TUGAS AKHIR – IS184853

IMPLEMENTASI ALGORITMA *TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS* UNTUK OTOMASI DAN OPTIMASI PEMBUATAN JADWAL MATA KULIAH DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

IMPLEMENTATION OF TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS ALGORITHM FOR AUTOMATION AND OPTIMIZATION COURSE TIMETABLING WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

NARENDRA PUSPA ADI NEGARA
NRP 05211540000087

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

IMPLEMENTASI ALGORITMA *TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS* UNTUK OTOMASI DAN OPTIMASI PEMBUATAN JADWAL MATA KULIAH DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

NARENDRA PUSPA ADI NEGARA
NRP 05211540000087

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

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

Halaman ini sengaja dikosongkan

FINAL PROJECT – IS184853

IMPLEMENTATION OF TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS ALGORITHM FOR AUTOMATION AND OPTIMIZATION COURSE TIMETABLING WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

NARENDRA PUSPA ADI NEGARA
NRP 05211540000087

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

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

Halaman ini sengaja dikosongkan

LEMBAR PENGESAHAN

IMPLEMENTASI ALGORITMA *TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS* UNTUK OTOMASI DAN OPTIMASI PEMBUATAN JADWAL MATA KULIAH 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:

NARENDRA PUSPA ADI NEGARA
NRP. 05211540000087

Surabaya, 16 Juli 2019

KEPALA
DEPARTEMEN SISTEM INFORMASI

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

Halaman ini sengaja dikosongkan

LEMBAR PERSETUJUAN

IMPLEMENTASI ALGORITMA *TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS* UNTUK OTOMASI DAN OPTIMASI PEMBUATAN JADWAL MATA KULIAH 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 :

NARENDRA PUSPA ADI NEGARA
NRP. 05211540000087

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)**

Halaman ini sengaja dikosongkan

IMPLEMENTASI ALGORITMA *TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS* UNTUK OTOMASI DAN OPTIMASI PEMBUATAN JADWAL MATA KULIAH DENGAN DOMAIN PERMASALAHAN DARI *INTERNATIONAL TIMETABLING COMPETITION 2019*

Nama Mahasiswa	**:**	**Narenda Puspa Adi Negara**
NRP	**:**	**05211540000087**
Departemen	**:**	**Sistem Informasi**
Dosen Pembimbing	**:**	**Ahmad Muklason, S.Kom., M.Sc., Ph.D**

ABSTRAK

International Timetabling Competition merupakan kompetisi penjadwalan internasional yang memiliki tujuan untuk memotivasi penelitian lebih lanjut mengenai permasalahan penjadwalan khususnya pada penjadwalan di bidang pendidikan. Dalam dunia pendidikan, permasalahan penjadwalan telah menjadi suatu topik yang sering sekali ditemui. Salah satu permasalahan penjadwalan yang terdapat pada perguruan tinggi adalah penjadwalan mata kuliah. Penjadwalan mata kuliah setidaknya dilakukan rutin setiap awal semester, dalam melakukan penjadwalan harus memperhatikan alokasi sumber daya yang terdapat pada universitas. Penjadwalan merupakan proses yang panjang, hal itu disebabkan karena dalam mengalokasikan sumber daya dalam suatu permasalahan penjadwalan harus memperhatikan berbagai aspek atau batasan yang telah ditetapkan agar mendapatkan hasil yang optimal. Permasalahan ini tergolong dalam permasalahan Non-Polynomial hard, dimana belum terdapat algoritma eksak untuk menyelesaikannya dalam waktu polynomial time. Dalam penyusunan tugas akhir ini dilakukan penjadwalan mata kuliah dengan menggunakan algoritma tabu search - simulated annealing hyper-heuristics. Dataset yang

digunakan adalah dataset yang diperoleh dari International Timetabling Competition 2019. Hasil dari tugas akhir ini menunjukkan bahwa algoritma Tabu Search – Simulated Annealing dapat digunakan dalam melakukan optimasi penjadwalan mata kuliah, dibuktikan bahwa hasil benchmark dengan solusi awal, algoritma Hill Climbing maupun algoritma Simulated Annealing menunjukkan rata-rata penalti yang didapatkan oleh Tabu – Simulated Annealing lebih rendah dari pada algoritma benchmark yang dilakukan, dengan nilai terbaik yaitu 666 dibandingkan dengan 1068 hasil algoritma Hill Climbing dan 760 hasil algoritma Simulated Annealing.

Kata kunci: international timetabling competition, hyper-heuristic, penjadwalan mata kuliah, algoritma Tabu Search, algoritma Simulated Annealing

IMPLEMENTATION OF TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS ALGORITHM FOR AUTOMATION AND OPTIMIZATION COURSE TIMETABLING WITH PROBLEM DOMAIN FROM INTERNATIONAL TIMETABLING COMPETITION 2019

Name	**:**	**Narendra Puspa Adi Negara**
NRP	**:**	**05211540000087**
Department	**:**	**Information Systems**
Supervisior	**:**	**Ahmad Muklason, S.Kom., M.Sc., Ph.D**

ABSTRACT

The International Timetabling Competition is an international scheduling competition that aims to motivate further research on scheduling issues especially in scheduling in the field of education. In the world of education, scheduling problems have become a topic that is often encountered. One of the scheduling problems found in higher education is scheduling courses. Course scheduling is conducted routinely at the beginning of each semester, in scheduling must pay attention to the allocation of resources contained in the university. Scheduling is a long process, it is because in allocating resources in a scheduling problem must pay attention to various aspects or limits that have been set in order to get optimal results. This problem is classified as a Non-Polynomial hard problem, where there is no exact algorithm to solve it in a polynomial time. In the preparation of this final project subject scheduling is done using a tabu search algorithm - simulated annealing hyper-heuristics. The dataset used is a dataset obtained from the 2019 International Timetabling Competition. The results of this final project shows that the Tabu Search – Simulated Annealing

algorithm can be used to optimize courses scheduling. It is proven that the benchmark results with Initial Solution, Hill Climbing algorithm, and Simulated Annealing algorithm show the average penalty obtained by Tabu – Simulated Annealing more lower than the benchmark algorithm, with the best value from Tabu Search – Simulated Annealing which is 660 compared to 1068 the results of Hill Climbing algorithm and 760 from the result of Simulated Annealing algorithm.

Keywords: international timetabling competition, hyper-heuristic, course scheduling, Half Tabu algorithm, Simulated Annealing algorithm

KATA PENGANTAR

Alhamdulillahi robbil 'alamin, segala puji bagi Allah SWT atas limpahan nikmat, rahmat, petunjuk, dan karunia-Nya, sehingga penulis dapat menyelesaikan laporan penelitian tugas akhir dengan judul, "**IMPLEMENTASI ALGORITMA TABU SEARCH – SIMULATED ANNEALING HYPER HEURISTICS UNTUK OTOMASI DAN OPTIMASI PEMBUATAN JADWAL MATA KULIAH DENGAN DOMAIN PERMASALAHAN DARI INTERNATIONAL TIMETABLING COMPETITION 2019**" yang menjadi salah satu syarat kelulusan pada Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember Surabaya.

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

1. Kedua orang tua penulis, Bapak Miranu Triantoro dan Ibu Farida Nursusanti yang tidak pernah lelah dan tak pernah henti memberikan dukungan, motivasi, serta doa yang tulus dan ikhlas demi yang terbaik untuk penulis.
2. Bapak Ahmad Muklason, S.Kom., M.Sc., Ph.D. selaku dosen pembimbing yang telah meluangkan banyak waktu dan selalu sabar membimbing serta memberikan arahan kepada penulis dalam pengerjaan tugas akhir ini.
3. Bapak Edwin Riksakomara. S.Kom., MT., dan Ibu Retno Aulia Vinarti, S.Kom., M.Kom., Ph.D., selaku dosen penguji yang telah memberikan kritik dan saran yang membangun dalam proses pengerjaan tugas akhir ini.
4. Saudari Prasasti Karunia Farista Ananto, yang tidak pernah berhenti memberikan dukungan, saran, serta bantuan dari

awal sampai akhir terbentuknya laporan penelitian tugas akhir.

5. Para pejuang ITC. Umar, Cut, Risma, yang telah berjuang dari nol dalam menyelesaikan tugas akhir ini, *kon sangar rek!*
6. Saudara seperjuangan Laboratorium RDIB yang selalu menemani dan mendukung serta menjadi tempat diskusi penulis dalam pengerjaan tugas akhir hingga tak kenal waktu saat berada Lab RDIB.
7. Teman-teman Lannister yang selalu memberikan dukungan dan kenangan sendiri bagi penulis. Jangan lupakan *beton terbang bersinar* rek!

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

Surabaya, 09 Juli 2019

Penulis

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR GAMBAR

DAFTAR TABEL

Halaman ini sengaja dikosongkan

DAFTAR KODE

Halaman ini sengaja dikosongkan

BAB I
PENDAHULUAN

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

1.1. Latar Belakang

Penjadwalan telah menjadi suatu masalah yang sering dihadapi oleh sebagian besar organisasi atau instansi, salah satu contoh adalah instansi yang bergerak dalam bidang pendidikan yaitu Universitas. Penjadwalan di universitas sangat diperlukan terlebih lagi bila menyangkut dengan masalah mata kuliah. Permasalahan dalam penjadwalan mata kuliah adalah bagaimana alokasi sumber daya yang optimal sehingga semua batasan dan tujuan yang diharapkan tercapai [1].

Penjadwalan yang optimal adalah penjadwalan yang dapat memuaskan batasan. Batasan dibagi menjadi dua yaitu *hard constraint* dan *soft constraint*, fokus utama dari optimasi penjadwalan ini adalah menghindari *hard constraint* dan meminimalkan pelanggaran atau biasa disebut penalty dari *soft constraint*[2]. Penjadwalan mata kuliah merupakan proses yang panjang, hal ini disebabkan karena banyak aspek-aspek yang harus diperhatikan seperti jumlah mata kuliah yang dibuka, dosen pengajar, mahasiswa, dan waktu [3]. Oleh sebab itu penjadwalan mata kuliah digolongkan menjadi permasalahan NP-*hard* (Non-Polynomial *Hard*) [2], [3]. NP-*hard* problem merupakan permasalahan yang tidak diketahui waktu polynomialnya sehingga waktu untuk menemukan solusi berkembang secara eksponensial berdasarkan ukuran masalah.

Melihat permasalahan di atas, solusi yang optimal sangat diperlukan untuk mengatasi permasalahan tersebut. Hyper-heuristic merupakan framework yang dapat memberikan hasil yang lebih optimal dibandingkan dengan algoritma meta-heuristic. Hyper-heuristic secara umum dapat diartikan sebagai gabungan atau kumpulan dari meta-heuristic yang berbeda dengan tujuan menggabungkan kelebihan dari masing-masing meta-heuristik. Hyper-heuristic memiliki kelebihan yaitu tidak memerlukan parameter tuning tetapi dengan mekanisme framework Hyper-heuristic proses tersebut bisa di otomasi [4]. Framework Hyper-heuristic yang akan digunakan untuk penelitian ini adalah Tabu Search [5] dan Simulated Annealing [6].

Tabu Search merupakan metaheuristic yang berfungsi untuk menemukan global optimum dengan cara melakukan proses pencarian dari satu solusi ke solusi berikutnya (neighbourhood solution) yang kemudian akan dipilih solusi terbaik dari solusi yang sekarang (current solution). Dalam algoritma Tabu Search untuk menghindari solusi yang memutar, maka solusi yang sudah pernah didapatkan akan masuk ke tabu list [5], [7].

Simulated Annealing merupakan algoritma metaheuristic yang menganalogikan perilaku benda logam pada proses pendinginan. Ketika logam diberikan panas dengan suhu yang tinggi maka molekul serta atom logam akan terlepas dan bergerak bebas, namun ketika didinginkan kembali molekul serta atom tersebut akan mengikat kembali dengan energi atau susunan yang lebih minimal (optimum). Kelebihan algoritma Simulated Annealing adalah mampu untuk menghindari local optimal solution meskipun pencarian yang dilakukan acak, algoritma ini tidak hanya menerima solusi yang lebih baik tetapi juga menerima solusi yang kurang bagus [8].

Dataset yang digunakan merupakan dataset dari International Timetabling Competition 2019 atau yang biasa disingkat

dengan ITC 2019. Dataset ITC 2019 merupakan dataset terbaru dari perlombaan internasional tersebut yang mempunyai banyak instance berbeda sehingga uji coba algoritma ini akan lebih akurat dan lebih up to date dengan perkembangan permasalahan penjadwalan.

Diharapkan dari penelitian ini dengan menggabungkan dua algoritma antara Tabu Search dengan Simulated Annealing akan memperoleh solusi yang lebih optimal untuk membantu menyelesaikan permasalahan penjadwalan khususnya terkait dengan penjadwalan mata kuliah.

1.2. Perumusan Masalah

Berdasarkan latar belakang yang telah diuraikan pada bagian sebelumnya, rumusan masalah yang diangkat dalam penelitian ini adalah,

1. Bagaimana model matematis pada permasalahan *International Timetabling Competition 2019*?
2. Bagaimana implementasi penerapan algoritma *Tabu Search – Simulated Annealing* dalam kerangka kerja *hyper-heuristics*?
3. Bagaimana performa dari algoritma *Tabu Search – Simulated Annealing Hyper-heuristic*, khususnya jika dibandingkan dengan algoritma benchmark?

1.3. Batasan Pengerjaan Tugas Akhir

Batasan permasalahan penelitian ini adalah,

1. Implementasi algoritma *Tabu Search – Simulated Annealing* menggunakan bahasa pemrograman Java.
2. Dataset yang digunakan dalam tugas akhir ini diperoleh dari *International Timetabling Competition* 2019.
3. Hasil akhir dari tugas akhir ini adalah sistem penjadwalan mata kuliah otomatis berbasis Java.

4. Penjadwalan yang dilakukan adalah penjadwalan mata kuliah tanpa menjadwalkan mahasiswa.

1.4. Tujuan Tugas Akhir

Berdasarkan rumusan dan batasan masalah yang telah diuraikan pada bagian sebelumnya, maka tujuan yang ingin dicapai dari penelitian tugas akhir ini adalah menerapkan dan melakukan eksperimen terhadap algoritma Tabu-Simulated Annealing dalam *framework* Hyper-Heuristics untuk proses optimasi penjadwalan mata kuliah secara otomatis dan optimal dengan kelas (*instance*) dan data yang lebih besar.

1.5. Manfaat Tugas Akhir

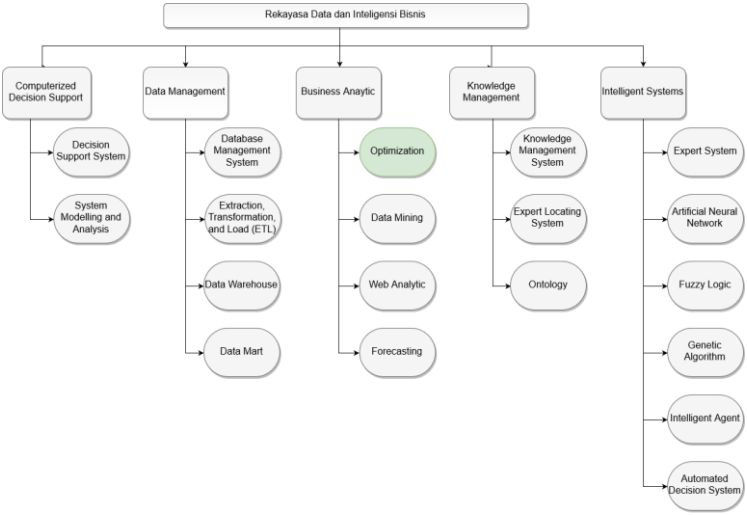
Beberapa manfaat yang diharapkan dapat tercapai pada penelitian ini adalah,

1. Membantu memecahkan masalah penjadwalan khususnya untuk penjadwalan mata kuliah
2. Memberikan referensi tambahan dalam lingkup optimasi khususnya pada bidang penjadwalan menggunakan *Tabu Search – Simulated Annealing Hyper-heuristic.*
3. Memberikan hasil yang lebih kompetitif dalam hal efisiensi waktu dan keefektifan dari penjadwalan mata kuliah.

1.6. Relevansi

Laboratorium Rekayasa Data dan Inteligensi Bisnis (RDIB) Jurusan Sistem Informasi ITS Surabaya memiliki lima topik utama, yaitu Computerized Decision Support, Data Management, Business Analytic, Knowledge Management, dan Intelligent System. Tugas akhir yang dikerjakan penulis adalah mengenai sub-topik dari Business Analytic yaitu optimasi yang termasuk salah satu

topik utama dari riset laboratorium RDIB. Mata kuliah yang bersangkutan dengan topik ini adalah Optimasi Kombinatorik dan Heuristik (OKH).



Gambar 1.1 Bagan Relevansi Laboratorium

Halaman ini sengaja dikosongkan

BAB II
TINJAUAN PUSTAKA

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

2.1. Studi Sebelumnya

Terdapat beberapa penelitian yang memiliki topik yang memiliki relevansi dengan penelitian yang sedang dikerjakan penulis untuk dijadikan sumber referensi dalam pengerjaan tugas akhir. Penelitian tserdahulu tersebut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Penelitian Terdahulu

No	Penelitian Terdahulu	
1.	Judul Penelitian	Simulated Annealing and Tabu Search in the Long Run: A Comparison on QAP Tasks [9]
	Tahun	1994
	Nama Peneliti	R. Battiti dan G. Tecchiolli
	Gambaran umum penelitian	Penilitian ini membahas mengenai kelemahan dan kelebihan dari algoritma Simulated Annealing dan Tabu Search dengan cara membandingkan hasil dari penerapan algoritma tersebut. Beberapa hal yang dibandingkan adalah fungsi evaluasi dan CPU Times. Hasil dari perbandingan tersebut membuktikan bahwa Tabu Search lebih unggul dari pada

No	Penelitian Terdahulu	
		Simulated Annealing jika menggunakan iterasi yang besar. Hasil yang sama juga diperoleh saat membandingkan hasil jika CPU Times dari masing-masing algoritma disamakan, Tabu Search memerlukan CPU Times yang lebih sedikit dari pada Simulated Annealing.
	Keterkaitan dengan Tugas Akhir	Penelitian tersebut menggunakan algoritma yang akan digunakan oleh pengerjaan tugas akhir ini yaitu Tabu Search dan Simulated Annealing. Dengan perbandingan yang dilakukan oleh penelitian tersebut, maka kelemahan dari masing-masing algoritma akan tertutupi dengan kelebihan dari algoritma lain.
2.	Judul Penelitian	*Heuristic Approaches for University Timetabling Problems* [10]
	Tahun	2006
	Nama Peneliti	Salwani Abdullah
	Permasalahan	*Course* Timetabling Problem
	Gambaran umum penelitian	Pada penelitian ini, peniliti menggunakan beberapa algoritma salah satunya adalah algoritma Tabu Search, dalam penelitian tersebut algoritma Tabu Search bisa mengungguli algoritma lain dan menghasilkan solusi yang lebih optimum. Dari hasil penggunaan metode Tabu Search – Variable Neighbourhood Algorithm, algoritma tersebut dapat menghasilkan rata-rata penalty yang kecil, yaitu 2 untuk small instance, 173 untuk medium instance, dan 852 untuk large/big instance.
	Keterkaitan dengan Tugas Akhir	Algoritma yang digunakan di penelitian ini sama dengan algoritma yang digunakan

No	Penelitian Terdahulu	
		dalam pengerjaan tugas akhir yaitu Tabu Search Algorithm.
3.	Judul Penelitian	Implementasi Algoritma Tabu Search pada Aplikasi Penjadwalan Mata Pelajaran (Studi Kasus SMA Negeri 4 Kendari) [7]
	Tahun	2016
	Nama Peneliti	M. P. Khairunnisa, B. Pramono, dan R. A. Saputra
	Gambaran umum penelitian	Penelitian ini menggunakan algoritma Tabu Search untuk mencari solusi yang optimal. Pada tahap awal peneliti melakukan Analisa kebutuhan data masukan seperti syarat yang ditentukan oleh pihak sekolah. Langkah langkah implementasi algoritma Tabu Search yang pertama adalah menentukan atau memilih solusi awal yang ditentukan secara acak, kemudian menentukan iterasi selanjutnya dengan mencari solusi alternatif yang tidak melanggar kriteria, langkah selanjutnya adalah memilih solusi terbaik dan dibandingkan dengan solusi awal untuk menentukan solusi yang optimum. Hasil dari penelitian ini menunjukkan bahwa algoritma Tabu Search dapat menghasilkan penjadwalan yang optimal dengan menekan adanya bentrok jadwal.
	Keterkaitan dengan Tugas Akhir	Algoritma yang digunakan penulis diatas menggunakan Tabu Search yang akan digunakan untuk mengerjakan penelitian tugas akhir ini.

2.2. Dasar Teori

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

2.2.1. Penjadwalan

Wren (1996) dalam [11] mengatakan bahwa penjadwalan merupakan alokasi yang sesuai pada kendala, sumber daya yang diberikan untuk objek yang ditempatkan dalam ruang waktu, sedemikian rupa untuk memenuhi sedekat mungkin dengan tujuan yang diinginkan.

Permasalahan penjadwalan digolongkan menjadi tiga kelas utama oleh Schaerf dalam [12] yaitu school timetabling, *course* timetabling, dan examination timetabling.

Permasalahan penjadwalan merupakan suatu NP-*hard* Problem [3], NP sendiri merupakan singkatan dari Non-Polynomial yang menandakan bahwa permasalahan ini belum mempunyai algoritma eksak sehingga belum mampu menemukan solusi yang optimal dalam waktu polinomial.

Aturan dan batasan adalah suatu perihal penting yang harus diperhatikan dalam penjadwalan. Terdapat dua jenis batasan yaitu *Hard Constraint* dan *Soft constraint* [2].

Hard Constraint merupakan batasan yang harus ditegakkan atau dipenuhi, sedangkan *Soft constraint* merupakan batasan yang diinginkan tetapi tidak mutlak penting dan boleh dilanggar serta sebisa mungkin terpenuhi, dalam dunia nyata biasanya tidak mungkin untuk memenuhi semua *Soft constraint* tersebut [13].

2.2.2. *International Timetabling Competition 2019* Dataset

International Timetabling Competition 2019 dataset adalah dataset yang dikeluarkan oleh The European Metaheuristics Network pada tahun 2019 untuk keperluan perlombaan internasional penjadwalan (timetabling). ITC sendiri telah diadakan sejumlah empat kali yaitu pada tahun 2002, 2007, 2011, dan 2019 [14]. Dataset tersebut memiliki 15 instances yang terdiri dari 5 grup data test dan 10 data early, disetiap instance memiliki jumlah *course*, jumlah class, jumlah *room*, jumlah *student*, dan distributions. Statistik dari dataset tersebut dapat dilihat pada Tabel 2.2.

Tabel 2.2 Statistika ITC 2019 Dataset

Instances		*Courses*	Classes	*Rooms*	*Students*	*Distributions*
Test Instances						
1	Lums-sum17	19	20	62	0	2
2	Bet-sum18	48	127	46	0	144
3	Pu-cs-fal07	44	174	13	2002	102
4	Pu-llr-spr07	603	803	56	27881	329
5	Pu-c8-spr07	1036	2418	211	29514	2004
Early Instances						
6	agh-fis-spr17	340	1239	80	1641	1220
7	agh-ggis-spr17	272	1852	44	2116	2690
8	bet-fal17	353	983	62	3018	1251

Instances		*Courses*	Classes	*Rooms*	*Students*	*Distributions*
9	iku-fal17	1206	2641	214	0	2902
10	mary-spr17	544	882	90	3666	3947
11	muni-fi-spr16	228	575	35	1543	740
12	muni-fsps-spr17	226	561	44	865	400
13	muni-pdf-spr16c	1089	2526	70	2938	2026
14	pu-llr-spr17	687	1001	75	27018	634
15	tg-fal17	36	711	15	0	501

Constraint pada dataset ITC dibagi menjadi dua yaitu *hard constraint* dan *soft constraint*. Dalam dataset ITC 2019, batasan *hard* dan soft ditandai dengan parameter required untuk *hard constraint* dan parameter penalty untuk *soft constraint*. Spesifikasi XML pendistribusian batasan dapat dilihat pada Kode 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>
```

Kode 2.1 Spesifikasi XML Distribusi Batasan

Batasan-batasan yang terdapat pada dataset ITC 2019 tercantum pada Tabel 2.3

Tabel 2.3 Daftar Batasan dan Pengertian

Batasan	Pengertian
SameStart	Kelas harus diadakan dengan *timeslot* yang sama, yang berarti bahwa. $C_{i.Start} = C_{j.Start}$ Dimana, $C_{i.Start}$ merupakan *timeslot* yang ditetapkan untuk kelas C_i.
SameTime	Kelas yang diberikan batasan ini harus diajarkan atau dimulai pada waktu yang sama, untuk kelas dengan panjang *timeslot* yang sama maka batasan ini sama dengan batasan *SameStart*. Perbedaannya adalah jika kelas dengan panjang *timeslot* yang berbeda, maka kelas yang lebih pendek dapat dimulai setelah kelas yang lebih panjang tetapi harus berakhir sebelum kelas yang lebih panjang. $(C_{i.Start} \leq C_{j.Start} \wedge C_{j.End} \leq C_{i.End}) \vee (C_{j.Start} \leq C_{i.Start} \wedge C_{i.End} \leq C_{j.End})$
DifferentTime	Kelas dengan batasan ini tidak boleh dimulai pada *timeslot* yang sama, dengan kata lain

Batasan	Pengertian
	tidak ada dua kelas dari batasan ini yang dapat tumpang tindih pada *timeslot* yang sama. $(C_{i.End} \leq C_{j.Start}) \vee (C_{j.End} \leq C_{i.Start})$
SameDays	Kelas dengan batasan harus diajarkan pada hari yang sama $(C_{i.Days}\ or\ C_{j.Days}) = C_{i.Days} \vee (C_{i.Days}\ or\ C_{j.Days}) = C_{j.Days}$
DifferentDays	Kelas harus diadakan berbeda hari dalam seminggu $(C_{i.Days}\ and\ C_{j.Days}) = 0$
SameWeeks	Kelas dengan batasan ini harus diajarkan pada minggu yang sama, $(C_{i.Weeks}\ or\ C_{j.Weeks}) = C_{i.Weeks} \vee (C_{i.Weeks}\ or\ C_{j.Weeks}) = C_{j.Weeks}$
DifferentWeeks	Kelas harus diadakan berbeda minggu dalam satu semester $(C_{i.Weeks}\ and\ C_{j.Weeks}) = 0$
Overlap	Kelas pada batasan ini memiliki waktu yang tumpang tindih dalam sehari, hari dalam seminggu, maupun minggu. $(C_{j.Start} < C_{i.End}) \wedge (C_{i.Start} < C_{j.End}) \wedge ((C_{i.Days}\ and\ C_{j.Days}) \neq 0) \wedge ((C_{i.Weeks}\ and\ C_{j.Weeks}) \neq 0)$
NotOverlap	Kelas yang diberikan batasan ini tidak boleh memiliki waktu yang sama atau tumpang tindih dengan kelas lain yang diberikan batasan. $(C_{i.End} < C_{j.Start}) \vee (C_{j.End} < C_{i.Start}) \vee ((C_{i.Days}\ and\ C_{j.Days}) = 0) \wedge ((C_{i.Weeks}\ and\ C_{j.Weeks}) = 0)$

Batasan	Pengertian
SameRoom	Kelas dengan batasan ini harus diletakkan pada ruang yang sama dengan kelas lain. $(C_{i.Room} = C_{j.Room})$
DifferentRoom	Kelas dengan batasan ini harus diletakkan pada ruang yang berbeda. $(C_{i.Room} \neq C_{j.Room})$
SameAttendees	Kelas dengan batasan ini tidak diperbolehkan tumpang tindih dengan kelas lain pada hari tersebut, jika tetap tumpang tindih maka kedua kelas tersebut harus diletakkan berjauhan agar peserta bias melakukan perjalanan antara dua kelas. $(C_{i.End} + C_{i.RoomTravel}\lvert C_{j.Room}\rvert \leq C_{j.Start}) \vee$ $(C_{j.End} + C_{j.RoomTravel}\lvert C_{i.Room}\rvert \leq C_{i.Start}) \vee$ $\big((C_{i.Days}\ and\ C_{j.Days}) = 0\big) \vee$ $\big((C_{i.Weeks}\ and\ C_{j.Weeks}) = 0\big)$
Precedence	Batasan ini merupakan penentuan prioritas antar kelas, kelas dengan batasan harus sesuai dengan urutan yang terdapat dalam batasan kendala. Untuk kelas yang memiliki pertemuan lebih dari satu kali dalam satu minggu atau dalam minggu yang berbeda maka batasan hanya berlaku pada pertemuan pertama, yaitu. 1. Kelas pertama yang dimulai pada minggu sebelumnya, atau 2. Dimulai pada minggu yang sama dan kelas pertama dimulai pada hari sebelumnya atau 3. Mulai pada minggu dan hari yang sama dalam seminggu dan kelas pertama lebih awal pada hari itu. $\big(first(C_{i.Weeks}) < first(C_{j.Weeks})\big) \vee$

Batasan	Pengertian
	$\Big[\big(first(C_{i.Weeks}) = first(C_{j.Weeks})\big) \wedge \Big[\big(first(C_{i.Days}) < first(C_{j.Days})\big) \vee \big(first(C_{i.Days}) = first(C_{j.Days})\big) \wedge (C_{i.End} \leq C_{j.Start})\Big]\Big]$
WorkDay(T)	Batasan ini tidak memperbolehkan terdapat lebih *Timeslot (T)* diantara awal kelas dan akhir kelas terakhir pada hari tertentu $\big((C_{i.Days} \; and \; C_{j.Days}) = 0\big)$ $\big((C_{i.Weeks} \; and \; C_{j.Weeks}) = 0\big)$ $\big(max(C_{i.End}, C_{j.Ends}) - min(C_{i.Start}, C_{j.Start}) \leq T\big)$
MinGap(G)	Dua kelas yang diajarkan pada hari yang sama, harus memiliki setidaknya gap *(G) timeslot*, yang berarti harus ada jarak *G* antara akhir kelas sebelumnya dengan awal kelas berikutnya. $\big((C_{i.Days} \; and \; C_{j.Days}) = 0\big) \vee$ $\big((C_{i.Weeks} \; and \; C_{j.Weeks}) = 0\big) \vee$ $\big((C_{i.End} + G \leq C_{j.Start})\big) \vee$ $\big((C_{j.End} + G \leq C_{i.Start})\big)$
MaxDays(MD)	Kelas yang diberikan batasan ini tidak boleh tersebar lebih dari *MD* hari dalam seminggu, yang berarti jumlah total kelas dalam batasan ini dalam seminggu tidak boleh melebihi *MD*. $countNonZeroBits$ $(C_{1.Days} \; or \; C_{2.Days} \; or \; \cdots \; C_{n.Days}) \leq MD$ Dimana countNonZeroBits(*x*) mengembalikan jumlah bit yang tidak nol.

Batasan	Pengertian
	Ketika batasan tersebut termasuk *Soft constraint*, penalti yang didapatkan dikalikan dengan jumlah hari yang melebihi konstanta dari *MD*.
MaxDayLoad(T)	Kelas yang diberikan batasan ini harus disebarkan selama beberapa hari dalam seminggu tetapi dengan syarat bahwa tidak melebihi slot *timeslot {T)* yang diberikan setiap hari. Berarti dalam setiap minggu $w \in \{0,1,\dots,nrWeeks-1\}$ dalam satu semester dan setiap hari dalam seminggu $d \in \{0,1,\dots,nrDays-1\}$, total jumlah slot yang dijadwalkan untuk kelas *C* tidak melebihi *T*. $$DayLoad(d,w) \leq T$$ $$DayLoad(d,w) = \sum_i \{C_{i.Length} | (C_{i.Days} \; and \; 2^d) \neq 0 \wedge (C_{i.Weeks} \; and \; 2^w) \neq 0\}$$ Dimana 2^d adalah *bit string* dengan hanya *non-zero bit* yang terdapat pada *d. S*edangkan 2^w adalah *bit string* dengan hanya *non-zero bit* yang terdapat pada *w.* Penalti yang didapatkan jika batasan ini termasuk *Soft constraint* adalah penalti tersebut dikalikan dengan jumlah konstanta *T* pada setiap hari dalam semester dan dibagi dengan jumlah minggu dalam semester. $$\left(penalty \times \sum_{w,d} max(DayLoad(d,w) - T, 0) \right) / \; nrWeeks$$
MaxBreaks(B,T)	Batasan ini membatasi jumlah *breaks* atau istirahat selama satu hari suatu kelas tertentu yang secara ringkas bisa dikatakan “tidak ada lebih dari *B* istirahat selama sehari”, adanya

Batasan	Pengertian
	break antara kelas jika ada lebih dari *T* slot waktu kosong diantaranya. Tetapi dua kelas yang berurutan dianggap satu blok jika jarak antara kelas tidak lebih besar dari *T* slot waktu. $\lvert MergeBlocks\ (\{\ (C_{.Start}, C_{.End}) \rvert$ $(C_{.Days}\ and\ 2^d) \neq 0 \wedge (C_{.Weeks}\ and\ 2^w) \neq 0$ $\}\ )\ \rvert\ \leq R + 1$
MaxBlocks(M,S)	Batasan ini berlaku untuk membatasi panjang *block* dari gabungan antar kelas pada sehari yang berarti tidak ada lebih dari *M* slot waktu dalam satu *block*. Dua kelas yang dikatakan satu *block* apabila *gap* diantara du akelas tidak lebih dari *T* slot waktu Untuk setiap *block*, jumlah slot waktu dari awal kelas sampai akhir kelas harus kurang dari *M* slot waktu. $max(\{B_{.End} - B_{.Start} \rvert B \in MergeBlocks(\{(C_{.Start}, C_{.End}) \rvert (C_{.Days}\ and\ 2^d) \neq 0 \wedge (C_{.Weeks}\ and\ 2^w) \neq 0\})\}) \leq M$

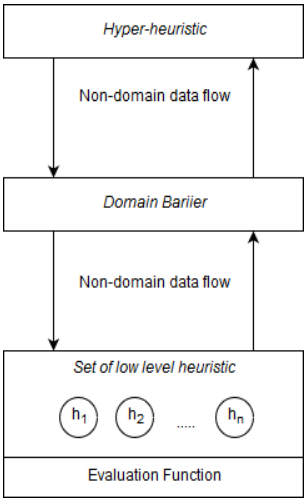
Dalam memodelkan dataset hal pertama yang harus dilakukan adalah menentukan variabel keputusan. Setelah variabel keputusan telah ditentukan, selanjutnya adalah menentukan objective function atau fungsi tujuan dari dataset. Fungsi tujuan untuk dataset ITC 2019 adalah meminimalisir adanya penalti (P) dalam penjadwalan.

$$F(x) = Min \sum_{i=1}^{n} P_i$$

2.2.3. Framework Hyper-Heuristic

Hyper-heuristic secara umum bertujuan untuk mengotomasi proses dengan menggunakan beberapa kumpulan pendekatan [4], pertama (1) dengan memilih dan menggunakan kombinasi heuristics yang berbeda, dan atau (2) mengeksplor heuristic yang sudah ada yang akhirnya menemukan sebuah heuristic baru yang lebih optimal.

Hyper-heuristic tidak langsung menuju ke solution space. Ilustrasi framework dari hyper-heuristic pada Gambar 2.2, menjelaskan bahwa metode ini bekerja pada tingkat abstraksi yang tinggi dan seringkali tidak mempunyai pengetahuan mengenai domain dari masalah. Metode ini bekerja pada low level heuristics sehingga tidak memerlukan parameter tuning karena tidak berpacu pada domain masalah saja [4], [15].



Gambar 2.1 Framework Hyper-heuristic

2.2.4. Tabu Search Algorithm

Tabu Search merupakan sebuah algoritma metaheuristic yang didesain untuk mencari dan mendapatkan solusi global optimum pada permasalahan penjadwalan [5] dengan begitu algoritma ini dapat menghindari keadaan terjebak pada local optima solution.

Algoritma Tabu Search menggunakan prinsip tabu list atau daftar tabu dimana daftar tersebut merupakan suatu memori yang akan menyimpan solusi yang pernah diperoleh dan lebih buruk atau tidak lebih baik dari solusi awal (current solution) [16]. Dengan tersimpannya solusi yang sudah pernah didapatkan, maka tidak akan terjadi solusi yang sama pada setiap iterasi yang dilakukan karena daftar yang ada pada tabu list tidak akan diikutsertakan pada iterasi yang selanjutnya [5] [17].

2.2.5. Simulated Annealing Algorithm

Simulated Annealing adalah algoritma metaheuristik yang mengaplikasikan penerapan atau proses pendinginan baja, proses tersebut meliputi pemanasan baja sampai suhu tertentu dan kemudian dilakukan pendinginan baja secara perlahan. Pada saat baja dipanaskan, atom-atom dalam baja tersebut bergerak bebas sehingga pada saat didinginkan atom tersebut menjadi terbatas gerakannya dan akhirnya membentuk atau menyusun susunan atom yang lebih teratur [8].

Peniruan proses pendinginan baja ini dilakukan dengan mengatur atau menentukan parameter kemudian dikontrol dengan fungsi pengontrol yang sesuai dengan konsep distribusi probabilitas Boltzman. Persamaan Boltzmann yaitu P(E)=e^(-E/t), dimana P merupakan Probabilitas Boltzmann atau peluang mencapai tingkat

energi E, T adalah suhu dan e adalah bilangan eksponensial [8].

Implementasi dari algoritma Simulated Annealing harus memperhatikan beberapa parameter, beberapa parameter ini dirangkum oleh David dan Harel dalam [18] yaitu,

1. Beberapa konfigurasi awal atau keadaan sistem sebagai titik mula.
2. Aturan umum yang digunakan untuk memperoleh konfigurasi baru.
3. Target atau biaya, fungsi yang harus diminimalkan.
4. Jadwal pendinginan dari parameter kontrol, seperti nilai awal dan kapan dan bagaimana mengubahnya.
5. Kondisi terminasi atau kondisi menghentikan proses.

Halaman ini sengaja dikosongkan

BAB III
METODOLOGI PENELITIAN

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

3.1. Metodologi Penelitian

Diagram metodologi pengerjaan Tugas Akhir dapat dilihat pada Gambar 3.1.

3.2. Tahapan Pelaksanaan Tugas Akhir

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

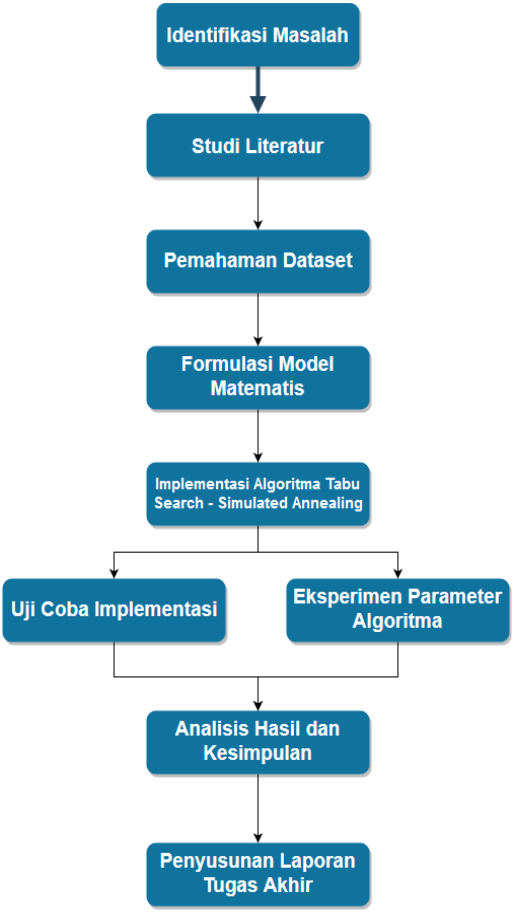
3.2.1. Identifikasi Masalah

Pada tahap ini dilakukan analisis untuk memahami studi kasus yang ada terkait dengan penjadwalan mata kuliah, di tahap ini akan dijabarkan permasalahan apa yang terjadi pata aspek penjadwalan mata kuliah mulai dari latar belakang permasalahan, perumusan masalah, penentuan batasan masalah, penentuan tujuan, manfaat tugas akhir hingga relevansi dari penelitian. Hal ini dilakukan untuk mendapatkan topik penelitian yang sesuai.

3.2.2. Studi Literatur

Tahap ini bertujuan untuk mencari sumber referensi atau informasi yang lebih banyak terkait dengan permasalahan yang ingin diteliti yaitu menyangkut dengan permasalahan penjadwalan mata kuliah yang

berasal dari paper, jurnal, maupun buku atau sumber bacaan lain. Dari tahap ini akan diperoleh algoritma yang akan digunakan untuk menyelesaikan permasalahan yang dilakukan oleh peneliti secara optimal.



Gambar 3.1 Metodologi Pengerjaan Tugas Akhir

3.2.3. Pemahaman Dataset

Tahap selanjutnya adalah pemahaman dataset, data yang digunakan adalah data dari International Timetabling Competition 2019 atau yang biasa disingkat ITC 2019. Pemahaman dataset ini meliputi pemahaman format dataset, pemahaman Batasan atau *constraint* dari dataset, serta model matematis dari dataset ITC 2019.

3.2.4. Formulasi Model Matematis

Pada tahap ini model matematis akan dibuat untuk menyelesaikan permasalahan, hasil dari tahap ini adalah beberapa variabel seperti:

a. Variabel keputusan
b. Fungsi tujuan
c. Batasan-batasan

Variabel Batasan terdiri dari *hard constraint* dan *soft constraint*, dimana *hard constraint* harus dipenuhi sedangkan untuk penalti dari *soft constraint* sebisa mungkin diminimalkan. Model yang dibuat kemudian akan diterjemahkan ke dalam struktur data yang akan digunakan untuk implementasi algoritma.

3.2.5. Implementasi Algoritma Tabu-Simulated Annealing

Tahap implementasi algoritma merupakan tahap utama dalam mengerjakan tugas akhir ini, langkah pertama adalah sistem akan membaca variabel yang diinputkan yaitu jumlah *timeslot* dengan parameter start sebagai awal dari slot dan panjang slot dengan parameter length, *courses* dengan parameter class id dan limit untuk membatasi jumlah *student* dalam suatu kelas serta parameter *room* id.

Selanjutnya *rooms* dengan parameter id dan *capacity* yang merupakan batas maksimal ruangan tersebut,

students dengan parameter *courses* id sebagai acuan mahasiswa tersebut mengambil *courses* apa saja. Serta terdapat parameter days dan weeks sebagai input untuk menentukan apakah ruangan bisa digunakan atau tidak dan dapat juga menentukan kelas yang tersedia atau tidak. Kemudian akan diproses sehingga mendapatkan solusi awal (initial solution). Langkah selanjutnya sistem akan membuat struktur lingkungan yaitu struktur yang akan digunakan untuk membuat pola acak pada iterasi selanjutnya, selain itu menentukan acceptance criteria juga perlu dilakukan, hal ini bertujuan untuk membuat sistem dapat menentukan apakah solusi selanjutnya bisa diterima untuk menggantikan solusi yang lama.

Stopping criteria, atau kriteria untuk menghentikan iterasi adalah hal yang harus diinisiasi selanjutnya, yaitu kriteria yang akan menghentikan iterasi untuk menghasilkan solusi selanjutnya. Setelah semua telah dipersiapkan, algoritma Simulated Annealing akan diimplementasikan dengan menggunakan acceptance criteria yang telah dibuat.

Implementasi dari algoritma ini akan merubah solusi awal dengan solusi selanjutnya yang lebih baik, jika ternyata solusi selanjutnya tidak lebih baik dengan solusi awal maka proses annealing akan dilakukan tetapi jika solusi berhasil lebih baik dari solusi awal maka algoritma Tabu Search akan diimplementasi untuk mencari apakah solusi tersebut masuk pada tabu list atau tidak.

Struktur lingkungan yang tidak terdapat pada tabu list akan diterima sebagai solusi yang baru dan kemudian struktur dari solusi tersebut akan dimasukkan ke dalam tabu list sehingga struktur lingkungan tersebut tidak akan digunakan kembali untuk iterasi selanjutnya. Suhu pada proses algoritma Simulated Annealing akan diturunkan

setiap kali melakukan iterasi. Dengan menggunakan dua algoritma ini sistem dapat menerima solusi yang tidak lebih baik dari solusi awal sehingga sistem tidak akan terjebak dalam solusi lokal optimal (local optimum solution)..

3.2.6. Uji Coba Implementasi

Pada tahap ini dilakukan uji coba pada dataset yang digunakan dalam penelitian, disini akan menghasilkan verifikasi dan validasi apakah algoritma yang telah diimplementasikan sesuai atau tidak dengan kebutuhan dan hasil yang diinginkan. Jika tidak maka akan dilakukan evaluasi terhadap algoritma yang telah diimplementasikan.

3.2.7. Eksperimen Parameter Algoritma

Tahap selanjutnya melakukan eksperimen terhadap parameter-parameter yang terdapat dalam kedua algoritma. Pada algoritma Tabu Search akan dilakukan eksperimen dengan mengubah Panjang tabu list yang dikehendaki, sedangkan untuk algoritma Simulated Annealing dilakukan perubahan terhadap beberapa parameter yaitu suhu awal, suhu akhir, koefisien penurunan suhu, dan banyaknya iterasi pada setiap penurunan suhu. Dengan melakukan eksperimen ini solusi yang didapatkan akan lebih optimum jika menemukan parameter yang sesuai dengan dataset.

3.2.8. Analisis Hasil dan Kesimpulan

Setelah dilakukannya uji coba dan eksperimen parameter, Analisa hasil dilakukan untuk menentukan parameter mana yang menghasilkan solusi atau keluaran yang paling optimal. Analisis hasil juga dilakukan pada setiap iterasi dengan cara membandingkan setiap iterasi yang dilakukan.

3.2.9. Penyusunan Laporan Tugas Akhir

Tahap terakhir dari penelitian ini adala penyusunan laporan penelitian dengan cara mengumpulkan semua dokumentasi dari penelitian mulai dari tahap masukan, proses, hingga keluaran dan juga menganalisis terhadap hasil akhir yang didapatkan. Pada tahap ini akan menghasilkan buku tugas akhir yang diharapkan dapat menjadi sumber referensi untuk penelitian selanjutnya.

BAB IV
PERANCANGAN

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

4.1. Pemahaman Data

Dalam perancangan tugas akhir ini, data yang digunakan adalah *International Timetabling Competition* 2019 Dataset atau biasa disebut ITC 2019 dataset. Dataset tersebut dapat diperoleh dari situs *www.itc2019.org* dimana dataset tersebut disediakan oleh panitia penyelenggara kompetisi penjadwalan internasional pada tahun 2019.

Pada subbab 0 sudah dijelaskan bahwa ITC2019 dataset terdiri dari 4 grup yaitu Test, Early, Middle, dan Late, tetapi pada pengerjaan tugas akhir ini hanya menggunakan data Test dan Early yang berjumlah 5 kelas test dataset (*instances*) dan early dataset yang berekstensi .xml.

Setiap kategori kelas dataset (*instances*), *tiny, small, medium, large, early,* memiliki jumlah *courses, classes, rooms, students, distributions* yang bebeda-beda. Sementara untuk setiap kelas dataset (*instances*) memiliki nilai optimasi yang bebeda-beda yaitu nilai *time, room, distribution,* dan *student.*

4.1.1. Inisiasi Data

Inisiasi data pada dataset ITC dimulai dengan *Node Elemen* pada setiap awal baris. *Elemen* pertama dalam ITC dataset adalah inisiasi problem yang akan digunakan

dalam penjadwalan mata kuliah diikuti dengan baris kedua yaitu inisiasi optimasi penjadwalan.

Pada baris pertama yaitu inisiasi problem memiliki 4 attribute data. Keempat attribute tersebut masing-masing memiliki maksud sebagai berikut.

1. Attribute "*name"*, menyatakan nama problem atau dataset tersebut.
2. Attribute *"nrDays"*, menyatakan jumlah hari yang tersedia selama penjadwalan.
3. Attribute *"slotsPerDay"*, menyatakan jumlah *timeslot* yang tersedia selama satu hari.
4. Attribute *"nrWeeks"*, menyatakan jumlah minggu yang tersedia selama satu semester.

Sedangkan pada baris kedua terdapat inisiasi optimisasi penjadwalan yang juga memiliki 4 attribute. Empat data tersebut memiliki arti sebagai berikut.

1. Attribute *"time"*, merupakan beban optimasi pada penalty dari pemilihan waktu.
2. Attribute *"room"*, merupakan beban optimasi pada penalty dari pemilihan ruang.
3. Attribute *"distribution"*, merupakan beban optimasi pada penalty dari pemilihan *constraint* atau batasan.
4. Attribute *"student"*, merupakan beban optimasi pada penalty dari pemilihan murid.

Gambar 4.1 merupakan contoh dari inisiasi problem dan optimasi penjadwalan dalam data *lums-sum17.xml* atau bisa dikatakan dataset *Tiny*.

```
<problem name="lums-sum17" nrDays="7" slotsPerDay="288" nrWeeks="9">
  <optimization time="1" room="1" distribution="10" student="10"/>
```

Kode 4.1 Inisiasi Problem dan Optimasi Data

Pada setiap dataset atau *instance* yang tersedia, terdapat 4 konten utama yaitu *rooms, courses, constraints,* dan *students* yang diinisiasi atau diawali dengan elemen pada setiap awal baris pertama dari konten tersebut. Inisiasi konten tersebut dapat dilihat sebagai berikut.

4.1.1.1. Inisiasi Konten *Rooms*

Baris selanjutnya setelah inisiasi problem dan optimasi penjadwalan adalah inisiasi elemen *rooms* yang menyatakan karakteristik dalam konten *room*. Di dalam elemen *room* terdapat dua attribut yaitu *"id"* yang menyatakan nomor ID dari ruang tersebut dan *"capacity"* yaitu kapasitas ruang tersebut.

Adapun selain elemen *room* terdapat dua anak elemen yaitu *"travel"* dan *"unavailable"*. Gambar 4.2 merupakan potongan dari inisiasi konten *rooms* pada salah satu data.

```
<rooms>
  <room id="1" capacity="40">
    <travel room="20" value="1"/>
    <travel room="21" value="1"/>
    <travel room="22" value="1"/>
    <travel room="23" value="1"/>
    <unavailable days="0010000" start="192" length="18" weeks="000010000000000"/>
    <unavailable days="0000100" start="144" length="24" weeks="000000000000100"/>
  </room>
```

Kode 4.2 Inisiasi Konten Rooms

Pada Kode 4.2, dapat diketahui bahwa ruangan pertama berkapasitas 40 orang, dan memiliki 4 anak elemen *travel* dan 2 anak elemen *unavailable*. Keterangan anak elemen tersebut adalah sebagai berikut.

1. Elemen travel *"room",* merupakan petunjuk jarak tempuh antar ID ruang. Sebagai contoh pada Gambar 4.2 terdapat ruang ID 1 yang memiliki jarak tempuh dengan ruang ID 20, 21, 22, dan 23.
2. Elemen travel *"value",* merupakan nilai jarak tempuh antar ruang dalam *timeslot*.
3. Elemen *"unavailable",* merupakan anak elemen yang menunjukkan pada minggu, hari dan *timeslot* keberapa ruang tersebut tidak bisa digunakan.

4.1.1.2. Inisiasi Konten *Courses*

Setelah inisiasi konten *room* adalah inisiasi konten *courses*. Di dalam *courses* terdapat anak elemen *course* yang memiliki satu maupun beberapa anak elemen *config*. Di dalam *config* juga terdapat satu atau beberapa elemen *subpart* yang dimana didalamnya mempunyai beberapa anak elemen *class*. Dimasing-masing anak elemen mempunyai attribute *id* yang menunjukkan ID dari elemen tersebut.

Untuk elemen *class*, memiliki attribute limit yang merupakan batas dari mahasiswa yang tercatat dalam *class* tersebut. Terdapat juga attribut *parent* yang menunjukkan bahwa murid yang mengambil kelas tersebut harus juga mengambil kelas *parent.* Elemen *class* sendiri memiliki beberapa anak elemen sebagai berikut.

1. Elemen *room "id",* merupakan anak elemen yang menunjukkan kelas tersebut bisa diselenggarakan pada ID ruang tertentu.
2. Elemen *room "penalty",* merupakan anak elemen yang menunjukkan nilai penalty jika

kelas tersebut masuk atau diselenggarakan pada *room* atau ruang terebut.

3. Elemen *"time",* merupakan anak elemen yang menunjukkan list minggu, hari, dan *timeslot* yang tersedia untuk kelas tersebut, serta attribut penalty jika kelas tersebut dilakukan pada waktu tertentu.

Kode 4.3 merupakan potongan inisiasi konten *courses* pada salah satu data.

```
<courses>
  <course id="1">
    <config id="1">
      <subpart id="1">
        <class id="1" limit="22">
          <room id="10" penalty="0"/>
          <room id="11" penalty="0"/>
          <room id="12" penalty="0"/>
          <time days="0100000" start="90" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0010000" start="90" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0001000" start="90" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0000100" start="90" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0100000" start="114" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0010000" start="114" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0001000" start="114" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0000100" start="114" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0100000" start="138" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0010000" start="138" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0001000" start="138" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0000100" start="138" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0100000" start="162" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0010000" start="162" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0001000" start="162" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0000100" start="162" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0100000" start="186" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0010000" start="186" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0001000" start="186" length="22" weeks="111111111111111" penalty="0"/>
          <time days="0000100" start="186" length="22" weeks="111111111111111" penalty="0"/>
        </class>
```

Kode 4.3 Inisiasi Konten Courses

4.1.1.3. Inisiasi Konten *Distributions*

Baris selanjutnya setelah inisiasi *courses* atau mata kuliah adalah inisiasi konten *distributions* atau yang biasa disebut sebagai batasan. Dalam dataset ITC terdapat 19 distribusi *constraints* yang dimana semua *constraint* tersebut bisa berupa *hard constraint* maupun

soft constraint. Dalam inisiasi *distributions*, terdapat elemen *distribution* yang mempunyai attribute sebagai berikut.

1. Attribut distribusi *"type",* merupakan tipe atau nama dari batasan tersebut.
2. Attribut distribusi *"required",* merupakan penunjuk bahwa kelas tersebut merupakan *hard constraint*.
3. Attribut distribusi *"penalty",* merupakan penunjuk bahwa kelas tersebut merupakan *soft constraint* dengan parameter sebagai nilai dari penalti batasan tersebut.

Didalam elemen *distribution* terdapat anak elemen *class* dengan parameter *id* yang menunjukkan bahwa batasan tersebut berlaku pada id kelas yang terdapat dalam batasan tersebut. Kode 4.4 merupakan inisiasi konten *distributions* pada salah satu data.

```
<distribution type="NotOverlap" required="true">
  <class id="158"/>
  <class id="162"/>
  <class id="167"/>
  <class id="174"/>
  <class id="159"/>
  <class id="160"/>
  <class id="165"/>
  <class id="166"/>
</distribution>
<distribution type="NotOverlap" required="true">
  <class id="83"/>
  <class id="84"/>
</distribution>
<distribution type="SameRoom" penalty="4">
  <class id="81"/>
  <class id="82"/>
</distribution>
<distribution type="SameRoom" penalty="4">
  <class id="83"/>
  <class id="84"/>
</distribution>
```

Kode 4.4 Inisiasi Konten Distributions

4.1.1.4. Inisiasi Konten *Students*

Inisiasi terakhir yang dilakukan adalah konten *students*. Inisiasi murid dimulai dengan elemen *student* yang memiliki attribut *id* sebagai id unik dari masing masing murid. Didalam elemen tersebut terdapat anak elemen yaitu *course* dengan attribute *id*, yang menunjukkan bahwa murid dengan id diatas harus masuk ke dalam *course* atau mata kuliah dengan id tersebut

Kode 4.5 merupakan potongan inisiasi dari konten *students* pada salah satu data.

```
<students>
  <student id="1">
    <course id="4"/>
  </student>
  <student id="2">
    <course id="1"/>
  </student>
  <student id="3">
    <course id="20"/>
    <course id="26"/>
  </student>
  <student id="4">
    <course id="5"/>
  </student>
```

Kode 4.5 Inisiasi Konten Students

4.2. Transformasi Model Matematis ke Struktur Data

Transformasi model matematis ke struktur data atau lebih dikenal dengan *pseudocode* memiliki tujuan untuk mempermudah implementasi algoritma terhadap dataset permasalahan.

Transformasi ke struktur data dilakukan terhadap semua model matematis dataset, yaitu fungsi tujuan (*objective function*), dan batasan (*constraint*).

4.2.1. Variabel Keputusan

Variabel keputusan merupakan variabel yang menjabarkan secara lengkap keputusan-keputusan yang akan dibuat. Berikut merupakan beberapa variabel keputusan yang terdapat pada dataset *International Timetabling Competition* 2019.

$C = class = \{c_1, \ldots, c_{|c|}\}$
$T = time = \{t_1, \ldots, t_{|t|}\}$
$R = room = \{r_1, \ldots, r_{|r|}\}$
$W = week = \{w_1, \ldots, w_{|w|}\}$
$D = day = \{1, \ldots, 7\}$
$S = start = \{1, \ldots, 288\}$
$L = length = \{1, \ldots, 288\}$
$E = end = \{1, \ldots, 288\}$
$T = time = \{1, \ldots, 288\}$

4.2.2. Fungsi Tujuan

Fungsi tujuan dari dataset ITC adalah meminimalisasi jumlah penalti yang didapat pada waktu meletakkan kelas pada ruang dan waktu serta apabila melanggar *soft constraint*.

$$Z = \min(P_1 + P_2 + P_3)$$

Dengan,

1. P_1, Nilai penalti terhadap pemilihan ruangan yang digunakan dalam penjadwalan
2. P_2, Nilai penalti terhadap pemilihan *timeslot* yang yang digunakan dalam penjadwalan
3. P_3, Nilai penalti terhadap pelanggaran *Soft Constraint.*

Penalti yang dihasilkan pada setiap penalti tersebut dikalikan dengan nilai optimasi pada masing-masing

dataset kemudian dijumlahkan sebagai nilai akhir fungsi tujuan.

$$z = \min \left(\sum_{i=1}^{sc} SC_i \times Opt_d + \sum_{j=1}^{r} P_{r.j} \times Opt_r + \sum_{k=1}^{t} P_{ts.k} \times Opt_t \right).$$

4.2.3. Transformasi *Distribution Constraint*

Pada dataset *International Timetabling Competition* terdapat 19 *constraint* atau batasan yang bisa menjadi batasan keras (*hard constraint*) maupun batasan lunak (*soft constraint*) sesuai dengan atribut pada batasan tersebut.

4.2.3.1. Constraint SameStart

Constraint pertama dalam dataset ITC adalah kelas harus diletakkan di waktu mulai yang sama dengan kelas lain yang berada dalam batasan tersebut.

$$C_1SS = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} s_i = s_j$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{SS} = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., c}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.2. Constraint SameTime

Constraint selanjutnya adalah kelas harus dijadwalkan pada waktu yang sama dengan kelas lain pada list kelas dalam batasan terebut.

$$C_2ST = \left(\sum_{i=1}^{C-1} \sum_{j=1+1}^{C} s_i \leq s_j \wedge e_j \leq e_i \right) \vee \left(\sum_{i=1}^{C-1} \sum_{j=1+1}^{C} s_j \leq s_i \wedge e_i \leq e_j \right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{ST} = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.3. Constraint DifferentTime

Constraint ketiga dalam dataset ITC adalah kebalikan dari *constraint* kedua (*SameTIme)* yaitu kelas yang ada di dalam batasan ini tidak boleh memiliki waktu yang sama dengan kelas lain.

$$C_3DT = \left(\sum_{i=0}^{C-1} \sum_{j=i+1}^{C} (e_i \leq s_j) \vee (e_j \leq s_i) \right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{DT} = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.4. Constraint SameDays

Constraint selanjutnya dalam dataset ITC adalah kelas dalam batasan ini harus dimulai atau dilakukan pada hari yang sama dengan kelas lain yang berada pada list batasan ini.

$$C_1SD = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} d_i = d_j$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{SD} = M \times penalty$$

Dimana,

d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.5. Constraint DifferentDays

Constraint ini membatasi bahwa tidak boleh ada kelas dalam batasan ini yang dilakukan di hari yang sama, semua kelas dalam batasan ini harus diletakkan pada hari yang berbeda.

$$C_5DD = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} d_i \neq d_j$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{DD} = M \times penalty$$

Dimana,

d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.6. Constraint SameWeeks

Constraint selanjutnya adalah kelas dalam batasan ini harus diletakkan dalam minggu yang bersamaan.

$$C_6SW = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} w_i = w_j$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{SW} = M \times penalty$$

Dimana,

w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.7. Constraint DifferentWeeks

Constraint ini membatasi bahwa kelas dalam batasan ini harus diletakkan pada minggu yang berbeda dengan kelas lainnya.

$$C_7DW = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} w_i \neq w_j$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{DW} = M \times penalty$$

Dimana,

w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.

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

4.2.3.8. Constraint SameRoom

Constraint ini membatasi bahwa kelas yang berada pada batasan ini harus diletakkan pada ruang yang sama.

$$C_8SR = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} r_i = r_j$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{SR} = M \times penalty$$

Dimana,

r = *{1, ..., R}* adalah ruangan yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.9. Constraint DifferentRoom

Constraint ini membatasi beberapa kelas dalam batasan bahwa tidak boleh diletakkan di ruang kelas yang sama, yang berarti harus diletakkan di tempat yang berbeda.

$$C_9DR = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} r_i \neq r_j$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{DR} = M \times penalty$$

Dimana,

r = *{1, ..., R}* adalah ruangan yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.10. Constraint Overlap

Constraint Overlap mengharuskan kelas dalam batasan ini memiliki waktu atau *timeslot* yang tumpang tindih antara kelas lain.

$$C_{10}O = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} s_j < e_i \wedge$$

$$s_i < e_j \wedge$$

$$\left(d_{ij} \neq 0 \wedge w_{ij} \neq 0\right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_O = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.

M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.11. Constraint NotOverlap

Berlawanan dengan *Overlap*, *constraint* ini mengharuskan kelas di dalam batasan memiliki waktu yang tidak tumpang tindih antara kelas lain, atau bisa dikatakan tidak boleh memiliki waktu yang sama.

$$C_{11}NO = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} e_i \leq s_j \vee$$

$$e_j \leq s_i \vee \left(d_{ij} = 0 \vee w_{ij} = 0\right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{NO} = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.12. Constraint SameAttendees

Kelas dalam batasan ini diharuskan bertemu pada waktu dan ruang yang ditentukan sehingga murid maupun dosen bisa menghadiri kelas tersebut.

$$C_{12}SA = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} e_i + t_{i.rTravel.j} \leq s_j \ \vee$$

$$e_j + t_{j.rTravel.i} \leq s_i \vee$$

$$\left(d_{ij} = 0 \vee w_{ij} = 0\right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{SA} = M \times penalty$$

Dimana,

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

e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.

$t_{rTravel}$ = *{1, ..., 288}* waktu tempuh antar ruang dalam *timeslot*.

M = jumlah pasangan kelas yang melanggar batasan

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

4.2.3.13. Constraint Precedence

Batasan ini memberikan daftar urutan, dimana kelas teratas dalam batasan ini harus didahulukan atau dijadwalkan di minggu, hari, atau waktu yang lebih awal dari kelas lain.

$$C_{13}P = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} \left(w_i < w_j \vee \left(w_i = w_j \wedge d_i < d_j \vee \left(d_i = d_j \wedge e_i \leq s_j\right)\right)\right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_P = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.

4.2.3.14. Constraint WorkDay(S)

Batasan ini mengharuskan bahwa pasangan kelas yang terdapat dalam batasan dimana waktu antara mulainya kelas pertama dan berakhirnya kelas terakhir tidak boleh melebihi S_{WD} *timeslot*

$$C_{14}WD = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} \left(d_{ij} = 0\right) \vee \left(w_{ij} = 0\right) \vee \left(max\left(e_i, e_j\right) - min\left(s_i, s_j\right) \leq S_{WD}\right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{WD} = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.
S_{WD} = jumlah *timeslot* maksimal batasan.

4.2.3.15. Constraint MinGap(G)

Pasangan kelas dalam batasan ini jika dijadwalkan pada minggu dan hari yang sama maka setidaknya harus ada jarak sebesar *G timeslot* yang memisahkan antara kedua kelas tersebut.

$$C_{15}MG = \sum_{i=1}^{C-1} \sum_{j=i+1}^{C} (d_{ij} = 0) \vee (w_{ij} = 0) \vee (e_i + G \leq s_j) \vee (e_j + G \leq s_i)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh setiap pasangan kelas (*M*) yang melanggar batasan ini.

$$totalPenalty_{MG} = M \times penalty$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.
M = jumlah pasangan kelas yang melanggar batasan.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.
G = jumlah jarak gap *timeslot* maksimal batasan.

4.2.3.16. Constraint MaxDays(D)

Kelas pada batasan ini tidak boleh dijadwalkan lebih dari *D* hari kerja dalam satu minggu.

$$C_{16}MD(D) = countNonzerobits \sum_{i=1}^{7} d_i \leq D$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan oleh jumlah hari (*X*) yang melebihi konstanta *D* yang diberikan.

$$totalPenalty_{MD} = X \times penalty$$

Dimana,

d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
X = jumlah hari yang melebihi konstanta *D.*
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.
D = jumlah maksimal hari kerja.

4.2.3.17. Constraint MaxDayLoad(S)

Batasan ini membatasi jumlah total Panjang slot yang ditetapkan untuk sekumpulan kelas tidak boleh lebih dari S_{MDL} *timeslot* setiap hari selama satu semester.

$$C_{17}MDL(S) = DayLoad(d,w) \leq S_{MDL}$$

$$DayLoad(d,w) = \sum_i \left\{ \left(l_i \middle| \left(\sum_{i=1}^{d} d_i \neq 0 \wedge 2^d \neq 0 \right) \wedge \left(\sum_{i=1}^{w} w_i \neq 0 \wedge 2^w \neq 0 \right) \right) \right\}$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan dengan jumlah slot yang melebihi konstanta S_{MDL} dan dibagi dengan jumlah minggu (*nrWeeks*)

$$totalPenalty_{MDL} = (penalty \times \sum_{w,d} \max(Dayload(d,w) - S, 0))) / nrWeeks$$

Dimana,

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

e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.

d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.

w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.

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

S_{MDL} = jumlah *timeslot* maksimal batasan.

4.2.3.18. Constraint MaxBreaks(R, S)

Batasan *MaxBreaks* membatasi jumlah jeda antar kelas dalam sehari yang dimana jeda tersebut tidak boleh

melebihi S_{MB} *timeslot* dan tidak boleh memiliki jeda lebih dari *R*.

$$C_{18}MB(R, S_{MB}) = MergeBlocks\{\sum_{i=1}^{c} s_i, e_i | (\sum_{i=1}^{d} d_i \wedge 2^d \neq 0) \wedge (\sum_{i=1}^{w} w_i \wedge 2^w \neq 0)\} | \leq R + 1$$

$$MergeBlocks = \sum_{i=1}^{c-1} \sum_{j=i+1}^{c} \left(e_{b_a} + S_{MB} \geq s_{b_b}\right) \wedge \left(e_{b_b} + S_{MB} \geq s_{b_a}\right) \Rightarrow \left(s_b = min\left(s_{b_a}, s_{b_b}\right) \wedge \left(e_b = min\left(e_{b_a}, e_{b_b}\right)\right)\right)$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan dengan jumlah jeda atau *breaks* tambahan ($R_{additional}$) yang dihitung setiap hari dalam seminggu dan kemudian dibagi dengan jumlah minggu (*nrWeeks*).

$$totalPenalty_{MBREAKS} = (penalty \times R_{additional})/nrWeeks$$

Dimana,

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

e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.

d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.

w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.

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

S_{MB} = jumlah *timeslot* maksimal.
R = jumlah *break* maksimal.

4.2.3.19. Constraint MaxBlock(M, S)

Constraint ini membatasi panjang atau jumlah waktu dari blok kelas dimana tidak boleh lebih dari *M timeslot*.

$$\max\{e_B - s_B | B \in MergeBlocks\{(s_c, e_c)| \sum_{i=1}^{c} (d_c \wedge 2^d) \neq 0 \wedge (w_c \wedge 2^w) \neq 0\})\} \leq M$$

Jika batasan termasuk *soft constraint*, penalti pada batasan akan dikalikan dengan jumlah blok yang melebihi slot waktu *M* yang dihitung setiap hari dalam seminggu dan kemudian dibagi dengan jumlah minggu (*nrWeeks*).

$$totalPenalty_{MBLOCK} = (penalty \times M_{over})/nrWeeks$$

Dimana,

s = *{1, ..., 288}* adalah waktu mulai kelas pada *timeslot* tertentu.
e = *{1, ..., 288}* adalah waktu berakhirnya kelas pada *timeslot* tertentu.
d = *{1, ..., 7}* adalah hari yang dijadwalkan pada kelas tersebut.
w = *{1, ..., W}* adalah minggu yang dijadwalkan pada kelas tersebut.
i,j = *{1, 2, ..., C}* adalah daftar kelas yang terdaftar pada batasan tersebut.
S_{MB} = jumlah *timeslot* maksimal batasan.
M = maksimal panjang blok kelas dalam *timeslot*

4.3. Implementasi Algoritma *Tabu Search-Simulated Annealing*

Implementasi algoritma *Tabu Search-Simulated Annealing* dengan kerangka Hyper-Heuristic digunakan untuk mencari solusi optimum. Implementasi algoritma tersebut dilakukan pada saat tahap perancangan telah menghasilkan solusi awal (*initial solution*). Terdapat juga *low level heuristic* yang akan digunakan adalah *move room* dan *move timeslot*. Solusi awal yang telah melalui *low level heuristic* akan dioptimasi menggunakan algoritma *Tabu Search-Simulated Annealing*. Adapun tahapan dalam implementasi akan dijelaskan lebih lanjut pada BAB V IMPLEMENTASI.

BAB V
IMPLEMENTASI

Bab ini menjelaskan proses pelaksanaan penelitian tugas akhir dan proses implementasi algoritma Tabu-Simulated Annealing Hyper-Heuristic terhadap Socha dataset dengan menggunakan IDE Netbeans 8.2 dan bahasa pemrogaman Java versi 8.

5.1. Pembentukan Solusi Awal

Subbab ini menjelaskan mengenai proses awal implementasi algoritma, yaitu menciptakan solusi awal (*generate initial solution*). Aktivitas-aktivitas yang terkait dengan pembuatan solusi awal antara lain adalah membaca file input, membuat beberapa tempat penyimpanan data antara lain data *Rooms, Courses,* dan *Distriutions*, serta melakukan pengurutan kelas. Aktivitas yang paling utama adalah pembentukan solusi awal yang telah memenuhi *hard constraint*.

5.1.1. Pembacaan File Input

Untuk pembentukan solusi awal, pertama yang dilakukan adalah membuat kode program untuk membaca file input.

Pada Kode 5.1 selain untuk membaca file input .xml, kode program juga terdapat inisiasi variabel yang digunakan untuk menyimpan nilai pada data file input. Beberapa nilai data yang disimpan dalam variabel adalah jumlah minggu, jumlah hari, jumlah ruangan, jumlah *timeslot*, dan jumlah batasan/*constraint*.

```
File fXmlFile = new File("src/itc2019/" + 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();

jWeek = Integer.parseInt(rootElement.getAttribute("nrWeeks"));
jDays = Integer.parseInt(rootElement.getAttribute("nrDays"));
jSlot = Integer.parseInt(rootElement.getAttribute("slotsPerDay")) + 1;

ReadRoom(problemChild);
ReadCourse(problemChild);
ReadCons(problemChild);
makeTS(jWeek, jDays, idRomBesar);
```

Kode 5.1 Pembacaan File Input

Dalam pembacaan file input juga dilakukan *sorting* untuk memilih urutan kelas yang dimana nantinya akan dijadwalkan berdasarkan kelas yang mempunyai jumlah daftar *timeslot* dan ruang yang sedikit dan kelas yang memiliki daftar *hard constraint* yang banyak. Kode dari *sorting* tersebut dapat dilihat pada Kode 5.2.

```
Collections.sort(listKelas, new groupSort(
        new sortTSKelas(), new sortRoomKelas(),
        new sortKlsHC(), new sortUrutanKls())
);
```

Kode 5.2 Pembacaan File Input

Selain variabel tersebut, terdapat beberapa *method* untuk menyimpan file input diantaranya *ReadRoom*, untuk menyimpan data ruang pada file input. *ReadCourse*, untuk menyimpan data mata kuliah dan *ReadCons*, untuk menyimpan data batasan atau *constraint* yang terdapat dalam file input.

5.1.2. Pembuatan *Method Read Room*

Selanjutnya adalah pembuatan *method* untuk menyimpan data ruang / *room*. Di dalam *method* ini

terdapat beberapa struktur data array maupun arraylist. Beberapa struktur data yang dibuat meliputi arraylist *Room - Capacity*, array 2-dimensi matriks *travelRoom*, dan array 4-dimensi matriks *unavailableRoom*.

5.1.2.1. Pembuatan *Arraylist Rooms – Capacity*

Pembuatan *arraylist rooms-capacity* dilakukan menggunakan struktur data arraylist yang berisi ID ruang dan nilai kapasitas dari ruang tersebut Kode 5.3 merupakan kode program pembuatan struktur data *arraylist rooms-capacity*.

```
void ReadRoom(Element problemChild) {
    NodeList rL = problemChild.getElementsByTagName("rooms");
    Node rN = rL.item(0);
    Element roomsElement = (Element) rN.getChildNodes();
    NodeList childRoomsL = (NodeList) roomsElement.getElementsByTagName("room");

    setjRoom(childRoomsL.getLength());
    ArrayList<Integer> idRom = new ArrayList<>();

    for (int rm = 0; rm < jRoom; rm++) {
        Node nRoom = childRoomsL.item(rm);
        Element roomElement = (Element) nRoom;
        int roomID, capRoom;
        roomID = Integer.parseInt(roomElement.getAttribute("id"));
        capRoom = Integer.parseInt(roomElement.getAttribute("capacity"));

        Room room = new Room(roomID, capRoom);
        listRoom.add(room);
        idRom.add(roomID);
    }
```

Kode 5.3 Pembuatan Arraylist *Rooms - Capacity*

5.1.2.2. Pembuatan Matriks *Travel Room*

Pembuatan matriks *travel room* dilakukan menggunakan struktur data array 2-dimensi dengan ukuran *room x room*. Matriks *travel room* digunakan untuk menampung nilai jarak antara kelas atau ruang satu dengan ruang yang lain. Kode program pembuatan matriks *travel room* ditunjukkan oleh Kode 5.4.

```
for (int room = 0; room < childRoomsL.getLength(); room++) {
    Node nRoom = childRoomsL.item(room);
    Element roomElement = (Element) nRoom;
    Room roomA = listRoom.get(room);
    int roomID = Integer.parseInt(roomElement.getAttribute("id")) - 1;

    NodeList travelList = (NodeList) roomElement.getElementsByTagName("travel");
    NodeList unRoomList = (NodeList) roomElement.getElementsByTagName("unavailable");

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

Kode 5.4 Pembuatan Matriks Travel *Room*

5.1.2.3. Pembuatan Matriks *Unavailable Rooms*

Matriks *unavailable rooms* adalah matriks yang menjelaskan apakah ruangan tersebut tidak bisa ditempati pada waktu yang telah ditentukan. Matriks *unavailable rooms* disusun menggunakan struktur data array 4-dimensi dengan ukuran matriks adalah *room x week x day x length*. Kode 5.5 merupakan kode program matriks *suitable rooms*.

```
for (int unrm = 0; unrm < unRoomList.getLength(); unrm++) {
    Node unavNode = unRoomList.item(unrm);

    Element unRoomElement = (Element) unavNode;

    String[] wk = unRoomElement.getAttribute("weeks").split("");
    String[] ds = unRoomElement.getAttribute("days").split("");
```

```
    int[] a = new int[wk.length];
    int[] b = new int[ds.length];
    int ts = Integer.parseInt(unRoomElement.getAttribute("start"));
    int te = Integer.parseInt(unRoomElement.getAttribute("start"))
           + Integer.parseInt(unRoomElement.getAttribute("length"));

    //MEMASUKKAN NILAI UNAVAILABLE ROOM KE ARRAY
    for (int j = 0; j < wk.length; j++) {
        a[j] = Integer.parseInt(wk[j]);
        if (a[j] == 1) {
            for (int k = 0; k < ds.length; k++) {
                b[k] = Integer.parseInt(ds[k]);
                if (b[k] == 1) {
                    for (int l = ts; l <= te; l++) {
                        unroom[room][j][k][l] = 1;
                    }
                }
            }
        }
    }
}
```

Kode 5.5 Pembuatan Matriks Unavailable *Rooms*

5.1.3. Pembuatan *Method Read Courses*

Setelah membuat *method read room* maka selanjutnya adalah pembuatan *method read rourses* untuk menyimpan data mata kuliah (*courses*), konfigurasi (*config)*, sub bagian (*subpart*), dan kelas (*Class*) yang dimana di dalam *method* ini terdapat beberapa struktur data arraylist.

Arraylist penyimpanan *courses*, *config*, dan *subpart* hanya menyimpan ID dari masing-masing bagian sedangkan untuk bagian *class*, selain id kelas terdapat attribut *limit*, *parent* dan *room.* Di dalam penyimpanan objek *class* terdapat penyimpanan *available room* dan *available time* dalam bentuk struktur data *Arraylist* dan dalam objek yang berbeda. Kegunaan *available room* dan *available day* adalah menyimpan list waktu dan ruang yang bisa digunakan oleh kelas tersebut.

Kode untuk penyimpanan data id setiap elemen ditunjukkan oleh Kode 5.6.

```
void ReadCourse(Element problemChild) {
    NodeList cL = problemChild.getElementsByTagName("courses");
    Node cN = cL.item(0);
    Element coursesElement = (Element) cN.getChildNodes();
    NodeList courseL = coursesElement.getElementsByTagName("course");
    jCourse = courseL.getLength();

    for (int i = 0; i < courseL.getLength(); i++) {
        Node courseN = courseL.item(i);
        Element courseE = (Element) courseN;
        NodeList configL = courseE.getElementsByTagName("config");

        courseId = Integer.parseInt(courseE.getAttribute("id"));
        Course course = new Course(courseId);
        for (int j = 0; j < configL.getLength(); j++) {
            Node configN = configL.item(j);
            Element configE = (Element) configN;
            NodeList subpartL = configE.getElementsByTagName("subpart");

            configId = Integer.parseInt(configE.getAttribute("id"));
            Config config = new Config(configId);
            for (int k = 0; k < subpartL.getLength(); k++) {
                Node subpartN = subpartL.item(k);
                Element subpartE = (Element) subpartN;
                NodeList classL = subpartE.getElementsByTagName("class");

                subpartId = Integer.parseInt(subpartE.getAttribute("id"));
                Subpart subpart = new Subpart(subpartId);
                for (int l = 0; l < classL.getLength(); l++) {
                    Node classN = classL.item(l);
                    Element classE = (Element) classN;
```

Kode 5.6 Penyimpanan Data *Course*, Config, dan Subpart

Selanjutnya adalah pembuatan Objek kelas dan pembuatan arraylist *available room* dan *available time*. Seperti yang di jelaskan diatas objek kelas memiliki 4 attribut yaitu *id*, *limit* untuk membatasi isi dari kelas, *parent* sebagai induk kelas, dan *room* dimana attribute *room* memiliki parameter *false* yang berarti kelas tersebut tidak membutuhkan ruangan. Adapun kode program *conflict matrix* terdapat pada Kode 5.7.

```
for (int l = 0; l < classL.getLength(); l++) {
    Node classN = classL.item(l);
    Element classE = (Element) classN;
    NodeList roomL = classE.getElementsByTagName("room");
    NodeList timeL = classE.getElementsByTagName("time");

    classId = Integer.parseInt(classE.getAttribute("id"));
    limit = Integer.parseInt(classE.getAttribute("limit"));
    if (classE.hasAttribute("parent")) {
        parent = Integer.parseInt(classE.getAttribute("parent"));
    } else {
        parent = -1;
    }
    hasRoom = !classE.hasAttribute("room");
    Kelas kelas = new Kelas(classId, parent, limit, hasRoom);
```

Kode 5.7 Pembuatan Objek Kelas

Selanjutnya adalah membuat struktur data Arraylist untuk menyimpan list waktu dan ruang yang tersedia. Kode program tersebut terdapat pada Kode 5.8.

```
for (int ruang = 0; ruang < roomL.getLength(); ruang++) {
    Node roomN = roomL.item(ruang);
    Element ruangE = (Element) roomN;
    int rg = Integer.parseInt(ruangE.getAttribute("id"));
    int pen = Integer.parseInt(ruangE.getAttribute("penalty"));

    RoomAss room = new RoomAss(rg, pen);
    kelas.addARoom(room);
}

for (int time = 0; time < timeL.getLength(); time++) {
    Node timeN = timeL.item(time);
    Element timeE = (Element) timeN;
    String[] wk = timeE.getAttribute("weeks").split("");
    String[] ds = timeE.getAttribute("days").split("");
```

```
int[] week = new int[wk.length];
int[] day = new int[ds.length];
for (int m = 0; m < wk.length; m++) {
    week[m] = Integer.parseInt(wk[m]);
}
for (int m = 0; m < ds.length; m++) {
    day[m] = Integer.parseInt(ds[m]);
}

int start = Integer.parseInt(timeE.getAttribute("start"));
int length = Integer.parseInt(timeE.getAttribute("length"));
int penT = Integer.parseInt(timeE.getAttribute("penalty"));

TimeAss timeslot = new TimeAss(week, day, start, length, penT);
kelas.addTS(timeslot);
```

Kode 5.8 Penyimpanan Available *Room* dan Available Day

5.1.4. Pembuatan *Method Read Cons*

Method Read Cons merupakan method yang menyimpan daftar batasan atau *constraint* yang terdapat dalam dataset ITC tersebut. Penyimpanan daftar tersebut menjadi dua bagian yaitu *Hard Constraint* dan *Soft constraint*. Penyimpanan batasan tersebut disimpan dalam objek Kelas dan juga dalam objek *Constraint*, kegunaan dari menyimpan ke dalam objek Kelas adalah agar mengetahui kelas tersebut memiliki batasan dengan kelas apa saja.

Pembeda dari *hard constraint* dan *soft constraint* adalah adanya attribut *penalty* untuk batasan lunak dan attribut *required* dengan parameter *true* untuk batasan keras. Di antara beberapa batasan terdapat batasan yang memiliki parameter untuk nantinya menjadi parameter apakah batasan tersebut memenuhi atau tidak.

Adapun kode program untuk menyimpan data batasan pada objek *constraints* ditujukan oleh Kode 5.9.

```
for (int distri = 0; distri < distriL.getLength(); distri++) {
    required = 0;
    penalty = 10000;
    int angka1 = -1, angka2 = -1;
    Node distribNode = distriL.item(distri);
    Element disElement = (Element) distribNode;
    NodeList disCild = disElement.getElementsByTagName("class");
    ArrayList<Integer> temp = new ArrayList<>();
    type = disElement.getAttribute("type");

    if (disElement.hasAttribute("required")) {
        required = 1;
    } else {
        penalty = Integer.parseInt(disElement.getAttribute("penalty"));
    }

    typeS = type.split("[),(]");
    if (typeS.length == 2) {
        angka1 = Integer.parseInt(typeS[1]);
    } else if (typeS.length == 3) {
        angka2 = Integer.parseInt(typeS[2]);
    }

    for (int cons = 0; cons < disCild.getLength(); cons++) {
        Node consNode = disCild.item(cons);
        Element consElement = (Element) consNode;
        classDis = Integer.parseInt(consElement.getAttribute("id"));
        temp.add(classDis);
    }
    Constraint cns = new Constraint(type, angka1, angka2, penalty, temp);
```

Kode 5.9 Penyimpanan Batasan pada Objek *Constraint*

Selanjutnya adalah menyimpan batasan lunak dan batasan keras kedalam setiap objek Kelas yang ditunjukkan pada Kode 5.10.

```
if (required == 1) {
    distrib.addHard(cns);
    for (int i = 0; i < temp.size(); i++) {
        Kelas kelas = listKelas.get(temp.get(i) - 1);
          System.out.println(kelas.getClassID());
        kelas.addclasHC(cns);
    }
} else {
    distrib.addSoft(cns);
    for (int i = 0; i < temp.size(); i++) {
        Kelas kelas = listKelas.get(temp.get(i) - 1);
          System.out.println(kelas.getClassID());
        kelas.addclasSC(cns);
    }
}
```

Kode 5.10 Penyimpanan Batasan pada Objek Kelas

5.1.5. Pembuatan Initial Solution

Initial Solution adalah solusi awal yang digunakan sebagai langkah awal dalam optimasi. Solusi awal ini berisi jadwal sebelum dilakukannya optimasi dengan menggunakan algoritma.

Dalam pembuatan solusi awal, algoritma yang diterapkan adalah Greedy Algorithm dan juga dengan menambah fungsi *Backtrack*, dimana urutan pertama dalam sebuah daftar waktu dan ruang yang tersedia pada salah satu mata kuliah dipilih pada urutan yang pertama, jika memenuhi maka dimasukkan pada slot yang tersedia tetapi jika dalam daftar ruang dan waktu tidak ada yang memenuhi maka fungsi *backtrack* berjalan untuk kembali ke urutan sebelumnya dan mencari pilihan ruang dan waktu yang lain, sedemikian hingga seluruh mata kuliah terjadwal.

Sehingga *soft constraint* diabaikan dan hanya memperhatikan *hard constraint* atau batasan keras. Jika mata kuliah telah semua terjadwal pada Objek Kelas dan telah lolos *hard constraint*, maka *intial solution* telah terbentuk.

Langkah awal untuk membuat solusi awal adalah dengan melakukan berbagai pembuatan objek dan pembuatan setter getter pada objek tersebut sehingga bisa diambil untuk melakukan pembuatan *initial solution.*

Setelah dilakukan pembuatan beberapa objek, kemudian dilakukan pembuatan method yang digunakan untuk mengecek *timeslot* yang tersedia. Method ini mengambil nilai dari daftar ruang dan waktu terpilih yang dimiliki oleh setiap kelas.

Sementara keluaran yang dihasilkan adalah *true* atau *false* yang biasa disebut dengan tipe *boolean*.

Jika indeks *timeslot* yang berada pada ruang dan waktu tersebut sedang digunakan sebagai *timeslot* kelas lain, maka method akan menghasilkan nilai *false*. Untuk lebih lengkapnya, method ditujukan pada Kode 5.11.

```
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;
                        }
                    }
                }
            }
        }
    }
    return true;

}
```

Kode 5.11 Method Pengecekan Timeslot

Setelah membuat *method* pengecekan *timeslot*, kemudian dilanjutkan dengan pembuatan *method* pengecekan *unavailable room*, method ini berfungsi untuk mengecek apakah ruang yang akan di pilih termasuk kedalam daftar ruang yang tidak bisa digunakan atau tidak. *Method unavailable room* selengkapnya ditunjukkan oleh Kode 5.12.

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

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

Kode 5.12 Method Pengecekan Unavailable *Room*

Jika mata kuliah tersebut telah mendapatkan slot, maka objek *timeslot* dan *room* pada Kelas akan terupdate menjadi sesuai dengan urutan pada daftar ruang dan waktu yang terpilih.

Untuk memastikan bahwa semua mata kuliah telah terjadwal, dibuat sebuah *method* menggunakan struktur data ArrayList seperti yang ditunjukkan oleh Kode 5.13.

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

Kode 5.13 Method Pengecekan Semua Mata Kuliah Terjadwal

5.1.6. Pembuatan Kode Program *Hard Constraint*

Pembuatan kode program *hard constraint* sesuai dengan model matematis *constraint* yang telah dijelaskan pada subbab 4.2 dimana terdapat 19

constraint yang kemudian akan diterjemahkan dalam kode program atau bahasa pemrograman Java. Pengecekan *hard constraint* membutuhkan inisiasi atau masukan yaitu *index Timeslot*, *index Room*, *index* Kelas, ArrayList daftar kelas, dan matriks *travel room*, yang nantinya masukan tersebut digunakan oleh setiap *constraint* yang tersedia. Kode dari pengambilan masukan untuk setiap *constraint* ditunjukkan pada Kode 5.14.

```
boolean HardConstraint(int TS, int Rom, Kelas kls,
        ArrayList<Kelas> listKls, int[][] travel) {
    this.listKelas = listKls;
    ArrayList<Constraint> HardCons = kls.getClasHC();
    boolean hc = false;
    if (HardCons.size() == 1) {
        return true;
    }
    for (int i = 0; i < HardCons.size(); i++) {
        Constraint Hard = HardCons.get(i);
        ArrayList<Integer> KlsCons = Hard.getConsClass();
        String type = Hard.getType();
        int angka1 = Hard.getAngka1();
        int angka2 = Hard.getAngka2();
        if (Cons1(type, angka1, angka2, TS, Rom, kls, KlsCons)
                || Cons2(type, TS, Rom, kls, KlsCons, travel)) {
            hc = true;
        } else {
            return false;
        }
    }
    return hc;
}
```

Kode 5.14 Pengambilan Data *Hard Constraint*

5.1.6.1. *Constraint Same Start*

Contraint pertama yaitu *same start* berhubungan dengan awal waktu dilaksanakan kegiatan mata kuliah. Kode program *hard contraint same start* ditunjukkan pada Kode 5.15.

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

Kode 5.15 *Hard Constraint* Same Start

5.1.6.2. *Constraint Same Time*

Hard constraint kedua merupakan pengecekan terhadap waktu kegiatan mata kuliah dan diharuskan memiliki waktu yang sama atau bersinggungan. Method *hard constraint* kedua selengkapnya dapat dilihat pada Kode 5.16.

```
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;
    int idA = kls.getClassID();
    boolean ST = false;
    for (int i = 0; i < listST.size(); i++) {
```

```
        Kelas klsB = SearchKls(listST.get(i));
        if (idA != klsB.getClassID()) {
            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;
            }
        } else {
            ST = true;
        }
    }
    return ST;
}
```

Kode 5.16 *Hard Constraint* Same Time

5.1.6.3. *Constraint Different Time*

Method hard constraint Different Time merupakan kebalikan dari *constraint Same Time*. Jika *same time* mengharuskan kelas dimulai di waktu yang sama maka di *different time* kelas harus dimulai di waktu yang berbeda.

Adapun kode program untuk *hard constraint* ketiga ditunjukkan pada Kode 5.17.

```
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;
    int idA = kls.getClassID();
    boolean DT = false;
    for (int i = 0; i < listDT.size(); i++) {
```

```
        Kelas klsB = SearchKls(listDT.get(i));
        if (idA != klsB.getClassID()) {
            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) {
                    DT = true;
                } else {
                    return false;
                }
            } else if (iTS == -1) {
                DT = true;
            }
        } else {
            DT = true;
        }
    }
    return DT;
}
```

Kode 5.17 *Hard Constraint* Different Time

5.1.6.4. *Constraint Same Days*

Constraint ke empat merupakan batasan yang juga melihat waktu berlangsungnya mata kuliah, pada *constraint* ini membatasi bahwa beberapa mata kuliah harus dilaksanakan di hari yang sama dalam seminggu.

Beberapa mata kuliah dikatakan sudah memenuhi batasan ini jika ada setidaknya satu hari setiap minggu dimana kelas tersebut dilakukan di hari yang sama.Kode program *constraint* ini ditunjukkan pada Kode 5.18.

```
boolean SameDays(int TS, Kelas kls, ArrayList<Integer> listSD) {
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean SD = false;
    int idA = kls.getClassID();
    for (int i = 0; i < listSD.size(); i++) {
        Kelas klsB = SearchKls(listSD.get(i));
        if (idA != klsB.getClassID()) {
            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;
            }
        }
        SD = true;
    }
    return SD;
}
```

Kode 5.18 *Hard Constraint* Same Days

5.1.6.5. *Constraint Different Days*

Selanjutnya adalah *constraint* atau batasan yang berkebalikan dari batasan sebelumnya yaitu *Same Days*, di batasan *Different Days* mata kuliah yang terdapat dalam batasan tidak boleh sama sekali memiliki hari yang sama antar kelas. Kode batasan ini dapat dilihat pada Kode 5.19.

```
boolean DifferentDays(int TS, Kelas kls, ArrayList<Integer> listDD) {
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean DD = false;
    int idA = kls.getClassID();
    for (int i = 0; i < listDD.size(); i++) {
        Kelas klsB = SearchKls(listDD.get(i));
        if (idA != klsB.getClassID()) {
            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)) {
                        DD = true;
                    } else {
                        DD = false;
                        break;
                    }
                }
                if (DD == false) {
                    return false;
                }
            } else if (iTS == -1) {
                DD = true;
            }
        } else {
            DD = true;
        }
    }
    return DD;
}
```

Kode 5.19 *Hard Constraint* Different Days

5.1.6.6. *Constraint Same Weeks*

Hampir sama dengan batasan *Same Days*, disini kelas harus memiliki setidaknya satu minggu dalam satu semester yang sama dengan kelas lain dalam daftar kelas di batasan. Kode method *Same Weeks* ditunjukkan pada Kode 5.20.

```
boolean SameWeeks(int TS, Kelas kls, ArrayList<Integer> listSW) {
    boolean SW = false;
    ArrayList<Integer> weekA = kls.getAvailableTS().get(TS).getWeeks();
    int idA = kls.getClassID();
    for (int i = 0; i < listSW.size(); i++) {
        Kelas klsB = SearchKls(listSW.get(i));
        if (idA != klsB.getClassID()) {
            int iTS = klsB.getTs();
            if (iTS != -1) {
                for (int k = 0; k < weekA.size(); k++) {
```

```
                    ArrayList<Integer> weekB = klsB.getAvailableTS().get(iTS).getWeeks();
                    if (weekA.get(k) == 1 && weekB.get(k) == 1) {
                        SW = true;
                        break;
                    } else {
                        SW = false;
                    }
                }
                if (SW == false) {
                    return false;
                }
            } else if (iTS == -1) {
                SW = true;
            }
        } else {
            SW = true;
        }
    }
    return SW;
}
```

Kode 5.20 *Hard Constraint* Same Weeks

5.1.6.7. *Constraint Different Weeks*

Constraint selanjutnya yang dibuat sebuah *method* adalah *different weeks*, method ini sama dengan method *different days* bedanya hanya terdapat pada hari dan minggu. Kode method ini dapat dilihat pada Kode 5.21.

```
boolean DifferentWeeks(int TS, Kelas kls, ArrayList<Integer> listDW) {
    boolean DW = false;
    ArrayList<Integer> weekA = kls.getAvailableTS().get(TS).getWeeks();
    int idA = kls.getClassID();
    for (int i = 0; i < listDW.size(); i++) {
        Kelas klsB = SearchKls(listDW.get(i));
        if (idA != klsB.getClassID()) {
            int iTS = klsB.getTs();
            if (iTS != -1) {
                for (int k = 0; k < weekA.size(); k++) {
                    ArrayList<Integer> weekB = klsB.getAvailableTS().get(iTS).getWeeks();
                    if (weekA.get(k) != 1 || weekB.get(k) != 1) {
                        DW = true;
                    } else {
                        DW = false;
                        break;
                    }
                }
```

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

Kode 5.21 *Hard Constraint* Different Weeks

5.1.6.8. *Constraint Same Room*

Batasan ini membutuhkan *index* ruang dari masing-masing kelas, batasan *same room* mengharuskan kelas mempunyai ruang yang sama dengan kelas lain dalam batasan. Kode 5.22 merupakan kode program dari batasan ini.

```
boolean SameRoom(int Rom, Kelas kls, ArrayList<Integer> listSR) {
    boolean SR = false;
    int roomA = kls.getAvailableroom().get(Rom).getRoom_id();
    int idA = kls.getClassID();
    for (int i = 0; i < listSR.size(); i++) {
        Kelas klsB = SearchKls(listSR.get(i));
        if (idA != klsB.getClassID()) {
            int iRM = klsB.getRoom();
            if (iRM != -1) {
                int roomB = klsB.getAvailableroom().get(iRM).getRoom_id();
                if (roomA == roomB) {
                    SR = true;
                } else {
                    return false;
                }
            } else if (iRM == -1) {
                SR = true;
            }
        } else {
            SR = true;
        }
    }
    return SR;
}
```

Kode 5.22 *Hard Constraint* Same *Room*

5.1.6.9. *Constraint Different Room*

Batasan ini adalah kebalikan dari batasan *same room*, dimana batasan ini mengharuskan kelas tidak boleh mempunyai ruang yang sama dengan kelas lain. Adapun kode program dari *method* ini ditunjukkan pada Kode 5.23.

```
boolean DifferentRoom(int Rom, Kelas kls, ArrayList<Integer> listDR) {
    boolean DR = false;
    int roomA = kls.getAvailableroom().get(Rom).getRoom_id();
    int idA = kls.getClassID();
    for (int i = 0; i < listDR.size(); i++) {
        Kelas klsB = SearchKls(listDR.get(i));
        if (idA != klsB.getClassID()) {
            int iRM = klsB.getRoom();
            if (iRM != -1) {
                int roomB = klsB.getAvailableroom().get(iRM).getRoom_id();
                if (roomA != roomB) {
                    DR = true;
                } else {
                    return false;
                }
            } else if (iRM == -1) {
                DR = true;
            }
        } else {
            DR = true;
        }
    }
    return DR;
}
```

Kode 5.23 *Hard Constraint* Different *Room*

5.1.6.10. *Constraint Overlap*

Batasan selanjutnya adalah batasan yang membatasi waktu dilakukan nya kelas, hampir sama dengan *same time* yang mengharuskan agar waktu kelas harus bersinggungan dengan kelas lain. Batasan ini membutuhkan *index timeslot*, *index* kelas dan daftar kelas dalam batasan. Adapun kode program ditunjukkan pada Kode 5.24.

```
boolean Overlap(int TS, Kelas kls, ArrayList<Integer> listOv) {
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;
    int idA = kls.getClassID();
    ArrayList<Integer> weeksA = kls.getAvailableTS().get(TS).getWeeks();
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean O = false;

    for (int i = 0; i < listOv.size(); i++) {
        Kelas klsB = SearchKls(listOv.get(i));
        if (idA != klsB.getClassID()) {
            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;
                ArrayList<Integer> weeksB = klsB.getAvailableTS().get(iTS).getWeeks();
                ArrayList<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();
                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) {
                                    O = true;
                                } else {
                                    return false;
                                }
                            }
                        } else {
                            return false;
                        }
                    }
                } else {
                    return false;
                }
            } else if (iTS == -1) {
                O = true;
            }
        } else {
            O = true;
        }
    }
    return O;
}
```

Kode 5.24 *Hard Constraint* Overlap

5.1.6.11. *Constraint Not Overlap*

Batasan selanjutnya merupakan kebalikan dari batasan sebelumnya, yaitu dibatasan ini mengharuskan kelas tidak boleh dilakukan bersamaan dengan kelas lain. *Mehod* ini membutuhkan masukan yang sama dengan *method* batasan sebelumnya. Kode program dapat dilihat pada Kode 5.25.

```
boolean NotOverlap(int TS, Kelas kls, ArrayList<Integer> listNotOv) {
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;
    int idA = kls.getClassID();
    ArrayList<Integer> weeksA = kls.getAvailableTS().get(TS).getWeeks();
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean NO = false;

    for (int i = 0; i < listNotOv.size(); i++) {
        Kelas klsB = SearchKls(listNotOv.get(i));
        if (idA != klsB.getClassID()) {
            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;
                ArrayList<Integer> weeksB = klsB.getAvailableTS().get(iTS).getWeeks();
                ArrayList<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();

                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;
                                }
                            } else {
                                NO = true;
```

```
                                }
                            }
                        } else {
                            NO = true;
                        }
                    }
                } else if (iTS == -1) {
                    NO = true;
                }
            } else {
                NO = true;
            }
        }
        return NO;
}
```

Kode 5.25 *Hard Constraint* Not Overlap

5.1.6.12. *Constraint Same Attendees*

Batasan selanjutnya adalah batasan yang memastikan kelas bisa dilaksanakan tepat waktu sehingga tidak ada kelas yang terlewat karena jarak antara kelas terlalu jauh. Batasan ini membutuhkan beberapa masukan yaitu *index timeslot*, *index* kelas, daftar kelas dalam batasan, ditambah dengan matriks *travel room*. Kode program dari batasan ini dapat dilihat dalam Kode 5.26.

```
boolean SameAttendees(int TS, int Rom, Kelas kls, int[][] valueTravel,
        ArrayList<Integer> listSA) {
    int trvl = 0;
    boolean noRoomA = false;
    boolean noRoomB = false;
    int roomA = 0;
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;
    int idA = kls.getClassID();
    ArrayList<Integer> weeksA = kls.getAvailableTS().get(TS).getWeeks();
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean SA = false;

    if (Rom < 0) {
        noRoomA = true;
    } else {
        roomA = kls.getAvailableroom().get(Rom).getRoom_id() - 1;
    }
```

```
for (int i = 0; i < listSA.size(); i++) {
    Kelas klsB = SearchKls(listSA.get(i));
    if (idA != klsB.getClassID()) {
        if (idA != klsB.getClassID()) {
            int iTS = klsB.getTs();
            int iRM = klsB.getRoom();
            if (klsB.isHasRoom() == false) {
                iRM = 999;
            } else {
                iRM = klsB.getRoom();
            }

            if (iTS != -1 && iRM != -1) {
                int startB = klsB.getAvailableTS().get(iTS).getStart();
                int lengthB = klsB.getAvailableTS().get(iTS).getLength();
                int endB = startB + lengthB;
                ArrayList<Integer> weeksB = klsB.getAvailableTS().get(iTS).getWeeks();
                ArrayList<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();
                int roomB = 0;
                if (iRM == 999) {
                    noRoomB = true;
                } else {
                    roomB = klsB.getAvailableroom().get(iRM).getRoom_id() - 1;
                }
                outerloop:
                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;
                            }
                        }
                    } else {
                        SA = true;
                    }
                }
            } else if (iTS == -1) {
                SA = true;
            }
        }
    } else {
        SA = true;
    }
}
return SA;
```

Kode 5.26 *Hard Constraint* Same Attendees

5.1.6.13. *Constraint Precedence*

Batasan ke-13 adalah batasan yang mengharuskan kelas dalam batasan tersebut sesuai dengan urutan kelas yang terdapat pada urutan di dalam batasan tersebut, yang berarti bahwa urutan pertama dalam batasan tersebut harus lebih dulu dijadwalkan dibandingkan dengan kelas pada urutan lain.

Dalam batasan ini masukan masih sama dengan batasan lain yaitu *index timeslot*, *index* kelas, dan list kelas dalam batasan. Tetapi perbedaan yang terdapat dalam batasan ini adalah di dalam *method* dilakukan pengecekan apakah *index timeslot kelas* yang dimasukkan merupakan urutan pertama atau tidak.

Adapun kode program dapat dilihat dalam Kode 5.27.

```
boolean Precedence(int TS, Kelas kls, ArrayList<Integer> listP) {
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;
    ArrayList<Integer> weeksA = kls.getAvailableTS().get(TS).getWeeks();
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean P = false;
    int klsA = listP.indexOf(kls.getClassID());
    int idA = kls.getClassID();

    for (int i = 0; i < listP.size(); i++) {
        if (i < klsA) {
            Kelas klsB = SearchKls(listP.get(i));
            if (idA != klsB.getClassID()) {
                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;
                    ArrayList<Integer> weeksB = klsB.getAvailableTS().get(iTS).getWeeks();
                    ArrayList<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();
```

```
                outerloop:
                for (int wk = 0; wk < weeksA.size(); wk++) {
                    if (weeksA.get(wk) < weeksB.get(wk)) {
                        P = true;
                        break outerloop;
                    } else if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
                        for (int dy = 0; dy < daysA.size(); dy++) {
                            if (daysA.get(dy) < daysB.get(dy)) {
                                P = true;
                                break outerloop;
                            } else if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
                                if ((endA >= startB)) {
                                    P = true;
                                    break outerloop;
                                } else {
                                    return false;
                                }
                            } else {
                                P = false;
                            }
                        }
                    } else {
                        P = false;
                    }
                }
            } else if (iTS == -1) {
                P = true;
            }
        } else {
            P = true;
        }
```

Kode 5.27 *Hard Constraint* Precedence

5.1.6.14. *Constraint Work Day*

Batasan selanjutnya adalah *work day*, batasan ini mengharuskan pasangan kelas yang dimana awal kelas pertama dan berakhirnya kelas terakhir tidak boleh melebihi *S timeslot.*

Batasan ini memiliki masukan yang berbeda dengan kelas lain, dimana banyak kelas lain hanya membutuhkan tiga masukan yaitu *index timeslot*, *index* kelas, dan daftar kelas, sedangkan di batasan ini ada tambahan masukan yaitu parameter *S* yang terdapat pada batasan *work day.* Adapun kode dari method ini dapat dilihat pada Kode 5.28.

```
boolean Workday(int TS, int S, Kelas kls, ArrayList<Integer> listWD) {
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;
    int idA = kls.getClassID();
    ArrayList<Integer> weeksA = kls.getAvailableTS().get(TS).getWeeks();
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean WD = false;

    for (int i = 0; i < listWD.size(); i++) {
        Kelas klsB = SearchKls(listWD.get(i));
        if (idA != klsB.getClassID()) {
            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;
                ArrayList<Integer> weeksB = klsB.getAvailableTS().get(iTS).getWeeks();
                ArrayList<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();

                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 ((Math.max(endA, endB)) - (Math.min(startA,
                                        startB)) <= S) {
                                    WD = true;
                                    break outerloop;
                                } else {
                                    return false;
                                }
                            } else {
                                WD = true;
                            }
                        }
                    } else {
                        WD = true;
                    }
                }
                if (WD == false) {
                    return false;
                }
            } else if (iTS == -1) {
                WD = true;
            }
        } else {
            WD = true;
        }
    }
    return WD;
}
```

Kode 5.28 *Hard Constraint* Work Day

5.1.6.15. *Constraint Min Gap*

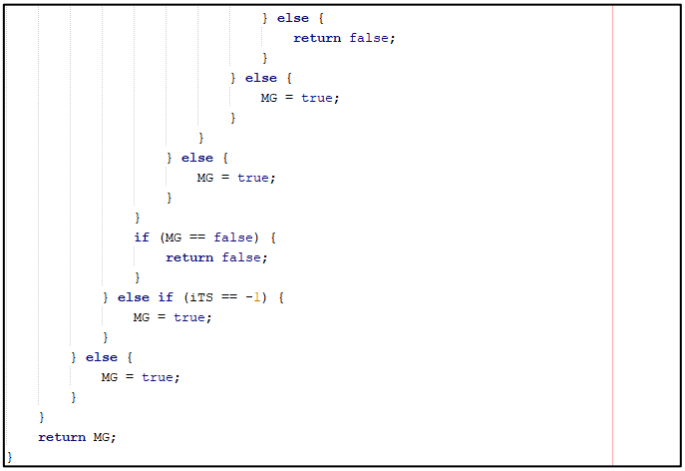
Batasan 15 atau *min gap* ini mengharuskan pasangan kelas yang terdapat dalam daftar batasan harus

mempunyai gap atau batas selama *G timeslot* antar kelas.

Masukan dalam batasan ini sama dengan masukan pada batasan *work day*, dimana yang berbeda adalah pada batasan *work day* menggunakan masukan *S timeslot*, sedangkan pada batasan ini membutuhkan masukan *G timeslot*. Kode *method* ini ditunjukkan pada Kode 5.29.

```
boolean Mingap(int TS, int G, Kelas kls, ArrayList<Integer> listMG) {
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;
    int idA = kls.getClassID();
    ArrayList<Integer> weeksA = kls.getAvailableTS().get(TS).getWeeks();
    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    boolean MG = false;

    for (int i = 0; i < listMG.size(); i++) {
        Kelas klsB = SearchKls(listMG.get(i));
        if (idA != klsB.getClassID()) {
            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;
                ArrayList<Integer> weeksB = klsB.getAvailableTS().get(iTS).getWeeks();
                ArrayList<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();
                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)) {
                                    MG = true;
                                    break outerloop;
```



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

Kode 5.29 *Hard Constraint* Min Gap

5.1.6.16. *Constraint Max Days*

Batasan selanjutnya membatasi berapa hari kelas tersebut dilaksanakan dalam satu minggu, pelaksanaan kelas tersebut tidak boleh melebihi dari *D* hari selama seminggu.

Masukan yang dibutuhkan dari *method* batasan ini adalah *index timeslot, index* kelas, dan parameter *D.* Kode program dapat dilihat pada Kode 5.30.

Pada method ini terdapat fungsi lain yaitu *countNonzeroBits* dimana di dalam fungsi tersebut menghitung berapa hari dalam seminggu kelas tersebut dilaksanakan, sehingga dibutuhkan masukan berupa Arraylist hari pada kelas tersebut. Kode dari fungsi tersebut dapat dilihat pada Kode 5.31.

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

Kode 5.30 Kode Program *Hard Constraint* Max Days

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

Kode 5.31 Fungsi Count Non Zero Bits

5.1.6.17. *Constraint Max Day Load*

Batasan ini membatasi jumlah waktu total yang ditetapkan untuk sekumpulan kelas tidak lebih dari *S timeslot* setiap hari. Kode program dapat dilihat pada Kode 5.32 dan fungsi *day load* pada Kode 5.33.

```
boolean MaxdayLoad(int TS, int S, Kelas kls
) {
    int length = kls.getAvailableTS().get(TS).getLength();
    ArrayList<Integer> weeks = kls.getAvailableTS().get(TS).getWeeks();
    ArrayList<Integer> days = kls.getAvailableTS().get(TS).getDays();
    boolean MDL = false;

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

Kode 5.32 *Hard Constraint MaxDayLoad*

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

Kode 5.33 Fungsi *DayLoad*

5.1.6.18. *Constraint Max Breaks*

Batasan *max breaks* membatasi jumlah jeda atau istirahat antar kelas selama setiap hari yang melebihi *S timeslot* dan tidak boleh mempunyai lebih dari *R* jeda perhari. Masukan untuk kode program ini adalah *arraylist* daftar kelas dalam batasan, nilai parameter *R* dan *S.* Adapun kode program ditunjukkan pada Kode

5.34 beserta dengan fungsi *MergeBlocks* pada Kode 5.35.

```
boolean Maxbreaks(int TS, int R, int S, Kelas kls,
        ArrayList<Integer> listMB
) {
    boolean MB = false;
    if (MergeBlocks(TS, S, kls, listMB) <= R + 1) {
        MB = true;
    } else {
        return false;
    }
    return MB;
}
```

Kode 5.34 ***Hard Constraint MaxBreaks***

```
private int MergeBlocks(int TS, int S, Kelas kls, ArrayList<Integer> listMB) {
    int initialBlock = 1;

    ArrayList<Integer> dayA = kls.getAvailableTS().get(TS).getDays();
    ArrayList<Integer> weekA = kls.getAvailableTS().get(TS).getWeeks();
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lenA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lenA;

    for (int i = 1; i < listMB.size(); i++) {
        Kelas klsB = SearchKls(listMB.get(i));
        int iTS = klsB.getTs();
        if (iTS != -1) {
            ArrayList<Integer> dayB = klsB.getAvailableTS().get(iTS).getDays();
            ArrayList<Integer> weekB = klsB.getAvailableTS().get(iTS).getWeeks();
            int startB = klsB.getAvailableTS().get(iTS).getStart();
            int lenB = klsB.getAvailableTS().get(iTS).getLength();
            int endB = startB + lenB;

            outerloop:
            for (int wk = 0; wk < weekA.size(); wk++) {
                if (weekA.get(wk) == 1 && weekB.get(wk) == 1) {
                    for (int ds = 0; ds < dayA.size(); ds++) {
                        if (dayA.get(ds) == 1 && dayB.get(ds) == 1) {
                            if (endA + S >= startB && endB + S >= startA) {
                                startA = Math.min(startA, startB);
                                endA = Math.max(endA, endB);
                            } else {
                                initialBlock++;
```

```
                                startA = startB;
                                endA = endB;
                                dayA = dayB;
                                weekA = weekB;
                                break outerloop;
                            }
                        }
                    }
                }
            }
        } else if (iTS == -1) {
            initialBlock += 0;
        }
    }
    return initialBlock;
}
```

Kode 5.35 Fungsi *MergeBlocks* Batasan *MaxBreaks*

Fungsi *MergeBlocks* bertujuan untuk menggabungkan antar kelas jika memenuhi kriteria bahwa pasangan kelas tersebut tidak mempunyai jeda.

5.1.6.19. *Constraint Max Blocks*

Hampir sama dengan batasan *max breaks*, batasan ini membatasi agar panjang *block* atau blok kelas (kelas yang digabungkan) tidak lebih dari *M* slot blok. Kode program batasan ini ditunjukkan pada Kode 5.36. dan 5.37.

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

    return MB;
}
```

Kode 5.36 *Hard Constraint MaxBlocks*

```
private int MergeBlockB(int TS, int M, int S, Kelas kls, ArrayList<Integer> listMBL) {

    int lengthBlock = 0;
    int lebih = 0;
    boolean blockBaru = true;

    ArrayList<Integer> daysA = kls.getAvailableTS().get(TS).getDays();
    ArrayList<Integer> weeksA = kls.getAvailableTS().get(TS).getWeeks();
    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;

    for (int i = 0; i < listMBL.size(); i++) {
        Kelas klsB = SearchKls(listMBL.get(i));
        int iTS = klsB.getTs();
        if (iTS != -1) {

            ArrayList<Integer> daysB = klsB.getAvailableTS().get(iTS).getDays();
            ArrayList<Integer> weeksB = klsB.getAvailableTS().get(iTS).getWeeks();
            int startB = klsB.getAvailableTS().get(iTS).getStart();
            int lengthB = klsB.getAvailableTS().get(iTS).getLength();
            int endB = startB + lengthB;

            outerloop:
            for (int wk = 0; wk < weeksA.size(); wk++) {
                for (int j = 0; j < daysA.size(); j++) {
                    if (weeksA.get(wk) == 1 && daysA.get(j) == 1 && weeksB.get(wk) == 1
                            && daysB.get(j) == 1) {
                        if (endA + S >= startB && endB + S >= startA) {
                            startA = Math.min(startA, startB);
                            endA = Math.max(endA, endB);
                                lengthBlock = endA - startA;
                                if (lengthBlock > M && blockBaru == true) {
                                    lebih += 1;
                                    blockBaru = false;
                                } else {
                                    lebih += 0;
                                }
                            } else {
                                lengthBlock = endA - startA;
                                if (lengthBlock > M && blockBaru == true) {
                                    lebih += 1;
                                } else {
                                    lebih += 0;
                                }
                                blockBaru = true;
                                startA = startB;
                                endA = endB;
                                daysA = daysB;
                                weeksA = weeksB;
                                break outerloop;
                            }
                        }
                    }
                }
            } else if (iTS == -1) {
                lengthBlock += 0;
            }
        }
        return lengthBlock;
    }
```

Kode 5.37 Fungsi *MergeBlocks Batasan MaxBlocks*

5.1.7. Pembuatan Kode Program *Soft constraint*

Pembuatan kode program *soft constraint* akan didasarkan sesuai dengan model matematis yang sudah dibuat, dimana terdapat 19 *constraint* yang nantinya kode program tersebut akan menghitung jumlah nilai penalti yang akan menjadi hasil perbandingan dari proses optimasi. Total nilai penalti dari *soft constraint* akan dihitung berdasarkan penalty dari masing-masing batasan yang melanggar dan kemudian dikalikan dengan bobot yang sudah di inisiasi di awal.

Pengecekan penalti *method soft constraint* membutuhkan masukan yaitu berupa ArrayList daftar kelas, matriks *travel room*, dan daftar batasan lunak. Kode masukan *soft constraint* ditunjukkan pada Kode 5.38.

```
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 5.38 Masukan *Soft constraint*

Setiap *method* batasan memerlukan daftar kelas yang mempunyai batasan tersebut dan juga memerlukan masukan berupa penalti yang terdapat pada setiap

batasan. Untuk kode masing masing batasan dapat dilihat sebagai berikut.

5.1.7.1. Constraint Same Start

Batasan lunak *same start* memiliki kode yang hampir sama dengan *hard constraint* batasan tersebut, perbedaannya terletak pada masukan dan juga keluaran dari *method* tersebut. Batasan ini dikatakan melanggar jika terjadi pasangan kelas yang tidak sesuai dengan kondisi penerimaan sehingga jumlah penalti akan di tambah dengan penalty yang telah di inisiasi. Kode batasan ini dapat dilihat pada Kode 5.39.

```
int SameStart(ArrayList<Integer> listSS, int penalty) {
    int startA, startB;
    int TSA, TSB;
    int jPenalty = 0;
    if (listSS.size() == 1) {
        return jPenalty;
    }
    for (int i = 0; i < listSS.size(); i++) {
        Kelas klsA = SearchKls(listSS.get(i));
        TSA = klsA.getTs();
        startA = klsA.getAvailableTS().get(TSA).getStart();
        for (int j = i + 1; j < listSS.size(); j++) {
            Kelas klsB = SearchKls(listSS.get(j));
            TSB = klsB.getTs();
            startB = klsB.getAvailableTS().get(TSB).getStart();
            if (startA == startB) {
                jPenalty += 0;
            } else {
                jPenalty += penalty;
            }
        }
    }
    return jPenalty;
}
```

Kode 5.39 *Soft Constraint* Same Start

5.1.7.2. *Constraint Same Time*

Masukan dalam batasan ini sama dengan masukan dari fungsi batasan lain yaitu, daftar kelas pada batasan ini dan penalti dari batasan *same time*. Jumlah penalti akan bertambah apabila terdapat pasangan kelas yang tidak

memenuhi kriteria penerimaan. Kode batasan ini ditunjukkan pada Kode 5.40.

```
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;
        int idA = klsA.getClassID();
        for (int j = i + 1; j < listST.size(); j++) {
            Kelas klsB = SearchKls(listST.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                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 5.40 *Soft Constraint* Same Time

5.1.7.3. *Constraint Different Time*

Batasan ini sama dengan batasan sebelumnya yaitu *same time*, perbedaan terletak pada *acceptance criteria* dari batasan tersebut. Jumlah penalti bertambah jika terdapat pasangan kelas yang tidak memenuhi kriteria dari batasan ini. Kode program batasan ini dapat dilihat pada Kode 5.41.

```
int DifferentTime(ArrayList<Integer> listDT, int penalty) {
    int startA, startB;
    int lengthA, lengthB;
    int endA, endB;
    int TSA, TSB;
    int jPenalty = 0;
    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;
        int idA = klsA.getClassID();
        for (int j = i + 1; j < listDT.size(); j++) {
            Kelas klsB = SearchKls(listDT.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                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 5.41 *Soft Constraint* Different Time

5.1.7.4. *Constraint Same Days*

Batasan *same days* melihat pasangan kelas yang mempunyai hari yang sama dalam melakukan kegiatan pembelajaran, jumlah penalty dari batasan ini bertambah jika terdapat pasangan kelas yang melanggar kriteria penerimaan yaitu pasangan kelas harus berada pada hari yang sama. Kode program ditunjukkan pada Kode 5.42.

```
int SameDays(ArrayList<Integer> listSD, int penalty) {
    int TSA, TSB, jPenalty = 0, salah = 0;
    for (int i = 0; i < listSD.size(); i++) {
        Kelas klsA = SearchKls(listSD.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        for (int j = i + 1; j < listSD.size(); j++) {
            Kelas klsB = SearchKls(listSD.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                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 {
                        salah += 1;
                    }
                    if (salah == daysA.size()) {
                        jPenalty += penalty;
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.42 *Soft Constraint* Same Days

5.1.7.5. *Constraint Different Days*

Batasan ini berlawanan dengan batasan *same days*, *acceptance criteria* dari batasan ini adalah pasangan kelas harus dijadwalkan pada hari yang berbeda dalam seminggu, jika terjadi pelanggaran maka jumlah penalti akan ditambahkan sesuai berapa penalti yang ditentukan oleh batasan, Kode fungsi batasan dapat dilihat pada Kode 5.43.

```
int DifferentDays(ArrayList<Integer> listDD, int penalty) {
    int TSA, TSB;
    int jPenalty = 0;

    for (int i = 0; i < listDD.size(); i++) {
        Kelas klsA = SearchKls(listDD.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        for (int j = i + 1; j < listDD.size(); j++) {
            Kelas klsB = SearchKls(listDD.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                daysB = klsB.getAvailableTS().get(TSB).getDays();
                for (int dy = 0; dy < daysA.size(); dy++) {
                    if ((daysA.get(dy) == 0 && daysB.get(dy) == 0)
                            || (daysA.get(dy) == 1 && daysB.get(dy) == 0)
                            || (daysA.get(dy) == 0 && daysB.get(dy) == 1)) {
                        jPenalty += 0;
                    } else {
                        jPenalty += penalty;
                        break;
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.43 *Soft Constraint* Different Days

5.1.7.6. *Constraint Same Weeks*

Batasan *same weeks* selaras dengan batasan *same days*, perbedaannya jika batasan *same days* melihat hari kelas tersebut dijadwalkan sedangkan *same weeks* melihat minggu dimana kelas tersebut dijadwalkan. Kode program ini dapat dilihat pada Kode 5.44.

```
int SameWeeks(ArrayList<Integer> listSW, int penalty) {
    int TSA, TSB;
    int jPenalty = 0, salah = 0;
    for (int i = 0; i < listSW.size(); i++) {
        Kelas klsA = SearchKls(listSW.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();
        for (int j = i + 1; j < listSW.size(); j++) {
            Kelas klsB = SearchKls(listSW.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                for (int wk = 0; wk < weeksA.size(); wk++) {
                    if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
                        jPenalty += 0;
                        break;
                    } else {
                        salah += 1;
                    }
                    if (salah == weeksA.size()) {
                        jPenalty += penalty;
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.44 *Soft Constraint* Same Weeks

5.1.7.7. *Constraint Different Weeks*

Batasan ini bertentangan dengan batasan *same weeks*. Jumlah penalty pada batasan ini akan bertambah jika terdapat pasangan kelas yang dijadwalkan pada minggu yang sama. Kode dari batasan ini dapat dilihat pada Kode 5.45.

```
int DifferentWeeks(ArrayList<Integer> listDW, int penalty) {
    int TSA, TSB;
    int jPenalty = 0;
    for (int i = 0; i < listDW.size(); i++) {
        Kelas klsA = SearchKls(listDW.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();

        for (int j = i + 1; j < listDW.size(); j++) {
            Kelas klsB = SearchKls(listDW.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                for (int wk = 0; wk < weeksA.size(); wk++) {
                    if ((weeksA.get(wk) == 0 && weeksB.get(wk) == 0)
                            || (weeksA.get(wk) == 1 && weeksB.get(wk) == 0)
                            || (weeksA.get(wk) == 0 && weeksB.get(wk) == 1)) {
                        jPenalty += 0;
                    } else {
                        jPenalty += penalty;
                        break;
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.45 *Soft Constraint* Different Weeks

5.1.7.8. *Constraint Same Room*

Jumlah penalti pada batasan ini bertambah apabila terdapat pasangan kelas yang dijadwalkan pada ruang yang sama. Pembuatan kode program *soft constraint* sama dengan kode program *hard constraint* hanya berbeda pada masukan dan keluaran. Kode program tersebut dapat dilihat pada Kode 5.46.

```
int SameRoom(ArrayList<Integer> listSR, int penalty) {
    int roomA, roomB;
    int RMA, RMB;
    int jPenalty = 0;

    for (int i = 0; i < listSR.size(); i++) {
        Kelas klsA = SearchKls(listSR.get(i));
        int idA = klsA.getClassID();
        RMA = klsA.getRoom();
        roomA = klsA.getAvailableroom().get(RMA).getRoom_id();
        for (int j = i + 1; j < listSR.size(); j++) {
            Kelas klsB = SearchKls(listSR.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                RMB = klsB.getRoom();
                roomB = klsB.getAvailableroom().get(RMB).getRoom_id();
                if (roomA == roomB) {
                    jPenalty += 0;
                } else {
                    jPenalty += penalty;
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.46 *Soft Constraint* Same *Room*

5.1.7.9. *Constraint Different Room*

Batasan ini merupakan antitetis dari batasan *same room*. Kriteria penerimaan dari fungsi ini adalah apabila ada pasangan kelas dijadwalkan pada ruang yang berbeda. Adapun kode dari batasan ini ditunjukkan pada Kode 5.47.

```
int DifferentRoom(ArrayList<Integer> listDR, int penalty) {
    int roomA, roomB;
    int RMA, RMB;
    int jPenalty = 0;

    for (int i = 0; i < listDR.size(); i++) {
        Kelas klsA = SearchKls(listDR.get(i));
        int idA = klsA.getClassID();
        RMA = klsA.getRoom();
        roomA = klsA.getAvailableroom().get(RMA).getRoom_id();
        for (int j = i + 1; j < 10; j++) {
            Kelas klsB = SearchKls(listDR.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                RMB = klsB.getRoom();
                roomB = klsB.getAvailableroom().get(RMB).getRoom_id();
                if (roomA != roomB) {
                    jPenalty += 0;
                } else {
                    jPenalty += penalty;
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.47 *Soft Constraint* Different *Room*

5.1.7.10. *Constraint Overlap*

Batasan *Overlap* membatasi pasangan kelas harus dijadwalkan pada waktu yang bersinggungan, batasan ini memberikan tambahan penalti jika terdapat pasangan kelas yang melanggar ketentuan tersebut. Adapun kode program ditunjukkan pada Kode 5.48.

```
int Overlap(ArrayList<Integer> listOv, int penalty) {
    int startA, startB, TSA, TSB, lengthA, lengthB, endA, endB;
    int jPenalty = 0, weekSalah = 0, daySalah = 0;
    for (int i = 0; i < listOv.size(); i++) {
        Kelas klsA = SearchKls(listOv.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        startA = klsA.getAvailableTS().get(TSA).getStart();
        lengthA = klsA.getAvailableTS().get(TSA).getLength();
        endA = startA + lengthA;
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        for (int j = i + 1; j < listOv.size(); j++) {
            Kelas klsB = SearchKls(listOv.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                startB = klsB.getAvailableTS().get(TSB).getStart();
                lengthB = klsB.getAvailableTS().get(TSB).getLength();
                endB = startB + lengthB;
                weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                daysB = klsB.getAvailableTS().get(TSB).getDays();
                if ((startB < endA) && (startA < endB)) {
                    outerloop:
                    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;
                                    }
                                }
                            }
                        } else {
                            weekSalah += 1;
                            if (weekSalah == weeksA.size()) {
                                jPenalty += penalty;
                            }
                        }
                    }
                } else {
                    jPenalty += penalty;
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.48 *Soft Constraint* Overlap

5.1.7.11. *Constraint Not Overlap*

Acceptance criteria dari batasan *not overlap* adalah mengatur pasangan kelas agar tidak dijadwalkan pada waktu yang sama. Sehingga kelas yang dijadwalkan pada minggu, hari, dan waktu yang berbeda lolos dari batasan ini. Kode program ini dapat dilihat pada Kode 5.49.

```
int NotOverlap(ArrayList<Integer> listNotOv, int penalty) {
    int startA, startB, lengthA, lengthB, TSA, TSB, endA, endB;
    int jPenalty = 0;

    for (int i = 0; i < listNotOv.size(); i++) {
        Kelas klsA = SearchKls(listNotOv.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        startA = klsA.getAvailableTS().get(TSA).getStart();
        lengthA = klsA.getAvailableTS().get(TSA).getLength();
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        endA = startA + lengthA;
        for (int j = i + 1; j < listNotOv.size(); j++) {
            Kelas klsB = SearchKls(listNotOv.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                startB = klsB.getAvailableTS().get(TSB).getStart();
                lengthB = klsB.getAvailableTS().get(TSB).getLength();
                endB = startB + lengthB;
                weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                daysB = klsB.getAvailableTS().get(TSB).getDays();
```

```
                outerloop:
                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;
                                }
                            } else {
                                jPenalty += 0;
                            }
                        }
                    } else {
                        jPenalty += 0;
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.49 *Soft Constraint* Not Overlap

5.1.7.12. *Constraint Same Attendees*

Batasan *same attendees* membatasi atau mengatur agar pasangan kelas yang terdapat pada daftar kelas di batasan bisa dilaksanakan tepat waktu, yang artinya waktu tempuh antara ruang kelas tidak boleh melebihi dari awal kelas selanjutnya. Kode program dari batasan ini dapat dilihat pada Kode 5.50.

```
int SameAttendees(int[][] valueTravel, ArrayList<Integer> listSA, int penalty) {
    int trvl = 0, roomA = 0, roomB = 0, jPenalty = 0;
    boolean noRoomA = false, noRoomB = false;
    int TSA, TSB, RMA, RMB, startA, startB, lengthA, lengthB, endA, endB;

    for (int i = 0; i < listSA.size(); i++) {
        Kelas klsA = SearchKls(listSA.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        RMA = klsA.getRoom();
        if (RMA < 0) {
            noRoomA = true;
        } else {
            roomA = klsA.getAvailableroom().get(RMA).getRoom_id() - 1;
        }
```

```
            startA = klsA.getAvailableTS().get(TSA).getStart();
            lengthA = klsA.getAvailableTS().get(TSA).getLength();
            endA = startA + lengthA;
            weeksA = klsA.getAvailableTS().get(TSA).getWeeks();
            daysA = klsA.getAvailableTS().get(TSA).getDays();
            for (int j = i + 1; j < listSA.size(); j++) {
                Kelas klsB = SearchKls(listSA.get(j));
                int idB = klsB.getClassID();
                if (idA != idB) {
                    TSB = klsB.getTs();
                    RMB = klsB.getRoom();
                    if (RMB < 0) {
                        noRoomB = true;
                    } else {
                        roomB = klsB.getAvailableroom().get(RMB).getRoom_id() - 1;
                    }
                    startB = klsB.getAvailableTS().get(TSB).getStart();
                    lengthB = klsB.getAvailableTS().get(TSB).getLength();
                    endB = startB + lengthB;
                    weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                    daysB = klsB.getAvailableTS().get(TSB).getDays();

                    if (noRoomA == false && noRoomB == false) {
                        trvl = valueTravel[roomA][roomB];
                    }
                    outerloop:
                    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;
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
        return jPenalty;
    }
```

Kode 5.50 *Soft Constraint* Same Attendees

5.1.7.13. *Constraint Precedence*

Batasan *precedence* membatasi urutan kelas yang berada pada daftar kelas pada *constraint* tersebut. Jumlah penalti akan bertambah jika ada kelas yang terjadwal mendahului kelas pada urutan yang lebih awal dari pada kelas tersebut. Kode program batasan ini ditunjukkan pada Kode 5.51.

```
int Precedence(ArrayList<Integer> listP, int penalty) {
    int startA, startB, lengthA, lengthB, endA, endB, TSA, TSB;
    int jPenalty = 0;

    for (int i = 0; i < listP.size(); i++) {
        Kelas klsA = SearchKls(listP.get(i));
        int idA = klsA.getClassID();
        TSA = klsA.getTs();
        startA = klsA.getAvailableTS().get(TSA).getStart();
        lengthA = klsA.getAvailableTS().get(TSA).getLength();
        endA = startA + lengthA;
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        for (int j = i + 1; j < listP.size(); j++) {
            Kelas klsB = SearchKls(listP.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                startB = klsB.getAvailableTS().get(TSB).getStart();
                lengthB = klsB.getAvailableTS().get(TSB).getLength();
                endB = startB + lengthB;
                weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                daysB = klsB.getAvailableTS().get(TSB).getDays();

                outerloop:
                for (int wk = 0; wk < weeksA.size(); wk++) {
                    if (weeksA.get(wk) > weeksB.get(wk)) {
                        break outerloop;
                    } else if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
                        for (int dy = 0; dy < daysA.size(); dy++) {
                            if (daysA.get(dy) > daysB.get(dy)) {
                                break outerloop;
                            } else if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
                                if (endA <= startB) {
                                    break outerloop;
                                } else {
                                    jPenalty += penalty;
                                    break outerloop;
                                }
                            } else if (daysA.get(dy) < daysB.get(dy)) {
                                jPenalty += penalty;
                                break outerloop;
                            }
                        }
                    } else if (weeksA.get(wk) < weeksB.get(wk)) {
                        jPenalty += penalty;
                        break outerloop;
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.51 *Soft Constraint* Precedence

5.1.7.14. *Constraint Work Days*

Batasan selanjutnya adalah *work day*, batasan ini mengharuskan pasangan kelas yang dimana awal kelas pertama dan berakhirnya kelas terakhir tidak boleh melebihi *S timeslot*. Adapun kode batasan ini ditunjukkan pada Kode 5.52.

```
int Workday(int S, ArrayList<Integer> listWD, int penalty) {
    int startA, startB, lengthA, lengthB, endA, endB, TSA, TSB;
    int jPenalty = 0;

    for (int i = 0; i < listWD.size(); i++) {
        Kelas klsA = SearchKls(listWD.get(i));
        TSA = klsA.getTs();
        int idA = klsA.getClassID();
        startA = klsA.getAvailableTS().get(TSA).getStart();
        lengthA = klsA.getAvailableTS().get(TSA).getLength();
        endA = startA + lengthA;
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        for (int j = i + 1; j < listWD.size(); j++) {
            Kelas klsB = SearchKls(listWD.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                TSB = klsB.getTs();
                startB = klsB.getAvailableTS().get(TSB).getStart();
                lengthB = klsB.getAvailableTS().get(TSB).getLength();
                endB = startB + lengthB;
                weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                daysB = klsB.getAvailableTS().get(TSB).getDays();
                outerloop:
                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) {
                                    break outerloop;
                                } else {
                                    jPenalty += penalty;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.52 *Soft Constraint* Work Days

5.1.7.15. *Constraint Min Gap*

Batasan ini mengatur batas minimal jarak antara kelas. Setidaknya pasangan kelas harus mempunyai *G* slot waktu antara kedua kelas tersebut. Besarnya *G* merupakan masukan bagi fungsi batasan ini. Kode program dari batasan *min gap* bisa dilihat pada Kode 5.53.

```
int Mingap(int G, ArrayList<Integer> listMG, int penalty) {
    int startA, startB;
    int lengthA, lengthB;
    int endA, endB;
    int jPenalty = 0;

    for (int i = 0; i < listMG.size(); i++) {
        Kelas klsA = SearchKls(listMG.get(i));
        int idA = klsA.getClassID();
        int TSA = klsA.getTs();
        startA = klsA.getAvailableTS().get(TSA).getStart();
        lengthA = klsA.getAvailableTS().get(TSA).getLength();
        endA = startA + lengthA;
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        for (int j = i + 1; j < listMG.size(); j++) {
            Kelas klsB = SearchKls(listMG.get(j));
            int idB = klsB.getClassID();
            if (idA != idB) {
                int TSB = klsB.getTs();
                startB = klsB.getAvailableTS().get(TSB).getStart();
                lengthB = klsB.getAvailableTS().get(TSB).getLength();
                endB = startB + lengthB;
                weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                daysB = klsB.getAvailableTS().get(TSB).getDays();

                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)) {
                                    break outerloop;
                                } else {
                                    jPenalty += penalty;
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    return jPenalty;
}
```

Kode 5.53 *Soft Constraint* Min Gap

5.1.7.16. *Constraint Max Days*

Batasan *max days* membatasi berapa hari kelas tersebut dilaksanakan dalam satu minggu, pelaksanaan kelas tersebut tidak boleh melebihi dari *D* hari selama seminggu. Masukan yang dibutuhkan dari *method* batasan ini adalah daftar kelas dalam batasan, nilai penalti, dan nilai *D*. Kode program dapat dilihat pada Kode 5.54. dan Kode 5.55 merupakan kode fungsi *countNonZeroBits* yang berfungsi untuk menghitung hari pada satu minggu dalam perkuliahan.

```
int Maxdays(int D, ArrayList<Integer> listMD, int penalty) {
    int lebih, jPenalty = 0;
    for (int i = 0; i < listMD.size(); i++) {
        Kelas kls = SearchKls(listMD.get(i));
        int TS = kls.getTs();
        daysA = kls.getAvailableTS().get(TS).getDays();
        if (countNonzeroBits(daysA) <= D) {
            jPenalty += 0;
        } else {
            lebih = (countNonzeroBits(daysA) - D);
            jPenalty += (lebih * penalty);
        }
    }
    return jPenalty;
}
```

Kode 5.54 *Soft Constraint* Max Days

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

Kode 5.55 Fungsi countNonZeroBits

5.1.7.17. *Constraint Max Day Load*

Constraint ini membatasi Panjang *timeslot* dari daftar kelas yang terdapat dalam batasan ini. Batasan ini memiliki dua *method*, yaitu *method max day load* itu sendiri dan *method DayLoad* yang berfungsi untuk menghitung Panjang *timeslot* per hari dalam seminggu.

Adapun kode program *MaxDayLoad* dapat dilihat pada Kode 5.56 Dan kode program *DayLoad* pada Kode 5.57

```
int MaxdayLoad(int S, ArrayList<Integer> listMDL, int penalty) {
    int jPenalty;
    int lebih = 0;
    int total = 0;
    Kelas klsA = SearchKls(listMDL.get(0));
    weeksA = klsA.getAvailableTS().get(0).getWeeks();
    daysA = klsA.getAvailableTS().get(0).getDays();
      for (int wk = 0; wk < weeksA.size(); wk++) {
          for (int dy = 0; dy < daysA.size(); dy++) {
    int jmlLength = DayLoad(0, 0, 0, weeksA.size(), daysA.size(),
            listMDL.size(), listMDL, total);
    if (jmlLength <= S) {
        lebih += 0;
    } else {
        lebih += jmlLength - S;
              }
          }
    }
    jPenalty = (lebih * penalty) / weeksA.size();
    return jPenalty;
}
```

Kode 5.56 ***Soft Contraints Max Day Load***

```
int DayLoad(int w, int d, int kls, int btsW, int btsD, int btsKls,
        ArrayList<Integer> list, int total) {

    Kelas ini = SearchKls(list.get(kls));
    int TS = ini.getTs();
    int length = ini.getAvailableTS().get(TS).getLength();
    weeksA = ini.getAvailableTS().get(TS).getWeeks();
    daysA = ini.getAvailableTS().get(TS).getDays();
    if (w < btsW) {
        if (d < btsD) {
            if (kls < btsKls) {
                if (weeksA.get(w) == 1) {
                    if (daysA.get(d) == 1) {
                        total += length;
                        DayLoad(w, d, ++kls, btsW, btsD, btsKls, list, total);
                    } else {
                        DayLoad(w, d, ++kls, btsW, btsD, btsKls, list, total);
                    }
                } else {
                    DayLoad(w, d, ++kls, btsW, btsD, btsKls, list, total);
                }
            } else {
                return total;
            }
        }
    }
    return total;
}
```

Kode 5.57 Fungsi *DayLoad*

5.1.7.18. *Constraint Max Breaks*

Pengertian batasan lunak *max breaks* sama dengan pengertian dari batasan keras *max breaks* dimana batasan ini membatasi jumlah jeda antar kelas, hal yang berbeda adalah dari masukan kode program, pada *soft constraint max breaks* membutuhkan nilai penalti batasan tersebut. Adapun kode program dari batasan ditunjukkan oleh Kode 5.58 dan fungsi *mergeblocks* pada Kode 5.59

```
int Maxbreaks(int R, int S, ArrayList<Integer> listMB, int penalty) {
    int jPenalty = 0;

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

Kode 5.58 ***Soft Constraint Max Breaks***

```
private int MergeBlocks(int S, ArrayList<Integer> listMB) {
    int initialBlock = 1;

    int startA, startB;
    int lengthA, lengthB;
    int endA, endB;

    for (int x = 0; x < listMB.size(); x++) {
        Kelas klsA = SearchKls(listMB.get(0));
        int TSA = klsA.getTs();
        startA = klsA.getAvailableTS().get(TSA).getStart();
        lengthA = klsA.getAvailableTS().get(TSA).getLength();
        endA = startA + lengthA;
        daysA = klsA.getAvailableTS().get(TSA).getDays();
        weeksA = klsA.getAvailableTS().get(TSA).getWeeks();

        for (int wk = 0; wk < weeksA.size(); wk++) {
            outerloop:
            for (int dy = 0; dy < daysA.size(); dy++) {
                for (int i = 1; i < listMB.size(); i++) {
                    Kelas klsB = SearchKls(listMB.get(i));
                    int TSB = klsB.getTs();
                    startB = klsB.getAvailableTS().get(TSB).getStart();
                    lengthB = klsB.getAvailableTS().get(TSB).getLength();
                    endB = startB + lengthB;
                    daysB = klsB.getAvailableTS().get(TSB).getDays();
                    weeksB = klsB.getAvailableTS().get(TSB).getWeeks();
                    if (weeksA.get(wk) == 1 && weeksB.get(wk) == 1) {
                        if (daysA.get(dy) == 1 && daysB.get(dy) == 1) {
                            if (endA + S >= startB && endB + S >= startA) {
                                startA = Math.min(startA, startB);
                                endA = Math.max(endA, endB);
```

```
                                } else {
                                    initialBlock++;
                                    startA = startB;
                                    endA = endB;
                                    daysA = daysB;
                                    weeksA = weeksB;
                                }
                            }
                        }
                    }
                }
            }
        }
        return initialBlock;
}
```

Kode 5.59 Fungsi *Merge Blocks* Batasan *Max Breaks*

5.1.7.19. *Constraint Max Blocks*

Batasan terakhir adalah batasan yang membatasi panjang blok kelas sehingga tidak lebih dari *M* slot waktu. Batasan *max blocks* juga menerima masukan yaitu nilai penalti dari batasan tersebut. Kode program beserta fungsi *merge blocks* akan ditunjukkan pada Kode 5.60 dan 5.61

```
int Maxblock(int M, int S, ArrayList<Integer> listMBL, int penalty) {
    int jPenalty = 0;

    if (MergeBlockB(M, S, listMBL) <= M) {
        return jPenalty;
    } else {
        jPenalty = (penalty * MergeBlockB(M, S, listMBL)) / weeksA.size();
    }
    return jPenalty;
}
```

Kode 5.60 *Constraint Max Blocks*

```
private int MergeBlockB(int M, int S, ArrayList<Integer> listMBL) {
    int lengthBlock = 0;
    int lebih = 0;
    boolean blockBaru = true;
    Kelas kls = SearchKls(listMBL.get(0));
    int TS = kls.getTs();

    int startA = kls.getAvailableTS().get(TS).getStart();
    int lengthA = kls.getAvailableTS().get(TS).getLength();
    int endA = startA + lengthA;
    daysA = kls.getAvailableTS().get(TS).getDays();
    weeksA = kls.getAvailableTS().get(TS).getWeeks();

    for (int i = 1; i < listMBL.size(); i++) {
        Kelas klsB = SearchKls(listMBL.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;
            daysB = klsB.getAvailableTS().get(iTS).getDays();
            weeksB = klsB.getAvailableTS().get(iTS).getWeeks();

            outerloop:
            for (int wk = 0; wk < weeksA.size(); wk++) {
                for (int dy = 0; dy < daysA.size(); dy++) {
                    if (weeksA.get(wk) == 1 && daysA.get(dy) == 1
                            && weeksB.get(wk) == 1 && daysB.get(dy) == 1) {
                        if (endA + S >= startB && endB + S >= startA) {
                            startA = Math.min(startA, startB);
                            endA = Math.max(endA, endB);
                            lengthBlock = endA - startA;
                            if (lengthBlock > M && blockBaru == true) {
                                lebih += 1;
                                blockBaru = false;
                            } else {
                                lebih += 0;
                            }
                        } else {
                            lengthBlock = endA - startA;
                            if (lengthBlock > M && blockBaru == true) {
                                lebih += 1;
                            } else {
                                lebih += 0;
                            }
```

```
                                blockBaru = true;
                                startA = startB;
                                endA = endB;
                                daysA = daysB;
                                weeksA = weeksB;
                            }
                        }
                    }
                }
            } else if (iTS == -1) {
                return lengthBlock;
            }
        }
        return lengthBlock;
    }
```

Kode 5.61 Fungsi *Merge Blocks* Batasan *Max Blocks*

5.2. Optimasi Solusi

Pada bagian ini akan dijelaskan mengenai bagaimana cara untuk mengoptimasi jadwal yang sudah terbentuk pada solusi awal dengan menggunakan implementasi algoritma *Tabu Search-Simulated Annealing Hyper-Heuristics*. Dalam tahapan optimasi solusi, aktivitas yang dilakukan adalah sebagai berikut, menyimpan solusi awal, pembuatan dan pemilihan *low level heuristic*, implementasi algoritma, dan penyimpanan solusi optimum.

5.2.1. Pembuatan *Low Level Heuristics*

Metode *low level heuristic* merupakan metode tingkat rendah yang digunakan untuk memindah urutan solusi hingga solusi baru tercipta dan menghasilkan nilai penalti baru. Pada tugas akhir ini, ditentukan *low level heuristic* berupa "*move*" yang memiliki variasi yaitu *"move timeslot", "move room",* serta *"move room* dan *timesloot"*. *Move* dilakukan dengan memilih satu atau beberapa kelas secara random kemudian memindahkan *timeslot* maupun *room* yang sudah terpilih di kelas

tersebut dan mengganti *timeslot* maupun *room* tersebut. Contoh *low level heuristic* ditunjukkan oleh Kode 5.62 dan Kode 5.63.

```
int[] move1TS() {

    int klsMove[] = new int[1];
    int ts = -1, tsBaru = -1;
    ArrayList<TimeAss> times;

    do {
        int kls = rand.nextInt(listKelas.size());
          System.out.println("movelts " + kls);
        Kelas kelas = listKelas.get(kls);
        if (kelas.isHasRoom()) {
            times = kelas.getAvailableTS();
            ts = kelas.getTs();
            tsBaru = ts;
            RoomAss room = kelas.getAvailableroom().get(kelas.getRoom());
            if ((ts + 1) >= times.size()) {
                ts = 0;
            }
            for (int i = ts; 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 5.62 Low Level Heuristic: Move Timeslot

```
int[] move1RM() {
    int klsMove[] = new int[1];
    int rm = -1, rmBaru = -1, ts;
    ArrayList<RoomAss> rooms;
    do {
        int kls = rand.nextInt(listKelas.size());
        Kelas kelas = listKelas.get(kls);
        if (kelas.isHasRoom()) {
            ts = kelas.getTs();
            TimeAss time = kelas.getAvailableTS().get(ts);
            rm = kelas.getRoom();
            rmBaru = rm;
            rooms = kelas.getAvailableroom();
            if ((rm + 1) >= rooms.size()) {
                rm = 0;
            }
```

```
            for (int i = rm; i < rooms.size(); i++) {
                if (i != rm) {
                    RoomAss room = rooms.get(i);
                    if (cekTS(time, room)) {
                        kelas.setRoom(i);
                        rmBaru = i;
                        klsMove[0] = kls;
                        break;
                    }
                }
            }

        }
    } while (rm == rmBaru);
    return klsMove;
}
```

Kode 5.63 Low Level Heuristic: Move *Room*

Selain *low level heuristic* yang ditunjukkan oleh Kode 5.62 dan 5.63, dalam pengerjaan tugas akhir ini juga dilakukan eksperimen dengan menggunakan jumlah kelas yang dilakukan *move*, yaitu *move 1 timeslot, move 2 timeslot, move 1 room, move 2 room, move 1 timeroom, move 2 timeroom.*

5.2.2. Implementasi Algoritma

Penerapan algoritma *Simulated Annealing* diimplementasikan dengan menggunakan persamaan Bolztmann sebagai *acceptance criteria* sesuai dengan pseudocode yang ditunjukkan pada Kode 5.64. Solusi baru dengan nilai penalti lebih kecil akan langsung diterima sebagai solusi terbaik. Sedangkan jika nilai penalti baru lebih tinggi maka solusi tersebut masuk dalam percabangan, percabangan tersebut melakukan pengecekan apakah nilai random memenuhi atau tidak persamaan Bolztmann, maka algoritma Tabu Search akan diimplementasikan sesuai dengan pseudocode

pada Kode 5.65 dengan *acceptance criteria* yaitu solusi yang dihasilkan tidak terdapat pada daftar pengecualian (*Tabu List*).

```
1: procedure Simulated Annealing
2:    current ← initial solution
3:    t ← initial temperature
4:    l ← initial length
5:    repeat
6:     for i = 1 to l do
7:       candidate ϵ N(current)
8:         if f(candidate) <= f(current) then
9:            current ← candidate
10:         else if
11:          exp(f(current) - f(candidate)=t) >
           random[0,1)then
12:    current ← candidate
13:  end if
14:    end for
15:    update l and t
16:   until stop condition
17: end procedure
```

Kode 5.64 Pseudocode Algoritma *Simulated Annealing*

```
1: procedure Tabu Search
2:   Tabu list T
3:   current ← initial solution
4:   best ← current
5:    repeat
6:       current ← arg min_{xϵN(current)} f(x), x: non-
         tabu
7:      if current < best then
8:          best ← current
9:      end if
10:         record the recent move in T
            delete the oldest entry if necessary
11:   until stop condition
12: end procedure
```

Kode 5.65 Pseudocode Algoritma *Tabu Search*

Berdasarkan pada penjelasan tersebut, solusi baru dapat diterima menjadi solusi terbaik jika memenuhi satu dari 2 syarat, yaitu jika (1) nilai penalti dari solusi yang baru lebih rendah daripada nilai penalti solusi sekarang, atau (2) nilai *random* lebih besar daripada nilai persamaan Bolztmann dan solusi tidak terdapat pada daftar tabu atau *Tabu List*. Sehingga implementasi algoritma *Tabu Search – Simulated Annealing* dapat dituliskan seperti pada Kode 5.64

```
if (delta > 0 && newPenalty != X) { //ACCEPT SOLUTION
    currentPenalty = newPenalty;
    penaltyTS = penaltyTSBaru;
    penaltyRM = penaltyRMBaru;
    penaltySC = penaltySCBaru;
    for (int i = 0; i < lkls.length; i++) {
        Kelas kelas = listKelas.get(lkls[i]);
        int id = kelas.getClassID();
        if (kelas.isHasRoom()) {
            int tsBackup = kelas.getTsBaru();
            int rmBackup = kelas.getRoomBaru();
            TimeAss time = kelas.getAvailableTS().get(tsBackup);
            RoomAss room = kelas.getAvailableroom().get(rmBackup);
            removeKls(time, room);

            int ts = kelas.getTs();
            int rm = kelas.getRoom();
            time = kelas.getAvailableTS().get(ts);
            room = kelas.getAvailableroom().get(rm);
            assign(id, time, room);
            kelas.setTsBaru(kelas.getTs());
            kelas.setRoomBaru(kelas.getRoom());
        } else {
            kelas.setTsBaru(kelas.getTs());
            kelas.setRoomBaru(kelas.getRoom());
        }
    }
} else if (delta <= 0) {
     //START TABU
    double p = Math.pow(Math.E, (-(Math.abs(delta) / temp)));
    double x = Math.random();
    if (p > x) { //REJECT CONDITION
        if (tabuList.contains(newPenalty)) {
            for (int i = 0; i < lkls.length; i++) {
                Kelas kelas = listKelas.get(lkls[i]);
                kelas.setTs(kelas.getTsBaru());
                kelas.setRoom(kelas.getRoomBaru());
            }

            if (tabuList.remainingCapacity() == 0) {
                tabuList.remove();
                tabuList.add(newPenalty);
            } else {
                tabuList.add(newPenalty);
            }
```

```
    } else { //ACCEPT SOLUTION + ADD TABU
        currentPenalty = newPenalty;
        penaltyTS = penaltyTSBaru;
        penaltyRM = penaltyRMBaru;
        penaltySC = penaltySCBaru;
        for (int i = 0; i < lkls.length; i++) {
            Kelas kelas = listKelas.get(lkls[i]);
            int id = kelas.getClassID();
            if (kelas.isHasRoom()) {
                int tsBackup = kelas.getTsBaru();
                int rmBackup = kelas.getRoomBaru();
                TimeAss time = kelas.getAvailableTS().get(tsBackup);
                RoomAss room = kelas.getAvailableroom().get(rmBackup);
                removeKls(time, room);

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

                kelas.setTsBaru(kelas.getTs());
                kelas.setRoomBaru(kelas.getRoom());
            } else {
                kelas.setTsBaru(kelas.getTs());
                kelas.setRoomBaru(kelas.getRoom());
            }
        }
        .
        if (tabuList.remainingCapacity() == 0) {
            tabuList.remove();
            tabuList.add(newPenalty);
        } else {
            tabuList.add(newPenalty);
        }
          if (tabu.size() >= tabuLength) {
              tabu.pop();
              tabu.push(storeResult);
          } else {
              tabu.push(storeResult);
          }
    }
} else {
    for (int i = 0; i < lkls.length; i++) {
        Kelas kelas = listKelas.get(lkls[i]);
        kelas.setTs(kelas.getTsBaru());
        kelas.setRoom(kelas.getRoomBaru());
    }
}
```

```
} else if (newPenalty == X) { //REJECT SOLUTION
    for (int i = 0; i < lkls.length; i++) {
        Kelas kelas = listKelas.get(lkls[i]);
        kelas.setTs(kelas.getTsBaru());
        kelas.setRoom(kelas.getRoomBaru());
    }
}
```

Kode 5.66 Implementasi Algoritma Tabu - Simulated Annealing

Optimasi menggunakan algoritma *Tabu Search-Simulated Annealing* terus dijalankan hingga *stopping criteria* yaitu berdasarkan jumlah iterasi yang ditetapkan.

5.2.3. Penyimpanan Solusi Optimum Akhir

Solusi optimum akhir yang telah terbentuk disimpan dalam file berekstensi .xml. Adapun kode program yang digunakan untuk penyimpanan solusi akhir sama dengan penyimpanan solusi awal yang telah ditunjukkan oleh Kode 5.65

```
public class WriteSol {

    ArrayList<RoomAss> room;
    ArrayList<TimeAss> time;

    WriteSol(String instance, ArrayList<Kelas> listKelas, int penalty) {
        String name = "src/solution/solution_" + instance;
        try {

            Collections.sort(listKelas, new sortIdKelas());

            DocumentBuilderFactory documentFactory = DocumentBuilderFactory.newInstance();

            DocumentBuilder documentBuilder = documentFactory.newDocumentBuilder();

            Document document = documentBuilder.newDocument();

            //ROOT ELEMENT
            Element root = document.createElement("solution");
            root.setAttribute("name", instance + " (penalti: " + penalty + ")");
            root.setAttribute("technique", "Tabu Search - Simulated Annealing");
            root.setAttribute("author", "Narendra");
            root.setAttribute("institution", "ITS");
            root.setAttribute("country", "Indonesia");
```

Kode 5.67 Penyimpanan Solusi File XML

Halaman ini sengaja dikosongkan

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini akan menjelaskan mengenai proses dan hasil uji coba serta analisis terhadap hasil yang diperoleh dari proses implementasi Algoritma Tabu-Simulated Annealing Hyper-Heuristic, termasuk eksperimen parameter yang digunakan.

6.1. Data Uji Coba

Dataset ITC menjadi data uji coba yang dilakukan pada pengerjaan tugas akhir ini.

6.2. Lingkungan Uji Coba

Pada subbab ini membahas mengenai lingkungan pengujian dalam pengerjaan penelitian tugas akhir terkait optimasi penjadwalan mata kuliah. Spesifikasi perangkat keras yang digunakan dalam implementasi ditunjukkan pada Tabel 6.1

Tabel 6.1 Spesifikasi Perangkat Keras

Perangkat Keras	Spesifikasi
Jenis	Laptop ASUS X456UF
Processor	Intel(R) Core i5-6200U 2.3GHz
RAM	8 GB
***Hard* Disk Drive**	500 GB

Sedangkan untuk spesifikasi perangkat lunak yang digunakan dalam pengerjaan penelitian tugas akhir ditunjukkan pada Tabel 6.2

Tabel 6.2 Spesifikasi Perangkat Lunak

Perangkat Lunak	Fungsi
Windows 10 64 bit	Sistem Operasi
Netbeans 8.2	Implementasi Algoritma

Perangkat Lunak	Fungsi
Sublime	Pengolahan Data dan Hasil
Microsoft Excel 2016	Pengolahan Hasil Uji Coba
Microsoft Word 2016	Penulisan Laporan

6.3. Penentuan Urutan Objek Pembentukan Solusi Awal

Penjelasan mengenai susunan serta urutan indeks kelas atau mata kuliah akan dijelaskan pada subbab ini. Pembentukan solusi awal melalui langkah yang telah dijelaskan pada subbab 5.1. Dalam pembentukan solusi awal, urutan indeks objek mata kuliah harus diperhatikan untuk mendapatkan solusi yang lolos *hard constraint*. Beberapa indikator yang digunakan dalam mengurutkan indeks objek mata kuliah adalah jumlah daftar waktu tersedia (*available times*), jumlah daftar ruang tersedia (*available rooms*), jumlah batasan keras per kelas (*hard constraint)*, dan jumlah kelas yang sudah di urutkan (*Sorted Class*).

6.3.1. Skenario Urutan Indikator 1 Kombinasi

Pada skenario pertama pembentukan solusi awal, tidak dilakukan dengan menggunakan kombinasi antar indikator. Urutan objek dilakukan hanya berdasarkan satu indikator terkait. Hasil dari eksperimen skenario pertama ditunjukkan pada Tabel 6.3 yang dimana angka di dalam kolom merupakan jumlah kelas yang belum terjadwalkan.

Tabel 6.3 Hasil Skenario 1 Pembentukan Solusi Awal

Instances	Indikator		
	Times	*Rooms*	*Hard Constraint*
Tiny	**0**	**0**	**0**
Small 1	12	9	19
Small 2	**0**	**0**	**0**
Medium	20	14	28

Instances	Indikator		
	Times	*Rooms*	*Hard Constraint*
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**

6.3.2. Skenario Urutan Indikator 2 Kombinasi

Berdasarkan Tabel 6.3, diketahui bahwa urutan indikator dari yang terbaik dan terendah adalah *Times, Rooms,* dan *Hard Constraint*. indikator *Times* dan *Rooms* memiliki hasil yang lebih baik daripada indikator lain. Sehingga pada eksperimen selanjutnya yaitu skenario menggunakan 2 kombinasi, hanya dilakukan dengan indikator *times* dan *rooms*. Hasil eksperimen tersebut ditunjukkan pada Tabel 6.4.

Tabel 6.4 Hasil Skenario 2 Pembentukan Solusi Awal – Times

Instances	Indikator			
	Time - Room	*Times - HC*	*Rooms - Times*	*Rooms - HC*
Tiny	**0**	**0**	**0**	**0**
Small 1	11	12	9	9
Small 2	**0**	**0**	**0**	**0**
Medium	17	20	10	14
Large	148	158	92	114
Early 1	46	39	59	63

Instances	Indikator			
	Time - Room	*Times - HC*	*Rooms - Times*	*Rooms - HC*
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**

6.3.3. Skenario Urutan Indikator 3 Kombinasi

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

Tabel 6.5 Hasil Skenario 3 Pembentukan Solusi Awal

Instances	Indikator		
	Room – Time – HC	*Time – HC - Room*	*Sorted Class*
Tiny	**0**	**0**	**0**
Small 1	9	12	14
Small 2	**0**	**0**	**0**
Medium	10	20	10
Large	92	158	93
Early 1	59	39	47
Early 2	116	101	122

Instances	Indikator		
	Room – Time – HC	*Time – HC - Room*	*Sorted Class*
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**

Berdasarkan Tabel 6.5, *Sorted Class* atau kombinasi dari urutan kelas yang sudah diurutkan menunjukkan hasil rata-rata yang paling bagus. Tetapi, berdasarkan tabel masih belum ada kombinasi yang bisa menjadwalkan semua dataset. Hanya dataset *Tiny* dan *Small 2* yang memenuhi semua *hard constraint.*

Uji menggunakan *backtrack* juga sudah digunakan untuk menemukan solusi dari data yang belum terjadwalkan semua. Tetapi proses dari *backtrack* tidak menghasilkan hasil yang memuaskan, yang artinya proses *backtrack* mengulang terus iterasi sehingga menyebabkan iterasi tidak bisa berhenti jika tidak menemukan solusi yang tepat.

Sehingga dalam pengerjaan atau penelitian tugas akhir ini dilakukan dengan menggunakan dua dataset yaitu *Tiny (lums-sum17.xml)* dan *Small2 (Pu-cs-fal07.xml)*, serta dengan menggunakan kombinasi *Sorted Class.*

6.4. Hasil Eksperimen Implementasi Algoritma

Hasil eksperimen dari optimasi data ITC menggunakan algoritma *Tabu Search – Simulated Annealing* akan dijelaskan pada subbab ini. Bagian ini juga menjelaskan

skenario beserta hasil skenario eksperimen yang telah dilakukan. Setiap eksperimen dijalankan sebanyak 10 kali. Tabel 6.6 menunjukkan daftar parameter yang digunakan. Sementara daftar nilai parameter ditunjukkan oleh Tabel 6.7. Nilai parameter didasarkan pada percobaan penulis secara acak, namun dalam rentang nilai yang masih memungkinkan berdasarkan beberapa penelitian sebelumnya.

Tabel 6.6 Daftar Parameter Percobaan

Parameter	Keterangan
LLH	Jumlah *low level heuristic* yang digunakan
T_0	Suhu awal algoritma Simulated Annealing
α	Penurunan suhu (*alpha*)
Nα	Jumlah iterasi setiap penurunan suhu
β	Kenaikan suhu pada proses reheating (*beta*)
Nβ	Jumlah iterasi setiap kenaikan suhu (*reheating*)
TL	Panjang *tabu list* solusi algoritma Tabu Search

Tabel 6.7 Daftar Skenario Eksperimen

Skenario	Iterasi (ribu)	Eksperimen	Nilai Parameter			
A	500	Suhu (T_0)	75	95	150	-
B	500	*Cooling Rate (α)*	0.4	0.6	0.8	0.999
C	500	LLH	3	3	6	-
D	500	*Tabu List*	3	5	10	-

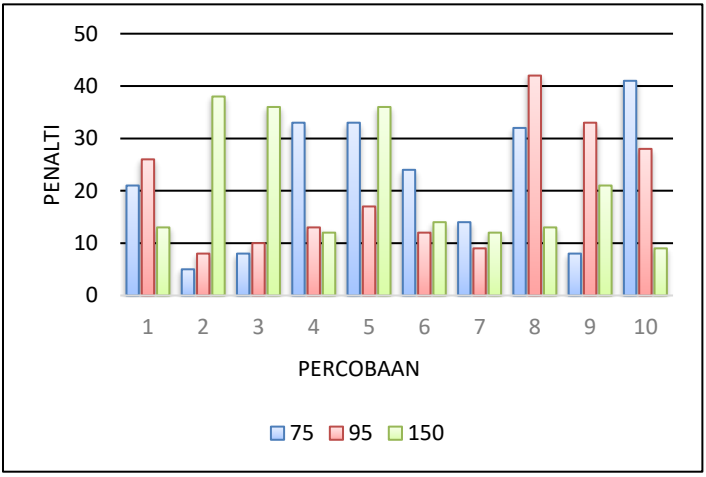
6.4.1. Skenario A – *Initial Temperature*

Pada skenario A, percobaan dilakukan dengan merubah parameter suhu awal menggunakan parameter seperti yang ditunjukkan pada Tabel 6.6. Adapun hasil eksperimen A ditunjukkan oleh Tabel 6.8.

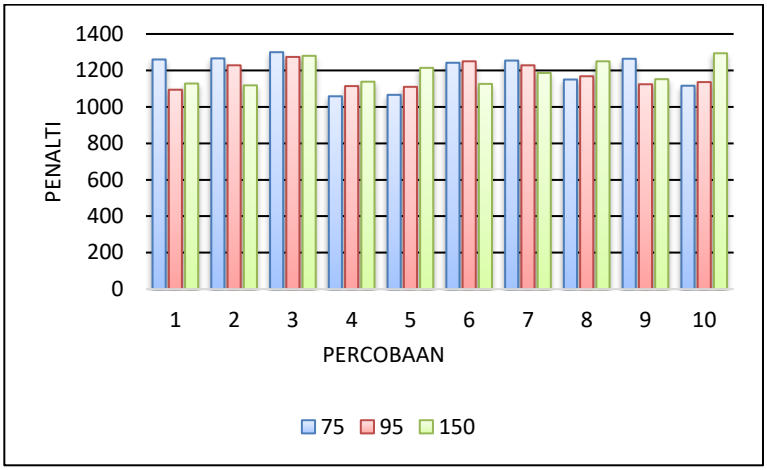
Tabel 6.8 Hasil Eksperimen Skenario A – Initial Temperature

Instance		Initial	75	95	150
tiny	Best	48	**5**	8	9
	Worst		41	42	**38**
Average			21.9	**19.8**	20.4
small	Best	1790	**1058**	1094	1118
	Worst		1300	**1274**	1294
Average			**1**197.6	**1172.6**	1188.6

Pada Tabel 6.8, *instance tiny* dengan nilai suhu 75 memperoleh hasil yang lebih optimal dari nilai parameter lain. Sedangkan rata-rata terendah didapatkan pada parameter 95. Pada *nstance small* penalti terendah didapatkan oleh nilai suhu 75, dan rata-rata terendah terdapat pada suhu 95. Perbandingan setiap percobaan *instances small* dan *tiny* ditunjukkan pada Gambar 6.1 dan 6.2.

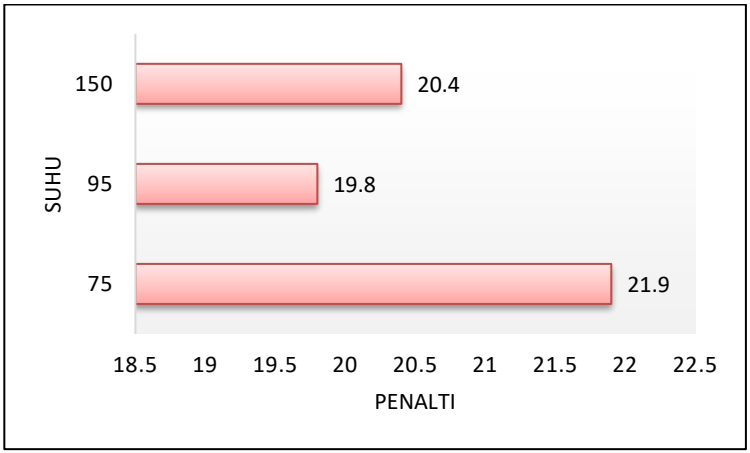


Gambar 6.1 Perbandingan Penalti pada Perubahan Suhu *Instance Tiny*

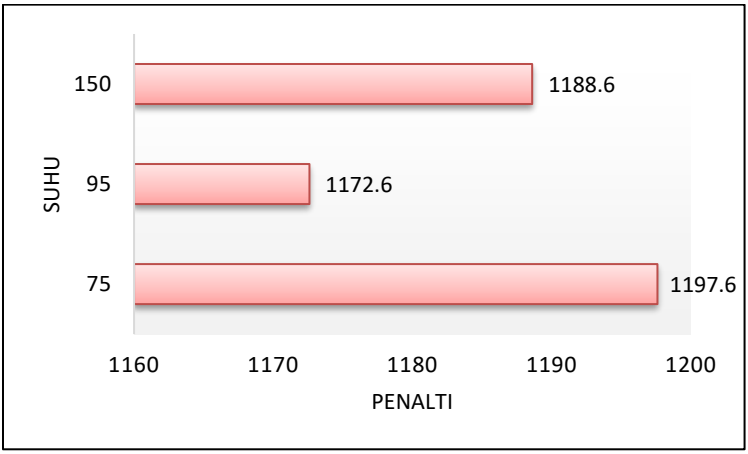


Gambar 6.2 Perbandingan Penalti pada Perubahan Suhu *Instance Small*

Sedangkan grafik rata-rata penalti pada setiap perubahan suhu akan ditunjukkan pada Gambar 6.3 dan 6.4.



Gambar 6.3 Rata-rata Penalti Perubahan Suhu (*Tiny*)



Gambar 6.4 Rata-rata Penalti Perubahan Suhu (Small)

6.4.2. Skenario B – *Cooling Rate*

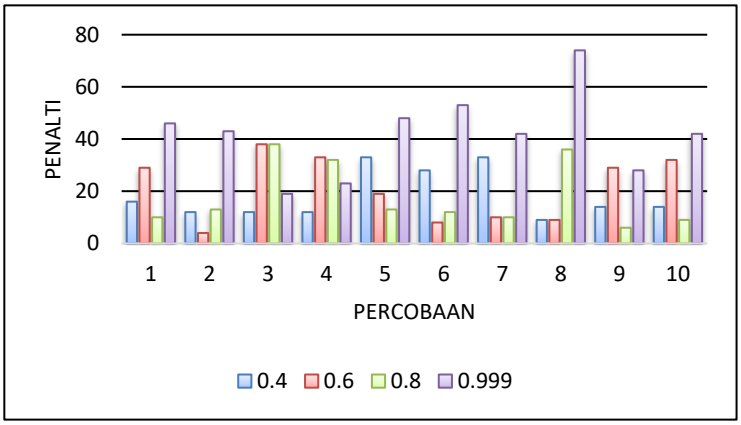
Pada skenario B, percobaan dilakukan dengan mengubah parameter *cooling rate* atau nilai koefisien penurunan suhu dengan parameter seperti yang ditunjukkan pada Tabel 6.6. Tabel 6.9 menunjukkan hasil eksperimen skenario B.

Tabel 6.9 Hasil Eksperimen Skenario B – Cooling Rate

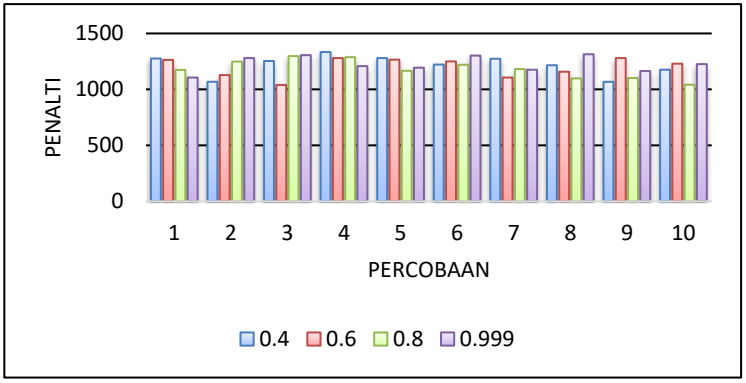
Instance		Initial	0.4	0.6	0.8	0.999
tiny	Best	48	9	**4**	6	19
	Worst		**33**	38	38	74
Average			18.3	21.1	**17.9**	41.8
small	Best	1790	1068	**1040**	1042	1106
	Worst		1334	**1280**	1298	1314
Average			1216.8	1200.2	**1181.8**	1227.6

Hasil dalam Tabel 6.9 pada *instance tiny* dan *small* ditemukan bahwa pada *cooling rate* 0.6 penalti yang

didapatkan lebih optimal dibandingkan dengan parameter lain. Pada nilai parameter 0.8 mempunyai rata-rata penalti yang lebih rendah dari parameter lain. Hasil dari 10 percobaan ditampilkan pada Gambar 6.5 dan Gambar 6.6

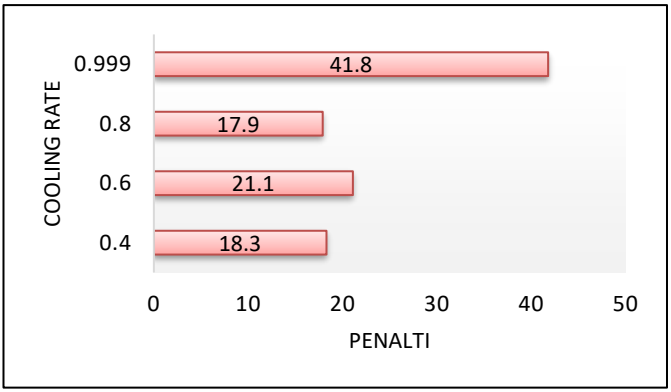


Gambar 6.5 Perbandingan Penalti Perubahan *Cooling Rate (Tiny)*

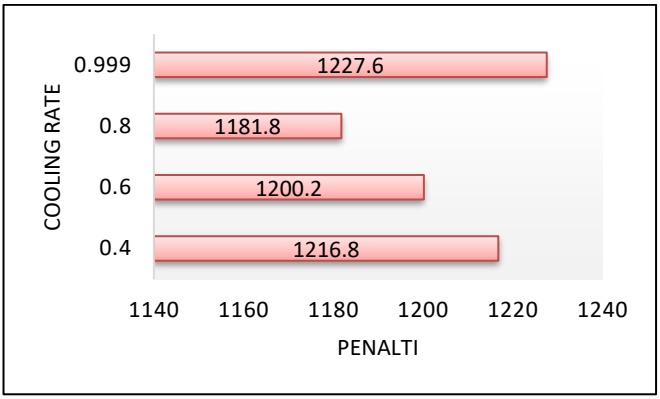


Gambar 6.6 Perbandingan Penalti Perubahan *Cooling Rate (Small)*

Dari Gambar 6.5 dan 6.6 dapat diperoleh rata-rata hasil perhitungan penalti dari perubahan *cooling rate*. Perubahan *cooling rate* mempunyai pengaruh terhadap cepat tidak nya penurunan suhu, Gambar 6.7 dan 6.8 merupakan rata-rata hasil penalty dari perubahan parameter *cooling rate.*



Gambar 6.7 Perbandingan Penalti Perubahan *Cooling Rate (Tiny)*



Gambar 6.8 Perbandingan Penalti Perubahan *Cooling Rate (Small)*

6.4.3. Skenario C – *Low Level Heuristics*

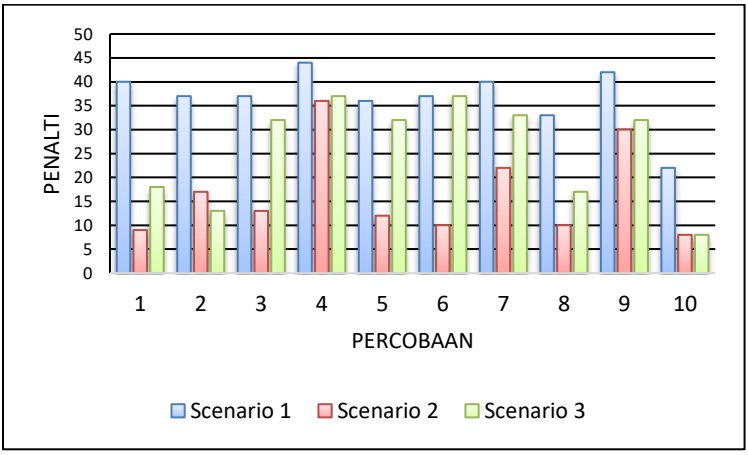
Eksperimen skenario C dilakukan dengan mengubah jumlah *low level heuristic* yang digunakan. Percobaan pertama dilakukan dengan menggunakan 3 parameter yaitu *move-1TS, move-1RM,* dan *move-1TSRM,* percobaan kedua menggunakan 3 parameter yaitu *move-2TS, move-2RM,* dan *move-2TSRM,* sedangkan untuk percobaan ketiga menggunakan semua LLH tersebut. Hasil eksperimen skenario C ditunjukkan pada Tabel 6.10.

Tabel 6.10 Hasil Eksperimen Skenario C – Low Level Heuristics

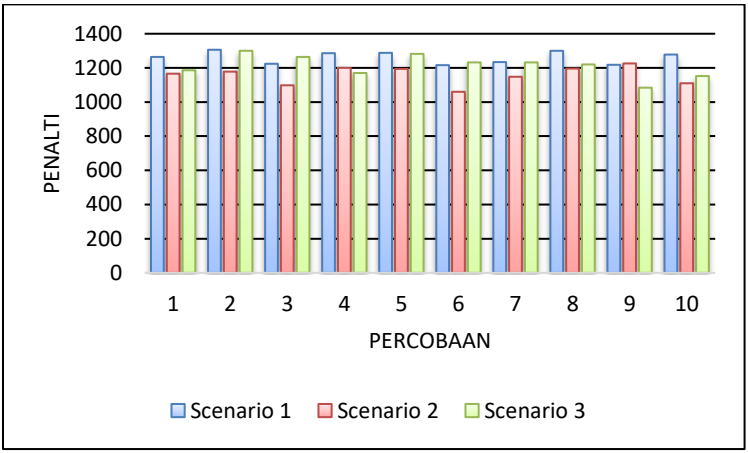
Instance		Initial	3 (1)	3 (2)	6
tiny	Best	48	22	**8**	8
	Worst		44	**36**	37
Average			36.8	**16.7**	25.9
small	Best	1790	1216	**1060**	1084
	Worst		1306	**1226**	1300
Average			1261.4	**1157.6**	1212.2

Berdasarkan Tabel 6.10 diketahui bahwa pada *instance tiny* maupun *small* dengan parameter kedua selalu mendapatkan hasil yang lebih baik dari pada percobaan dengan nilai parameter lain.

Data hasil percobaan akan ditunjukkan pada Gambar 6.9 dan 6.10 yang dimana merupakan perbandingan hasil penalti perubahan *LLH.*

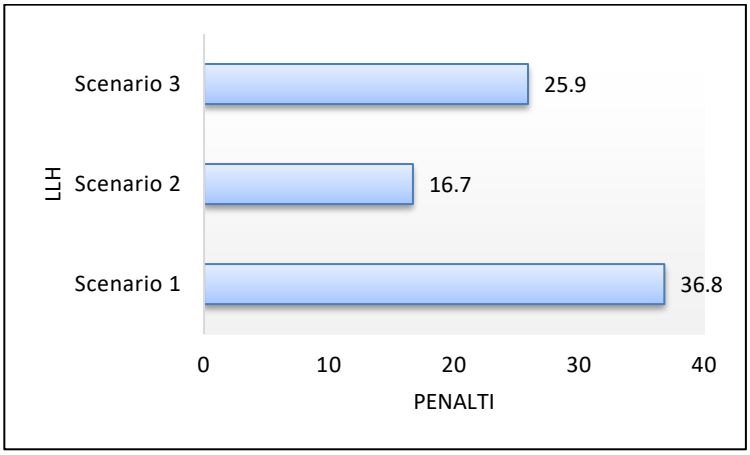


Gambar 6.9 Perbandingan Penalti Perubahan Skenario *LLH (Tiny)*

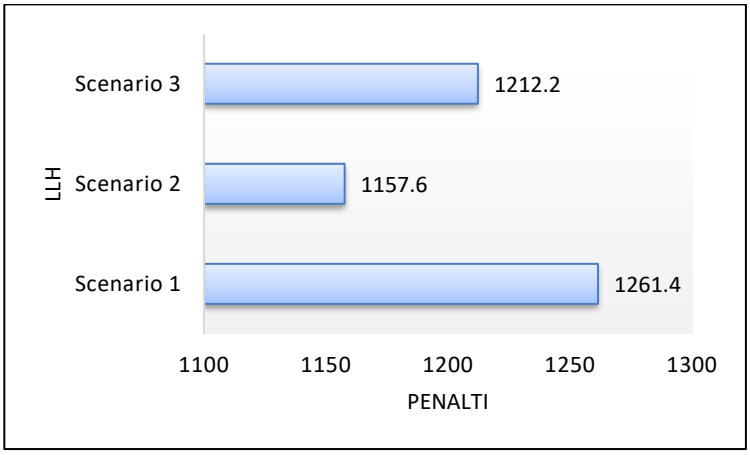


Gambar 6.10 Perbandingan Penalti Perubahan Skenario *LLH (Small)*

Sedangkan untuk hasil rata-rata penalti semua percobaan ditunjukkan oleh Gambar 6.11 dan 6.12.



Gambar 6.11 Rata-rata Penalti Perubahan Skenario *(tiny)*



Gambar 6.12 Rata-rata Penalti Perubahan Skenario *(small)*

6.4.4. Skenario D – *Tabu Length*

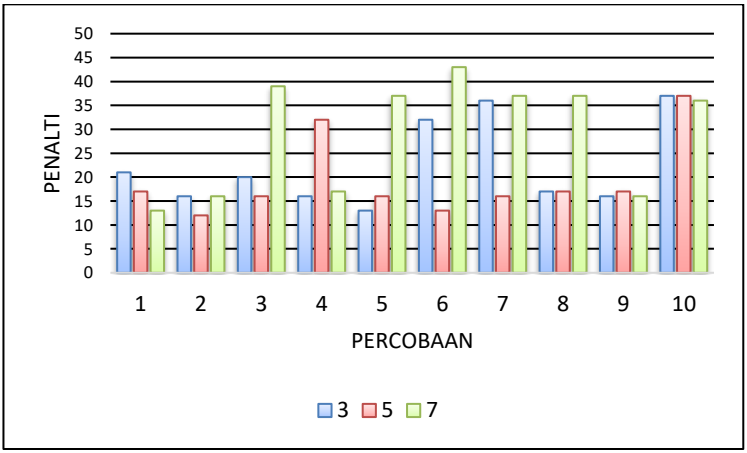
Eksperimen skenario D dilakukan dengan mengubah *tabu length* atau panjang daftar tabu untuk menyimpan nilai yang tidak lebih baik dari penalti sebelumnya.

Parameter diubah sesuai dengan Tabel 6.7. Adapun hasil eksperimen ditunjukkan oleh Tabel 6.11.

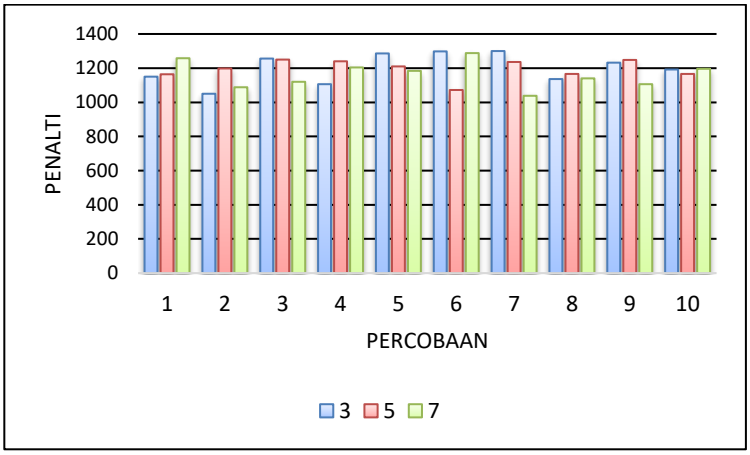
Tabel 6.11 Hasil Eksperimen Skenario D

Instance		Initial	3	5	7
tiny	Best	48	13	**12**	13
	Worst		**37**	**37**	43
Average			22.4	**19.3**	29.1
small	Best	1790	1050	1072	**1038**
	Worst		1300	**1250**	1288
Average			**1**200.6	1195	**1162.2**

Tabel 6.11 menunjukkan nilai 5 pada parameter *tabu length* di *instance tiny* memperoleh hasil lebih baik dibandingkan dengan nilai parameter lain, sedangkan pada *instance small* nilai parameter 7 mendapatkan nilai lebih baik dari pada nilai parameter lain. Grafik perbedaan hasil penalty pada setiap percobaan ditunjukkan pada Gambar 6.13 dan 6.14

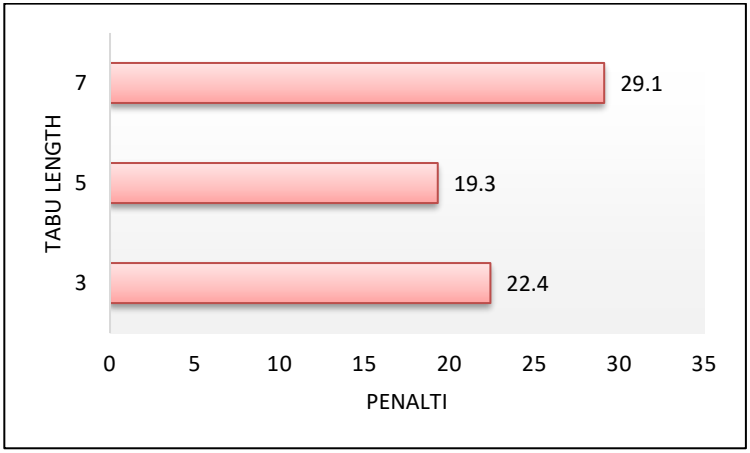


Gambar 6.13 Perbandingan Perubahan Penalti Tabu List (Tiny)

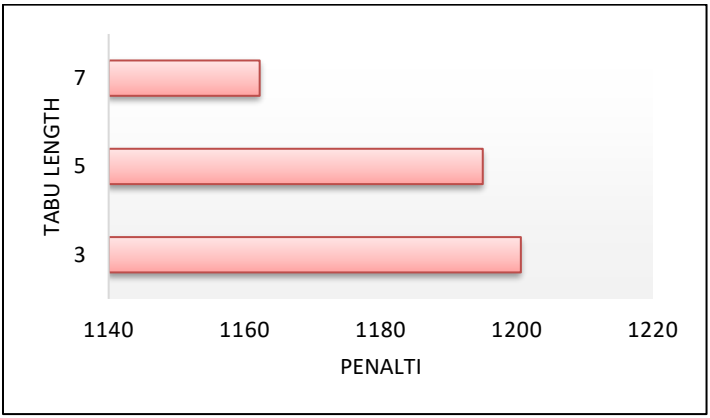


Gambar 6.14 Perbandingan Perubahan Penalti Tabu List (Small)

Sedangkan untuk hasil rata-rata penalty yang disebabkan oleh perubahan *tabu length* atau besar tabu ditampilkan dalam grafik pada Gambar 6.15 dan 6.16



Gambar 6.15 Rata-rata Penalti Perubahan Tabu Length (*Tiny*)



Gambar 6.16 Rata-rata Penalti Perubahan Tabu Length (Snall)

6.5. Perbandingan Hasil Eksperimen

Pada subbab ini dilakukan pembandingan solusi optimasi melalui nilai penalti. Pembandingan solusi optimasi dilakukan dengan pembandingan terhadap hasil *benchmark*. Hasil *benchmark* adalah hasil yang didapatkan dari percobaan menggunakan algoritma lain. Dalam penelitian ini algoritma *Benchmark* yang diujikan dengan *Tabu Search – Simulated Annealing* adalah algoritma *Hill Climbing* dan *Simulated Annealing* tanpa *Tabu.*

Hasil paling optimal dari semua eksperimen dipilih yang nantinya akan dibandingkan dengan hasil *benchmark*. Nilai parameter yang didapatkan dari hasil uji parameter diatas adalah sebagai berikut

1. *Temperature*, 95.
2. *Cooling Rate*, 0,8.
3. *Low Level Heuristics*, 3 (*Move-2 Timeslot, Move-2 Room, Move-2 TimeRoom*)
4. *Tabu Length*, 5.

Dalam penelitian ini algoritma untuk *benchmark* adalah menggunakan algoritma *Hill* Climbing dan algoritma *Simulated Annealing* tanpa *Tabu Search*. Perbandingan hasil optimasi dan hasil *benchmark* ditunjukkan pada Tabel 6.12.

Tabel 6.12 Perbandingan Hasil Optimasi Penelitian Dengan Algoritma Benchmark *Instance Tiny*

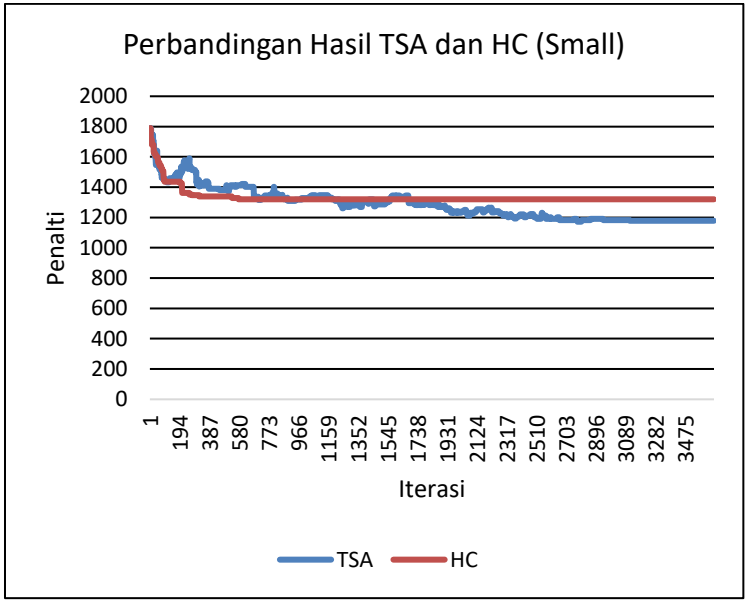
Iteration	Tabu - SA		Simulated Annealing		Hill Climbing	
	Tiny					
	Best	**Average**	**Best**	**Average**	**Best**	**Average**
5000	**5**	**17.9**	10	21.9	16	22.9
10000	**5**	**12.7**	13	28.6	16	19.2
50000	**6**	**12.2**	8	22.4	16	22.5
100000	**5**	**9.5**	9	27	12	27.4
250000	**5**	**9.5**	14	25.6	16	29.8
500000	**4**	**7.6**	9	20.3	12	24
750000	**5**	**9.6**	10	18.6	12	27.2
1000000	**4**	**6.9**	5	22.9	12	20.5

Tabel 6.13 Perbandingan Hasil Optimasi Penelitian Dengan Algoritma Benchmark Instance *Small*

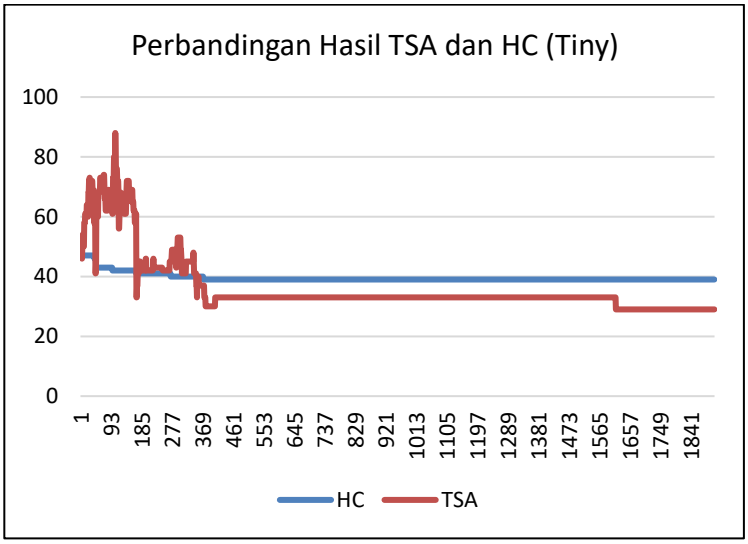
Iteration	Tabu - SA		Simulated Annealing		Hill Climbing	
	Small					
	Best	**Average**	**Best**	**Average**	**Best**	**Average**
5000	**836**	**989.6**	1016	1053.6	1080	1230.8
10000	**712**	**864.2**	812	954	1080	1215.4
50000	**666**	**896.2**	848	990.4	1078	1206
100000	**766**	**893.4**	888	985.6	1068	1158
250000	808	**879.4**	**806**	891.6	1074	1186.6
500000	**736**	**845**	806	942.2	1070	1169.8
750000	**774**	**839.2**	780	913	1074	1189.6
1000000	**766**	**836**	794	913	1074	1175.8

Dari perbandingan nilai penalti pada Tabel 6.12, dapat dilakukan analisis dan disimpulkan bahwa algoritma Tabu-Simulated Annealing yang diimplementasikan pada penelitian tugas akhir ini, mendapatkan hasil yang lebih bagus dan lebih optimum untuk *instance tiny* dan *small*.

Grafik perbandingan algoritma *Tabu Search* dan *Hill Climbing* ditunjukkan pada Gambar 6.



Gambar 6.17 Perbandingan Hasil TSA dan HC (Small)



Gambar 6.18 Perbandingan Hasil TSA dan HC (Tiny)

Pada Gambar 6. Dan 6. Dapat disimpulkan bahwa pada algoritma *Hill Climbing* selalu berhenti pada titik penalti tertentu, hal tersebut yang dinyatakan dengan terjebak pada *local optima*.

Sedangkan untuk algoritma *Tabu Search – Simulated Annealing* terlihat naik turun perolehan penalti, hal ini disebabkan algoritma *tabu – simulated annealing* masih menerima solusi yang tidak lebih baik daripada solusi awal. Hal itu lah yang menyebabkan algoritma *tabu search – simulated annealing* tidak berhenti pada solusi *local optima*

BAB VII
KESIMPULAN DAN SARAN

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

7.1. Kesimpulan

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

1. Algoritma Tabu-Simulated Annealing berbasis Hyper-heuristics mampu menyelesaikan permasalahan penjadwalan mata kuliah (*University Course Timetabling Problem*)..
2. Pada algoritma Tabu-Simulated Annealing Hyper-Heuristics tanpa pengembangan, semua parameter yaitu *initial temperature, cooling rate,* jumlah *low level heuristics*, jumlah *tabu list* dan jumlah iterasi berpengaruh pada solusi yang dihasilkan. Pengaruh paling signifikan adalah perubahan jumlah iterasi, semakin banyak iterasi maka semakin besar kemungkinan menghasilkan penalty yang semakin optimum.
3. Algoritma Tabu-Simulated Annealing Hyper-heuristics lebih unggul dalam menghasilkan solusi yang optimum dibandingkan dengan menggunakan algoritma *Hill Climbing* yang rawan atau sering berhenti pada *local optimum*.

7.2. Saran

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

1. Pada penelitian ini hanya menggunakan dua *instance* yaitu pada *test instance tiny1* dan *small2*. Dalam penelitian kedepan diharapkan dapat menggunakan *test instance* yang lain maupun *early instances.*
2. Dalam penelitian kedepan, diharapkan bisa mengoptimasi *source code* yang telah peneliti buat sehingga dapat mendapatkan hasil yang lebih optimal dari pada hasil penelitian penulis.
3. Penelitian dalam tugas akhir ini hanya melakukan eksperimen sebanyak 5 eksperimen dengan parameter yang berbeda dan iterasi sejumlah 10 kali. Dalam penelitian kedepan, dapat menggunakan variasi parameter lain dalam ekperimen parameter selain yang telah diujicobakan, terutama dengan kombinasi yang lebih kompleks dan mewakili seluruh kombinasi total.
4. Penelitian ini hanya menggunakan *low level heuristics* berupa *move* yaitu *move room* dan *move timeslot*. Untuk mendapatkan hasil yang lebih variatif, *low level heuristics* dapat ditambahkan seperti *swap, reverse,* dan sebagainya. sehingga memungkinan ditemukan solusi yang lebih optimum.
5. Tidak ada pengembangan algoritma yang dilakukan pada penelitian ini. Penemuan dan pengembangan algoritma juga dapat diimplementasikan untuk menemukan hasil yang lebih optimum.

BIODATA PENULIS

Narendra Puspa Adi Negara adalah nama lengkap penulis, lahir di Blitar pada hari Selasa tanggal 04 Juni 1996. Penulis adalah anak kedua dari orang tua bernama Bapak Miranu Triantoro dan Ibu Farida Nursusanti. Penulis menempuh pendidikan dasar di SD Negeri Sananwetan 3 Blitar pada tahun 2004 hingga 2009. Setelah lulus dari sekolah dasar, penulis melanjutkan pendidikan formal di bangku sekolah menengah pertama di SMP Negeri 1 Blitar dan lulus pada tahun 2012. Pada tahun yang sama, penulis melanjutkan pendidikan formal di SMA Negeri 1 Blitar dan lulus pada tahun 2015. Setelah selesai menempuh Pendidikan menengah atas, pada tahun 2015, penulis melanjutkan pendidikan tinggi di Departemen Sistem Informasi, Fakultas Teknologi Informasi dan Komunikasi, Insitut Teknologi Sepuluh Nopember (ITS) Surabaya, melalui jalur Seleksi Bersama Masuk Perguruan Tinggi Negeri (SBMPTN) tahun 2015.

Selama menjalani pendidikan tinggi di FTIK ITS, penulis tergolong aktif dalam berbagai kegiatan, baik organisasi maupun kepanitiaan. Penulis pernah menjadi koordinator berbagai acara seperti koordinator logistik *Information System Expo 2017*, koordinator perkamjin Bursa Karir ITS-37 2019, dan berbagai staf acara jurusan maupun institut. Penulis pernah menjabat menjadi BPK atau Badan Pengawas Kepengurusan Himpunan Mahasiswa Sistem Informasi pada tahun 2016.

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

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

DAFTAR PUSTAKA

[1] S. Daskalaki and T. Birbas, "Efficient solutions for a university timetabling problem through integer programming," *European Journal of Operational Research*, vol. 160, no. 1, pp. 106–120, Jan. 2005.

[2] S. Abdullah and H. Turabieh, "Generating University *Course* Timetable Using Genetic Algorithms and Local Search," in *2008 Third International Conference on Convergence and Hybrid Information Technology*, Busan, Korea, 2008, pp. 254–260.

[3] K. Socha, J. Knowles, and M. Sampels, "A MAX-MIN Ant System for the University *Course* Timetabling Problem," in *Ant Algorithms*, vol. 2463, M. Dorigo, G. Di Caro, and M. Sampels, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 1–13.

[4] A. Muklason, "Solver Penjadwal Ujian Otomatis Dengan Algoritma Maximal Clique dan Hyper-heuristics," p. 8, 2017.

[5] A. Hertz, "Tabu search for large scale timetabling problems," *European Journal of Operational Research*, vol. 54, no. 1, pp. 39–47, Sep. 1991.

[6] N. Basir, W. Ismail, and N. M. Norwawi, "A Simulated Annealing for Tahmidi *Course* Timetabling," *Procedia Technology*, vol. 11, pp. 437–445, 2013.

[7] M. P. Khairunnisa, B. Pramono, and R. A. Saputra, "Implementasi Algoritma Tabu Search pada Aplikasi Penjadwalan Mata Pelajaran (Studi Kasus: SMA Negeri 4 Kendari)," vol. 2, p. 10, Dec. 2016.

[8] H. A. Shiddiq, Sugiono, and C. F. M. Tantrika, "Implementasi Algoritma Simulated Annealing pada Penjadwalan Produksi untuk Meminimasi Makespan (Studi

Kasus di PT. Gatra Mapan, Karang Ploso, Malang)," vol. 3, no. 1, p. 10.

[9] R. Battiti and G. Tecchiolli, "Simulated annealing and Tabu search in the long run: A comparison on QAP tasks," *Computers & Mathematics with Applications*, vol. 28, no. 6, pp. 1–8, Sep. 1994.

[10] S. Abdullah, "Heuristic Approaches for University Timetabling Problems," p. 226, Jun. 2006.

[11] A. Wren, "Scheduling, timetabling and rostering — A special relationship?," in *Practice and Theory of Automated Timetabling*, vol. 1153, E. Burke and P. Ross, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 46–75.

[12] A. Schaerf, "A Survey of Automated Timetabling," p. 41.

[13] E. K. Burke and S. Petrovic, "Recent research directions in automated timetabling," *European Journal of Operational Research*, vol. 140, no. 2, pp. 266–280, Jul. 2002.

[14] T. Muller, H. Rudova, and Z. Mullerova, "University *course* timetabling and International Timetabling Competition 2019," p. 27, 2019.

[15] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-Heuristics: An Emerging Direction in Modern Search Technology," in *Handbook of Metaheuristics*, vol. 57, F. Glover and G. A. Kochenberger, Eds. Boston: Kluwer Academic Publishers, 2003, pp. 457–474.

[16] E. K. Burke, *Search methodologies*. New York: Springer, 2013.

[17] A. Marom, "Optimasi Penjadwalan Mata Kuliah Otomatis Menggunakan Algoritma Tabu-Simulated Annealing Hyper-Heuristics," p. 189, 2019.

[18] R. Davidson and D. Harel, "Drawing graphs nicely using simulated annealing," *ACM Transactions on Graphics*, vol. 15, no. 4, pp. 301–331, Oct. 1996.

LAMPIRAN A: Hasil Optimasi Penjadwalan

Tabel A.1 Hasil Optimasi Penjadwalan *Instance Tiny*

Class ID	Day	*Room*	Start	Week
1	1111000 (Senin - Kamis)	45	114 - 136 (09.30 - 10.40)	111111111 (Setiap Minggu)
2	1111000 (Senin - Kamis)	45	96 - 111 (08.00 - 09.15)	111111111 (Setiap Minggu)
3	1111000 (Senin - Kamis)	24	138 - 153 (11.30 - 12.45)	111111111 (Setiap Minggu)
4	0010100 (Rabu, Jumat)	59	96 - 130 (08.00 - 10.50)	111111111 (Setiap Minggu)
5	1111000 (Senin - Kamis)	8	96 - 111 (08.00 - 09.15)	111111111 (Setiap Minggu)
6	1111000 (Senin - Kamis)	40	138 - 160 (11.30 - 13.21)	111111111 (Setiap Minggu)

Class ID	Day	*Room*	Start	Week
7	1111000 (Senin - Kamis)	28	126 - 148 (10.30 - 12.20)	111111111 (Setiap Minggu)
8	1111000 (Senin - Kamis)	23	120 - 142 (10.00 - 11.50)	111111111 (Setiap Minggu)
9	1111000 (Senin - Kamis)	8	132 - 154 (11.00 - 12.50)	111111111 (Setiap Minggu)
10	1111000 (Senin - Kamis)	25	120 - 142 (10.00 - 11.50)	111111111 (Setiap Minggu)
11	1111000 (Senin - Kamis)	22	96 - 118 (08.00 - 09.50)	111111111 (Setiap Minggu)
12	1111000 (Senin - Kamis)	29	96 - 111 (08.00 - 09.15)	111111111 (Setiap Minggu)
13	1111000 (Senin - Kamis)	38	120 - 142 (10.00 - 11.50)	111111111 (Setiap Minggu)

Class ID	Day	*Room*	Start	Week
14	1111000 (Senin - Kamis)	26	144 - 159 (12.00 - 13.15)	111111111 (Setiap Minggu)
15	1111000 (Senin - Kamis)	38	96 - 118 (08.00 - 09.50)	111111111 (Setiap Minggu)
16	1111000 (Senin - Kamis)	62	96 - 154 (08.00 - 12.50)	111111111 (Setiap Minggu)
17	1111000 (Senin - Kamis)	61	144 - 166 (12.00 - 13.50)	111111111 (Setiap Minggu)
18	1111000 (Senin - Kamis)	27	96 - 118 (08.00 - 09.50)	111111111 (Setiap Minggu)
19	1111000 (Senin - Kamis)	12	97 - 118 (08.00 - 09.50)	111111111 (Setiap Minggu)
20	1111000 (Senin - Kamis)	24	98 - 118 (08.00 - 09.50)	111111111 (Setiap Minggu)