakultas Teknik Universitas Necmettin Erbakan yang berlokasi di Konya, Turki dan menggunakan data semester musim gugur tahun 2016-2017. Permasalahan *timetabling* pada penelitian ini mengadopsi *hard constraint* dan *soft constraint* dari *International Timetabling Competition 2007*. Tujuan dari penelitian ini adalah untuk menganalisis ketersediaan jadwal yang akurat dengan lebih sedikit ruang kelas yang digunakan atau *timeslots* per hari yang lebih sedikit.
Hasil Penelitian	Hasil penelitian ini menunjukkan bahwa algoritma *Iterated Local Search* mampu menemukan solusi yang *feasible* dengan pengurangan *timeslots* per hari dari 10 menjadi 8 *timeslots* per hari.
Relevansi Penelitian	Penelitian ini menggunakan studi kasus dengan karakteristik yang sama dengan data ITC2007 dan penerapan algoritma *Iterated Local Search* dapat memenuhi karakteristik tersebut.

Tabel 2.3 Penelitian Sebelumnya 3

Judul	GOAL solver: a hybrid local search based solver for high school timetabling[5]
Penulis / tahun	George Henrique Godim da Fonseca, Haroldo Gambini Santos, Túlio Ângelo Machado Toffolo, Samuel Souza Brito, Marcone Jamilson Freitas Souza/ 2014
Deskripsi Umum Penelitian	Penelitian ini bercerita tentang pendekatan local search untuk permasalahan penjadwalan sekolah menengah yang juga permasalahan dari ITC2011, penelitian juga memenangkan kompetisi ITC2011. Algoritma yang digunakan pada penelitian ini adalah algoritma *Simulated Annealing* dan *Iterated Local Search*.

Hasil Penelitian	Pada [5], pendekatan *local search* menghasilkan solusi yang layak untuk hamper semua *instance* yang ada pada ITC2011 dan memenangkan kompetisi tersebut.
Relevansi Penelitian	Penelitian [5] menggunakan algoritma *Iterated Local Search* yang dapat dijadikan referensi sebagai penelitian tugas akhir dan juga penelitian [5] merupakan pemenang ITC2011.

2.2 Dasar Teori

Pada sub bab ini akan menjelaskan mengenai dasar teori yang digunakan untuk mendukung pengerjaan tugas akhir.

2.2.1 Penjadwalan

Permasalahan penjadwalan merupakan permasalahan kombinatorial yang rumit karena memiliki pilihan alternatif solusi yang luas dan banyak dijumpai lokal optimal. Permasalahan tersebut menjadi salah satu permasalahan kombinatorial yang mendapatkan banyak perhatian dari para peneliti. Beberapa diantaranya membuktikan bahwa permasalahan tersebut bertipe *NP-hard* (*non deterministic polynomial -time hard*) atau tipe permasalahan yang sulit untuk diselesaikan untuk ukuran yang besar[2].

Setiap permasalahan penjadwalan memiliki batasan, batasan tersebut terbagi menjadi dua yang biasa disebut *hard constraint* dan *soft constraint*. *Hard constraint* sendiri adalah batasan-batasan yang harus dipenuhi dan tidak boleh dilanggar[7]. Sedangkan untuk *soft constraint* sendiri adalah batasan yang tidak harus dipenuhi, namun apabila terpenuhi maka lebih baik[7]. Biasanya *soft constraint* memiliki pinalti apabila tidak terpenuhi. Suatu solusi dari permasalahan penjadwalan dapat dikatakan *feasible* apabila memenuhi *hard constraint*, dan diusahakan untuk memenuhi *soft constraint*.

2.2.2 *International Timetabling Competition 2019 Dataset*

International Timetabling Competition 2019 (ITC2019) sendiri adalah lomba tentang permasalahan penjadwalan yang berbasis Internasional. ITC sudah pernah dilaksanakan empat kali, yang

pertama adalah ITC2002, yang kedua ITC2007, yang ketiga ITC2011, dan yang keempat adalah ITC2019. Pada penelitian tugas akhir ini akan menggunakan data yang telah disediakan oleh ITC2019. Permasalahan penjadwalan yang telah disediakan oleh ITC2019 mencakup mulai dari *Rooms* (ruangan), *Classes* (kelas) beserta struktur *Courses* (mata kuliah), *Distribution Constraint* (distribusi batasan), dan *Students* dengan kebutuhan masing-masing. Pada permasalahan penjadwalan ITC2019, semua biaya direpresentasikan sebagai penalti bilangan bulat non-negatif yang mana semakin kecil penalti maka akan semakin baik solusi tersebut[3]. *Dataset* ITC2019 terbagi menjadi tiga kategori, yaitu data *early*, data *middle*, dan data *late.* Sedangkan data yang akan digunakan pada penelitian tugas akhir ini adalah *early dataset*. Pada *early dataset* terdapat 10 *instances* yang memiliki karakteristik sendiri-sendiri, perbedaan tersebut terletak pada jumlah dari *Courses* (mata kuliah), *Classes* (kelas), *Rooms* (ruang kelas), *Students* (siswa), dan jumlah *Distributions Constraints* (batasan distribusi). Berikut penjelasan karakteristik dari *test dataset* dan *early dataset* yang dijelaskan pada Tabel 2.4.

Tabel 2.4 Karakteristik Test Dataset dan Early Dataset

Nama	Total Jumlah				
	Courses	**Classes**	**Rooms**	**Students**	**Distributions**
Test Tiny	19	20	62	0	2
Test Small1	48	127	46	0	144
Test Small2	44	174	13	2002	102
Test Medium	603	803	56	27.881	329
Test Large	1.036	2.418	211	29.514	2004
Early Instance 1	340	1.239	80	1.641	1.220
Early Instance 2	272	1.852	44	2.116	2.690
Early Instance 3	353	983	62	3.018	1.251

Nama	Total Jumlah				
	Courses	**Classes**	**Rooms**	**Students**	**Distributions**
Early Instance 4	1.206	2.641	214	-	2.902
Early Instance 5	544	882	90	3.666	3.947
Early Instance 6	228	575	35	1.543	740
Early Instance 7	226	561	44	865	400
Early Instance 8	1.089	2.526	70	2.938	2.026
Early Instance 9	687	1.001	75	27.018	634
Early Instance 10	36	711	15	-	501

Permasalahan penjadwalan pada ITC2019 memiliki *Decision Variable* (Variabel Keputusan), *Constraint* (Batasan), dan *Objective Function* (Fungsi Tujuan) yang dijelaskan sebagai berikut:

- *Decision Variable* (Variabel Keputusan)
  Variabel keputusan yang terdapat pada ITC2019 ada dua, yaitu mengalokasikan setiap mata kuliah yang ada pada ruang kelas dan *timeslots* yang ada dan mengalokasikan siswa pada setiap kelas sesuai dengan mata kuliah yang diambil.
- *Constraints* (Batasan)
  Terdapat dua batasan pada ITC2019 yaitu *hard constraint* dan *soft constraint*. Setiap batasan dapat berupa *hard constraint* atau *soft constraint* dengan ditandainya melalui *required* sebagai *hard constraint* lalu *penalty* sebagai *soft constraint* seperti pada contoh Gambar 2.1.

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

Gambar 2.1 Spesifikasi dari Distribution Constraints

Berikut adalah apa saja *constraint* yang terdapat pada ITC2019 yang akan dijelaskan pada Tabel 2.5.

Tabel 2.5 Macam-macam constraint pada ITC2019

Constraint	**Definisi**
SameStart	Semua kelas dalam batasan ini harus dimulai pada waktu yang bersamaan
SameTime	Semua kelas dalam batasan ini dijadwalkan pada waktu yang sama
DifferentTime	Kelas dalam batasan ini tidak diperbolehkan dijadwalkan dalam waktu yang bersamaan
SameDays	Semua kelas dalam batasan ini dijadwalkan dalam hari yang sama
DifferentDays	Semua kelas dalam batasan ini harus dijadwalkan pada hari yang berbeda
SameWeeks	Semua kelas pada batasan ini dijadwalkan pada minggu yang sama
DifferentWeeks	Kelas pada batasan ini tidak boleh dijadwalkan pada minggu yang sama
SameRoom	Semua kelas dalam batasan ini harus dijadwalkan ke ruang kelas yang sama
DifferentRoom	Semua kelas dalam batasan ini harus dijadwalkan ke ruang kelas yang berbeda
Overlap	Dua kelas dalam batasan ini harus memiliki waktu yang tumpang tindih

NotOverlap	Kelas dalam batasan ini tidak bisa tumpang tindih dalam waktu yang sama
SameAttendees	Kelas dalam batasan ini harus mempertimbangkan waktu untuk siswa berpindah antar kelas
Precedence	Batasan ini memprioritaskan kelas dimana kelas dengan prioritas yang tinggi dijadwalkan pada minggu, hari, ataupun jadwal yang lebih awal
WorkDay (S)	Batasan ini mencegah atau memberikan penalti ketika sepasang kelas melebihi durasi *timeslot* yang telah ditentukan
MinGap (G)	Minimal jeda waktu yang dijadwalkan pada setiap dua kelas yang terdapat pada batasan ini
MaxDays (D)	Kelas yang berada pada batasan ini tidak dapat dijadwalkan lebih dari *D* hari dalam satu minggu
MaxDayLoad (S)	Pada batasan ini penjadwalan kelas yang ada di dalam daftar, pada setiap harinya tidak boleh melebihi *timeslots* yang sudah disediakan, oleh karena itu kelas yang ada pada daftar ini harus disebar ke hari lainnya dalam satu minggu
MaxBreaks (R,S)	Jumlah jeda untuk istirahat yang tidak boleh melebihi *R* per hari
MaxBlock (M,S)	Batasan ini membatasi jumlah waktu (diukur sebagai *M timeslot*) dimana satu set kelas dapat dijadwalkan secara berurutan yang masing-masing dipisahkan oleh tidak lebih dari *S timeslot*

- *Objective Function* (Fungsi Tujuan)
  Fungsi Tujuan pada permasalahan ITC2019 adalah meminimalkan biaya atau *cost* yang direpresentasikan melalui *penalty* guna menciptakan solusi yang lebih optimal.

2.2.3 *Hyperheuristic*

Hyper-heuristic adalah metodologi otomatis untuk memilih atau menghasilkan heuristik untuk memecahkan masalah pencarian komputasi yang sulit[8], sebuah *metaheuristic* yang

menghasilkan serangkaian *low level metaheuristic* dalam upaya untuk menyelesaikan masalah pencarian dan optimasi yang sulit[4]. Pendekatan *hyperheuristic* bertujuan untuk: (1) memilih dan mengkombinasikan *heuristics* yang lebih sederhana (*heuristic selection*) , atau (2) menghasilkan *heuristics* baru dari komponen *heuristics* yang sudah ada (*heuristic generation*)[9].

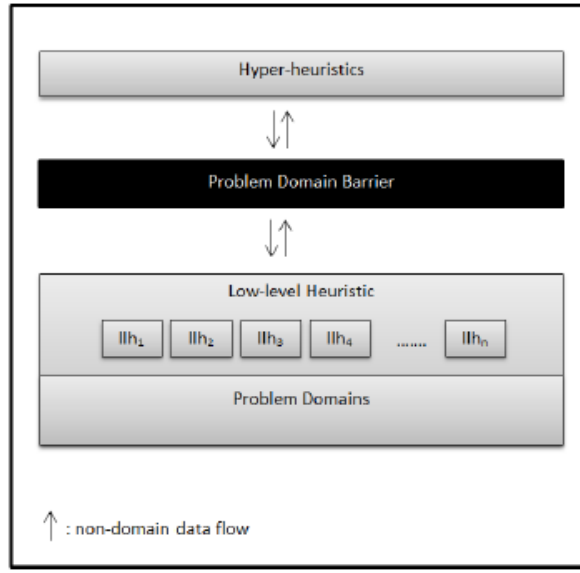
Pada [9], strategi *hyperheuristic* tidak bersinggungan langsung dengan *solution space* dikarenakan terdapat *problem domain barrier* dan hanya bersinggungan dengan *low-level heuristics* sebagaimana yang diilustrasikan pada Gambar 2.2. Sementara untuk *metaheuristic* bekerja di atas *search space* berupa *solution space*, berbeda dengan *hyperheuristic* yang berupa *lower-level heuristics*. Untuk metode *hyperheuristic* sendiri tidak memerlukan parameter tuning untuk setiap domain masalah secara manual karena tidak tergantung pada domain masalah tertentu saja dan dapat melakukan otomasi untuk parameter tuning ini.

Berdasarkan tujuannya, *hyperheuristic* dapat dikategorikan menjadi dua, yaitu:

1. *Heuristic Selection*: *Selection Constructive, Selection Perturbative*
2. *Heuristic Generation: Generation Constructive, Generation Perturbative*

Penelitian [10] menunjukkan berbagai contoh *Hyperheuristic* yang ditujukan untuk *timetabling*, salah satunya adalah *low level heuristic move* dan *swap* yang termasuk pada kategori *generation perturbative hyperheuristic* yang juga nantinya akan digunakan pada penelitian kali ini.

Gambar 2.2 Framework Hyper-heuristics



2.2.4 *Iterated Local Search*

Iterated Local Search (ILS) merupakan metaheuristic yang mengadaptasi improvement heuristic dengan proses berulang yang dapat menghasilkan rantai dari sebuah solusi[11]. ILS adalah metodologi yang relatif sederhana, telah berhasil di berbagai domain[4]. Tujuan dari ILS adalah untuk meningkatkan Pencarian Multi-Restart stokastik dengan pengambilan sampel di lingkungan yang lebih luas dari kandidat solusi dan menggunakan teknik Pencarian Lokal untuk memperbaiki solusi yang lebih optimal. ILS mengeksplorasi urutan solusi yang dibuat sebagai *perturbations* dari solusi terbaik saat ini, yang hasilnya disempurnakan menggunakan *embedded heuristic*[12]. Berikut merupakan *pseudocode* dari algoritma ILS yang ditunjukkan pada Gambar 2.3 [13].

Gambar 2.3 Pseudocode Iterated Local Search

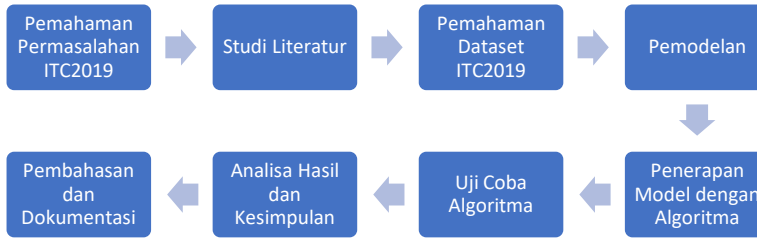
```
procedure Iterated Local Search
    s_0 = GenerateInitialSolution
    s* = LocalSearch(s_0)
    repeat
        s' = Perturbation(s*, history)
        s*' = LocalSearch(s')
        s* = AcceptanceCriterion(s*, s*', history)
    until termination condition met
end
```

Algoritma ILS dimulai dengan pembentukan solusi awal dan akan terus melakukan pencarian solusi yang lebih baik hingga menemui kriteria yang diinginkan[13].

Pada *Local Search* dan *Perturbation* yang terdapat pada *ILS*, menggunakan kerangka kerja *Hyperheuristic. Low level heuristic* yang digunakan adalah metode *move* dan *swap*. Keuntungan dari penggunaan algoritma *ILS-Hyperheuristic* adalah solusi kandidat yang diperoleh melalui *perturbation* seringkali memiliki kualitas yang lebih baik dan waktu yang lebih cepat[14]. Pada penelitian [4], penggunaan *ILS-Hyperheuristic* menghasilkan solusi yang kompetitif dan performa yang lebih baik bila dibandingkan dengan penelitian lain yang menggunakan metode *Genetic Algorithm Hybrid Iterative Forward Search*, *Adaptive Tabu Search*, *SAT* dan *MaxSAT*, *Lower Bound* dengan *ILP*, *Iterative Tabu Search* dengan *ILP solver*.

BAB III
METODOLOGI PENELITIAN

Pada bab ini akan menjelaskan tentang metodologi yang akan digunakan untuk mengerjakan tugas akhir. Metodologi ini digunakan sebagai panduan dan memberikan alur pengerjaan tugas akhir yang sistematis. Alur metodologi pengerjaan tugas akhir dapat dilihat pada Gambar 3.1.



Gambar 3.1 Metodologi Pengerjaan Tugas Akhir

3.1 Pemahaman Permasalahan ITC2019

Pemahaman permasalahan yang dilakukan adalah memahami studi kasus secara menyeluruh tentang *International Timetabling Competition 2019* (ITC2019) yang utamanya tentang topik penjadwalan untuk mengetahui detail topik dan objek yang diteliti pada penelitian tugas akhir yang akan dikerjakan. Hasil dari tahapan ini adalah topik permasalahan yang dijadikan bahan pada penelitian tugas akhir.

3.2 Studi Literatur

Pada tahap studi literatur, mengumpulkan penelitian, buku, serta dokumen yang terkait tentang topik tugas akhir yang dikerjakan. Topik terkait tugas akhir ini adalah tentang penjadwalan mata kuliah secara otomatis. Pada tahap ini juga menentukan algoritma yang digunakan untuk menyelesaikan permasalahan penelitian tugas akhir. Algoritma yang digunakan pada penelitian tugas akhir ini adalah algoritma *Iterated Local Search*

dan diperkuat menggunakan *Hyperheuristic* yang sesuai pada permasalahan penjadwalan.

3.3 Pemahaman Dataset ITC2019

Setelah pemahaman permasalahan dan algoritma telah dipahami, tahap berikutnya adalah tahap pemahaman dataset ITC2019. Data didapatkan melalui mendaftar pada kompetisi ITC2019 melalui website ITC2019. Setelah data didapatkan dilakukan pemahaman terkait format dataset yaitu tentang variabel keputusan beserta batasan-batasan yang terdapat pada data ITC2019. Data masukan yang digunakan adalah berupa hari (terdapat 7 hari dalam setiap minggu), *timeslots* (untuk *timeslots* terdapat 288 *timeslots* per harinya dan tiap *timeslots* diberikan waktu selama 5 menit), minggu (untuk jumlah minggu perkuliahan berbeda-beda pada setiap *instance* berantara 13 hingga 19 minggu perkuliahan), ruangan, mata kuliah, dan siswa yang akan menjadi variabel keputusan. Terdapat juga beberapa batasan yang sudah dijelaskan pada sub bab 2.2.2 Tabel 2.4 yang akan menjadi *hard constraint* dan *soft constraint.* Untuk jumlah ruangan, mata kuliah, siswa, dan juga *Distributions Constraint* (batasan) berbeda-beda untuk setiap *Instance* seperti pada Tabel 2.4.

3.4 Permodelan

Pada tahap pemodelan, permasalahan dianalisis untuk menentukan fungsi tujuan dan batasan masalah. Setelah fungsi tujuan dan batasan masalah ditentukan sesuai yang telah dijelaskan pada sub bab 2.2.2, langkah selanjutnya adalah menerjemahkan permasalahan ke dalam model matematika.

3.5 Implementasi Algoritma Iterated Local Search - Hyperheuristic

Pada tahap ini, implementasi yang dilakukan menggunakan input dari ITC2019 dataset. Dalam penerapan algoritma *Iterated Local Search - Hyperheuristic*, yang dilakukan pertama kali adalah menentukan solusi awal terlebih dahulu yang digunakan untuk mencari iterasi berikutnya sehingga menghasilkan solusi yang lebih baik. Lalu pada langkah selanjutnya dilakukan

perbaikan solusi awal menggunakan *local search* seperti yang dijelaskan pada sub bab 2.2.2 dengan menggunakan keraangka kerja *hyperheuristic* yang mana solusi akan di *update* apabila terdapat solusi yang lebih baik hingga mendapatkan hasil yang paling optimal. Setelah itu dilakukan *perturbation* seperti yang dijelaskan pada sub bab 2.2.2 untuk menggunakan kerangka kerja *hyperheuristic* pada solusi terakhir. *Perturbation* dilakukan untuk menemukan solusi optimal yang lebih baik dari sebelumnya. Setelah solusi terbaru telah ditemukan maka akan dilakukan kembali *local search* dan dilakukan perulangan pada *perturbation* kembali hingga memenuhi kriteria yang diinginkan.

3.6 Uji Coba Algoritma

Tahap uji coba dilakukan dengan memasukkan dataset dari ITC2019 ke dalam algoritma yang telah diimplementasikan. Uji coba dilakukan dengan memasukkan seluruh instance yang telah dijelaskan pada sub bab 2.2.2 *International Timetabling Competition 2019 Dataset* dan berguna untuk mengetahui apakah keseluruhan *hard constraint* sudah terpenuhi juga sebisa mungkin memenuhi semua *soft constraint*. Apabila dirasa masih belum sesuai dengan kriteria yang diinginkan maka akan dilakukan evaluasi dan peninjauan ulang terhadap algoritma.

3.7 Analisis Hasil dan Kesimpulan

Tahapan ini dilakukan analisis terkait solusi yang dihasilkan oleh setiap *instance* dan dilakukan analisis terkait performa algoritma dalam menyelesaikan permasalahan penjadwalan mata kuliah dan pemenuhan fungsi tujuan. Dari hasil analisis dapat diketahui terkait kinerja algoritma dan dapat dilakukan penarikan kesimpulan terkait keseluruhan *instance* dan juga kinerja algoritma.

3.8 Pengerjaan Laporan Tugas Akhir

Pada tahap ini melakukan dokumentasi terkait hal-hal yang telah dilakukan selama pengerjaan tugas akhir ini serta hasil analisis yang telah dilakukan. Hasil pada tahap ini adalah

laporan tugas akhir yang mana sebagai bukti dari seluruh pengerjaan penelitian tugas akhir yang telah dilakukan.

BAB IV
PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan mengenai data yang digunakan untuk tugas akhir ini serta konversi model matematis kedalam bentuk struktur data yang digunakan untuk implementasi algoritma.

4.1 Pemahaman Data

Data yang digunakan pada tugas akhir ini menggunakan dataset kompetisi dalam *International Timetabling Competition* 2019 yang diperoleh dari situs resmi ITC 2019 yaitu www.itc2019.org. Dataset ini terdiri dari empat kelompok data yaitu data *test, early, middle,* dan *late*. Pada data *test* sendiri terdiri dari 5 subset yaitu *tiny*, *small1*, *small2*, *medium*, dan *large.* Namun pada penelitian tugas akhir ini dibatasi hanya pada data *test* pada subset *tiny* (lums-sum17.xml) dan *small2* (pu-cs-fal07.xml).

Setiap data memiliki jumlah mata kuliah (*course*), kelas (*class*), ruangan (*room*), batasan (*distribution constraint*), dan siswa *(student*), yang berbeda-beda. Namun pada penelitian tugas akhir ini dibatasi hanya pada jumlah mata kuliah, kelas, ruangan, dan batasan.

4.2 Formulasi Model Matematis

Pada tahap ini melakukan pendefinisian fungsi tujuan, variabel keputusan, dan batasan – batasan (*hard constraint* dan *soft constraint*) menjadi model matematis untuk mempermudah pemahaman konsep untuk menyelsaikan permasalahan.

4.2.1 Fungsi Tujuan

Fungsi tujuan yang akan dicapai untuk pembuatan model yaitu meminimalkan penalti (*cost*) untuk segala sumber daya yang digunakan dalam penjadwalan mata kuliah. Biaya / penalti yang harus dikeluarkan dalam penyusunan jadwal harus seminimal

mungkin sehingga persamaan dari fungsi tujuan dapat dilihat pada persamaan (4.1).

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

Dengan nilai :

P1 = Nilai penalti bagi ruangan yang dipilih untuk digunakan dalam menjadwalkan kelas dari suatu mata kuliah

P2 = Nilai penalti bagi timeslot yang dipilih untuk digunakan dalam menjadwalkan kelas dari suatu mata kuliah

P3 = Nilai penalti bagi soft constraint yang berhasil dipenuhi

Nilai penalti yang dihitung berasal dari 3 sumber yaitu P1, P2, dan P3, dimana P1 sendiri dapat dinyatakan pada persamaan (4.2).

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

Dengan nilai :

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

PRc = Nilai penalti dari ruangan yang digunakan oleh kelas suatu mata kuliah

Dimana

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

P2 menghitung nilai penalti dari timeslot yang digunakan oleh kelas suatu mata kuliah, dimana persamaan P2 dapat dilihat pada persamaan (4.3).

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

Dengan nilai :

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

PTc = Nilai penalti dari timeslot yang digunakan oleh kelas suatu mata kuliah

Dimana

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

P3 menghitung nilai penalti dari masing-masing batasan yang telah didefinisikan pada suatu dataset (soft constaint) yang dinyatakan pada persamaan (4.4).

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

Dengan nilai :

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

PDCi = Vektor yang menunjukkan soft constraint

Dimana

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

4.2.2 Variabel Keputusan

Variabel keputusan merupakan variabel yang menjabarkan keputusan-keputusan yang akan dibuat. Berikut beberapa

variabel keputusan yang terdapat pada dataset *International Timetabling Competition 2019*.

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

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

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

W = *Week* $\{w_1,\ldots,w_{|W|}\}$

D = *Day* $\{d_1,\ldots,d_{|D|}\}$

S = *Start* $\{1,\ldots,288\}$

L = *Length* $\{1,\ldots,288\}$

E = *End* $\{1,\ldots,288\}$

4.2.3 Distribution Constraints

Terdapat 19 batasan yang berlaku sebagai *hard constraint* maupun *soft constraint* bergantung dari aturan yang telah ditetapkan pada setiap dataset. Batasan-batasan tersebut telah terdefinisikan pada setiap dataset beserta kelas-kelas yang memiliki batasan tersebut. Contoh batasan dalam dataset dapat dilihat pada Tabel 4.1 yang mana menjelaskan batasan dari dataset Small2.

Tabel 4.1 Macam-Macam Constraint pada Dataset Small2

No	Nama	Tipe	Jumlah Distribution
1	SameAttendees	*Hard*	52 Distribusi
2	NotOverlap	*Hard*	8 Distribusi
3	SameTime	*Hard*	8 Distribusi
4	SameRoom	*Soft*	8 Distribusi
5	NotOverlap	*Soft*	26 Distribusi

Untuk batasan yang berlawanan seperti contoh *SameTime* dan *DifferentTime* tidak akan tumpang tindih, dalam artian kelas dalam kedua batasan tersebut tidak akan sama satu sama lain.

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

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

Dengan nilai :

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

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

2) Same Time
   Semua kelas dalam batasan ini harus berlangsung dalam rentang waktu yang sama.

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

Dengan nilai :

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

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

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

3) Different Time

Semua kelas pada batasan ini tidak boleh berlangusng dalam rentang waktu yang sama.

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

Dengan nilai :

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

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

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

4) Same Days
Semua kelas dalam batasan ini harus ditawarkan pada hari yang sama.

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

Dengan nilai :

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

d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu

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

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

Dengan nilai :

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

d = {1,2, …, 7} adalah urutan hari yang dijadwalkan pada kelas tertentu

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

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

Dengan nilai :

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

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

7) Different Weeks
Semua kelas dalam batasan ini tidak boleh diselenggarakan pada minggu yang sama pada satu semester.

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

Dengan nilai :

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

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

8) Same Room
Semua kelas pada batasan ini harus dijadwalkan pada ruangan yang sama.

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

Dengan nilai :

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

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

9) Different Room
Semua kelas dalam batasan ini harus dijadwalkan pada ruangan yang berbeda satu sama lain.

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

Dengan nilai :

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

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

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

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

Dengan nilai :

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

s = {1,2, …, 288} adalah waktu kelas dimulai yang terjadwal pada timeslot tertentu

e = {1,2, …, 288} adalah waktu kelas berakhir 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

11) Not Overlap

Tidak boleh ada dua kelas dalam batasan ini yang saling tumpah tindih.

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

Dengan nilai :

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

s = {1,2, …, 288} adalah waktu kelas dimulai yang terjadwal pada timeslot tertentu

e = {1,2, …, 288} adalah waktu kelas berakhir 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

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

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

Dengan nilai :

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

c_{ij} = jumlah mahasiswa yang mengambil mata kuliah *i* dan *j*

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

Dimana $v_{ij,}$

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

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

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

Dengan nilai :

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

s = {1,2, …, 288} adalah waktu kelas dimulai yang terjadwal pada timeslot tertentu

e = {1,2, …, 288} adalah waktu kelas berakhir 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

14) Work Day(s)

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

$$D14 = \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} d_{ij} = 0\right) \vee \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} w_{ij} = 0\right) \vee \left(\left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} max(e_i, e_j)\right) - \left(\sum_{i=1}^{c-1}\sum_{j=i+1}^{c} min(s_i, s_j)\right) \leq S\right)$$

Dengan nilai :

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

s = {1,2, …, 288} adalah waktu kelas dimulai yang terjadwal pada timeslot tertentu

e = {1,2, …, 288} adalah waktu kelas berakhir 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

15) Min Gap (G)

Dua kelas yang dijadwalkan pada hari yang bersamaan harus memiliki setidaknya timeslot sebesar G yang terpisah (antara berakhirnya kelas pertama dan dimulainya kelas kedua)

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

Dengan nilai :

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

s = {1,2, …, 288} adalah waktu kelas dimulai yang terjadwal pada timeslot tertentu

e = {1,2, …, 288} adalah waktu kelas berakhir 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

16) Max Days (D)

Kelas – kelas pada batasan ini tidak boleh dijadwalkan ke lebih dari D hari kerja yang berbeda dalam satu minggu.

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

Dengan nilai :

i,j = {1, 2, 3, …, 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

17) Max Day Load (S)

Membatasi jumlah total waktu yang ditetapkan untuk sekumpulan kelas tidak lebih dari S timeslot tiap hari selama satu semester.

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

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

Dengan nilai :

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

l = {1,2, …, 288} adalah waktu durasi berlangsungnya kelas yang terjadwa; 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

18) Max Breaks (R, S)

Semua kelas yang masuk dalam daftar ini tidak boleh dijadwalkan melebihi jumlah jeda sebanyak R antar kelas selama sehari dengan jumlah slot sebanyak S.

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

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

Dengan nilai :

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

19) Max Block (M, S)

Semua kelas yang ada dalam batasan ini tidak boleh dijadwalkan melebihi batas maksimal hari yang telah ditentukan setiap minggunya.

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

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

4.3 Pembentukan Initial Solution

Solusi awal dibuat secara otomatis dengan hanya memperhatikan *hard constraint* yang telah ditetapkan, sehingga tidak melihat *soft constraint*. Setiap solusi dideskripsikan dalam bentuk *arraylist object* yang berisi daftar *class* dari setiap *course* beserta jadwalnya. Masing-masing solusi terdiri dari kelas yang dilengkapi dengan identitas, *start timeslot, days, weeks,* dan *room* yang digunakan kelas tersebut. Untuk membuat solusi awal ini terdapat beberapa langkah yang harus dilakukan.

4.3.1 Pembacaan *File*

Pembacaan *file* dilakukan untuk membaca *file* untuk dijadikan sebagai data masukkan dalam proses inisiasi solusi dan optimasi. File yang dimasukan untuk diproses memiliki ekstensi XML, sehingga file ini terlebih dahulu harus dapat dibaca untuk kemudian disimpan masing-masing nilainya. Untuk membaca *file* menggunakan bahasa pemrograman Java dan menggunakan *DOM-parser*.

4.3.2 Penyimpanan Konten *Rooms*

Pada konten *rooms* terdapat beberapa nilai yang harus disimpan diantaranya adalah kode identitas (ID), kapasitas (capacity), waktu yang tidak tersedia (unavailable), dan waktu perjalanan antar ruangan (travel room). Penyimpanan konten *rooms* dilakukan dengan membangun struktur *object oriented* dengan bahasa pemorgraman Java.

4.3.3 Penyimpanan Konten *Courses*

Pada konten courses ini cukup kompleks karena terdapat beberapa nilai yang harus tersimpan diantaranya identitas (id) setiap courses dan hierarki yang dimiliki corse tersebut yang terdiri dari config, subpart, dan class dimana masing-masing dari atribut tersebut juga memiliki identitas. Informasi ini disimpan kedalam class class baru berbentuk objek yang. Atribut yang menyimpan informasi lebih banyak adalah class Class yang terdiri dari identitas class (id), keterkaitan antar class (parent-child), kuota (limit), kebutuhan ruangan (room = required), pilihan ruangan yang dapat digunakan dengan nilai penaltinya, dan pilihan slot jadwal yang dapat digunakan dengan nilai penaltinya. Penyimpanan konten *courses* dilakukan dengan membangun struktur *object oriented* dengan bahasa pemorgraman Java.

4.3.4 Penyimpanan Konten *Distribution Constraint*

Konten batasan berisikan informasi berupa metode untuk 19 batasan yang berlaku pada domain permasalahan ITC 2019. Setaip batasan didefinisikan dalam bentuk metode sesuai

dengan ketentuan dimana metode disimpan dalam satu kelas yang sama. Penyimpanan konten *Distribution Constraint* dilakukan dengan membangun struktur *object oriented* dengan bahasa pemorgraman Java.

4.3.5 Pembuatan Kode Program *Hard Constraint*

Setiap batasan yang sudah dijelaskan pada sub bab 4.2.3 merupakan batasan dalam bentuk matematis, sehingga harus diubah atau dikonversi menjadi kode program dengan menggunakan bahasa pemrograman Java. Pembuatan kode *hard constraint* ini bertujuan sebagai batasan pembentuk solusi awal dan hasil solusi optimum terakhir dimana batasan-batasan ini harus terpenuhi.

4.3.6 Pembentukan *Initial Solution*

Setelah membentuk kode program *hard constraint* selanjutnya adalah membentuk solusi awal sebagai jadwal awal sebelum di optimasi. Pada initial solution ini menerapkan algoritma Greedy dimana urutan pertama dalam sebuah daftar mata kuliah diletakkan pada slot tersedia pertama. Artinya, pada saat membuat initial solution ini sekaligus melakukan cek atau verifikasi *hard constraint*. Jika semua mata kuliah telah terjadwal dan memenuhi *hard constraint*, maka *initial solution* dianggap sudah terbentuk.

4.3.7 Pembuatan Kode Program *Soft Constraint*

Soft Constraint ini untuk menghitung nilai penalti yang akan menjadi hasil perbandingan dari optimasi. Total nilai penalti akan dihitung berdasarkan penjadwalan mata kuliah. Kemudian semua nilai dari penalti tersebut dijumlahkan dan menghasilkan total nilai penalti yang akan digunakan sebagai pembanding setiap solusi yang dihasilkan pada optimasi.

4.4 Pembentukan *Low Level Heuristic*

Low Level Heuristic digunakan untuk memindah urutan solusi sehingga menghasilkan nilai penalti yang baru dan berbeda. *Low level heuristic* yang digunakan adalah *move. Move* dilakukan untuk memindahkan timeslot satu mata kuliah atau

beberapa mata kuliah kedalam timeslot lain secara acak yang terpilih.

4.5 Implementasi Algoritma *Iterated Local Search*

Setelah membuat *initial solution* dan mengetahui penaltinya, selanjutnya dilakukan optimasi agar nilai penalti seminimum mungkin. Optimasi ini dilakukan dengan implementasi algoritma *Iterated Local Search*. Beberapa aktivitas yang dilakukan sebelum implementasi algoritma ILS sendiri adalah menyimpan solusi awal dalam struktur data tertentu, pembuatan dan pemilihan *Low Level Heuristic*, implementasi algoritma *Local Search*, implementasi pertubasi dan penyimpanan solusi optimum akhir.

Halaman ini sengaja dikosongkan

BAB V
IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai penerapan dari perancangan yang telah ditentukan sebelumnya dan proses implentasi algoritma *Iterated Local Search* terhadap dataset ITC 2019.

5.1 Pemahaman Data

Data yang digunakan pada tugas akhir ini menggunakan dataset kompetisi dalam *International Timetabling Competition* 2019 yang diperoleh dari situs resmi ITC 2019 yaitu www.itc2019.org. Dataset ini terdiri dari empat kelompok data yaitu data *test, early, middle,* dan *late*. Pada data *test* sendiri terdiri dari 5 subset yaitu *tiny*, *small1*, *small2*, *medium*, dan *large.* Namun pada penelitian tugas akhir ini dibatasi hanya pada data *test* pada subset *tiny* (lums-sum17.xml) dan *small2* (pu-cs-fal07.xml).

Setiap data memiliki jumlah mata kuliah (*course*), kelas (*class*), ruangan (*room*), batasan (*distribution constraint*), dan siswa *(student*), yang berbeda-beda. Namun pada penelitian tugas akhir ini dibatasi hanya pada jumlah mata kuliah, kelas, ruangan, dan batasan.

5.1.1 Inisiasi Konten Dataset

Permasalahan yang didefinisikan dari dataset *tiny* dan *small2* yaitu memiliki jumlah minggu (nrWeeks), jumlah hari (nrDays) dan jumlah timeslot untuk setiap hari (slotsPerDay), dimana hal tersebut didefinisikan pada awal file data XML pada Kode Program 5.1.

Setelah mendefinisikan permasalahan pada baris problem, selanjutnya diikuti dengan optimization yang berisi pembobotan untuk masing-masing *penalty* terkait dengan *time*, *room*, dan *distribution constraint* seperti yang terlihat pada Kode Program 5.2.

```
<?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 Program 5.1 Inisiasi Jumlah *Days, Weeks, d*an *Timeslots*

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

Kode Program 5.2 Inisiasi pembobotan *penalty*

5.1.2 Inisiasi Konten *Rooms*

Konten *rooms* berisi 3 atribut yang dapat dilihat pada Kode Program 5.3 yaitu nomor atau ID setiap ruangan beserta kapasitas ruangan, dimana nilai ini menunjukkan masing-masing ruangan memiliki kapasitas untuk jumlah mahasiswa yang menggunakan ruangan tersebut. Selanjutnya terdapat *travel room* yang menunjukkan waktu yang dihabiskan untuk jarak tempuh saat perpindahan ruangan terjadi. Waktu perpindahan tersebut hanya terdaftar pada salah satu ruangan. Setelah itu terdapat *Unavailable* yang menunjukkan waktu ketika ruangan tersebut tida dapat digunakan dan memiliki atribut *weeks, days, start* dan *length*.

```
<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
    times are present -->
    <!-- not available on Mondays and Tuesdays betwen 8:30
    - 10:30, all weeks -->
    <unavailable days="110000" start="102" length="24"
```

```
    weeks="1111111111111"/> <!-- not available on Fridays
    betwen 12:00 - 24:00, odd weeks only -->
    <unavailable days="000100" start="144" length="144"
    weeks="1010101010101"/> </room>
</rooms>
```

Kode Program 5.3 Kode Kontan Ruangan

5.1.3 Inisiasi Konten *Courses*

Konten Courses memiliki identitas setiap mata kuliah, identitas konfigurasi yang menunjukkan jenis mata kuliah yang terdiri dari tiga jenis yaitu sesi perkuliahan, seminar dan laboratorium. Selajutnya ada subpart yang merupakan turunan dari konfigurai, dan yang terakhir adalah identitas kelas dan batas kuota (limit) untuk menunjukkan identitas kelas dan batasan dari kuota kelas yang dapat diikuti oleh mahasiswa. Format dari konten *courses* dapat dilihat pada Kode Program 5.4.

```
<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 Program 5.4 Kode konten Courses

Selain dari itu terdapat beberapa kelas yang memiliki *parent-child* yang erat kaitannya antara satu sama lainnya atribut *room* yang akan bernilai *false* apabila *class* tersebut tidak memerlukan ruangan untuk pelaksanaannya. Jika ada *class* yang memiliki atribut *parent* ini maka siswa yang mengambil kelas itu harus juga mengambil *parent* class nya.

5.1.4 Inisiasi Konten Student

Setiap mahasiswa memiliki identitas masing-masing yang unik dan daftar matakuliah yang harus diambil. Dari setiap mata kuliah juga memiliki identitas dari matakuliah itu sendiri. Untuk contoh kode dari konten *student* ini dapat dilihat pada Kode Program 5.5.

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

Kode Program 5.5 Kode konten student

5.1.5 Inisiasi Konten Distribution Constraint

Terdapat dua jenis batasan yaitu *hard constraint* dan *soft constraint. Hard Constraint* merupakan syarat yang wajib dipenuhi, yaitu ditandai dengan atribut “required=true” , sementara *soft constraint* merupakan penalti yang akan mempengaruhi *cost* pada hasil akhir. Terdapat 19 jenis *distribution constraint* yang ada dimana dalam setiap *distribution constraint* terdapat *class* yang dikenai batasan-batasan tersebut. Contoh penulisan batasan dapat dilihat pada Kode Program 5.6.

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

Kode Program 5.6 Contoh XML dari Distribution Constraint

5.2 Pembacaan File Input

Untuk membentuk *initial solution* langkah pertama yang dilakukan adalah membaca file yang menjadi *input* untuk program. File yang dimasukan untuk diproses memiliki ekstensi XML, sehingga file ini terlebih dahulu harus dapat dibaca untuk kemudian disimpan masing-masing nilainya.

```
public ReadFile(String filename) throws
ParserConfigurationException,
        SAXException, IOException {

File fXmlFile = new File("src/Test/" + filename);
DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
DocumentBuilder dBuilder =
dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(fXmlFile);
doc.getDocumentElement().normalize();

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

namaFile = rootElement.getAttribute("name");
```

Kode Program 5.7 Script Pembacaan File Input

Kode Program 5.7 menunjukkan fungsi koding yang membaca file dengan menyimpan atribut dari *tag* XML yang terdiri dari *root element* dan *child element*.

5.3 Penyimpanan Konten Rooms

Pada konten *rooms* terdapat beberapa nilai yang harus disimpan diantaranya adalah kode identitas (ID), kapasitas (*capacity*), waktu yang tidak tersedia (*unavailable*), dan waktu perjalanan antar ruangan (*travel room*). Nilai-nilai tersebut disimpan dalam sebuah class berbentuk objek yang dapat dilihat pada

```
public class Room {
```

```
private int idRoom;
private int roomcap;
private ArrayList <Integer> travelroom = new ArrayList<>();

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

Kode Program 5.8 Script Penyimpanan Konten Room

Kode Program 5.8 menunjukkan identitas Room yang memiliki tipe integer, kapasitas ruangan (roomcap) yang memiliki tipe integer, dan terdapat waktu perpindahan ruangan (travelroom) dalam bentuk array list.

5.4 Penyimpanan Konten Courses

Pada konten courses ini cukup kompleks karena terdapat beberapa nilai yang harus tersimpan diantaranya identitas (id) setiap courses dan hierarki yang dimiliki *course* tersebut yang terdiri dari *config*, *subpart*, dan *class* dimana masing-masing dari atribut tersebut juga memiliki identitas. Informasi ini disimpan kedalam kelas-kelas baru berbentuk objek. Atribut yang menyimpan informasi lebih banyak adalah kelas *Class* yang terdiri dari identitas *class* (id), keterkaitan antar *class* (*parent-child*), kuota (*limit*), kebutuhan ruangan (*room = required*), pilihan ruangan yang dapat digunakan dengan nilai penaltinya, dan pilihan slot jadwal yang dapat digunakan dengan nilai penaltinya. Konten course ini disimpan dalam class berbentuk objek yang dapat dilihat pada

```
public class Kelas {
    private int room = -1;
    private int roomLama=-1;
    private int ts = -1;
    private int tsLama = -1;
    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 Program 5.9 Script Penyimpanan Konten Course

Kode Program 5.9 menunjukkan identitas dari setiap Kelas yang terdapat pada Course yang memiliki tipe integer, limit dari setiap kelas yang memiliki tipe integer, parent dari kelas tersebut yang berisi identitas kelas lain dengan tipe integer, dan juga nilai yang menunjukkan bahwa kelas tersebut membutuhkan ruangan (room) untuk dijadwalkan atau tidak dengan tipe boolean.

5.5 Pembuatan Kode Program Hard Constraint

Konten hard constraint berisikan informasi berupa metode untuk 19 batasan yang berlaku pada domain permasalahan ITC 2019. Setaip batasan didefinisikan dalam bentuk metode sesuai dengan ketentuan dimana metode disimpan dalam satu kelas yang sama. Hard Constraint merupakan batasan / syarat yang harus terpenuhi dimana konversi yang dilakukan menggunakan bahasa pemrograman Java. Untuk kode program konversi pengkodean setiap batasan adalah sebagai berikut.

1. Same Start

Hard Constraint pertama berhubungan dengan Same Start sehingga semua kelas dalam batasan ini harus dimulai pada

waktu yang bersamaan. Kode program dari *hard constraint* pertama dapat dilihat pada Kode Program 5.10.

```
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;
            } else {
               return false;
            }
} else if (iTS == -1) {
            smStart=true;
         }
}
      return smStart;
   }
```

Kode Program 5.10 Script Hard Constraint 1

Metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki waktu yang sama, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

2. Same Time

Hard Constraint kedua berhubungan dengan Sama Time sehingga setiap kelas harus berlangsung dalam rentang waktu yang sama. Kode program dari *hard constraint* dapat dilihat pada Kode Program 5.11.

```
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;
            }
} else if (iTS == -1) {
            ST = true;
        }
    }
    return ST;
  }
```

Kode Program 5.11 Script Hard Constraint 2

Metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki durasi waktu yang sama, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

3. Different Time

Hard Constraint ketiga berhubungan dengan Different Time sehingga setiap kelas tidak boleh berlangsung dalam rentang waktu yang sama. Kode program dari *hard constraint* dapat dilihat pada Kode Program 5.12.

```
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;
}
} else if (iTS == -1) {
            difTime = true;
         }
}
      return difTime;
   }
```

Kode Program 5.12 Script Hard Constraint 3

Dimana metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki durasi waktu yang berbeda satu sama lain, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

4. Same Days

Hard Constraint keempat berhubungan Same Days sehingga kelas yang masuk daftar ini harus dijadwalkan di hari yang sama. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.13.

```
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;
                    break;
                } else {
                    SD = false;
                }
            }
if (SD==false) {
                return false;
            }
} else if (iTS == -1) {
            SD = true;
        }
    }
    return SD;
  }
```

Kode Program 5.13 Script Hard Constraint 4

Dimana metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki hari yang sama antara satu dengan yang lainnya, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

5. Different Days

Hard Constraint kelima berhubungan dengan Different Days sehingga kelas yang masuk daftar ini tidak boleh dijadwalkan di hari yang sama. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.14.

```
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;
} else{
 DD = false;
break;
}
```

Kode Program 5.14 Script Hard Constraint 5

Dimana metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki hari yang berbeda antara satu dengan yang lainnya, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

6. Same Weeks

Hard Constraint keenam berhubungan dengan Same Weeks sehingga kelas yang masuk daftar ini harus dijadwalkan di minggu yang sama. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.15.

```
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 Program 5.15 Script Hard Constraint 6

Dimana metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki minggu yang sama antara satu dengan yang lainnya, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

7. Different Weeks

Hard Constraint ketujuh berhubungan dengan Different Weeks sehingga kelas yang masuk daftar ini harus dijadwalkan di minggu yang berbeda. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.16.

```
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{
DW = false;
break;
}
```

Kode Program 5.16 Script Hard Constraint 7

Dimana metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki minggu yang berbeda antara satu dengan yang lainnya, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

8. Same Rooms

Hard Constraint kedelapan berhubungan dengan Same Rooms sehingga kelas yang masuk daftar ini harus dijadwalkan di ruangan yang sama. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.17.

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

Kode Program 5.17 Script Hard Constraint 8

Dimana metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki ruangan yang sama antara satu dengan yang lainnya, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

9. Different Rooms

Hard Constraint kesembilan berhubungan dengan Different Rooms sehingga kelas yang masuk daftar ini harus dijadwalkan di ruangan yang berbeda. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.18.

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

Kode Program 5.18 Script Hard Constraint 9

Dimana metode ini memiliki parameter waktu yang telah terjadwal, objek kelas, dan daftar kelas yang berada pada batasan ini. Kemudian kelas tersebut yang sudah terjadwal akan dibandingkan apakah memiliki ruangan yang berbeda antara satu dengan yang lainnya, jika iya maka pengembalian nilai benar, jika tidak maka pengembalian nilai salah.

10. Overlap

Hard Constraint kesepuluh berhubungan dengan Overlap sehingga kelas yang masuk daftar ini harus dijadwalkan tumpah tindih pada durasi yang berlangsung, hari dan minggu. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.19.

```
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 Program 5.19 Script Hard Constraint 10

Dimana metode ini melakukan perulangan pada daftar kelas yang terjadwal pada daftar kelas batasan ini, lalu melakukan cek apakah waktu mulai kelas dan berakhirnya kelas saling tumpang tindih dan mengecek dengan melakukan perulangan pada minggu dan hari dari jadwal perulangan.

11. Not Overlap

Hard Constraint kesebelas berhubungan dengan Not Overlap sehingga kelas yang masuk daftar ini tidak boleh dijadwalkan tumpah tindih pada durasi yang berlangsung, hari dan minggu. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.20.

```
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 Program 5.20 Script Hard Constraint 11

Dimana metode ini melakukan perulangan pada daftar kelas yang terjadwal pada daftar kelas batasan ini, lalu melakukan cek apakah waktu mulai kelas dan berakhirnya kelas saling tumpang tindih dan mengecek dengan melakukan perulangan pada minggu dan hari dari jadwal perulangan. Jika saling tumpang tindih, maka pernyataan tidak dapat terpenuhi. Jika tidak tumpang tindih, maka pernyataan benar.

12. Same Attendees

Hard Constraint ke dua belas berhubungan dengan Same Attendees sehingga kelas yang terdaftar pada batasan ini dijadwalkan berdasarkan kriteria dengan mempertimbangkan waktu perpindahan antar kelas sehingga tidak terjadi konflik untk mereka yang mengambil dan mengajar kelas tersebut. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.21.

```
for (int i = 0; i < listSA.size(); i++) {
if (idKLSA != idKLSB){
if (klsB.isHasRoom() == false) {
                iRM = 999;
            }
if (iRM == 999) {
noRoomB = true;
} else {
roomB                                                    =
klsB.getAvailableroom().get(iRM).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){
SA = true;
break outerloop;
} else {
return false;
}
} else {
SA =true;
}
```

Kode Program 5.21 Script Hard Constraint 12

Dimana metode ini mengecek apakah identitas kelas yang ada termasuk dalam daftar ini atau tidak, lalu memastikan apakah kelas itu membutuhkan ruangan untuk jadwal yang ditentukan.

Kemudian mempertimbangan jarak tempuh yang diperlukan antar kelas satu dengan lainnya dan dilakukan perulangan untuk cek apakah pasangan antar kelas tersebut dijadwalkan dalam waktu yang sama sehingga para dosen dan mahasiswa bisa mengestimasi waktu yang dihabiskan. Jika pernyataan tersebut benar, maka pengembalian nilainya benar. Jika tidak, maka salah.

13. Precedence

Hard Constraint ke tiga belas berhubungan dengan Precedence sehingga kelas yang terdaftar harus dijadwalkan pada minggu, hari, atau timeslot yang lebih awal. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.22.

```
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 Program 5.22 Script Hard Constraint 13

Langkah pertama yang diakukan adalah mengecek apakah kelas yang terjadwal pada waktu awal dengan melakukan perulangan apakah minggu kelas pertama lebih awal dibandingkan kelas berikutnya. Lalu cek perulangan hari dengan kondisi yang sama dan diikuti dengan waktu mulai kelas kedua lebih keci atau sama dengan waktu mulai kelas pertama. Jika benar maka pengembaliannya benar, jika tidak maka salah.

14. Work days (S)

Hard Constraint ke empat belas berhubungan dengan Work Days sehingga kelas yang terdafttar pada batasan ini tidak boleh dijadwalkan melebihi timeslot yang telah ditentukan. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.23.

```
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;
} else {
return false;
}
```

Kode Program 5.23 Script Hard Constraint 14

Dimana metode ini melakukan perulangan pada jadwal kelas, untuk mengecek pada minggu berapa kelas dijadwalkan, 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.

15. Min Gap (G)

Hard Constraint ke lima belas berhubungan dengan Min Gap sehingga dua kelas yang dijadwalkan pada hari yang bersamaan

harus memiliki setidaknya timeslot sebesar G yang terpisah (antara berakhirnya kelas pertama dan dimulainya kelas kedua). Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.24.

```
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 Program 5.24 Script Hard Constraint 15

Metode ini melakukan cek terhadap jumlah slot yang dihabiskan untuk selisih jeda antara suatu kelas dengan kelas lain. Langkah pertama yang dilakukan adalah perulangan minggu kemudian perulangan hari dan kemudian dicek apakah waktu berakhirnya kelas pertama ditambah dengan sejumlah minimal slot lebih kecil bila dibandingkan dengan waktu mulai kelas berikutnya. Jika pernyataan terpenuhi, maka pengembalian nilai benar, jika tidak maka salah.

16. Max Days (D)

Hard Constraint ke enam belas berhubungan dengan Max Days sehingga kelas – kelas pada batasan ini tidak boleh dijadwalkan ke lebih dari D hari kerja yang berbeda dalam satu minggu. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.25. Dimana metode ini 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 Program 5.25 Script Hard Constraint 16

Kode Program 5.26 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 Program 5.26 Script Count Non Zero Bits

17. Max Day Load (S)

Hard Constraint ke tujuh belas berhubungan dengan Max Day Load sehingga membatasi jumlah total waktu yang ditetapkan untuk sekumpulan kelas tidak lebih dari S timeslot tiap hari selama satu semester. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.27. Dimana metode ini melakukan pengecekan terhadap jumlah durasi waktu yang dihabiskan dalam sehari tidak boleh melebihi sejumlah S slot yang telah ditentukan.

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

Kode Program 5.27 Script Hard Constraint 17

Kode Program 5.28 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 Program 5.28 Script Pengecekan Dayload

18. Max Breaks (R, S)

Hard Constraint ke delapan belas berhubungan dengan Max Breaks sehingga semua kelas yang masuk dalam daftar ini tidak boleh dijadwalkan melebihi jumlah jeda sebanyak R antar kelas selama sehari dengan jumlah slot sebanyak S. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.29. Dimana 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 Program 5.29 Script Hard Constraint 18

19. Max Block (M, S)

Hard Constraint terakhir berhubungan dengan Max Block sehingga semua kelas yang ada dalam batasan ini tidak boleh dijadwalkan melebihi batas maksimal hari yang telah ditentukan setiap minggunya. Kode program dari *hard constraint* ini dapat dilihat pada Kode Program 5.30. Dimana 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 Program 5.30 Script Hard Constraint 19

5.6 Pembuatan Initial Solution

Initial Solution merupakan solusi awal yang digunakan sebagai jadwal awal sebelum di optimasi. Pada initial solution ini menerapkan algoritma *Greedy* dimana urutan pertama dalam sebuah daftar mata kuliah diletakkan pada slot tersedia pertama. Pada saat membuat *initial solution* ini sekaligus melakukan cek atau verifikasi *hard constraint* dan hanya memerhatikan *hard constraint* dan mengabaikan nilai dari *soft constraint*. Jika semua mata kuliah telah terjadwal dan memenuhi hard constraint, maka initial solution dianggap sudah terbentuk. Pembuatan solusi awal ini menggunakan *script* yang ditampilkan pada Kode Program 5.31

```
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 Program 5.31 Script Initial Solution

Metode pada Kode Program 5.32 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 Program 5.32 Script Cek Available Timeslot

Apabila telah dilakukan pengecekan terhadap *timeslot* dan ruangan, maka dibuat fungsi untuk menjadwalkan kelas ke dalam *timeslot* dan ruangan yang tersedia, seperti tampak pada Kode Program 5.33.

```
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 Program 5.33 Script Menjadwalkan Kelas

Kemudian metode selanjutnya adalah Kode Program 5.34 yang berfungsi menghapus jadwal kelas jika ternyata kelas tidak cocok berada pada jadwal tersebut. Dilakukan penetapan kembali untuk kelas tersebut menjadi -1 dan pada timeslot dirubah menjadi 0.

```
void removeKls( 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] = 0;
                  }
               }
```

Kode Program 5.34 Script Menghapus Jadwal Kelas

Initial solution yang sudah terbentuk selanjutnya disimpan untuk tahap automasi. Hal yang dilakukan untuk menyimpan *initial solution* ini terdiri yaitu penyimpanan *arraylist*.

5.7 Pembuatan Kode Program Soft Constraint

Soft Constraint ini untuk menghitung nilai penalti yang akan menjadi hasil perbandingan dari optimasi. Total nilai penalti akan dihitung berdasarkan penjadwalan mata kuliah. Kemudian semua nilai dari penalti tersebut dijumlahkan dan menghasilkan total nilai penalti yang akan digunakan sebagai pembanding setiap solusi yang dihasilkan pada optimasi.

```
int totalPenalty(ArrayList<Kelas> listKls, int[][] travel,
Distributions distri) {
      this.listKelas = listKls;
      this.distrib = distri;
      int penalty;
```

```
        int jmlPenalty = 0;
        ArrayList<Constraint> SoftCons = distrib.getSoftC();
        for (int i = 0; i < SoftCons.size(); i++) {
            Constraint Soft = SoftCons.get(i);
            ArrayList<Integer> KlsCons = Soft.getConsClass();
            String type = Soft.getType();
            int angka1 = Soft.getAngka1();
            int angka2 = Soft.getAngka2();
            penalty = Soft.getPenalty();
            if (type.contains("WorkDay") || type.contains("MinGap")
                    || type.contains("MaxDays") ||
type.contains("MaxDayLoad")
                    || type.contains("MaxBreaks") ||
type.contains("MaxBlock")) {
                jmlPenalty += Cons1(type, angka1, angka2, KlsCons,
penalty);
            } else {
                jmlPenalty += Cons2(type, KlsCons, travel, penalty);
            }
        }
        return jmlPenalty;
    }
```

Kode Program 5.35 Menghitung Total Penalty Soft Constraint

Kode Program 5.35 digunakan untuk setiap kali akan menghitung total penalty dari *soft constraint* yang ada.

1. Same Start

Soft Constraint pertama berhubungan dengan Same Start sehingga semua kelas dalam batasan ini harus dimulai pada waktu yang bersamaan. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.36.

```
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().get(TSA).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;
              }
           }
       }
       return jPenalty;
    }
```

Kode Program 5.36 Script Soft Constraint 1

2. Same Time

Soft Constraint kedua berhubungan dengan Sama Time sehingga setiap kelas harus berlangsung dalam rentang waktu yang sama. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.37

```
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().get(TSA).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;
            }
}
      }
      return jPenalty;
   }
```

Kode Program 5.37 Script Soft Constraint 2

3. Different Time

Soft Constraint ketiga berhubungan dengan Different Time sehingga setiap kelas tidak boleh berlangsung dalam rentang waktu yang sama. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.38.

```
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;
}
}
      }
      return jPenalty;
   }
```

Kode Program 5.38 Script Soft Constraint 3

4. Same Days

Soft Constraint keempat berhubungan Same Days sehingga kelas yang masuk daftar ini harus dijadwalkan di hari yang sama. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.39.

```
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;
                } else {
                    jPenalty += penalty;
                }
            }
        }
        return jPenalty;
    }
```

Kode Program 5.39 Script Soft Constraint 4

5. Different Days

Soft Constraint kelima berhubungan dengan Different Days sehingga kelas yang masuk daftar ini tidak boleh dijadwalkan di hari yang sama. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.40.

```
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;
} else {
jPenalty += penalty;
break;
}
```

Kode Program 5.40 Script Soft Constraint 5

6. Same Weeks

Soft Constraint keenam berhubungan dengan Same Weeks sehingga kelas yang masuk daftar ini harus dijadwalkan di minggu yang sama. Dimana jika batasan tersebut dilanggar akan

dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.41.

```
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 Program 5.41 Script Soft Constraint 6

7. Different Weeks

Soft Constraint ketujuh berhubungan dengan Different Weeks sehingga kelas yang masuk daftar ini harus dijadwalkan di minggu yang berbeda. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.42.

```
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++;
}else{
  jPenalty += penalty;
break;
}
```

Kode Program 5.42 Script Soft Constraint 7

8. Same Rooms

Soft Constraint kedelapan berhubungan dengan Same Rooms sehingga kelas yang masuk daftar ini harus dijadwalkan di ruangan yang sama. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.43.

```
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 Program 5.43 Script Soft Constraint 8

9. Different Rooms

Soft Constraint kesembilan berhubungan dengan Different Rooms sehingga kelas yang masuk daftar ini harus dijadwalkan di ruangan yang berbeda. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.44.

```
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 Program 5.44 Script Soft Constraint 9

10. Overlap

Soft Constraint kesepuluh berhubungan dengan Overlap sehingga kelas yang masuk daftar ini harus dijadwalkan tumpah tindih pada durasi yang berlangsung, hari dan minggu. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.45.

```
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 Program 5.45 Script Soft Constraint 10

11. Not Overlap

Soft Constraint kesebelas berhubungan dengan Not Overlap sehingga kelas yang masuk daftar ini tidak boleh dijadwalkan tumpah tindih pada durasi yang berlangsung, hari dan minggu. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.46.

```
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 Program 5.46 Script Soft Constraint 11

12. Same Attendees

Soft Constraint ke dua belas berhubungan dengan Same Attendees sehingga kelas yang terdaftar pada batasan ini dijadwalkan berdasarkan kriteria dengan mempertimbangkan waktu perpindahan antar kelas sehingga tidak terjadi konflik untk mereka yang mengambil dan mengajar kelas tersebut. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.47.

```
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 Program 5.47 Script Soft Constraint 12

13. Precedence

Soft Constraint ke tiga belas berhubungan dengan Precedence sehingga kelas yang terdaftar harus dijadwalkan pada minggu, hari, atau timeslot yang lebih awal. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.48.

```
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 Program 5.48 Script Soft Constraint 13

14. Work days (S)

Soft Constraint ke empat belas berhubungan dengan Work Days sehingga kelas yang terdafttar pada batasan ini tidak boleh dijadwalkan melebihi timeslot yang telah ditentukan. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.49.

```
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 Program 5.49 Script Soft Constraint 14

15. Min Gap (G)

Soft Constraint ke lima belas berhubungan dengan Min Gap sehingga dua kelas yang dijadwalkan pada hari yang bersamaan harus memiliki setidaknya timeslot sebesar G yang terpisah (antara berakhirnya kelas pertama dan dimulainya kelas kedua). Dimana jika batasan tersebut dilanggar akan dikenakan penalti.

Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.50.

```
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 Program 5.50 Script Soft Constraint 15

16. Max Days (D)

Soft Constraint ke enam belas berhubungan dengan Max Days sehingga kelas – kelas pada batasan ini tidak boleh dijadwalkan ke lebih dari D hari kerja yang berbeda dalam satu minggu. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.51

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

Kode Program 5.51 Script Soft Constraint 16

Kode Program 5.52 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 Program 5.52 Script Count Non Zero Bits

17. Max Day Load (S)

Soft Constraint ke tujuh belas berhubungan dengan Max Day Load sehingga membatasi jumlah total waktu yang ditetapkan untuk sekumpulan kelas tidak lebih dari S timeslot tiap hari selama satu semester. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.53

```
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 Program 5.53 Script Soft Constraint 17

Kode Program 5.54 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 Program 5.54 Script Pengecekan Dayload

18. Max Breaks (R, S)

Soft Constraint ke delapan belas berhubungan dengan Max Breaks sehingga semua kelas yang masuk dalam daftar ini tidak boleh dijadwalkan melebihi jumlah jeda sebanyak R antar kelas selama sehari dengan jumlah slot sebanyak S. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.55.

```
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 Program 5.55 Script Soft Constraint 18

19. Max Block (M, S)

Soft Constraint terakhir berhubungan dengan Max Block sehingga semua kelas yang ada dalam batasan ini tidak boleh dijadwalkan melebihi batas maksimal hari yang telah ditentukan setiap minggunya. Dimana jika batasan tersebut dilanggar akan dikenakan penalti. Kode program dari *soft constraint* ini dapat dilihat pada Kode Program 5.56.

```
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 Program 5.56 Script Soft Constraint 19

5.8 Pembuatan Low Level Heuristic

Low level heuristic (LLH) yang digunakan adalah *move. Move* dilakukan untuk memindahkan timeslot satu mata kuliah atau beberapa mata kuliah kedalam timeslot lain secara acak yang terpilih. LLH *move* yang digunakan memiliki beberapa variasi seperti *moveRM* yaitu memindah ruangan yang digunakan kelas, *moveTS* untuk memindah *timeslot* yang digunakan oleh kelas, dan *moveRMTS* untuk memindah ruangan dan waktu dari kelas. Penerapan *move* ini ditampilkan pada Kode Program 5.57.

```
int[] move1TS(){
        int klsMove[] = new int[1];
        int ts =-1, tsBaru =-1;
        ArrayList<TimeAss> times;

        do {
           int kls = rand.nextInt(listKelas.size());
           Kelas kelas = listKelas.get(kls);
           if (kelas.isHasRoom()) {
              times = kelas.getAvailableTS();
              ts = kelas.getTs();
              tsBaru = ts;
              RoomAss room =
kelas.getAvailableroom().get(kelas.getRoom());
              for (int i = 0; i < times.size(); i++) {
                 if (i!=ts) {
                    TimeAss time = kelas.getAvailableTS().get(i);
                    if (cekTS(time, room)) {
                       kelas.setTs(i);
                       tsBaru = i;
                       klsMove[0] = kls;
                       break;
                    }
                 }
              }
           } else {
              times = kelas.getAvailableTS();
              ts = kelas.getTs();
              tsBaru = ts;
              int j = ts;
              if (times.size()!=1) {
                 do {
                    j = rand.nextInt(times.size());
                 } while (ts==j);
                 kelas.setTs(j);
                 tsBaru = j;
                 klsMove[0] = kls;
              }
           }
        } while (ts==tsBaru);
```

```
        return klsMove;
    }
```

Kode Program 5.57 Script Low Level Heuristic moveTS

Pada Kode Program 5.57 hal yang pertama dilakukan adalah mengacak indeks kelas yang akan dipindahkan, apabila ketika dilakukan pengecekan *available timeslot* hasilnya adalah memungkinkan maka dilakukan pemindahan *timeslot* kelas tersebut. Namun apabila *timeslot* yang tersedia pada kelas tersebut tidak dapat lagi dimasuki maka yang dilakukan adalah melakukan kembali pengacakan indeks kelas yang akan dipindah. Method *move* berhenti apabila sudah terdapat kelas yang dipindah.

5.9 Implementasi Algoritma Iterated Local Search

Implementasi algoritma *Iterated Local Search Hyper-Heuristic* dilakukan untuk mengoptimasi solusi yang telah didapat dari *initial solution*. Algoritma *Iterated Local Search* diimplementasikan dengan cara melakukan *Local Search* pada *initial solution*, lalu dilakukan sejumlah iterasi dengan memberikan *Perturbation* atau gangguan menggunakan *Hyper-heuristic* dan dilakukan *Local Search* kembali dan melakukan pembaruan pada solusi sesuai dengan kriteria yang diinginkan

5.9.1 Algoritma Local Search

Local Search yang digunakan disini adalah *Hill Climbing*. Pada *Local Search Hill Climbing* dan *Simmulated Annealing* diberikan juga LLH yang digunakan untuk mencari solusi yang lebih baik.

```
int randLH = rand.nextInt(6);
        switch (randLH){
            case 0:
                klsLH = moveRMTS();
                break;
            case 1:
                klsLH = moveRMTS2();
                break;
            case 2:
                klsLH = move1RM();
                break;
            case 3:
                klsLH = move2RM();
                break;
            case 4:
                klsLH = move1TS();
                break;
            case 5:
                klsLH = move2TS();
                break;
            default:
                break;
        }
```

Kode Program 5.58 LLH Local Search

Pada Kode Program 5.58 dilakukan pemilihan LLH dengan cara random untuk menentukan LLH apa yang akan digunakan. Setelah menjalankan *method* LLH yang dipilih, pengecekan *hard constraint* akan dilakukan dan menghitung *penalty* dari solusi yang baru, apabila hard constraint tidak terpenuhi atau memiliki penalty lebih besar maka solusi akan dikembalikan lagi seperti sebelum menjalankan LLH dan akan dilakukan perulangan hingga menemukan solusi yang lebih baik atau limit dari perulangan telah tercapai.

5.9.2 Perturbation

Pada *perturbation* atau yang bisa disebut gangguan, hal yang dilakukan hampir sama seperti proses *Local Search*. Pemberian

gangguan dilakukan dengan menggunakan LLH yang ada dan diberikan pada awal setiap iterasi. Solusi dengan nilai penalti lebih kecil akan diterima dan menggantikan solusi sebelumnya. Namun apabila solusi yang dihasilkan memiliki nilai penalti lebih besar maupun solusi yang dihasilkan tidak *feasible*, maka solusi akan dikembalikan menjadi solusi terakhir yang telah diterima.

```
if (delta > 0 && newPenalty != X) {
            currPenalty = newPenalty;
            penaltyTS = newPenaltyTS;
            penaltyRM = newPenaltyRM;
            penaltySC = newPenaltySC;
            for (int i = 0; i < klsLH.length; i++) {
               Kelas kelas = listKelas.get(klsLH[i]);
               int idKls = kelas.getClassID();
               if (kelas.isHasRoom()) {
                  int tsLama = kelas.getTsLama();
                  int rmLama = kelas.getRoomLama();
                  TimeAss time =
kelas.getAvailableTS().get(tsLama);
                  RoomAss room =
kelas.getAvailableroom().get(rmLama);
                  removeKls(time, room);

                  int ts = kelas.getTs();
                  int rm = kelas.getRoom();
                  time = kelas.getAvailableTS().get(ts);
                  room = kelas.getAvailableroom().get(rm);
                  assign(idKls, time, room);

                  kelas.setTsLama(ts);
                  kelas.setRoomLama(rm);
               } else {
                  kelas.setTsLama(kelas.getTs());
                  kelas.setRoomLama(kelas.getRoom());
               }
            }
         } else {
            for (int i = 0; i < klsLH.length; i++) {
               Kelas kelas = listKelas.get(klsLH[i]);
```

```
                kelas.setTs(kelas.getTsLama());
                kelas.setRoom(kelas.getRoomLama());
            }
        }
```

Kode Program 5.59 Algoritma penerimaan solusi

Kode Program 5.59 menjelaskan alur penerimaan untuk setiap solusi yang telah terbentuk dan telah *feasible*.

5.9.3 Perhitungan Penalti Akhir

Nilai penalti akhir merupakan total dari nilai penalti untuk *timeslot* yang digunakan, nilai penalti ruangan yang digunakan, dan penalti dari *soft constraint*. Nilai penalti akhir akan menentukan solusi yang paling optimum dalam beberapa eksperimen yang dilakukan.

```
if (hardC) {
            newPenaltyTS = soft.countTSPenalty(listKelas) *
bobot[0];
            newPenaltyRM = soft.countRMPenalty(listKelas)
* bobot[1];
            newPenaltySC = soft.totalPenalty(listKelas, travel,
distri) * bobot[2];
            newPenalty = newPenaltyTS + newPenaltyRM
+newPenaltySC;
        }
```

Kode Program 5.60 Menghitung Penalti dari Solusi Baru

Kode Program 5.60 menunjukkan untuk setiap solusi yang *feasible* akan dilakukan perhitungan terkait keseluruhan penalti dari solusi tersebut. Selanjutnya, penalti tersebut akan dibandingkan dengan solusi sebelumnya. Apabila solusi baru memiliki nilai penalti lebih kecil, maka akan menjadi solusi terkini yang akan ditunjukkan pada Kode Program 5.61.

```
currPenalty = newPenalty;
penaltyTS = newPenaltyTS;
penaltyRM = newPenaltyRM;
penaltySC = newPenaltySC;
```

Kode Program 5.61 Menerima Penalti Terbaru

Halaman ini sengaja dikosongkan

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini akan dijelaskan mengenai hasil implementasi yang sudah dikerjakan pada bab sebelumnya.

6.1 Data Uji Coba

Data yang digunakan untuk uji coba adalah data dari ITC 2019. Data dari ITC 2019 terdiri dari dua tipe, yaitu *test* dan *early*.

6.2 Pembuatan Initial Solution

Pada bagian ini menjelaskan tentang urutan indeks mata kuliah yang digunakna untuk pembentukan *initial solution*. Urutan indeks mata kuliah harus diperhatikan untuk mendapatkan solusi yang lolos *hard constraint*. Beberapa indikator yang digunakan untuk mengurutkan adalah jumlah daftar waktu tersedia (*available times*), jumlah daftar ruang tersedia (*available rooms*), jumlah *hard constraint* per kelas, dan jumlah kelas yang sudah di urutkan (*sorted class*).

6.2.1 Skenario 1 Indikator

Skenario 1 ini tidak dilakukan kombinasi indikator, sehingga urutan objek hanya berdasarkan 1 indikator terkait. Hasil skenario 1 ini ditampilkan pada Tabel 6.1, dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah kelas yang belum dijadwalkan.

Tabel 6.1 Hasil Skenario 1 Pembuatan Initial Solution

Instances	Indikator		
	Timeslot	Rooms	Hard Constraint
Test-Tiny	**0**	**0**	**0**
Test-Small 1	12	9	19
Test-Small 2	**0**	**0**	**0**
Test-Medium	20	14	28
Test-Large	158	114	174
Early 1	39	63	66
Early 2	101	127	154

Early 3	268	280	285
Early 4	120	139	158
Early 5	11	18	16
Early 6	28	35	37
Early 7	34	49	45
Early 8	83	131	125
Early 9	29	16	29
Early 10	28	62	76
Rata-rata	**62**	**70**	**81**

6.2.2 Skenario 2 Indikator

Pada Tabel 6.1 menunjukkan bahwa indikator rooms dan timeslot memiliki hasil yang lebih baik daripada hard constraint. Maka dari itu, pada skenario 2 ini melibatkan 2 indikator yang berhubungan dengan *rooms* dan *timeslot*. Hasil yang didapatkan pada skenario ini dapat dilihat pada Tabel 6.2.

Tabel 6.2 Hasil Skenario 2 Pembuatan Initial Solution

Instances	**Indikator**			
	Time - Room	**Times - HC**	**Rooms - Times**	**Rooms - HC**
Test-Tiny	**0**	**0**	**0**	**0**
Test-Small 1	11	12	9	9
Test-Small 2	**0**	**0**	**0**	**0**
Test-Medium	17	20	10	14
Test-Large	148	158	92	114
Early 1	46	39	59	63
Early 2	116	101	116	127
Early 3	268	268	266	280
Early 4	114	120	128	139
Early 5	13	11	14	18
Early 6	25	28	33	35
Early 7	39	34	42	49
Early 8	91	83	124	131
Early 9	28	29	14	16
Early 10	26	28	43	62
Rata-rata	**63**	**62**	**56**	**70**

6.2.3 Skenario 3 Indikator

Pada Tabel 6.2 menunjukkan bahwa kombinasi *Rooms-Times* dan *Times-HC* memiliki hasil yang paling baik daripada kombinasi lain. Pada skenario ini melibatkan 3 kombinasi indikator, dimana indikator *Rooms-Times* dan *Times-HC* terdapat pada susunan pertama dan kedua. Hasil dari skenario ini dapat dilihat pada Tabel 6.3.

Tabel 6.3 Hasil Skenario 3 Pembuatan Initial Solution

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

Pada Tabel 6.3, indikator *Sorted Class* atau kombinasi kelas yang sudah diurutkan menunjukkan hasil rata-rata yang paling bagus.

Pada Tabel 6.1, Tabel 6.2, dan Tabel 6.3 menunjukkan bahwa tidak ada kombinasi indikator yang mampu menjadwalkan semua kelas pada dataset early. Hanya pada dataset tiny dan small saja yang berhasil dijadwalkan semua kelas nya. Terdapat

pula uji coba yang dilakukan untuk menangani kelas-kelas yang tidak terjadwalkan dengan menggunakan *backtrack,* dimana metode ini mengulang terus iterasi sehingga menyebabkan iterasi tidak bisa berhenti jika belum menemukan solusi yang tepat, juga sudah dilakukan untuk menemukan solusi dari data yang belum terjadwalkan.

Namun uji coba yang dilakukan dengan metode *bactktrack* tidak meghasilkan hasil yang memuaskan. Sehingga, pada penelitian ini dilanjutkan dengan 2 dataset yaitu *Test-Tiny* dan *Test-Small 2*. Sementara itu kombinasi yang digunakan sebagai pembentuk solusi awal sebelum dilakukan proses optimasi adalah *sorted class* karena mampu menghasilkan rata-rata yang paling baik.

6.3 Hasil Initial Solution

Pada subbab ini menjelaskan mengenai hasil implementasi *initial solution* awal terhadap dataset. Dari implementasi yang dilakukan, didapatkan nilai *penalty* masing-masing *instance.* Untuk hasil *initial solution* awal dapat dilihat pada Tabel 6.4.

Tabel 6.4 Hasil Initial Solution Awal

Instance	Avg Initial Solution
Test-Tiny	48
Test-Small 2	1790

Setelah mendapatkan nilai *initial solution* awal, nilai tersebut dilanjutkan untuk di optimasi dengan berbagai skenario yang sudah didefinisikan pada proses berikutnya.

6.4 Hasil Eksperimen Algoritma Iterated Local Search

Pada bagian ini menjelaskan mengenai implementasi algoritma *Iterated Local Search* untuk melakukan optimasi solusi pada initial solution yang sudah dibangun pada proses sebelumnya. Terdapat 2 macam algoritma yang diterapkan, yaitu ILS menggunakan Hill Climbing sebagai Local Search dan Simmulated Annealing sebagai Local Search.Terdapat pula

beberapa penjelasan dan hasil dari skenario-skenario yang dilakukan untuk dapat menghasilkan solusi yang optimal. Pada Tabel 6.5 menunjukkan beberapa nilai parameter yang dilibatkan dalam skenario-skenario.

Tabel 6.5 Parameter Percobaan

Parameter	Keterangan
LLH	*Low level heuristics* yang digunakan
Max Iterasi LS	Iterasi maksimal pada Algoritma Local Search
Iterasi ILS	Jumlah Iterasi yang dilakukan pada ILS
To (khusus ILS SA)	Suhu awal algoritma Simmulated Annealing
Alpha (khusus ILS SA)	Penurunan suhu

6.4.1 Iterated Local Search dengan Hill Climbing

Pada bagian ini menjelaskan hasil implementasi dari algoritma *Iterated Local Search* dengan menggunakan *local search Hill Climbing.*

6.4.1.1 Skenario 1

Pada Skenario 1 melakukan percobaan terhadap parameter Iterasi ILS dengan *range* 10 hingga 10000. Untuk parameter Max Iterasi ILS dan LLH tidak berubah nilainya. Sehingga luaran pada skenario 1 ini adalah mengetahui dampak variabel dari Iterasi ILS. Hasil dari skenario 1 ditampilkan pada Tabel 6.6, Tabel 6.7, dan Tabel 6.8 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.6 Hasil Skenario 1 (Iterasi 10, 50, dan 100)

Max. Iter. LS	10					
Iterasi ILS	10		50		100	
Jumlah LLH	6		6		6	
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	42	1580	39	1310	39	1220
2	42	1482	42	1180	43	1150
3	37	1592	43	1240	41	1208
4	38	1492	22	1226	43	1108
5	41	1594	38	1222	42	1192
6	39	1554	38	1340	37	1194
7	39	1506	41	1184	37	1092
8	43	1454	41	1154	37	1280
9	41	1568	38	1154	43	1188
10	43	1568	43	1242	23	1162
Average Penalty	**40.5**	**1539**	**38.5**	**1225.2**	**38.5**	**1179.4**
Worst	43	1594	43	1340	43	1280
Best	37	1454	22	1154	23	1092

Tabel 6.6 menunjukkan hasil Sekenario 1 dengan iterasi sejumlah 10, 50, dan 100. Hasil menunjukkan bahwa iterasi 100 memiliki nilai rata-rata penalti yang lebih baik dari pada iterasi 10 dan 50. Nilai rata-rata yang dihasilkan alogoritma dengan iterasi adalah 1179.4 untuk data Small dan 38.5 untuk data Tiny.

Tabel 6.7 Hasil Skenario 1 (Iterasi 500, 1.000, dan 1.500)

Max. Iter. LS	10					
Iterasi ILS	500		1000		1500	
Jumlah LLH	6		6		6	
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	18	1086	22	1076	17	1222
2	21	1100	18	1048	18	1074
3	22	1034	22	1034	22	1038
4	17	1138	17	1102	21	1100

Max. Iter. LS	10					
Iterasi ILS	500		1000		1500	
Jumlah LLH	6		6		6	
5	39	1208	19	1036	19	1196
6	21	1112	17	1066	17	1070
7	21	1022	18	1198	22	1006
8	42	1224	37	998	21	1038
9	21	998	21	1160	18	1062
10	17	1036	21	1104	17	1044
Average	**23.9**	**1095.8**	**21.2**	**1082.2**	**19.2**	**1085**
Worst	42	1224	37	1198	22	1222
Best	17	998	17	998	17	1006

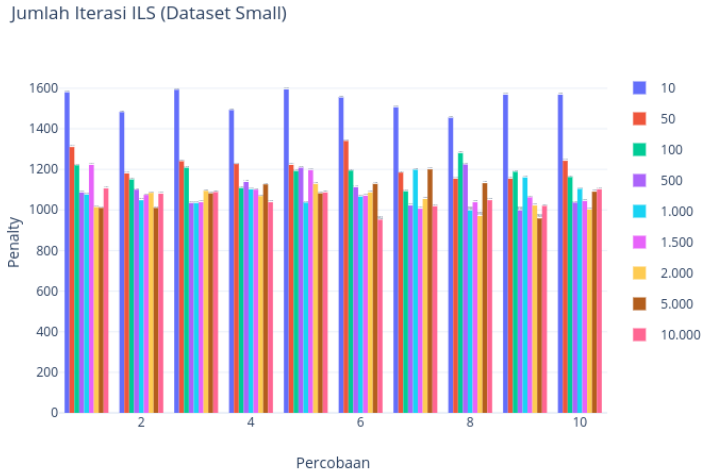
Tabel 6.7 menunjukkan hasil Sekenario 1 dengan iterasi sejumlah 500, 1000, dan 1500. Hasil menunjukkan bahwa iterasi 1000 memiliki nilai rata-rata penalti yang lebih baik ketika menggunakan data Small, namun ketika menggunakan data Tiny yang menghasilkan nilai rata-rata lebih baik adalah iterasi 1500. Nilai rata-rata terbaik yang dihasilkan alogoritma menggunakan data Tiny adalah 19.2 dengan iterasi 1500 dan untuk data Small adalah 1082.2 dengan iterasi 1000.

Tabel 6.8 Hasil Skenario 1 (Iterasi 2.000, 5.000, 10.000)

Max. Iter. LS	10					
Iterasi ILS	2000		5000		10000	
LLH	6		6		6	
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	21	1014	22	1010	22	1106
2	42	1084	17	1010	22	1080
3	21	1094	22	1082	17	1088
4	22	1068	42	1126	21	1038
5	17	1128	22	1082	22	1086
6	17	1086	18	1128	18	952
7	19	1054	23	1200	22	1018
8	22	970	23	1132	17	1048

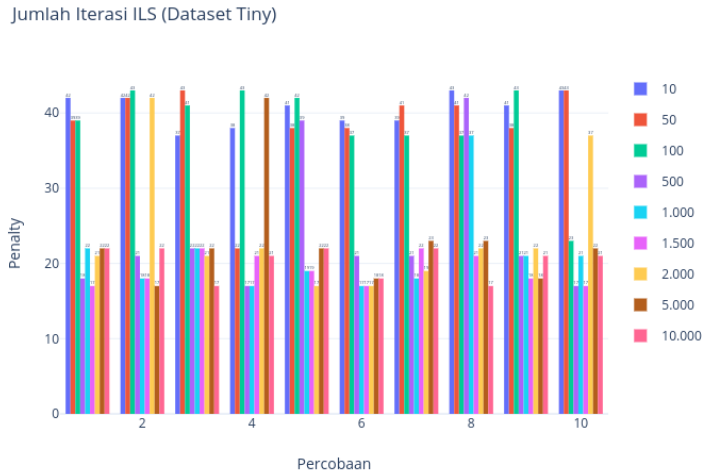
Max. Iter. LS	10					
Iterasi ILS	2000		5000		10000	
LLH	6		6		6	
9	22	1022	18	958	21	1020
10	37	1002	22	1090	21	1102
Average	**24**	**1052.2**	**22.9**	**1081.8**	**20.3**	**1053.8**
Worst	42	1128	42	1200	22	1106
Best	17	970	17	958	17	952

Tabel 6.8 menunjukkan hasil Sekenario 1 dengan iterasi sejumlah 2000, 5000, dan 10000. Hasil menunjukkan bahwa iterasi 2000 memiliki nilai rata-rata penalti yang lebih baik ketika menggunakan data Small, namun ketika menggunakan data Tiny yang menghasilkan nilai rata-rata lebih baik adalah iterasi 10000. Nilai rata-rata terbaik yang dihasilkan alogoritma menggunakan data Tiny adalah 20.3 dengan iterasi 10000 dan untuk data Small adalah 1052.2 dengan iterasi 2000.



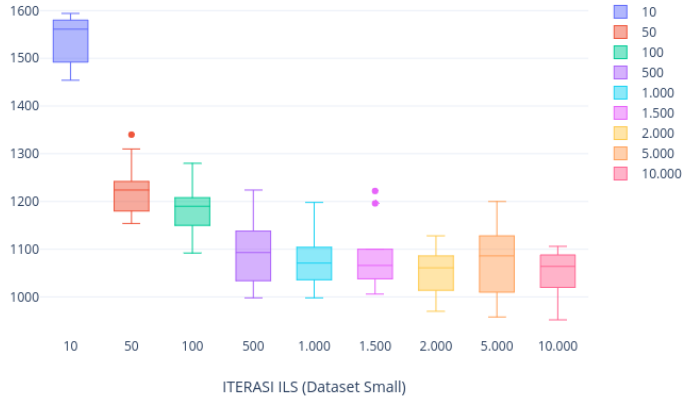
Gambar 6.1 Grafik Iterasi ILS Small

Pada Gambar 6.1 menunjukkan hasil penalty setiap iterasi setiap percobaan pada data Small.

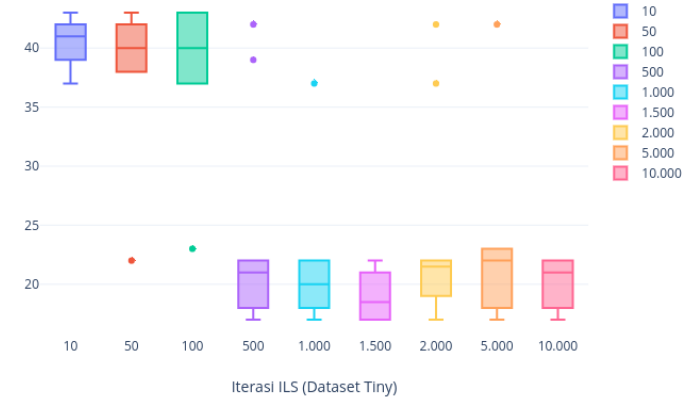


Gambar 6.2 Grafik Iterasi ILS Tiny

Gambar 6.2 menunjukkan hasil penalty setiap iterasi setiap percobaan pada data Tiny. Pada kedua grafik tersebut menunjukkan bahwa jika iterasi ILS terlalu kecil maka hasil penalty akan lebih buruk. Pada data small, ketika iterasi dilakukan cenderung menunjukkan performa yang stabil pada setiap iterasi. Ditunjukkan pada Gambar 6.3 yaitu diagram boxplot yang menampilkan adanya sedikit saja data *outlier* pada iterasi ke 50 dan 1500. Beda hal nya dengan ketika data tiny dilakukan uji coba iterasi ILS nya. Pada Gambar 6.4 menunjukkan boxplot iterasi ILS pada data tiny. Pada data tiny, ketika dilakukan uji coba iterasi ILS cenderung tidak stabil dikarenakan banyak data yang cenderung menghasilkan nilai penalty yang lebih tinggi atau lebih rendah dari persebaran.



Gambar 6.3 Box Plot Iterasi ILS Small



Gambar 6.4 Box Plot Iterasi ILS Tiny

6.4.1.2 Skenario 2

Pada Skenario 2 melakukan percobaan terhadap parameter Max Iterasi LS dengan *range* 2 hingga 100. Untuk parameter Iterasi ILS dan LLH tidak berubah nilainya. Pada Tabel 6.9 dipilih Iterasi ILS 1500, karena memilih iterasi yang masih belum mencapai solusi yang optimal atau belum terkena local optima, karena ingin mengetahui dampak dari variael Iterasi LS. Pada skenario ini ingin mengetahui dampak variabel dari Iterasi ILS tersebut dan Max Iterasi LS. Hasil dari skenario 2 ditampilkan pada Tabel 6.9 dan Tabel 6.10 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

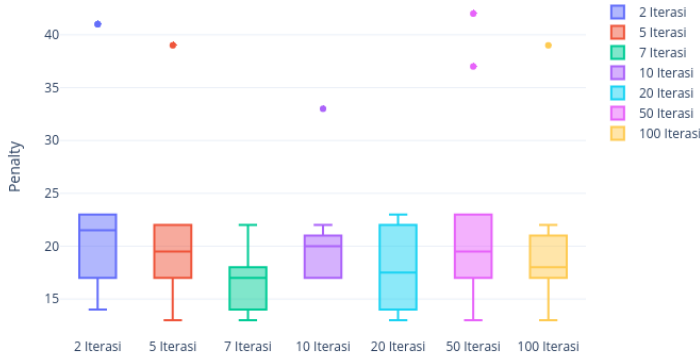
Tabel 6.9 Hasil Skenario 2 (Iterasi 2, 5, dan 7)

Iterasi ILS	1500					
Max. Iter. LS	2		5		7	
LLH	6		6		6	
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	17	1172	18	1138	22	998
2	21	1058	13	1158	18	1072
3	41	1108	21	1170	17	1008
4	14	1082	22	1060	14	1150
5	23	1102	17	1034	19	1054
6	22	1110	17	1038	13	1026
7	17	1128	39	1118	17	1098
8	22	1158	14	1118	18	1064
9	21	1118	22	1118	13	1108
10	41	1006	22	1194	14	1062
Average	**23.9**	**1104.2**	**20.5**	**1114.6**	**16.5**	**1064**
Worst	41	1172	39	1194	22	1150
Best	14	1006	13	1034	13	998

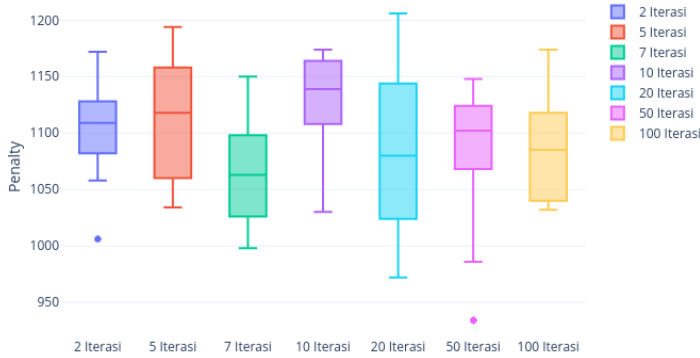
Tabel 6.10 Hasil Skenario 2 (Iterasi 10, 20, dan 50)

Iterasi ILS	1500					
Max. Iter. LS	10		20		50	
LLH	6		6		6	
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	17	1048	14	1106	18	1114
2	17	1134	17	1066	18	1090
3	22	1144	23	972	23	1124
4	21	1122	22	1144	42	1068
5	21	1108	22	1024	22	1114
6	19	1164	18	1158	13	1074
7	17	1030	13	1094	17	934
8	21	1154	13	1206	37	1148
9	33	1174	17	1012	17	986
10	19	1164	18	1062	21	1124
Average	20.7	1184	17.7	1084.4	22.8	1077.6
Worst	33	1174	23	1206	42	1148
Best	17	1030	13	972	13	934

Dari perbandingan Tabel 6.9 dan Tabel 6.10**Error! Reference source not found.** menunjukkan bahwa iterasi 7 menghasilkan nilai penalty yang paling kecil baik bagi dataset tiny maupun small, sehingga iterasi ini yang digunakan untuk skenario berikutnya. Selain memiliki penalty paling kecil, iterasi 7 ini memiliki performa yang stabil baik bagi data tiny maupun small, dimana tidak ada nilai yang melebihi atau kurang dari persebaran data yang ada. Persebaran tersebut dapat dilihat pada Gambar 6.5 dan untuk data tiny serta Gambar 6.6 dan untuk data small.

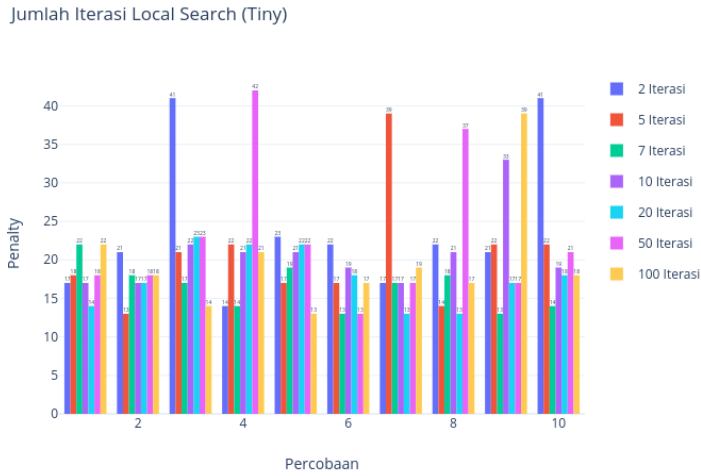


Gambar 6.5 Box Plot Iterasi LS Tiny

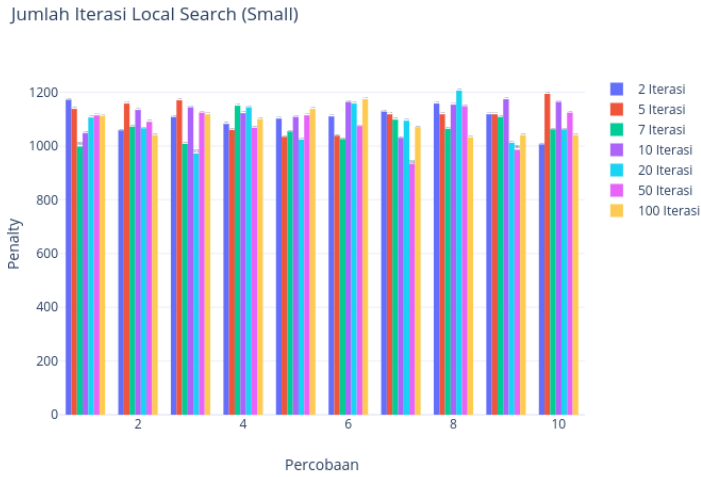


Gambar 6.6 Box Plot Iterasi LS Small

Sedangkan untuk perbandingan keseluruhan percobaan iterasi dapat dilihat pada Gambar 6.7 untuk data Tiny dan Gambar 6.8 untuk data Small.



Gambar 6.7 Grafik Iterasi LS Tiny



Gambar 6.8 Grafik Iterasi ILS Small

6.4.1.3 Skenario 3

Pada Skenario 3 melakukan percobaan terhadap parameter LLH dengan penggunaan kombinasi 5 LLH dari 6 LLH yang ada. Untuk parameter Iterasi ILS dan Max Iterasi LS tidak berubah nilainya. Iterasi ILS yang digunakan adalah 7, karena pada Tabel 6.9 iterasi 7 menghasilkan penalty yang paling kecil diantara iterasi lainnya baik dari dataset tiny maupun small. Pada Tabel 6.11 merupakan LLH apa saja yang digunakan untuk kombinasi.

Tabel 6.11 LLH yang digunakan

ID	Jenis LLH	Keterangan
a	Move1TS	1 kelas dipindah kedalam timeslot yang berbeda
b	Move2TS	2 kelas dipindah kedalam timeslot-timeslot yang berbeda
c	Move1RM	1 kelas dipindah kedalam ruangan yang berbeda
d	Move2RM	2 kelas dipindah kedalam ruangan-ruangan kelas yang berbeda
e	moveRMTS	1 kelas dapat dipindah ruangannya saja atau dipindah timeslotnya saja atau dipindah kedua-duanya
f	moveRMTS2	2 kelas dapat dipindah ruangannya saja atau dipindah timeslotnya saja atau dipindah kedua-duanya

1. Kombinasi pertama adalah kombinasi dari LLH move1TS, move2TS, move1RM, move2RM dan moveRMTS (abcde) yang dapat dilihat pada Tabel 6.12 dimana nilai-nilai yang ada didalam kolom tersebut

merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.12 Kombinasi 5 LLH 1

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abcde	
Percobaan	TINY	SMALL
1	18	1032
2	17	1146
3	22	1092
4	41	1032
5	21	1172
6	17	1104
7	23	1168
8	17	1186
9	18	1128
10	14	1132
Average	20.8	1119.2
Worst	41	1186
Best	14	1032

2. Kombinasi kedua adalah kombinasi dari LLH move1TS, move2TS, move1RM, move2RM, moveRMTS2 (abcdf) yang dapat dilihat pada Tabel 6.13 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.13 Kombinasi 5 LLH 2

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abcdf	
Percobaan	TINY	SMALL
1	17	1176
2	19	1128

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abcdf	
3	37	1152
4	17	976
5	17	1040
6	19	1082
7	23	1154
8	19	1088
9	37	1084
10	22	1086
Average	22.7	1096.6
Worst	37	1176
Best	17	976

3. Kombinasi ketiga adalah kombinasi dari LLH move1TS, move2TS, move1RM, moveRMTS, moveRMTS2 (abcef) yang dapat dilihat pada Tabel 6.14 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.14 Kombinasi 5 LLH 3

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abcef	
Percobaan	TINY	SMALL
1	22	1058
2	18	1138
3	22	1076
4	18	1162
5	19	1232
6	18	1080
7	17	1114
8	19	1252
9	22	1186

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abcef	
10	18	1108
Average	19.3	1140.6
Worst	22	1252
Best	17	1058

4. Kombinasi keempat adalah kombinasi dari LLH move1TS, move1RM, move2RM, moveRMTS, moveRMTS2 (acdef) yang dapat dilihat pada Tabel 6.15 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.15 Kombinasi 5 LLH 4

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	acdef	
Percobaan	TINY	SMALL
1	42	1128
2	42	1146
3	38	1024
4	42	1212
5	37	1100
6	38	1136
7	41	972
8	37	1078
9	39	990
10	38	1084
Average	39.4	1087
Worst	42	1212
Best	37	972

5. Kombinasi kelima adalah move1TS, move2TS, move2RM, moveRMTS, moveRMTS2 (abdef) yang

dilihat pada Tabel 6.16 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.16 Kombinasi 5 LLH 5

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abdef	
Percobaan	TINY	SMALL
1	14	1086
2	38	1150
3	17	1086
4	18	1194
5	13	1108
6	17	1078
7	21	1084
8	18	956
9	34	1144
10	22	1116
Average	21.2	1100.2
Worst	38	1194
Best	13	956

6. Kombinasi keenam adalah move2TS, move1RM, move2RM, moveRMTS, moveRMTS2 yang dilihat pada Tabel 6.17 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.17 Kombinasi 5 LLH 6

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bcdef	
Percobaan	TINY	SMALL
1	19	1180

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bcdef	
2	21	1106
3	22	1070
4	17	1080
5	21	1124
6	18	1170
7	22	1084
8	22	1112
9	22	954
10	17	1108
Average	20.1	1098.8
Worst	22	1180
Best	17	954

6.4.1.4 Skenario 4

Pada Skenario 4 melakukan percobaan terhadap parameter LLH dengan penggunaan kombinasi 4 LLH dari 6 LLH yang ada. Untuk parameter Iterasi ILS dan Max Iterasi LS tidak berubah nilainya. Iterasi ILS yang digunakan adalah 7, karena pada Tabel 6.9 iterasi 7 menghasilkan penalty yang paling kecil diantara iterasi lainnya baik dari dataset tiny maupun small. Pada Tabel 6.10 merupakan LLH apa saja yang digunakan untuk kombinasi.

1. Kombinasi pertama adalah move1TS, move2TS, move1RM, move2RM yang dapat dilihat pada Tabel 6.18 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.18 Kombinasi 4 LLH 1

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abcd	
Percobaan	TINY	SMALL
1	22	1194
2	18	968
3	22	1188
4	17	958
5	13	1136
6	23	1190
7	17	1168
8	22	1112
9	18	1284
10	22	1132
Average	**19.4**	**1133**
Worst	23	1284
Best	13	958

2. Kombinasi kedua adalah move1TS, move2TS, move1RM, moveRMTS yang dapat dilihat pada Tabel 6.19 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.19 Kombinasi 4 LLH 2

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abce	
Percobaan	TINY	SMALL
1	33	1070
2	23	1102
3	18	1064
4	39	1188
5	22	962
6	17	1076

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abce	
7	17	1084
8	18	1134
9	21	1128
10	17	1068
Average	22.5	1087.6
Worst	39	1188
Best	17	962

3. Kombinasi ketiga adalah move1TS, move2TS, move1RM, moveRMTS2 yang dapat dilihat pada Tabel 6.20 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.20 Kombinasi 4 LLH 3

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abcf	
Percobaan	TINY	SMALL
1	19	1148
2	37	1146
3	37	1124
4	22	1030
5	42	1030
6	17	1232
7	17	1124
8	22	1204
9	38	988
10	22	1124
Average	27.3	1115
Worst	42	1232
Best	17	988

4. Kombinasi keempat adalah move1TS, move1RM, move 2RM, moveRMTS yang dapat dilihat pada Tabel 6.21 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.21 Kombinasi 4 LLH 4

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	acde	
Percobaan	TINY	SMALL
1	41	1166
2	42	1116
3	41	1012
4	38	1114
5	37	1188
6	33	1046
7	41	1132
8	41	1052
9	38	1010
10	38	1014
Average	39	1085
Worst	42	1188
Best	33	1010

5. Kombinasi kelima adalah move1TS, move1RM, move2RM, moveRMTS2 yang dapat dilihat pada Tabel 6.22 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.22 Kombinasi 4 LLH 5

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	acdf	
Percobaan	TINY	SMALL

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	acdf	
1	43	1144
2	33	1166
3	42	1106
4	38	1094
5	37	1112
6	38	1150
7	41	1088
8	41	1014
9	36	1132
10	37	1238
Average	38.6	1124.4
Worst	43	1238
Best	33	1014

6. Kombinasi keenam adalah move1TS, move1RM, moveRMTS, moveRMTS2 yang dapat dilihat pada Tabel 6.23 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.23 Kombinasi 4 LLH 6

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	acef	
Percobaan	TINY	SMALL
1	42	1152
2	38	1090
3	42	1030
4	38	1212
5	39	1122
6	42	1126
7	42	1152

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	acef	
8	34	1140
9	33	1194
10	37	1152
Average	38.7	1137
Worst	42	1212
Best	33	1030

7. Kombinasi ketujuh adalah move1TS, move2RM, moveRMTS, moveRMTS2 yang dapat dilihat pada Tabel 6.24 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.24 Kombinasi 4 LLH 7

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	adef	
Percobaan	TINY	SMALL
1	38	1186
2	38	978
3	37	1140
4	42	1048
5	33	1136
6	41	1146
7	37	1138
8	37	1012
9	41	990
10	42	1168
Average	38.6	1094.2
Worst	42	1186
Best	33	978

8. Kombinasi kedelapan adalah move1TS, move2TS, moveRMTS, moveRMTS2 yang dapat dilihat pada Tabel 6.25 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.25 Kombinasi 4 LLH 8

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abef	
Percobaan	TINY	SMALL
1	22	1156
2	22	1008
3	22	1174
4	22	1030
5	19	1064
6	22	1156
7	18	1112
8	22	1120
9	22	1238
10	23	1176
Average	21.4	1123.4
Worst	23	1238
Best	18	1008

9. Kombinasi kesembilan adalah move2TS, move1RM, move2RM, moveRMTS dapat dilihat pada Tabel 6.26 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.26 Kombinasi 4 LLH 9

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bcde	
Percobaan	TINY	SMALL

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bcde	
1	17	1096
2	14	1122
3	21	1150
4	17	1204
5	14	986
6	15	1128
7	13	1134
8	21	1036
9	18	1166
10	41	1122
Average	19.1	1114.4
Worst	41	1204
Best	13	986

10. Kombinasi kesepuluh adalah move2TS, move1RM, move2RM, moveRMTS2 dapat dilihat pada Tabel 6.27 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.27 Kombinasi 4 LLH 10

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bcdf	
Percobaan	TINY	SMALL
1	38	1126
2	42	1116
3	13	1108
4	22	1058
5	18	1028
6	22	1130
7	38	954

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bcdf	
8	14	1180
9	21	986
10	17	1012
Average	24.5	1069.8
Worst	42	1180
Best	13	954

11. Kombinasi kesebelas adalah move2TS, move2RM, moveRMTS, moveRMTS2 dapat dilihat pada Tabel 6.28 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.28 Kombinasi 4 LLH 11

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bdef	
Percobaan	TINY	SMALL
1	15	1050
2	22	1024
3	22	1164
4	18	1128
5	18	1066
6	17	1226
7	17	1068
8	17	1118
9	21	890
10	17	1156
Average	18.4	1089
Worst	22	1226
Best	15	890

12. Kombinasi keduabelas adalah move2TS, move1RM, moveRMTS, moveRMTS2 dapat dilihat pada Tabel 6.29 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.29 Kombinasi 4 LLH 12

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bcef	
Percobaan	TINY	SMALL
1	23	1126
2	18	1168
3	21	980
4	18	1100
5	14	1068
6	22	1150
7	18	1036
8	19	1048
9	22	1216
10	18	1134
Average	19.3	1102.6
Worst	23	1216
Best	14	980

13. Kombinasi ketigabelas adalah move1RM, move2RM, moveRMTS, moveRMTS2 dapat dilihat pada Tabel 6.30 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.30 Kombinasi 4 LLH 13

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	cdef	
Percobaan	TINY	SMALL

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	cdef	
1	42	1198
2	41	1274
3	38	1178
4	42	1236
5	38	1360
6	41	1172
7	41	1296
8	39	1274
9	41	1210
10	37	1244
Average	40	1244.2
Worst	42	1360
Best	37	1172

14. Kombinasi keempatbelas adalah move1TS, move2TS, move2RM, moveRMTS dapat dilihat pada Tabel 6.31 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.31 Kombinasi 4 LLH 14

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abde	
Percobaan	TINY	SMALL
1	18	1080
2	18	1154
3	22	1106
4	17	992
5	22	1020
6	17	1066
7	18	1102
8	18	1054

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abde	
9	22	1062
10	18	1162
Average	19	1079.8
Worst	22	1162
Best	17	992

15. Kombinasi keduabelas adalah move1TS, move2TS, move2RM, moveRMTS2 dapat dilihat pada Tabel 6.32 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.32 Kombinasi 4 LLH 15

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abdf	
Percobaan	TINY	SMALL
1	17	1136
2	19	956
3	15	1154
4	21	1134
5	37	1020
6	22	1042
7	17	1074
8	14	1228
9	23	1072
10	19	1080
Average	20.4	1089.6
Worst	37	1228
Best	14	956

6.4.1.5 Skenario 5

Pada Skenario 5 melakukan percobaan terhadap parameter LLH dengan penggunaan kombinasi 3 LLH dari 6 LLH yang ada. Untuk parameter Iterasi ILS dan Max Iterasi LS tidak berubah nilainya. Iterasi ILS yang digunakan adalah 7, karena pada Tabel 6.9 iterasi 7 menghasilkan penalty yang paling kecil diantara iterasi lainnya baik dari dataset tiny maupun small. Pada Tabel 6.11 merupakan LLH apa saja yang digunakan untuk kombinasi.

1. Kombinasi pertama adalah move1TS, move2TS, move1RM dapat dilihat pada Tabel 6.33 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.33 Kombinasi 3 LLH 1

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abc	
Percobaan	TINY	SMALL
1	22	1242
2	18	1228
3	19	1160
4	18	1212
5	22	1086
6	23	1170
7	23	1218
8	23	1196
9	23	1164
10	18	1144
Average	20.9	1182
Worst	23	1242
Best	18	1086

2. Kombinasi kedua adalah move1TS, move2TS, move2RM dapat dilihat pada Tabel 6.34 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.34 Kombinasi 3 LLH 2

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abd	
Percobaan	TINY	SMALL
1	21	1078
2	17	1054
3	23	1208
4	17	1174
5	18	1128
6	38	1144
7	19	1080
8	23	1132
9	14	1158
10	22	1184
Average	21.2	1134
Worst	38	1208
Best	14	1054

3. Kombinasi ketiga adalah move1TS, move2TS, moveRMTS dapat dilihat pada Tabel 6.35 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.35 Kombinasi 3 LLH 3

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abe	
Percobaan	TINY	SMALL

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abe	
1	23	1078
2	23	1054
3	23	1208
4	23	1174
5	23	1128
6	19	1144
7	23	1080
8	23	1132
9	19	1158
10	23	1184
Average	22.2	1134
Worst	23	1208
Best	19	1054

4. Kombinasi keempat adalah move1TS, move2TS, moveRMTS2 dapat dilihat pada Tabel 6.36 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.36 Kombinasi 3 LLH 4

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abf	
Percobaan	TINY	SMALL
1	23	1086
2	38	1058
3	42	954
4	23	1114
5	23	1088
6	22	1030
7	23	1156

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	abf	
8	18	980
9	22	1156
10	19	1144
Average	25.3	1076.6
Worst	42	1156
Best	18	954

5. Kombinasi kelima adalah move1TS, move1RM, move2RM dapat dilihat pada Tabel 6.37 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.37 Kombinasi 3 LLH 5

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	acd	
Percobaan	TINY	SMALL
1	20	1174
2	42	1164
3	41	1194
4	38	1120
5	42	1132
6	43	1194
7	42	1098
8	41	1150
9	41	1164
10	41	1160
Average	39.1	1155
Worst	43	1194
Best	20	1098

6. Kombinasi keenam adalah move2TS, move2RM, moveRMTS dapat dilihat pada Tabel 6.38 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.38 Kombinasi 3 LLH 6

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bde	
Percobaan	TINY	SMALL
1	17	1076
2	22	1098
3	18	1006
4	22	1026
5	13	1140
6	22	1012
7	17	1000
8	14	1068
9	17	1126
10	21	1048
Average	18.3	1060
Worst	22	1140
Best	13	1000

7. Kombinasi ketujuh adalah move2TS, move2RM, moveRMTS2 dapat dilihat pada Tabel 6.39 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.39 Kombinasi 3 LLH 7

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bdf	
Percobaan	TINY	SMALL

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bdf	
1	15	1182
2	14	940
3	25	1114
4	18	968
5	18	1092
6	41	1086
7	18	1094
8	37	974
9	42	1136
10	22	1198
Average	25	1078.4
Worst	42	1198
Best	14	940

6.4.1.6 Skenario 6

Pada Skenario 6 melakukan percobaan terhadap parameter LLH dengan penggunaan kombinasi 2 LLH dari 6 LLH yang ada. Untuk parameter Iterasi ILS dan Max Iterasi LS tidak berubah nilainya. Iterasi pada Local Search yang digunakan adalah 7, karena pada Tabel 6.9 iterasi 7 menghasilkan penalty yang paling kecil diantara iterasi lainnya baik dari dataset tiny maupun small. Pada Tabel 6.11 merupakan LLH apa saja yang digunakan untuk kombinasi.

1. Kombinasi pertama adalah move1TS, move2TS dapat dilihat pada Tabel 6.40 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.40 Kombinasi 2 LLH 1

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	ab	
Percobaan	TINY	SMALL
1	28	1254
2	28	1222
3	28	1218
4	28	1158
5	28	1198
6	28	1238
7	28	1152
8	28	1212
9	28	1204
10	28	1140
Average	28	1199.6
Worst	28	1254
Best	28	1140

2. Kombinasi kedua adalah move2TS, move1RM dapat dilihat pada Tabel 6.41 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.41 Kombinasi 2 LLH 2

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bc	
Percobaan	TINY	SMALL
1	22	1068
2	18	1182
3	19	1192
4	19	1068
5	18	1048

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bc	
6	22	1126
7	18	1052
8	18	1070
9	18	1178
10	23	1098
Average	19.5	1108.2
Worst	23	1192
Best	18	1048

3. Kombinasi ketiga adalah move2TS, moveRMTS2 dapat dilihat pada Tabel 6.42 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.42 Kombinasi 2 LLH 3

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	bf	
Percobaan	TINY	SMALL
1	18	1068
2	34	1182
3	18	1192
4	22	1068
5	22	1048
6	22	1126
7	18	1052
8	43	1070
9	23	1178
10	23	1098
Average	24.3	1108.2
Worst	43	1192
Best	18	1048

4. Kombinasi keempat adalah move1RM, moveRMTS2 dappat dilihat pada Tabel 6.43 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.43 Kombinasi 2 LLH 4

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	cf	
Percobaan	TINY	SMALL
1	33	1274
2	39	1256
3	38	1242
4	34	1242
5	42	1166
6	43	1208
7	42	1218
8	39	1180
9	42	1246
10	42	1204
Average	39.4	1223.6
Worst	43	1274
Best	33	1166

5. Kombinasi kelima adalah moveRMTS, moveRMTS2 dapat dilihat pada Tabel 6.44 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.44 Kombinasi 2 LLH 5

Iterasi ILS	1500	
Max. Iter. LS	7	
LLH	ef	
Percobaan	TINY	SMALL
1	38	1204
2	42	1292
3	39	1204
4	39	1260
5	38	1204
6	42	1254
7	42	1286
8	43	1224
9	38	1286
10	42	1246
Average	40.3	1246
Worst	43	1292
Best	38	1204

6.4.1.7 Kesimpulan Skenario ILS-HC

Pada Tabel 6.45 menunjukkan model kombinasi terbaik dari setiap skenario kombinasi LLH.

Tabel 6.45 Kesimpulan Skenario

Model	TINY	SMALL
Kombinasi 5	*abcef**	*acdef*
	*19.3^*	*1087*
Kombinasi 4	*bdef*	*bcdf*
	19.1	*1069.8*
Kombinasi 3	*bde*	*abf*
	18.3	*1076.6*
Kombinasi 2	*bc*	*bf*
	19.5	*1105*

**Model*

^Rata-rata Fungsi Tujuan

Dari Tabel 6.45 menunjukkan skenario kombinasi terbaik dari setiap dataset baik tiny maupun small. Pada dataset tiny akan menghasilkan penalty paling kecil jika menggunakan kombinasi LLH move2TS, move2RM, dan moveRMTS. Sedangkan untuk dataset small dapat menghasilkan penalty terkecil jika menggunakan kombinasi LLH move2TS, move1RM, move2RM, dan moveRMTS2.

6.4.2 Iterated Local Search dengan Simmulated Annealing

Pada bagian ini menjelaskan hasil implementasi dari algoritma *Iterated Local Search* dengan menggunakan *local search Simmulated Annealing*.

6.4.2.1 Skenario 1

Pada skenario 1 ini mengubah nilai cooling rate (alpha) untuk mengetahui pengaruh dari variabel cooling rate terhadap performa algoritma. Cooling rate yang digunakan adalah 0.1, 0.5, 0.7, dan 0.9. Hasil dari skenario 1 ini dapat dilihat di Tabel 6.46 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.46 Hasil Skenario 1 ILS-SA

Cool Rate	0.1		0.5		0.7		0.9	
Temp	90							
Iterasi ILS	500							
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	14	924	14	680	13	924	14	808
2	14	872	14	1094	13	750	14	904
3	10	680	14	906	14	788	9	824
4	14	970	13	902	14	838	13	818

Cool Rate	0.1		0.5		0.7		0.9	
Temp	90							
Iterasi ILS	500							
5	10	910	14	968	9	792	14	892
6	10	1070	10	896	14	802	13	924
7	10	890	14	922	14	844	13	798
8	10	844	13	740	13	824	10	864
9	13	788	10	758	14	978	14	1056
10	14	954	30	932	9	840	13	936
Avg	**11.9**	**890.2**	**14.6**	**879.8**	**12.7**	**838**	**12.7**	**882.4**
Worst	14	1070	30	1094	14	978	14	1056
Best	10	680	10	680	9	750	9	798

Tabel 6.46 menunjukkan bahwa *cooling rate* 0.7 menghasilkan solusi lebih optimal ketika menggunakan data small dibandingkan percobaan lainnya dengan nilai rata-rata 838 dan *cooling rate* 0.1 lebih optimal ketika menggunakan data Tiny dengan rata-rata 11.9.

6.4.2.2 Skenario 2

Pada skenario 2 ini dilakukan pengubahan suhu awal dengan nilai iterasi ILS dan cooling rate tetap. Skenario ini memiliki tujuan untuk mengetahui pengaruh suhu awal. Hasil skenario ini dapat dilihat pada Tabel 6.47 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.47 Hasil Skenario 2 ILS-SA

Cool Rate	0.7							
Temp	92		95		97		100	
Iterasi ILS	500							
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	34	848	13	758	14	790	13	852

Cool Rate	0.7							
Temp	92		95		97		100	
Iterasi ILS	500							
2	18	896	13	844	17	876	9	824
3	13	872	13	1052	11	1024	26	752
4	14	662	14	812	9	804	10	866
5	13	772	13	780	13	890	9	858
6	10	904	10	888	13	964	13	768
7	10	950	14	834	14	666	14	862
8	10	768	13	938	9	900	13	896
9	10	838	14	970	9	900	14	614
10	14	868	10	854	10	910	13	908
Avg	**14.6**	**837.8**	**12.7**	**873**	**11.9**	**872.4**	**13.4**	**820**
Worst	34	950	14	1052	17	1024	26	908
Best	10	662	10	758	9	666	9	614

Tabel 6.47 menunjukkan bahwa *temperatur* 100 menghasilkan solusi lebih optimal ketika menggunakan data small dibandingkan percobaan lainnya dengan nilai rata-rata 820 dan *temperature* 97 lebih optimal ketika menggunakan data Tiny dengan rata-rata 11.9.

6.4.2.3 Skenario 3

Pada skenario ini parameter yang diubah adalah iterasi ILS dengan nilai cooling rate dan suhu awal tetap. Skenario ini bertujuan untuk mengetahui pengaruh variabel iterasi terhadap performa algoritma. Hasil dari skenario ini dapat dilihat pada Tabel 6.48 dimana nilai-nilai yang ada didalam kolom tersebut merupakan jumlah penalti pada setiap solusi akhir yang dihasilkan pada tiap percobaannya.

Tabel 6.48 Hasil Skenario 3 ILS-SA

Cool Rate	0.7							
Temp	100							
Iterasi ILS	1000		1500		2000		5000	
Percobaan	TINY	SMALL	TINY	SMALL	TINY	SMALL	TINY	SMALL
1	14	884	10	1030	10	800	9	1046
2	10	914	9	718	9	656	6	936
3	14	784	9	892	6	872	10	822
4	9	940	9	790	10	792	10	854
5	9	890	14	850	10	934	9	802
6	9	844	9	742	10	678	6	770
7	13	918	10	798	6	620	9	742
8	10	810	13	914	10	636	9	730
9	9	990	9	864	9	926	9	882
10	14	882	9	906	9	866	9	902
Avg	**11.1**	**885.6**	**10.1**	**850.4**	**8.9**	**778**	**8.6**	**848.6**
Worst	14	990	14	1030	10	934	10	1046
Best	9	784	9	718	6	620	6	730

Tabel 6.48 menunjukkan bahwa Iterasi sebanyak 2000 menghasilkan solusi lebih optimal ketika menggunakan data small dibandingkan percobaan lainnya dengan nilai rata-rata 778 dan Iterasi sebanyak 5000 lebih optimal ketika menggunakan data Tiny dengan rata-rata 8.6.

6.4.2.4 Kesimpulan Skenario ILS-SA

Pada skenario yang telah dilakukan, hasil dari ILS-SA yang terbaik dapat dilihat pada Tabel 6.49.

Tabel 6.49 Kesimpulan ILS-SA

Cool Rate	0.7	
Temp	100	
Iterasi ILS	2000	5000

Data	SMALL	TINY
Average	778	8.6
Worst	934	10
Best	620	6

Berdasarkan Tabel 6.45 dibandingkan dengan Tabel 6.49, algoritma ILS dengan penggunaan SA sebagai Local Search menghasilkan nilai yang lebih baik dibandingkan dengan Hill Climbing sebagai Local Search.

6.4.3 Pembahasan Hasil Skenario

Dari hasil yang telah didapatkan pada proses sebelumnya, diperoleh beberapa bahasan sebagai berikut.

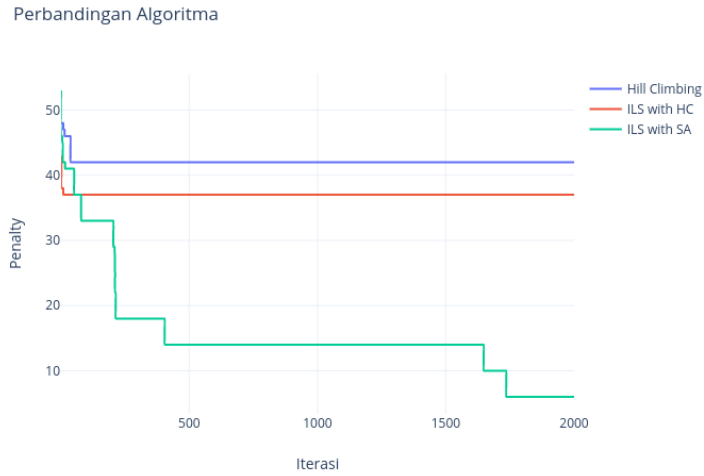
1. Pemilihan jumlah Iterasi LS memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana semakin banyak jumlah Iterasi LS yang dilakukan maka hasil fungsi tujuan akhir yang didapatkan semakin baik. Namun jika Iterasi LS terlalu banyak, maka solusi yang dihasilkan dapat terjebak di local optima.
2. Pemilihan jumlah Max Iterasi LS juga memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana semakin banyak jumlah Max Iterasi pada LS yang dilakukan maka hasil fungsi tujuan akhir yang didapatkan semakin baik. Namun jika Max Iterasi LS terlalu banyak, maka solusi yang dihasilkan dapat terjebak pada local optima.
3. Pemilihan *low level heuristics* memiliki pengaruh terhadap fungsi tujuan akhir, yang mana meskipun metode yang dipakai hanya *move* namun penerapan metode *move* dengan kombinasi atribut yang dipindah, menghasilkan nilai fungsi tujuan akhir yang lebih baik.
4. Pemilihan *cooling rate* memiliki pengaruh terhadap fungsi tujuan akhir, dimana semakin cooling rate dekat

dengan 1, maka akan ada kemungkinan akan menerima solusi yang lebih buruk itu besar, namun tidak menutup kemungkinan untuk mendapatkan solusi yang lebih baik.

5. Pemilihan suhu awal memiliki pengaruh terhadap fungsi tujuan akhir, dimana semakin besar suhu awal, maka akan ada kemungkinan meneirma solusi yang lebih buruk itu besar, sehingga nilai yang didapatkan akan memiliki nilai yang berfluktuasi dan lebih memungkinkan ntuk mencapai solusi yang lebih baik

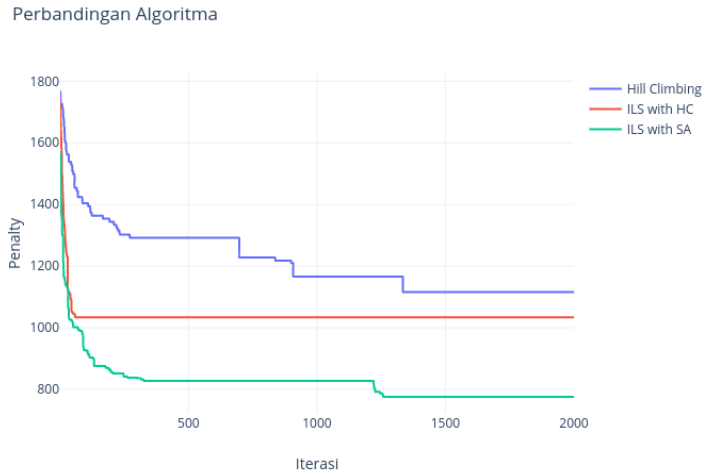
6.5 Perbandingan Hasil Eksperimen dengan Algoritma Lain

Pada bagian ini dilakukan perbandingan performa algoritma *Iterated Local Search* menggunakan *Hill Climbing* sebagai *Local Search* dan juga *Simmulated Annealing* sebagai *Local Search.* Perbandingan performa dilakukan dengan menerapkan algoritma lain sebagai pembanding hasil solusi optimal. Algoritma lain yang diterapkan adalah *Hill Climbing* dan setelah algoritma ini diterapkan kemudian hasilnya dibandingkan dengan ILS. Uji coba dilakukan menggunakan dataset *Tiny* dan juga dataset *Small*. Hasil uji coba dapat dilihat pada Gambar 6.9 dan juga Gambar 6.10.



Gambar 6.9 Perbandingan Algoritma (Dataset Tiny)

Dari Gambar 6.9 menunjukkan bahwa performa algoritma ILS memiliki performa yang lebih baik dengan menghasilkan solusi terbaik, yaitu dengan penalty atau fungsi tujuan sejumlah 37 untuk ILS dengan menggunakan *Local Search Hill Climbing* dan penalty 6 dengan menggunakan *Local Search Simmulated Annealing* setelah melakukan iterasi sebanyak 2000 kali. Sedangkan untuk Hill Climbing saja menghasilkan penalty sebesar 42.



Gambar 6.10 Perbandingan Algoritma (Dataset Small)

Gambar 6.10 menggunakan Dataset *Small* menunjukkan bahwa performa algoritma ILS memiliki performa yang lebih baik dengan menghasilkan solusi terbaik, yaitu dengan penalty atau fungsi tujuan sejumlah 1034 untuk ILS dengan menggunakan *Local Search Hill Climbing* dan penalty 776 dengan menggunakan *Local Search Simmulated Annealing* setelah melakukan iterasi sebanyak 2000 kali. Sedangkan untuk Hill Climbing saja menghasilkan penalty sebesar 1116.

Halaman sengaja dikosongkan

BAB VII
KESIMPULAN DAN SARAN

Bab kesimpulan dan saran membahas mengenai kesimpulan proses penelitian yang telah dilakukan dan saran yang diusulkan baik untuk perusahaan maupun untuk penelitian serupa di masa mendatang.

7.1 Kesimpulan

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

1. Penerapan algoritma *Iterated Local Search* untuk optimasi penjadwalan mata kuliah terbukti mampu menghasilkan nilai fungsi tujuan akhir yang lebih baik. Sebagai bukti penelitian tugas akhir ini telah menerapkan algoritma *Iterated Local Search* untuk optimasi solusi awal dan hasilnya mendukung pernyataan tersebut
2. Pemilihan jumlah Iterasi dari ILS memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana semakin banyak jumlah Iterasi dari ILS yang dilakukan maka hasil fungsi tujuan akhir yang didapatkan semakin baik. Namun jika Iterasi LS terlalu banyak, maka solusi yang dihasilkan dapat terjebak di local optima.
3. Pemilihan jumlah Max Iterasi dari Local Search juga memiliki pengaruh terhadap nilai fungsi tujuan akhir, yang mana semakin banyak jumlah Max Iterasi pada Local Search yang dilakukan maka hasil fungsi tujuan akhir yang didapatkan semakin baik. Namun jika Max Iterasi pada Local Search terlalu banyak, maka solusi yang dihasilkan dapat terjebak pada local optima.

4. Pemilihan *low level heuristics* memiliki pengaruh terhadap fungsi tujuan akhir, yang mana meskipun metode yang dipakai hanya *move* namun penerapan metode *move* dengan kombinasi atribut yang dipindah dan beragam, menghasilkan nilai fungsi tujuan akhir yang lebih baik dibanding hanya menggunakan satu *low level heuristic* saja.
5. Performa algoritma *Iterated Local Search* lebih baik dibandingkan algoritma *hill climbing* untuk melakukan penjadwalan mata kuliah terutama dengan menggunakan *Simmulated Annealing* sebagai *Local Search*. Sebagai bukti penelitian tugas akhir ini telah melakukan perbandingan hasil uji coba penerapan algoritma *Iterated Local Search* dengan algoritma *hill climbing*, yang mana hasilnya mendukung pernyataan tersebut.

7.2 Saran

Dalam pengerjaan tugas akhir, terdapat beberapa saran yang diharapkan dapat bermanfaat bagi perusahaan maupun untuk pengembangan penelitian ke depan, yaitu:

1. Pada penelitian tugas akhir ini hanya menghasilkan solusi untuk 2 *instance* dari keseluruhan 15 *instance* yang terdapat pada studi kasuk ITC 2019. Penelitian selanjutnya dapat melakukan pendekatan yang berbeda guna memberikan solusi untuk keseluruhan *instance*.
2. Penggunaan LLH pada penelitian tugas akhir ini hanya menggunakan method *move* saja, sehingga untuk ragam dari LLH masih terbatas. Untuk penelitian selanjutnya dapat menggunakan LLH yang bermacam- macam seperti salah satu contohnya *swap*. Dengan semakin ragamnya LLH yang digunakan, diharapkan dapat

meningkatkan performa algoritma dan menghasilkan solusi yang lebih optimal.

3. Penelitian tugas akhir ini tidak melakukan pengembangan algoritma yang mana dapat dilakukan oleh penelitian selanjutnya. Dengan melakukan pengembangan algoritma pada penelitian selanjutnya dapat menambah peluang untuk menemukan solusi yang lebih optimal.

DAFTAR PUSTAKA

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

[2] D. A. R. Wati and Y. A. Rochman, "Model Penjadwalan Matakuliah Secara Otomatis Berbasis Algoritma Particle Swarm Optimization (PSO)," *J. Rekayasa Sist. Ind.*, vol. 2, no. 1, pp. 22–31, 2013.

[3] T. Müller, H. Rudová, Z. Müllerová, T. Müller, H. Rudová, and Z. Müllerová, "University course timetabling and International Timetabling Competition 2019," 2019.

[4] J. A. Soria-alcaraz, E. Özcan, J. Swan, G. Kendall, and M. Carpio, "Iterated local search using an add and delete hyper-heuristic for university course timetabling," *Appl. Soft Comput. J.*, pp. 1–13, 2015.

[5] G. H. G. da Fonseca, H. G. Santos, T. Â. M. Toffolo, S. S. Brito, and M. J. F. Souza, "GOAL solver: a hybrid local search based solver for high school timetabling," *Ann. Oper. Res.*, vol. 239, no. 1, pp. 77–97, 2014.

[6] K. Alaykiran and M. Hacibeyoglu, "Using Iterated Local Search to Solve the Course Timetabling Problem at Engineering Faculty of Necmettin Erbakan University," no. 12, pp. 40–43, 2016.

[7] E. K. Burke and G. Kendall, *Search Methodologies*. 2013.

[8] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A Classification of Hyper-heuristics Approaches," *Handb. Metaheuristics, Second Ed.*, vol. 146, pp. 449–468, 2009.

[9] A. Muklason, "Solver Penjadwal Ujian Otomatis Dengan Algoritma Maximal Clique dan Hyper-heuristics," *Semin. Nas. Teknol. Informasi, Komun. dan Ind. 9*, pp. 18–19, 2017.

[10] N. Pillay, "Hyper-Heuristics for Educational Timetabling," *Proc. ninth Int. Conf. Pract. theory*

Autom. timetabling (PATAT 2012), no. August, pp. 29–31, 2012.

[11] T. Stützle and R. Ruiz, "Iterated Local Search," in *Handbook of Heuristics*, R. Martí, P. M. Pardalos, and M. G. C. Resende, Eds. Cham: Springer International Publishing, 2018, pp. 579–605.

[12] J. Brownlee, *Clever Algorithms*. 2011.

[13] O. C. Martin, H. R. Lourenço, and T. Stützle, *Iterated Local Search*, no. May 2014. 2003.

[14] T. Stutzle and R. Ruiz, "Iterated Local Search : A Concise Review," no. April, 2018.

BIODATA PENULIS

Penulis bernama Umar Rizki Kusumo Widayu. Penulis akrab di sapa sebagai Umar atau Rizki. Penulis lahir di Surabaya, tepatnya pada tanggal 14 Februari 1997. Penulis menempuh pendidikannya sejak sekolah dasar hingga perguruan tinggi yaitu SD Muhammadiyah 4 Surabaya, SMP Negeri 12 Surabaya, SMA Negeri 16 Surabaya, dan Institut Teknologi Sepuluh Nopember pada departemen Sistem Informasi. Penulis mulai berkuliah pada tahun 2015. Selama masa perkuliahan, penulis aktif dalam bidang organisasi kemahasiswaan antara lain staff pada departemen Sosial Masyarakat di Himpunan Mahasiswa Sistem Informasi pada periode 2016/2017. Selanjutnya pada periode 2017/2018 penulis aktif sebagai ketua departemen Sosial Masyarakat di Himpunan Mahasiswa Sistem Informasi.

Dalam rangka mendapatkan gelar Sarjana Komputer, maka penulis masuk ke dalam Laboratorium Rekayasa Data dan Intelegensia Bisnis dengan topik tugas akhir Penyelesaian Permasalahan Otomasi Dan Optimasi Penjadwalan Mata Kuliah Menggunakan Algoritma Iterated Local Search Hyper-Heuristic Dengan Domain Permasalahan Dari International Timetabling Competition 2019. Penulis dapat dihubungi via email : umarrizki14@gmail.com untuk kepentingan penelitian.

LAMPIRAN A: Jadwal Hasil Optimasi

Id Kelas	Week	Day	Start	Length	Ruangan
1	1-9	Senin, Selasa, Rabu, Kamis	132	22	45
2	1-9	Senin, Selasa, Rabu, Kamis	114	15	45
3	1-9	Senin, Selasa, Rabu, Kamis	144	15	8
4	1-9	Selasa, Kamis	96	34	19
5	1-9	Senin, Selasa, Rabu, Kamis	108	15	24
6	1-9	Senin, Selasa, Rabu, Kamis	192	22	28
7	1-9	Senin, Selasa, Rabu, Kamis	168	22	28
8	1-9	Senin, Selasa, Rabu, Kamis	126	22	28
9	1-9	Senin, Selasa, Rabu, Kamis	138	22	27
10	1-9	Senin, Selasa, Rabu, Kamis	120	22	22
11	1-9	Senin, Selasa, Rabu, Kamis	108	22	203

A-2

Id Kelas	Week	Day	Start	Length	Ruangan
12	1-9	Senin, Selasa, Rabu, Kamis	96	15	27
13	1-9	Senin, Selasa, Rabu, Kamis	96	22	38
14	1-9	Senin, Selasa, Rabu, Kamis	144	15	26
15	1-9	Senin, Selasa, Rabu, Kamis	120	22	38
16	1-9	Selasa, Kamis	96	58	62
17	1-9	Senin, Selasa, Rabu, Kamis	144	22	38
18	1-9	Senin, Selasa, Rabu, Kamis	96	22	8
19	1-9	Senin, Selasa, Rabu, Kamis	96	22	12
20	1-9	Senin, Selasa, Rabu, Kamis	120	22	8