OPTIMASI PENJADWALAN SIDANG SKRIPSI MENGGUNAKAN ALGORITME GENETIKA TERDISTRIBUSI
(STUDI KASUS : PRODI TEKNIK INFORMATIKA FAKULTAS ILMU KOMPUTER UNIVERSITAS BRAWIJAYA)

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Putri Bunga Rahmalita
NIM: 155150201111037

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

OPTIMASI PENJADWALAN SIDANG SKRIPSI MENGGUNAKAN ALGORITME GENETIKA TERDISTRIBUSI (STUDI KASUS : PRODI TEKNIK INFORMATIKA FAKULTAS ILMU KOMPUTER UNIVERSITAS BRAWIJAYA)

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Putri Bunga Rahmalita
NIM: 155150201111037

Skripsi ini telah diuji dan dinyatakan lulus pada
03 Mei 2019
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Agus Wahyu Widodo, S.T, M.Cs	Drs. Muh Arif Rahman, M.Kom
NIP: 19740805 200112 1 001	NIP: 19660423 199111 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710581 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 29 April 2019

Putri Bunga Rahmalita

NIM: 155150201111037

PRAKATA

Puji dan syukur penulis panjatkan kepada Allah SWT atas segala rahmat dan ata izin-Nya penulis dapat menyelesaikan skripsi yang berjudul "Optimasi Penjadwalan Sidang Skripsi Menggunakan Algoritme Genetika Terdistribusi (Studi Kasus: Prodi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya)".

Selama pengerjaan skripsi ini tidak lepas dari bantuan, bimbingan, semangat serta doa yang diberikan oleh berbagai pihak kepada penulis. Oleh karena itu, ucapan terimakasih akan disampaikan kepada:

1. Keluarga penulis, Elita Alam, Thifaal Rafkha, Teddy Hidayat, Fitri Aida yang selalu memberikan dukungan, semangat, dan doa selama proses pengerjaan skripsi dan selalu sabar menampung segala keluh kesah penulis selama mengerjakan skripsi.
2. Bapak Agus Wahyu Widodo, S.T, M.Cs selaku dosen pembimbing 1 yang selalu meluangkan waktu untuk membimbing dan mengarahkan penulis dengan sabar, segala kritik dan saran yang membangun, dukungan dan doa yang diberikan kepada penulis.
3. Bapak Drs. Muh Arif Rahman, M.Kom selaku dosen pembimbing 2 yang selalu meluangkan waktu untuk membimbing dan mengarahkan penulis dalam mengerjakan dokumen skripsi serta kritik dan saran yang sangat membantu penulis.
4. Moch Satrio Laksono Putro yang selalu bersedia menemani, mendoakan dan memberikan dukungan kepada penulis selama mengerjakan skripsi.
5. Sahabat-sahabat penulis, Diajeng Tania Ananda, Arsya Monica Pravina, Desy Andriani, Putri Stephanie Lesilolo, Afina Putri dan Carlista Naba yang selalu membantu, memberi dukungan, dan mendoakan dalam pengerjaan skripsi.
6. Sahabat-sahabat penulis Muhammad Rafi Farhan, Rafely Chandra, Muhammad Vidi, dan Agri yang bersedia membantu dengan sabar, dan memberikan semangat kepada penulis dalam mengerjakan skripsi.
7. Sahabat-sahabat penulis Rifkho Achmad Bawazir, Talitha Raissa, Azizah Noor Rahman, Dewi Yunisa, Audi P, Fajar, M. Luthfi Abdurrahman, Vicrhanica Rhamzi, Azis Permana, Rizky Armanda, Rizqinov, Aldi, Devan, Aat, Wiwit, Femi dan Fykar yang selalu memberikan semangat dan doa dalam mengerjakan skipsi.
8. Adik-adik penulis di Kebiroan Pusat Komunikasi dan Informasi Badan Eksekutif Mahasiswa FILKOM UB yaitu Anggtaslih Mutiara Fathony, Nadya Rahmasari, Muhammad Ghiffari, Hanif Naufal Ashari, Hawim Mahfudah, Miftachul Robherta, Hetti Marthaningrum, Hummam Aly Baziyad, Ade Listiawan, Inas Nabila, Gerwin Jonathan Henri, Muhammad Dandi Darojat, Renaldy Dwisma Febriarta, Felicia Marvela Evanita, Muhammad Fadhil Risyad, dan Mochamad Dwi Fadly yang selalu memberikan semangat dan doa kepada penulis.
9. Teman-teman seperjuangan Olimpiade Brawijaya yang memberi dukungan dan doa kepada penulis Ima Fitananda, Mohammad Al-Rasyid, Nadya Putri Rahayu, Noviola Berlianita, Elfatina Risti, Bella Rayani.

10. Teman-teman seperjuangan BEM FILKOM yang memberi dukungan dan doa kepada penulis Dzaky Fadhillah Guci, Fathanasa, Agung Wahyu Setiobudi, Alisryad Pahlevi, Anggara Dwi P, Muhammad Fattah Na'im.
11. Teman-teman Kedai Kopi Antara yaitu Mas Fikri, Ardhie, Azim, Lia, Kyra, Elsa, Mikel, Clara, Wahyu, Bolenk, Bless, dan Markipik yang memberi semangat dan doa kepada penulis.
12. Semua pihak yang tidak dapat disebutkan satu persatu, terimakasih banyak atas bantuan, semangat dan doa yang diberikan kepada penulis.

Penulis menyadari bahwa skripsi yang dibuat masih jauh dari kata sempurna, maka dari itu kritik dan saran yang membangun sangat diharapkan penulis untuk perbaikan kedepannya. Akhir kata, semoga skripsi ini bermanfaat bagi pembaca.

Malang, 29 April 2019

Penulis
putribungarahmalita@gmail.com

ABSTRAK

Putri Bunga Rahmalita, Optimasi Penjadwalan Sidang Skripsi Dengan Menggunakan Algoritme Genetika Terdistribusi (Studi Kasus: Prodi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya)

Pembimbing: Agus Wahyu Widodo, S.T, M.Cs dan Drs. Muh Arif Rahman, M.Kom

Ada beberapa kendala pada penjadwalan sidang skripsi di Prodi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya yang membuat penjadwalan sidang skripsi tidak efektif. Perbedaan yang signifikan pada jumlah mahasiswa dan dosen pada tahun 2017 serta pendaftaran sidang skripsi dalam waktu yang berdekatan seringkali menjadi masalah dalam menjadwalkan sidang skripsi. Penjadwalan yang tidak efektif dan tidak efesien akan memakan waktu yang lama dalam pengerjaannya. Maka dari itu, dibuat sistem yang dapat melakukan penjadwalan sidang skripsi dengan menggunakan metode algoritme genetika terdistribusi.

Algoritme genetika terdistribusi melakukan inisialisasi kromosom secara acak. Selanjutnya populasi akan dipecah menjadi beberapa subpopulasi, dan akan melalui tahap reproduksi yaitu *crossover* dengan metode *one-cut point* dan mutasi dengan metode *reciprocal exchange mutation* dan *random mutation.* Setelah itu dilakukan evaluasi untuk menghitung nilai *fitness*. Selanjutnya dengan metode *elitism selection* pada proses seleksi akan dipilih individu yang akan dipertahankan untuk generasi selanjutnya. Pada algoritme genetika terdistribusi akan dilakukan proses migrasi untuk menambah keberagaman individu dengan menukar individu dari subpopulasi satu dengan lainnya.

Berdasarkan hasil pengujian, nilai parameter yang optimal pada penjadwalan sidang skripsi yaitu dengan jumlah populasi sebesar 11, jumlah generasi sebesar 1750, *crossover rate* 0,7 mutation *rate* 0,3 dan jumlah sub populasi 5 dengan rata-rata nilai *fitness* sebesar 0.00010232. Dari hasil pengujian, jika semakin banyak jumlah populasi dan generasi, maka semakin luas daerah pencarian solusi, akan tetapi waktu komputasi menjadi semakin lama.

Kata kunci: penjadwalan, skripsi, algoritme genetika terdistribusi

ABSTRACT

Putri Bunga Rahmalita, Thesis Scheduling Optimization Using Distributed Genetic Algorithm (Case Study: Informatics Engineering Study Program, Faculty of Computer Science, Brawijaya University)

Supervisors: Agus Wahyu Widodo, S.T, M.Cs and Drs. Muh Arif Rahman, M.Kom

There are several problems on thesis scheduling in Informatics Engineering Study Program Faculty of Computer and Science Brawijaya University that makes thesis scheduling ineffective. Significant differences in the number of students and lecturers in 2017 as well as thesis trial registration in the adjacent time is often a problem in scheduling thesis hearings. Ineffective and inefficient scheduling will take a long time in the process. Therefore, a system can be made using a distributed genetic algorithm method to do thesis trial scheduling.

Distributed genetic algorithms randomize chromosomes. Furthermore, the population will be divided into several subpopulations, and will go through the reproductive stage, namely *crossover* and mutation, then through evaluation to calculate the *fitness* value. After that, the individual *elitism selection* method in the selection process will be selected for the next generation. In the distributed genetic algorithm a migration process will be carried out to increase the diversity of individuals by exchanging individuals from one subpopulation to another.

Based on the results of the test, the optimal parameter value in the thesis trial scheduling is the population of 11, the number of generations is 1750, the *crossover rate* is 0.7 mutation *rate* 0.3 and the number of sub-populations is 5 with an average *fitness* value of 0.00010232. From the results of the test, if there is more population and generation, the wider the search area for the solution will be, with a cost of a longer computing time.

Keywords: *scheduling*, *thesis*, *distributed genetic algorithms*

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Sidang skripsi akan dihadapi oleh seorang mahasiswa untuk diuji hasil penelitian akhir yang telah dilakukan untuk mendapatkan gelar sarjana. Untuk merencanakan dan membantu dalam melakukan sidang skripsi agar lebih optimal dibutuhkan sebuah penjadwalan sidang skripsi. Penjadwalan yang tidak efektif dan tidak efesien akan memakan waktu yang lama. Sebuah kegiatan akan berjalan dengan baik jika adanya pengaturan waktu pada penjadwalan yang efektif agar dapat meminimalkan kendala (Yang and Jat, 2011).

Prodi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya (FILKOM UB) merupakan prodi dengan jumlah mahasiswa sebanyak 565 orang untuk angkatan 2014 dan 955 untuk angkatan 2015 (FILKOM, 2018). Sedangkan jumlah dosen pada prodi Teknik Informatika sebanyak 60 orang (FILKOM UB, 2018). Dengan memperhatikan jumlah mahasiswa dan jumlah dosen yang memiliki perbedaan cukup besar, dapat membuat penjadwalan sidang skripsi semakin lama. Selain itu, jika banyak mahasiswa yang mendaftar untuk melakukan sidang skripsi pada periode akhir semester dan dalam waktu yang berdekatan maka seringkali bagian akademik mendapatkan kesulitan ketika akan menjadwalkan sidang skripsi. Beberapa kendala lainnya adalah pada saat mengatur jadwal dosen satu dengan dosen lainnya yang memiliki jadwal yang berbeda sehingga harus menyamakan jadwal dosen yang menjadi penguji dan pembimbing skripsi. Selain itu juga saat mengatur jadwal ruangan yang digunakan untuk sidang skripsi, karena beberapa ruangan digunakan untuk kegiatan belajar mengajar sehingga hanya terdapat beberapa ruangan yang kosong untuk melaksanakan sidang skripsi. Dengan adanya kendala tersebut pengaturan secara manual akan memakan waktu yang lama, karena penjadwalan sidang skripsi membutuhkan penempatan waktu, ruangan, penguji sidang skirpsi, pembimbing skripsi yang memiliki jadwal terbaik untuk melakukan sidang skripsi.

Ada beberapa penelitian yang telah dilakukan sebelumnya mengenai penjadwalan, salah satunya yaitu penjadwalan mata pelajaran menggunakan metode *tabu search* (Khoirul L.M.A., Widodo and Setiawan, 2017). Akan tetapi pada penelitian ini metode tabu search masih memiliki total penalti sebesar 302 untuk penjadwalan pada 8 kelas. Sehingga peneliti menyarankan agar penelitian ini dapat dikembangkan lebih lanjut dengan menambahkan penalti pelanggaran lain dan juga dapat menggabungkan metode *tabu search* dengan metode heuristic lainnya agar lebih optimal. Penelitian lainnya mengenai penjadwalan dengan objek penjadwalan dinas pegawai pada PT Kereta Api Indonesia (KAI) daerah operasi 7 Stasiun Besar Kediri menggunakan algoritme genetika (Ula, Ratnawati and Wicaksono, 2018).

Solusi yang dihasilkan oleh algoritme genetika masih sering terjebak dalam daerah yang sama atau daerah lokal optimum, hal ini dapat terjadi karena kurangnya variasi individu dalam populasi (Mahmudy, 2015). Maka dari itu pada

penelitian ini akan menggunakan metode algoritme genetika terdistribusi, agar dapat membuat keragaman individu yang akan menghasilkan solusi yang optimal. Sebelumnya penelitian dengan menggunakan algoritme genetika terdistribusi telah dilakukan, seperti penelitian mengenai keputusan pengemasan dalam masalah container (Vladimir I et al., 2002). Penelitian ini membuktikan bahwa periode migrasi akan memberikan kualitas operasi algoritme yang lebih baik. Selain itu penelitian lainnya yang menerapkan algoritme genetika terdistribusi adalah pada optimasi portofolio saham (Raissa, Widodo and Setiawan, 2017). Menurut peneliti untuk mendapatkan hasil yang lebih baik dapat menggunakan mekanisme struktur algoritme genetika terdistribusi yang berbeda.

Berdasarkan permasalahan tersebut serta dari penelitian sebelumnya, penulis akan menerapkan sistem untuk optimasi penjadwalan sidang skripsi menggunakan algoritme genetika terdistribusi. Diharapkan dalam penelitian ini algoritme genetika terdistribusi dapat memberikan hasil penjadwalan sidang skripsi yang optimal.

1.2 Rumusan Masalah

1. Bagaimana menyatakan solusi dari penjadwalan sidang skripsi menjadi struktur algoritme genetika terdistribusi?
2. Bagaimana operator-operator algoritme genetika terdistribusi diterapkan pada masalah optimasi penjadwalan sidang skripsi?
3. Berapa nilai parameter algoritme genetika terdistribusi yang digunakan untuk mendapatkan hasil yang optimal?

1.3 Tujuan

1. Mendapatkan struktur algoritme genetika terdistribusi sebagai solusi dari penjadwalan sidang skripsi.
2. Mendapatkan struktur operator-operator algoritme genetika terdistribusi dari penjadwalan sidang skripsi.
3. Mendapatkan nilai yang optimal dari parameter algoritme genetika terdistribusi.

1.4 Manfaat

Penelitian ini diharapkan memberikan manfaat dengan mendapatkan suatu model algoritme genetika terdistribusi berikut dengan nilai parameternya yang sesuai agar penjadwalan sidang skripsi menjadi lebih efektif dan efisien.

1.5 Batasan Masalah

Pembahasan dalam penelitian ini perlu diberikan batasan agar lebih terarah dan menghindari pembahasan yang tidak sesuai topik dan terlalu kompleks, maka dibuatlah batasan masalah sebagai berikut:

1. Data yang digunakan pada penelitian ini merupakan data dari Akademik Fakultas Ilmu Komputer Universitas Brawijaya.
2. Data jadwal dosen yang digunakan hanya jadwal mengajar dosen tersebut di Fakultas Ilmu Komputer Universitas Brawijaya pada semester genap 2017/2018.
3. Data mahasiswa yang digunakan hanya mahasiswa prodi teknik informatika angkatan 2014 yang telah lulus pada semester genap 2017/2018.
4. Penelitian ini melakukan optimasi penjadwalan sidang skripsi mingguan.
5. Data ruangan yang digunakan pada penelitian ini adalah 8 ruangan, yaitu ruang seminar di gedung F lantai 4 Fakultas Ilmu Komputer Universitas Brawijaya.
6. Penelitian ini menggunakan 6 sesi untuk penjadwalan sidang skripsi.

1.6 Sistematika Pembahasan

Sistematika pembahasan pada penulisan skripsi adalah sebagai berikut:

BAB I PENDAHULUAN

Bab I memuat latar belakang masalah, rumusan masalah, tujuan, manfaat, batasan masalah, serta sistematika pembahasan.

BAB II LANDASAN KEPUSTAKAAN

Bab II memuat tentang kajian pustaka dan teori-teori yang berkaitan dengan topik penelitian dari penulis yang dijadikan acuan untuk menerapkan algoritme genetika terdistribusi pada penentuan komposisi optimal untuk penjadwalan skripsi.

BAB III METODOLOGI PENELITIAN

Bab III memuat alur kerja yang akan dilakukan untuk menerapkan algoritme genetika terdistribusi dalam menentukan komposisi optimal pada penjadwalan sidang skripsi.

BAB IV PERANCANGAN

Bab IV memuat tentang bagaimana rancangan untuk menentukan hasil yang optimal pada penjadwalan sidang skripsi dengan algoritme genetika terdistribusi.

BAB V HASIL DAN PEMBAHASAN

Bab V memuat tentang pembahasan dari hasil yang didapatkan dari optimasi penjadwalan sidang skripsi.

BAB VI PENUTUP

Bab VI memuat kesimpulan dari penelitian ini yang meliputi hasil, kelebihan dan kekurangan dari penentuan komposisi optimal dengan algoritme genetika terdistribusi, tingkat akurasi. Selain itu bab ini juga berisi saran untuk penelitian selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini berisi penjabaran mengenai penelitian-penelitian sebelumnya serta teori-teori yang berhubungan dengan penelitian objek dan metode penelitian ini yaitu penjadwalan, algoritme genetika, serta algoritme genetika terdistribusi.

2.1 Kajian Pustaka

Penelitian sebelumnya mengenai masalah penjadwalan telah dilakukan dengan menggunakan metode *tabu search* mengenai penjadwalan mata pelajaran (Khoirul L.M.A., Widodo and Setiawan, 2017). Pada penelitian ini peneliti ingin mengetahui hasil optimasi penjadwalan mata pelajaran menggunakan metode *tabu search* dengan mempertimbangkan ketersediaan guru mata pelajaran, jumlah kelas, kurikulum sekolah, pemilihan waktu untuk mata pelajaran tertentu, serta alokasi waktu setiap mata pelajaran. Metode Tabu Search dilakukan dengan solusi awal yang berupa jadwal akan dibangkitkan secara acak, setelah itu solusi akhirnya merupakan jadwal mata pelajaran dengan nilai total penalti paling kecil pada tiap iterasi. Pada penelitian ini ditunjukkan bahwa total nilai penalti dapat dipengaruhi oleh hasil pembangkitan solusi awal jadwal mata pelajaran dan jumlah kelas yang dijadwalkan. Pada pengujiannya, semakin banyak jumlah kelas yang dijadwalkan, jumlah total penalti yang didapatkan akan semakin meningkat. Hal ini dikarenakan kompleksitas penyusunan dan sumber daya yang harus dijadwalkan meningkat, sedangkan jumlah guru mata pelajaran tertentu dan slot waktu yang dijadwalkan tetap. Hasil terbaik pada penelitian ini terdapat pada percobaan ke 4 dengan jumlah iterasi sebanyak 20, pengujian kedua dengan nilai penalti total 302 (Khoirul L.M.A., Widodo and Setiawan, 2017).

Selain metode *tabu search*, metode lain seperti algoritme genetika juga digunakan dalam mengatasi masalah optimasi, seperti optimasi penjadwalan pada dinas pegawai PT Kereta Api Indonesia (KAI) Daerah Operasi 7 Stasiun Besar Kediri (Ula, Ratnawati and Wicaksono, 2018). Pada penelitian tersebut sistem penjadwalan dikatakan benar apabila jadwal yang dihasilkan telah sesuai dengan aturan yang telah ditetapkan, akan tetapi karena algoritme genetika berjalan secara acak, maka jadwal yang dihasilkan untuk penjadwalan dinas pegawai juga secara acak. Pada algoritme genetika masih kurangnya variasi individu dalam sebuah populasi menyebabkan solusi selalu terjebak di daerah yang sama. Untuk mengatasi hal tersebut telah dilakukan penelitian menggunakan algoritme genetika terdistribusi untuk menyelesaikan masalah seperti keputusan pengemasan dalam masalah container (Vladimir I et al., 2002) dan masalah optimasi portofolio saham (Raissa, Widodo and Setiawan, 2017). Pada kedua penelitian tersebut algoritme genetika terdistribusi dinilai lebih baik dibandingkan dengan algoritme genetika biasa karena adanya pemecahan populasi menjadi beberapa sub-populasi, selain itu dengan adanya proses migrasi yang dapat menambah keberagaraman individu. Pada algoritme genetika terdistribusi proses reproduksi juga dapat dilakukan dengan cara yang berbeda untuk menghasilkan

keturunan yang lebih baik, seperti dapat menggunakan beberapa metode mutasi dan *crossover*. Hasil yang didapatkan kedua penelitian ini menyatakan bahwa algoritme genetika terdistribusi yang digunakan sudah baik untuk mengatasi permasalahan tersebut. Rangkuman kajian pustaka terdapat dalam sebuah tabel yaitu pada Tabel 2.1.

Tabel 2.1 Kajian Pustaka

No	Judul	Metode	Hasil Penelitian
1	Optimasi Penjadwalan Mata Pelajaran Menggunakan Metode Tabu Search (Studi Kasus : SMKN 2 Singosari) (Khoirul L.M.A., Widodo and Setiawan, 2017)	Tabu Search	Dengan metode Tabu Search, pada solusi awal yang berupa jadwal akan dibangkitkan secara random, setelah itu akan dicari solusi akhirnya dengan Tabu List berbentuk *array* yang merupakan solusi jadwal mata pelajaran dengan nilai total penalti paling kecil dari tiap iterasi. Hasil terbaik pada penelitian ini terdapat pada percobaan ke 4 dengan jumlah iterasi sebanyak 20, pengujian kedua dengan nilai penalti total 302.
2	Penjadwalan Dinas Pegawai Menggunakan Algoritme Genetika Pada PT Kereta Api Indonesia (KAI) Daerah Operasi 7 Stasiun Besar Kediri (Ula, Ratnawati and Wicaksono, 2018)	Algoritme Genetika	Penelitian ini berhasil melakukan penjadwalan dinas pegawai dengan melakukan pembangkitan representasi nilai kromosom secara acak dibangkitkan untuk membentuk jadwal dinas pegawai dengan panjang gen sebanyak 98 yang mewakili 14 pegawai dalam 7 hari. Pada penelitian ini di generasi ke 50 telah mengalami konvergen. Dapat dikatakan pada algoritme genetika individu cepat mengalami konvergensi dini karena kurangnya variasi individu.
3	The Application of the Distributed Genetic Algorithm to the Decision of the Packing in	Algoritme Genetika Terdistribusi	Penelitian ini menemukan konfigurasi algoritme yang paling efektif, atas dasar hasil temuan itu ditarik kesimpulan bahwa jenis masalah pada

	Containers Problem (Vladimir I et al., 2002)		penelitian ini dianggap dapat diselesaikan secara efektif dengan bantuan algoritme genetika. Hasil dari penelitian ini ditemukan 82,8125% sebagai keputusan terbaik dengan ukuran populasi/ elit sebesar 8/8, 32/16 dan 512/32.
4	Penentuan Portofolio Saham Optimal Menggunakan Algortime Genetika Terdistribusi (Raissa, Widodo and Setiawan, 2017)	Algoritme Genetika Terdistribusi	Pada penelitian ini parameter yang paling berpengaruh adalah jumlah sub-populasi karena pada saat ada interaksi antar sub-populasi dapat menjaga keragaman variasi individu. Hasil penelitian ini menunjukkan jumlah ukuran populasi optimal sebanyak 80 populasi, lalu untuk jumlah generasi optimal yaitu 150 generasi. Nilai cr optimal sebesar 0,8 dan nilai mr optimal sebesar 0,2, pada mr rata-rata nilai *fitness* sebesar 0,8939737. Sedangkan jumlah sub-populasi optimal adalah 10 dengan rata-rata *fitness* 0,9020893. Peneliti mengatakan bahwa algoritme genetika terdistribusi yang digunakan sudah cukup baik dalam menentukan portofolio saham optimal.

Berdasarkan kajian pustaka yang telah diuraikan, dapat diketahui bahwa ada beberapa metode yang dapat digunakan untuk melakukan optimasi, akan tetapi setiap metode memiliki kekurangan dan kelebihannya masing-masing. Algoritme genetika merupakan salah satu metode yang baik untuk melakukan optimasi dalam jumlah waktu yang wajar, akan tetapi jika diterapkan pada masalah yang lebih kompleks, waktu yang dibutuhkan untuk melakukan proses akan menjadi lebih lama dan biasanya solusi masih terjebak dalam daerah yang sama atau pada daerah optimum lokal. Maka dari itu penulis menggunakan algoritme genetika terdistribusi agar solusi dapat dihasilkan secara optimal, karena pada algoritme genetika terdistribusi populasi akan dipecah menjadi sub-populasi dan ditambahkan operasi migrasi untuk menambah keberagaman individu.

2.2 Penjadwalan Sidang Skripsi

Penjadwalan merupakan proses yang berhubungan dengan pengalokasian sumber daya untuk suatu kegiatan selama periode waktu tertentu yang berfungsi untuk mengoptimalkan satu atau lebih tujuan (Đumić et al., 2018). Penjadwalan ini digunakan untuk memanfaatkan sumber daya yang tersedia dengan dirancang agar memenuhi semua persyaratan, karena jika tidak hal tersebut hanya akan membuang-buang waktu dan sumber daya (Jain, Jain and Chande, 2010).

Penjadwalan sidang skripsi pada Fakultas Ilmu Komputer Universitas Brawijaya telah dilakukan menggunakan sistem untuk penjadwalan. Akan tetapi, pada sistem tersebut masih memiliki kekurangan yaitu sulitnya mengatur jadwal dosen penguji yang memiliki keminatan yang sama dengan mahasiswa yang akan diuji, sehingga menyebabkan penjadwalan sidang skripsi seringkali dilakukan secara manual. Penjadwalan sidang skripsi dilakukan dengan mempertimbangkan kesesuaian topik skripsi yang dibawakan oleh mahasiswa, setelah itu akan dilihat daftar dosen penguji yang sesuai dengan keminatan mahasiswa tersebut. Setelah itu, akan dilihat jadwal dari dosen penguji dan dosen pembimbing skripsi yang sesuai untuk melaksanakan sidang skripsi.

2.3 Algoritme Genetika

Algoritme genetika (*genetic algorithm,* GA) merupakan algoritme berbasis populasi yang memungkinkan digunakan sebagai solusi optimal dari sebuah masalah dengan ruang pencarian yang sangat luas dan kompleks. Algoritme genetika melakukan proses pencarian dan optimasi dengan cara yang berbeda dibandingkan dengan proses biasa (Michalewicz, 1996).

Seiring dengan perkembangan teknologi saat ini, algoritme genetika juga semakin berkembang untuk dapat menyelesaikan permasalahan di berbagai bidang. Algoritme genetika juga dapat digunakan untuk menyelesaikan masalah yang kompleks dengan banyak variabel seperti variabel kontinyu, diskrit atau campuran keduanya (Haupt and Haupt, 2004). Algoritme genetika bekerja dengan menirukan proses genetika dan seleksi alami untuk menghasilkan kromosom terbaik setelah melewati sekian generasi. Selanjutnya kromosom terbaik akan diuraikan untuk menjadi sebuah solusi yang diharapkan mendekati nilai optimal (Mahmudy, 2015).

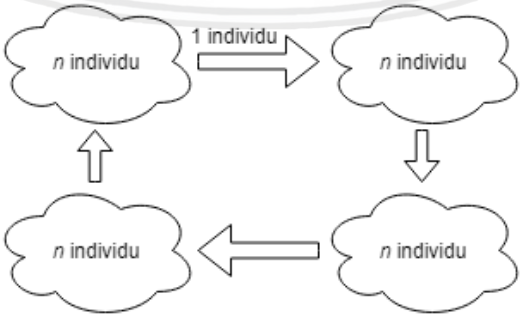
Pada algoritme genetika, kromosom dalam populasi akan dibangkitkan secara acak. Setiap kromosom dalam sebuah populasi akan menjalani satu atau lebih proses reproduksi untuk menghasilkan keturunan yang nantinya akan menjadi generasi berikutnya dari solusi. Pada saat yang sama, algoritma ini seharusnya dapat mempertahankan keragaman individu sehingga dapat mencari ruang pencarian yang besar. Hal tersebut membuat GA menjadi algoritme pencarian dan optimisasi yang baik (Salman Yussof, 2012).

2.4 Algoritme Genetika Terdistribusi

Algoritme genetika terdistribusi (*distributed genetic algorithms*, DGA) menggunakan beberapa sub-populasi dan tiap subpopulasi menggunakan operator genetika yang berbeda pada tiap prosesnya seperti metode *crossover*, mutasi, dan seleksi yang berbeda. Pada dasarnya algoritme genetika terdistribusi merupakan algoritme yang berasal dari pengembangan algoritme genetika. Pada algoritme genetika terdistribusi sebuah populasi akan dibagi menjadi unit yang lebih kecil yang disebut sub-populasi yang nantinya akan dikembangkan secara paralel dan kemudian memungkinkan untuk terjadinya perpindahan antar sub-populasi (Indra and Subanar, 2014). Sub-populasi dikembangkan dan diproses secara paralel karena komputasi secara paralel dapat meningkatkan kinerja komputasi dari algoritma, karena itu algoritme genetika terdistribusi sering disebut *parallel genetic algorithms* (PGA) (Defersha and Chen, 2010).

Pada algoritme genetika biasanya solusi akan sering terjebak dalam daerah yang sama atau pada daerah optimum lokal yang disebabkan oleh kurangnya variasi atau keragaman individu pada suatu populasi (Mahmudy, 2015). Adanya algoritme genetika terdistribusi digunakan untuk menambah keragaman individu, sehingga hasil yang didapatkan akan lebih optimal. Setiap sub-populasi akan diterapkan proses *crossover,* mutasi dan seleksi dengan operasi yang berbeda pula untuk melihat operasi mana yang paling optimal untuk menyelesaikan sebuah masalah. Pada DGA akan ada proses yang disebut migrasi, dimana kromosom diambil dari satu sub-populasi untuk dimutasi atau digantikan dengan kromosom di sub-populasi lain. Proses migrasi digunakan untuk menambah variasi individu. Jika tidak ada proses migrasi maka tidak akan ada interaksi antar sub-populasi, karena dengan adanya migrasi tercipta pula interaksi antar sub-populasi sehingga dapat menjaga keragaman dan variasi pada sub-populasi (Yussof and Razali, 2012).

Berikut disediakan gambar mengenai mekanisme migrasi yang ada di algoritme genetika terdistribusi pada Gambar 2.1 dimana terdapat empat sub-populasi dan pada setiap generasi tertentu setiap satu individu dari suatu sub-populasi dipindahkan ke sub-populasi yang lain.



Gambar 2.1 Mekanisme Migrasi
Sumber : (Mahmudy, 2015)

2.4.1 Representasi Kromosom

Kromosom merupakan suatu nilai yang menjadi satu kesatuan yang dapat berupa biner, float, integer ataupun karakter (Damayanti and Cholissodin, 2018). Bentuk kromosom yang akan digunakan dalam penelitian ini menggunakan bilangan *integer.* Kromosom dibentuk untuk merepresentasikan nama mahasiswa dan nama dosen dengan bilangan acak, hal ini disebabkan karena sebagai daerah pencarian solusi optimal sebuah populasi awal akan dibangkitkan secara acak (Mahmudy, 2009). Pada tahap ini pula akan ditentukan nilai ukuran populasi (*popsize*) untuk menentukan jumlah individu dalam sebuah populasi. Individu sendiri merupakan salah satu solusi dari suatu permasalahan yang nantinya akan dipilih menjadi solusi terbaik.

Pada DGA sebuah populasi nantinya akan dipecah menjadi beberapa subpopulasi. Sub-populasi ini digunakan untuk mengurangi waktu komputasi (*runtime*), meningkatkan kemampuan hasil pencarian solusi serta menjaga keragaman populasi karena akan diproses dalam lingkup yang lebih dalam lagi (Mahmudy, 2015).

2.4.2 Reproduksi

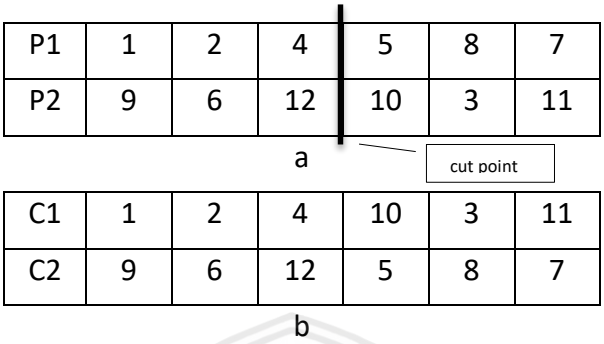
Reproduksi pada algoritme genetika terdistribusi bertujuan untuk menghasilkan anak/keturunan (*child/offspring*) yang digunakan untuk menambah jumlah variasi. Ada dua jenis reproduksi pada algoritme genetika terdistribusi yaitu *crossover* (tukar silang) dan mutasi, dimana kedua jenis reproduksi ini memiliki metode yang berbeda-beda untuk menghasilkan keturunan dan biasanya setiap penelitian menggunakan metode yang berbeda yang bersifat spesifik pada masalah penelitian tersebut. Pada reproduksi untuk dapat menentukan jumlah keturunan (*offspring*) yang dihasilkan proses *crossover* dan mutasi terhadap ukuran populasi dapat ditentukan dengan nilai tingkat *crossover* (*crossover rate*, cr) dan nilai tingkat mutasi (*mutation rate*, mr) yang selanjutnya jumlah keturunan untuk proses *crossover* akan dihitung berdasarkan hasil dari cr dikalikan dengan ukuran populasi (*popsize*). Sedangkan untuk jumlah keturunan proses mutasi dihitung menggunakan mr dikalikan popsize (Mahmudy, 2015).

2.4.3 *Crossover*

Crossover adalah salah satu jenis reproduksi dengan proses awal memilih induk secara acak dari populasi untuk menghasilkan keturunan dengan cara menyilangkan antara dua individu yang dipilih pada sub-populasi yang sama. *Crossover* digunakan agar algoritme dapat mencari jalur baru untuk menemukan jalur yang lebih baik (Yussof et al., 2009). Banyak metode yang dapat digunakan pada tahap *crossover* seperti metode *one-cut point, extended intermediate crossover,* dan lainnya.

Pada penelitian ini akan digunakan metode *one-cut point*, yang mana dengan nilai acak akan dipilih satu titik potong untuk melakukan pertukaran dari setiap *parent* agar memberikan hasil *child*/*offspring*. Hal pertama yang dilakukan untuk proses *crossover* adalah menentukan dua *parent* secara acak. Setelah kedua

parent telah terpilih, selanjutnya adalah menentukan titik potong secara acak, setelah itu semua gen setelah titik potong akan dilakukan pertukaran. Contoh proses *crossover* terdapat pada Gambar 2.2.



Gambar 2.2 Proses *Crossover*, (a) *Parent*, (b) *Child*

Pada gambar 2.2 terlihat bahwa *parent* yang terpilih adalah P1 dan P2. Setelah *parent* terpilih ditentukan titik potong (*cut point)* secara acak, pada gambar tersebut dapat dilihat titik potong yang telah ditentukan. Selanjutnya nilai gen yang berada setelah titik potong tersebut ditukar. Nilai gen P1 setelah titik potong ditukar dengan nilai gen P2 setelah titik potong dan hasilnya disimpan kedalam *child* C1 dan C2.

2.4.4 Mutasi

Proses mutasi merupakan teknik pertukaran yang membutuhkan 1 *parent* yang berasal dari populasi yang sama untuk menghasilkan 1 *child*/offspring (Indra and Subanar, 2014). Mutasi dilakukan dengan cara memasukkan jumlah populasi dan *mutation rate* (*mr*) kemudian menghitung jumlah *child* yang didapatkan, lalu melakukan pemilihan induk secara acak dari populasi yang ada. Terdapat beberapa jenis mutasi seperti *insertion mutation*, *reciprocal exchange mutation, random mutation,* dan lainnya. Pada penelitian ini akan menerapkan metode *reciprocal exchange mutation* dan metode *random mutation*.

Mutasi dengan metode *reciprocal exchange mutation* merupakan metode mutasi yang sederhana denhan memilih posisi secara acak yang kemudian menukarkan nilai pada posisi tersebut (Mahmudy, 2015). Proses awal metode *reciprocal exchange mutation* adalah dengan menentukan 2 titik mutasi secara acak, menukar 2 gen yang telah terpilih dan hasil disimpan pada *child*. Mutasi dengan metode *random mutation* dilakukan dengan penambahan atau pengurangan terhadap nilai gen yang terpilih (Suryaningrum, Ratnawati and Setiawan, 2017). Pada *random mutation*, diperlukan 1 *parent* untuk menghasilkan 1 keturunan atau *offspring.* Setelah itu nilai dari *parent* tersebut akan diganti secara acak dan hasilnya akan disimpan pada *child.* Contoh reproduksi dengan metode *reciprocal exchange mutation* terdapat pada Gambar 2.3, sementara reproduksi dengan menggunakan metode *random mutation* terdapat pada Gambar 2.4.

P1	1	2	4	10	3	11
C3	1	3	4	10	2	11

Gambar 2.3 Proses Mutasi dengan *reciprocal exchange mutation*

Pada gambar 2.3 terlihat bahwa *parent* yang terpilih adalah P1. Setelah *parent* terpilih ditentukan dua titik mutasi, dan pada gambar tersebut gen yang terpilih adalah gen dengan nilai 2 dan 3. Selanjutnya kedua gen tersebut melakukan pertukaran dan hasilnya disimpan kedalam *child* C3.

P2	5	4	7	1	2	3
C4	5	10	7	1	2	3

Gambar 2.4 Proses Mutasi dengan *random mutation*

Pada gambar 2.4 terlihat bahwa *parent* yang terpilih adalah P2. Setelah *parent* terpilih ditentukan satu titik mutasi, dan pada gambar tersebut gen yang terpilih adalah gen dengan nilai 4. Selanjutnya gen tersebut melakukan mengubah nilainya secara acak, sehingga nilai gen yang sebelumnya 4, berubah menjadi 10 dan hasilnya disimpan kedalam *child* C4.

2.4.5 Evaluasi

Pada evaluasi akan dilakukan perhitungan nilai *fitness* pada setiap kromosom untuk mengetahui kualitas dari setiap individu dengan mengukur seberapa baik sebuah individu sehingga individu dengan nilai terbaik merupakan solusi terbaik yang telah diperoleh (Raissa, Widodo and Setiawan, 2017). Nilai *fitness* akan sangat penting untuk menentukan solusi, karena semakin besar nilai *fitness* maka semakin baik individu untuk dijadikan solusi (Mahmudy, 2015).

2.4.6 Menghitung *Fitness*

Menghitung nilai *fitness* dibutuhkan untuk mengukur tingkat optimal sebuah individu. Nilai *fitness* akan berpengaruh pada solusi terbaik dari sebuah individu, karena semakin besar nilai *fitness* yang dihasilkan individu tersebut maka semakin besar pelung individu tersebut menjadi calon solusi terbaik (Mahmudy, 2015). Rumus untuk menghitung nilai *fitness* pada setiap penelitian biasanya berbeda tergantung kepada objek penelitian tersebut. Pada penelitian ini dihasilkan rumus *fitness* pada Persamaan 2.1. Persamaan tersebut digunakan karena pada penelitian ini ingin mendapatkan nilai *fitness* dengan nilai yang besar dan memiliki jumlah nilai penalti yang kecil.

$$fitness = \frac{1}{1 + (total\ penalti)} \tag{2.1}$$

2.4.7 Seleksi

Seleksi digunakan untuk memilih individu yang nantinya akan digunakan pada generasi selanjutnya. Pada tahap ini nilai *fitness* akan diurutkan dan individu yang memiliki nilai *fitness* yang tinggi akan memiliki peluang lebih tinggi juga untuk terpilih. Proses seleksi memiliki beberapa metode dalam pengerjaannya, seperti *elitism selection*, *roulette wheel selection, rank selection, tournament selection,*

steady state selection, dan *boltzmann selection.* Pada penelitian ini akan digunakan metode *elitism selection* pada tahap seleksinya. Hal ini dikarenakan pada penelitian mengenai *review* dari berbagai metode seleksi pada algoritme genetika menunjukkan bahwa metode *elitism selection* menghasilkan solusi yang lebih baik dibandingkan dengan metode seleksi yang lainnya karena *elitism selection* sendiri dapat mencegah individu terbaik untuk menjalani proses reproduksi sehingga dapat melewatinya tanpa modifikasi apa pun ke generasi berikutnya (Sharma and Mehta, 2013) dan menjamin individu yang terbaik akan selalu dipilih (Mahmudy, 2015). Individu-individu terbaik yang memiliki nilai *fitness* tinggi akan dipilih untuk menjadi generasi berikutnya.

2.4.8 Migrasi

Untuk menambah keragaman pada setiap sub-populasi yang ada dibutuhkan proses migrasi. Migrasi merupakan proses yang penting pada algoritme genetika terdistribusi dengan menukar individu antar sub-populasi. Kromosom yang bermigrasi akan mempengaruhi populasi dalam proses reproduksinya. Pada GA keragaman populasi yang dihasilkan terbilang rendah karena tidak adanya interaksi antar populasi. Dengan adanya Migrasi populasi yang telah dipecah menjadi sub-populasi dan telah di proses akan berinteraksi satu sama lain dan berbagi hasil.

Pada migrasi terdapat cara untuk menukar individu, yaitu menukar individu terbaik atau individu secara acak (Yussof and Razali, 2012). Pada penelitian ini akan diterapkan menukar individu terbaik dengan individu terburuk. Hal tersebut disebabkan karena dengan menukar individu yang memiliki nilai *fitness* yang rendah akan lebih baik hasilnya jika ditukar dengan individu yang memiliki nilai *fitness* yang tinggi.

BAB 3 METODOLOGI

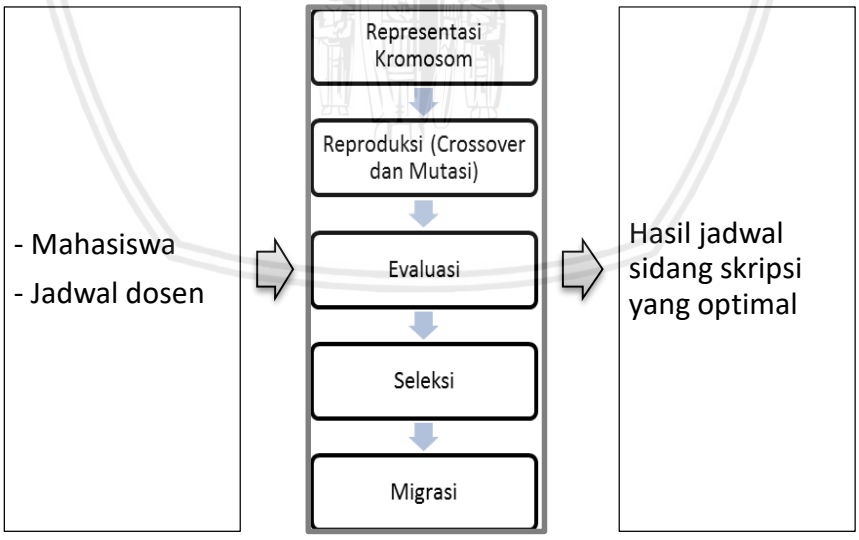
Bab ini akan dijelaskan bagaimana metodologi penelitian yang akan dilakukan pada penelitian yang berjudul "Optimasi Penjadwalan Sidang Skripsi Menggunakan Algoritme Genetika Terdistribusi (Studi Kasus : Prodi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya)". Pada bab ini akan dibahas mengenai Tipe Penelitian, S*trate*gi Penelitian, Partisipan Penelitian, Lokasi Penelitian, Teknik Pengumpulan Data, Analisis Kebutuhan, Perancangan Algoritme, dan Pengujian.

3.1 Tipe Penelitian

Tipe penelitian yang digunakan pada penelitian yang berjudul "Optimasi Penjadwalan Sidang Skripsi Menggunakan Algoritme Genetika Terdistribusi (Studi Kasus : Prodi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya)" adalah tipe non implementatif analitik.

3.2 S*trate*gi Penelitian

Pada penelitian ini s*trate*gi penelitian yang dilakukan adalah dengan mengumpulkan data parameter penjadwalan sidang skripsi seperti Data mahasiswa prodi Teknik Informatika FILKOM UB, jadwal kegiatan mengajar dosen di FILKOM UB, jadwal ruangan pada FILKOM UB. Setelah data dikumpulkan, maka akan dilakukan proses yang ditunjukan pada Gambar 3.1.



Gambar 3.1 Proses Model Optimasi Penjadwalan SIdang Skripsi

Pada penelitian ini proses untuk melakukan optimasi penjadwalan sidang skripsi dimulai dengan sistem membaca data mahasiswa, jadwal mengajar dosen, dan jadwal ruangan. Setelah itu akan memasukkan (*input*) berupa parameter algoritme genetika terdistribusi, yaitu jumlah populasi, jumlah generasi, jumlah

subpopulasi, nilai proses *crossover* (*crossover rate*), dan nilai proses mutasi (*mutationarate*) yang kemudian akan dilanjutkan ke dalam metode algoritme genetika terdistribusi. Dimulai dengan merepresentasi kromosom dan membetuk sub-populasi, setelah itu masuk ke tahap reproduksi dengan melakukan proses *crossover* dan proses mutasi, lalu menghitung nilai *fitness* pada proses evaluasi, melakukan proses seleksi, dan yang terakhir akan dilakukan proses migrasi. Setelah melakukan migrasi, maka akan direpresentasikan bentuk kromosom baru dengan menggabungkan subpopulasi menjadi sebuah populasi. Setelah itu akan didapatkan hasil yang optimal danasistem akan memberikan hasil keluaran (*output*) berupa jadwal sidang skripsi yang optimal.

3.3 Partisipan Penelitian

Pada penelitian ini melibatkan partisipan mahasiswa Prodi Teknik Informatika FILKOM UB, dosen Prodi Teknik Informatika FILKOM UB, dan sekretaris jurusan Teknik Informatika FILKOM UB. Mahasiswa dan dosen nantinya yang akan dibantu dalam mendapatkan jadwal sidang skripsi, sementara sekretaris jurusan akan dibantu dalam menjadwalkan sidang skripsi.

3.4 Lokasi Penelitian

Lokasi penelitian ini bertempat di Laboratorium Komputasi Cerdas dan Visualisasi Fakultas Ilmu Komputer Universitas Brawijaya.

3.5 Teknik Pengumpulan Data

Pada penelitian ini data yang diambil berbentuk sekunder, data didapatkan dari Fakultas Ilmu Komputer Universitas Brawijaya. Data ini kemudian akan digunakan untuk proses penelitian yang dilakukan dalam mencapai jadwal sidang skripsi yang optimal. Data yang dibutuhkan antara lain:

1. Data mahasiswa prodi Teknik Informatika FILKOM UB angkatan 2014 yang telah memperoleh gelar sarjana pada semester genap tahun 2017/2018. Bentuk data berupa nama dan keminatan mahasiswa. Data mahasiswa yang digunakan berjumlah 275 mahasiswa.
2. Data dosen prodi Teknik Informatika FILKOM UB tahun 2017/2018. Bentuk data berupa nama, keminatan major dosen, dan keminatan minor dosen. Data dosen yang digunakan berjumlah 60 dosen.
3. Data jadwal kegiatan mengajar dosen di FILKOM UB. Bentuk data berupa nama dosen, jam, hari. Pada data jadwal kegiatan mengajar akan di representasikan pada bentuk excel dengan menuliskan nama-nama dosen yang tidak dapat mengajar pada sesi tertentu.
4. Data jadwal pengujian sidang skripsi oleh dosen di FILKOM UB pada semester genap tahun 2017/2018. Bentuk data berupa nama dosen, keminatan, jam pengujian, hari pengujian, keterangan dosen sebagai pembimbing atau penguji.

3.6 Analisis Kebutuhan

Analisis Kebutuhan dilakukan untuk mengetahui kebutuhan apa saja yang diperlukan dalam membangun sistem dalam penelitian ini, dari kebutuhan fungsional maupun kebutuhan perangkat.

3.6.1 Kebutuhan Fungsional

Kebutuhan fungsional pada penelitian ini merupakan kebutuhan yang berisi proses apa saja yang harus ada di dalam sistem yang nantinya dilakukan oleh sistem. Kebutuhan fungsional antara lain sebagai berikut:

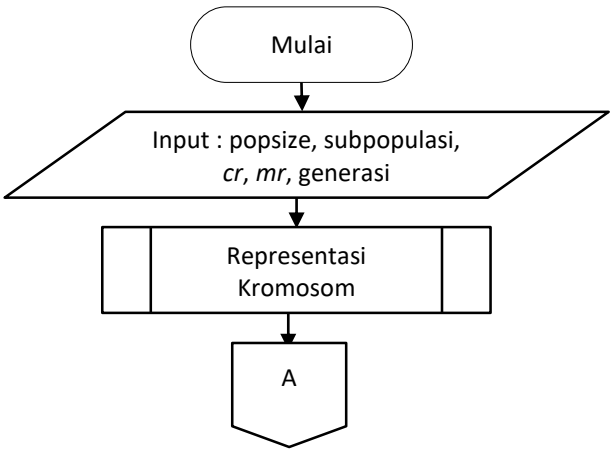
1. Sistem dapat melakukan proses Algoritme Genetika Terdistribusi,
2. Sistem dapat melakukan penjadwalan sidang skripsi.

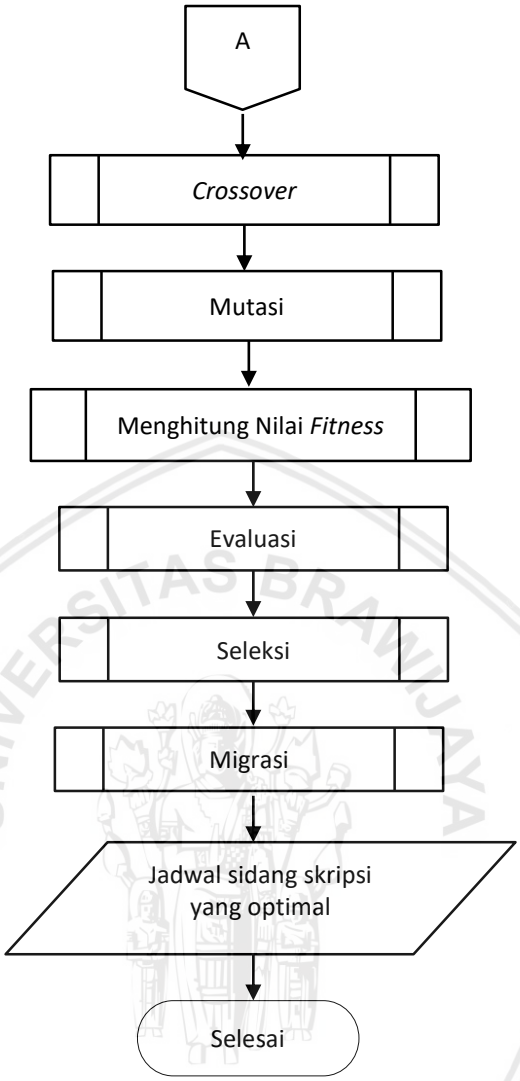
3.6.2 Kebutuhan Perangkat

1. Kebutuhan perangkat keras yang diperlukan adalah sebagai berikut:
   a. Laptop
      - Processor Intel® Core™ I5-5200U CPU @ 2.70 GHZ
      - Hard Disk 500 GB
      - RAM 4 GB
2. Kebutuhan perangkat lunak yang diperlukan adalah sebagai berikut:
   a. Sistem Operasi
      - Microsoft Windows10
   b. Bahasa Pemrograman
      - Python Anaconda Versi 3.6
   c. Lingkungan Pengembangan Terpadu
      - Spyder 3.2.6

3.7 Perancangan Algoritme

Pada penelitian ini tahap perancangan algoritme akan menjelaskan mengenai rancangan pada penelitian ini. Pada penelitian ini sistem akan memproses data yang kemudian akan dilanjutkan dengan tahapan-tahapan algoritme genetika terdistribusi untuk mendapatkan jadwal sidang skripsi yang optimal. Alur algoritme ditunjukkan pada Gambar 3.2.





Gambar 3.2 Diagram Alir Perancangan Algoritme

3.8 Pengujian

Pengujian akan dilakukan dengan manggunakan data yang sama. Pengujian ini akan melakukan 5 kali percobaan untuk setiap parameter. Uji populasi dilakukan untuk mendapatkan hasil ukuran populasi yang optimal. Setelah itu melakukan pengujian generasi agar mengetahui generasi optimal. Jika telah mendapatkan ukuran populasi dan jumlah generasi yang optimal, langkah selanjutnya adalah melakukan nilai tingkat proses *crossover* yaitu *crossover rate* (cr) dan nilai tingkat proses mutasi yaitu *mutation rate* (mr) dengan menggunakan ukuran populasi dan jumlah generasi yang telah optimal. Setelah semua pengujian dilakukan akan dievaluasi hasil dari pengujian tersebut untuk melihat apakah jumlah populasi, generasi, *crossover rate* (cr) dan *mutation rate* (mr) telah optimal dalam menyelesaikan masalah penjadwalan sidang skripsi.

BAB 4 PERANCANGAN

Masalah utama pada penelitian ini terdapat pada bagaimana membuat penjadwalan sidang skripsi dengan optimal yang sesuai dengan jadwal mahasiswa, dosen, dan ruang yang tepat. Penjadwalan merupakan proses yang penting dalam melakukan sidang skripsi, karena penjadwalan yang optimal akan memberikan manfaat yang baik dengan tidak membuang-buang waktu.

Untuk mengatasi permasalahan penjadwalan sidang skripsi yang optimal digunakan Algoritme Genetika Terdistribusi. Jika pada algoritme genetika, konvergensi dini sering terjadi karena biasanya solusi sering terjebak dalam daerah optimum lokal. Konvergensi dini terjadi karena kurangnya keragaman individu pada suatu populasi yang menyebabkan hampir semua individu bernilai sama sebelum tercapainya titik optimum yang diinginkan. Dengan menggunakan algoritme genetika terdistribusi diharapkan dapat memberikan hasil yang lebih beragam karena pada setiap sub-populasi bisa menggunakan proses reproduksi yang berbeda. Selain itu pada algoritme genetika terdistribusi terdapat mekanisme migrasi yang dapat menjaga keragaman individu sehingga dapat memeberikan hasil yang lebih optimal.

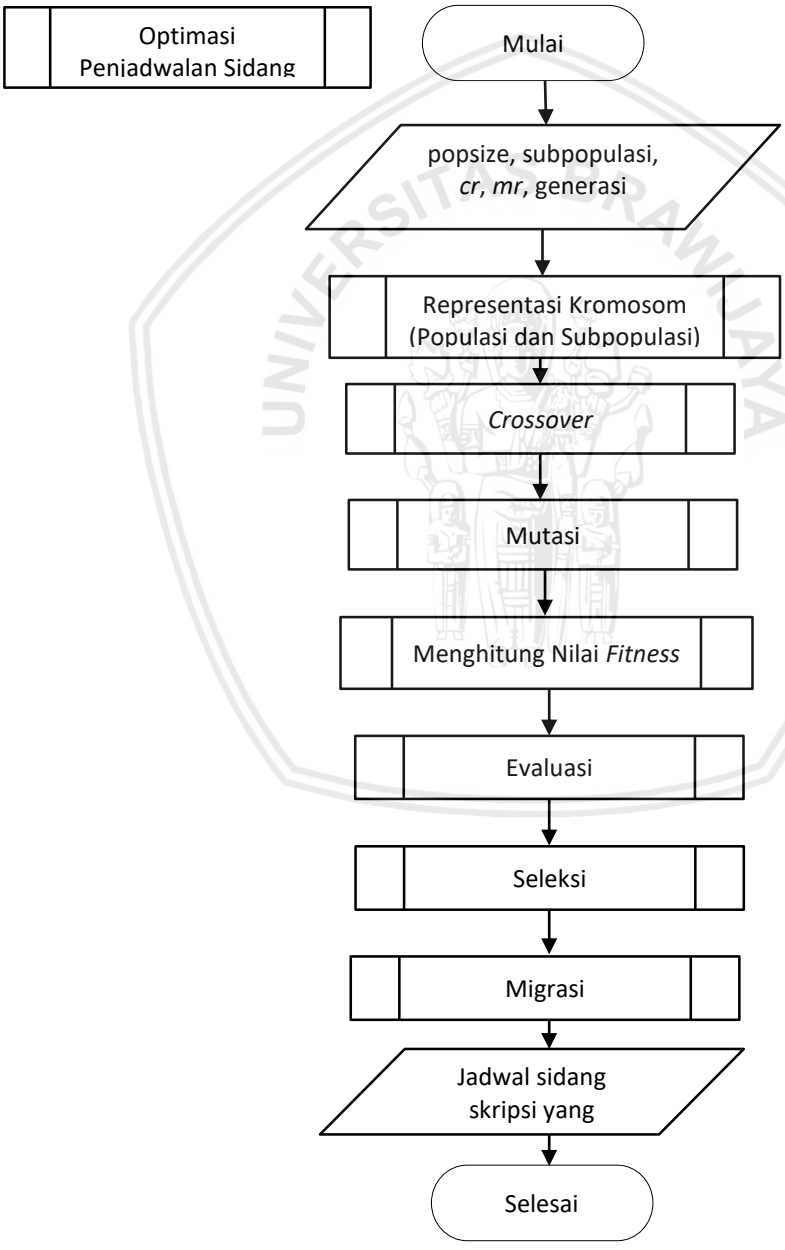
Perancangan algoritme akan dijelaskan pada bab ini yang berupa *flowchart* pada setiap proses yang akan dilakukan dalam menentukan jadwal yang optimal pada panjadwalan sidang skripsi. *Flowchart* yang ada pada bab ini adalah *flowchart* untuk proses *crossover* dengan menggunakan metode *one-cut point crossover*, proses mutasi dengan menggunakan metode *random mutation* dan *reciprocal exchange mutation*, proses menghitung nilai *fitness*, proses evaluasi, proses seleksi dengan menggunakan metode *elitism selection* dan dilanjutkan dengan diagram alir untuk mekanisme migrasi.

Proses perhitungan manual juga akan dijelaskan pada bab ini mengenai penerapan algoritme genetika terdistribusi untuk mengatasi permasalahan penjadwalan sidang skripsi agar optimal. Selain itu, pada bab ini juga akan menjelaskan mengenai perancangan pengujian yang akan dilakukan. Perancangan pengujian ini dilakukan untuk mengetahui bagaimana pengaruh parameter yang digunakan pada algoritme genetika terdistribusi dalam permasalahan penjadwalan sidang skripsi yang optimal. Perancangan pengujian terdiri dari perancangan pengujian ukuran populasi, perancangan pengujian jumlah generasi, dan perancangan pengujian kombinasi cr dan mr.

4.1 Perancangan Algoritme

Perancangan algoritme penelitian ini yaitu memasukkan nilai sub-populasi, *popsize*, *cr*, *mr*, dan jumlah generasi. Jumlah sub-populasi sudah ditentukan oleh program sehingga tidak perlu ada inisialisasi lagi, selain jumlah sub-populasi parameter algoritme genetika lainnya akan menjadi masukkan dari pengguna. Selanjutnya akan dilakukan proses inisialisasi nilai gen pada setiap kromosom untuk setiap sub-populasi secara acak, selanjutnya sub-populasi akan melalui

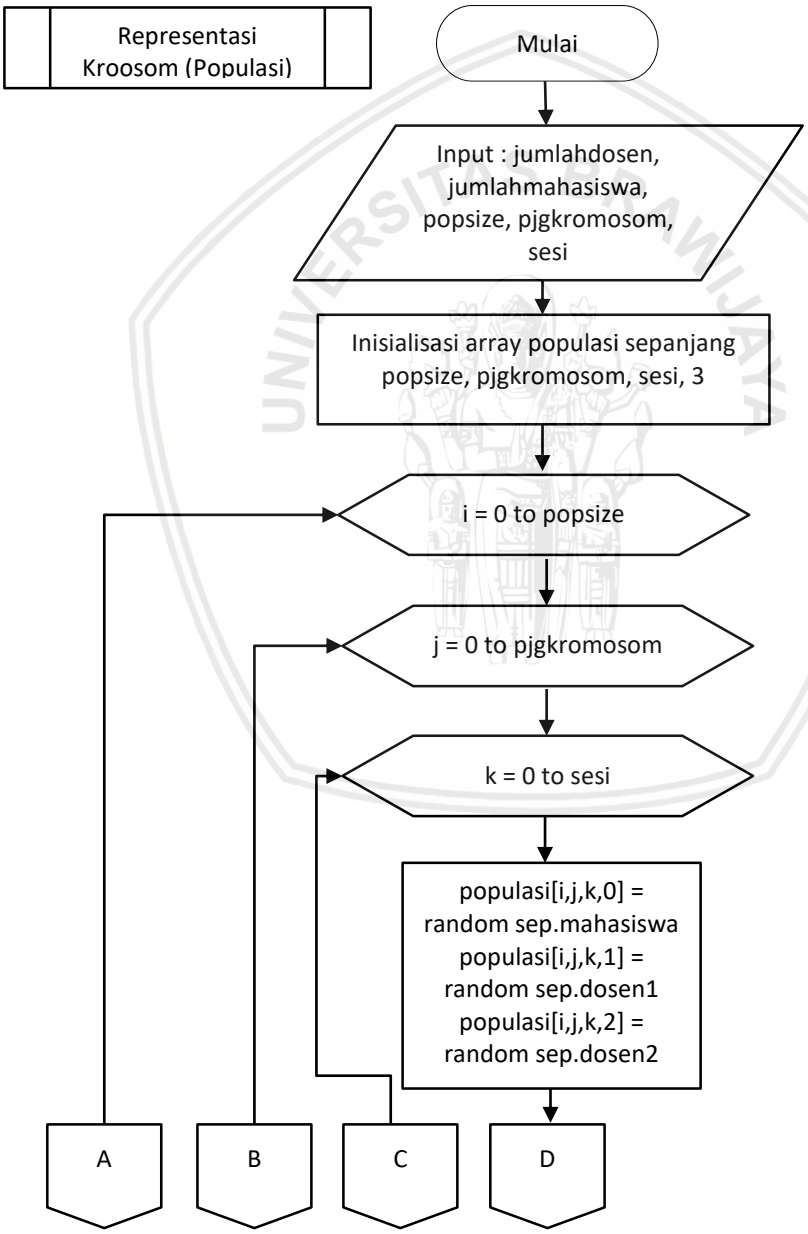
proses reproduksi yaitu proses mutasi dan proses *crossover* untuk menghasilkan keturunan (*offspring*). Pada proses mutasi akan digunakan metode *random mutation* dan *reciprocal exchange* mutation, sedangkan untuk proses *crossover* akan digunakan metode *one-cut* point. Setelah mendapatkan keturunan akan dilakukan proses evaluasi untuk menggabungkan *parent* dan *child* dalam setiap sub-populasi tersebut yang kemudian akan dilakukan perhitungan nilai *fitness*. Setelah mendapatkan hasil nilai *fitness*, dilakukan proses seleksi dengan metode *elitism selection* sehingga akan menghasilkan individu terbaik. Tahap terakhir yaitu proses migrasi yang dilakukan dengan menukar satu individu terbaik pada sebuah sub-populasi ke sub-populasi yang lain begitu juga sebaliknya. Diagram alir proses algoritme genetika terdistribusi dapat dilihat pada Gambar 4.1.

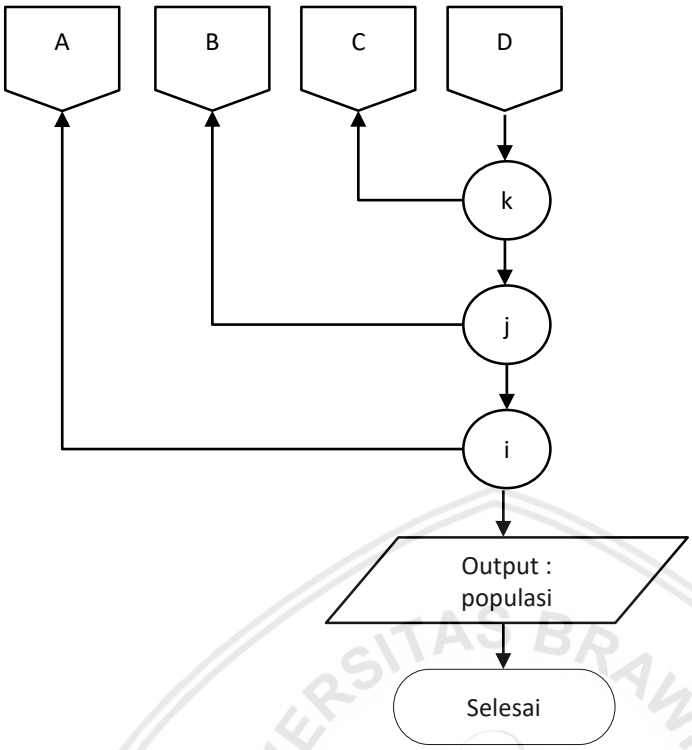


Gambar 4.1 Diagram Alir Optimasi Penjadwalan Sidang Skripsi

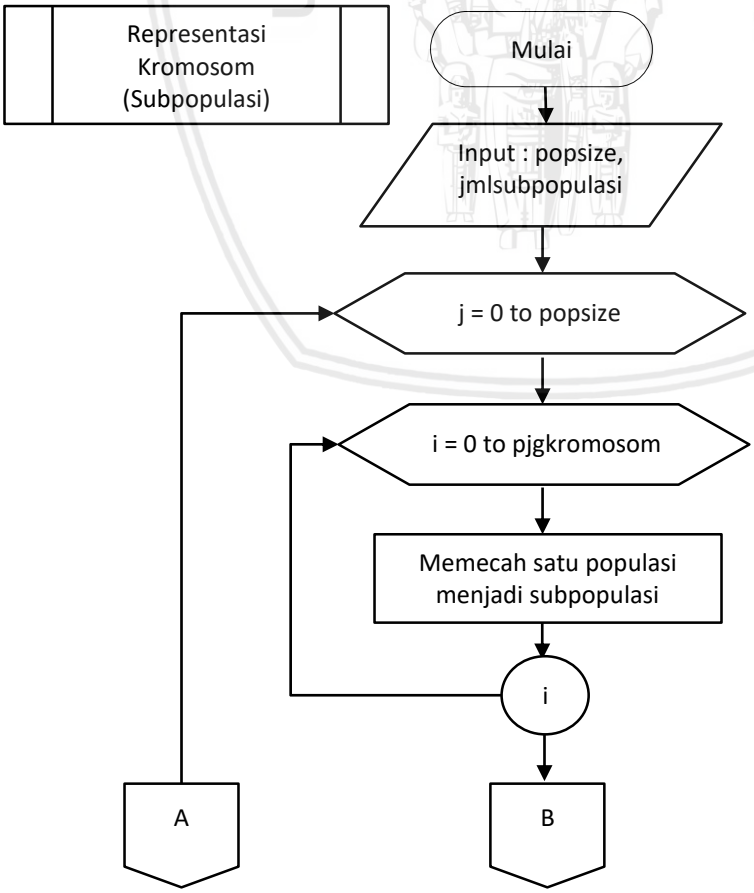
4.1.1 Representasi Kromosom

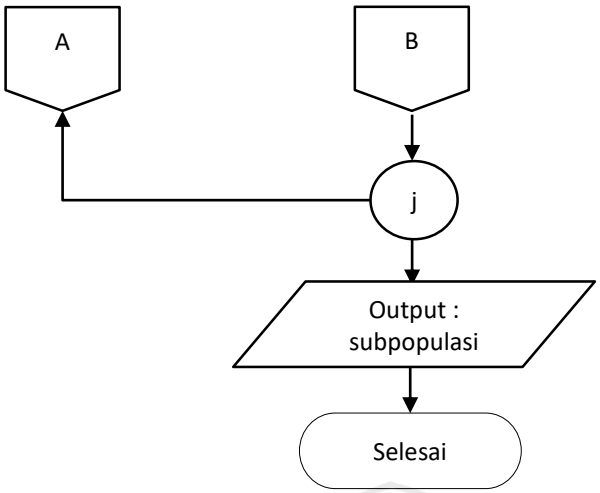
Tahap pertama pada algoritme genetika terdistribusi adalah representasi kromosom yang digunakan untuk membentuk populasi awal berdasarkan jumlah *popsize* dan jumlah gen untuk setiap subpopulasi. Representasi kromosom dibangkitkan secara acak berdasarkan jumlah mahasiswa yang dimasukkan. Proses representasi kromosom diawali dengan menginisialisasikan jumlah populasi dan subpopulasi, lalu masuk ke dalam perulangan sebanyak *popsize* dan jumlah gen yang digunakan untuk membangkitkan nilai kromosom secara acak diseluruh *popsize* sejumlah dengan jumlah gen pada setiap subpopulasi. Diagram alir proses representasi kromosom dapat dilihat pada Gambar 4.2 dan Gambar 4.3.





Gambar 4.2 Diagram Alir Representasi Kromosom (Populasi)

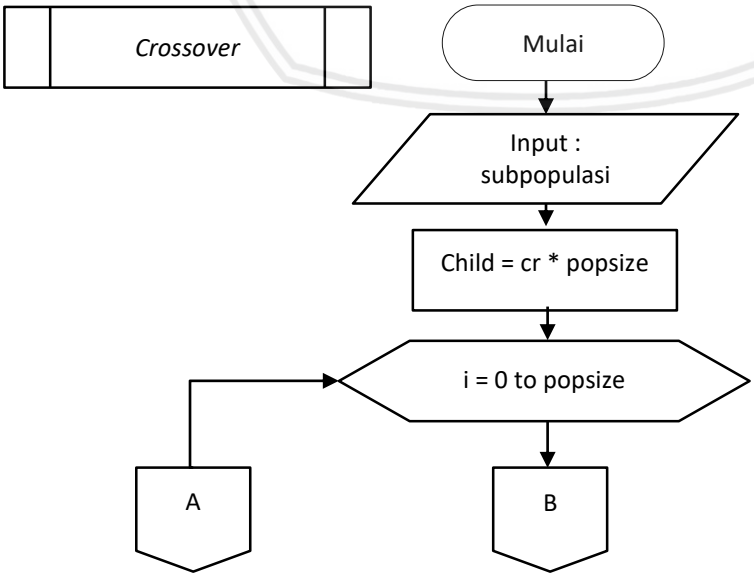


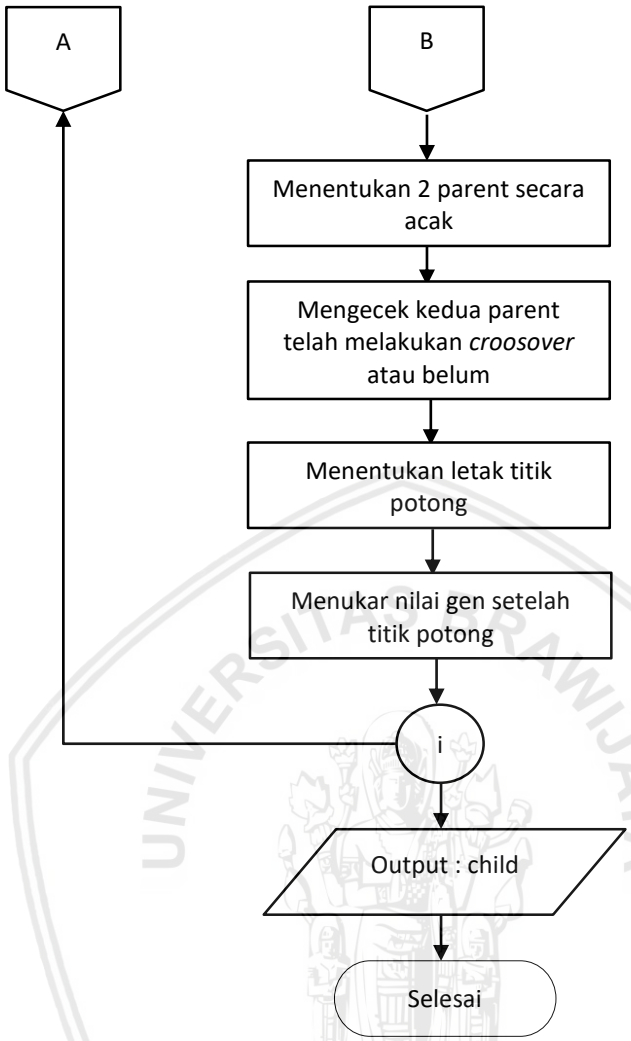


Gambar 4.3 Diagram Alir Representasi Kromosom (Subpopulasi)

4.1.2 *Crossover*

Setelah melalui tahap representasi kromosom, tahap selanjutnya adalah tahap reproduksi yaitu proses *crossover* untuk menghasilkan keturunan (*child*) dari *parent* pada setiap subpopulasi. Proses *crossover* pada penelitian ini menggunakan metode *one-cut point*. Tahap pertama yaitu menghitung jumlah *child* yang akan dihasilkan dengan mengalikan nilai *cr* dan *popsize.* Selanjutnya menentukan dua *parent* secara acak. Setelah *parent* didapatkan, tahap selanjutnya adalah menentukan titik potong secara acak. Dari titik potong (*cut point*) ini dilakukan proses *one-cut point crossover*. Nilai *child* akan didapatkan dari penggabungan dua *parent* yang telah melalui proses pemotongan. Setelah nilai *child* didapatkan, selanjutnya hasil *child* akan ditampilkan. Diagram alir proses *crossover* dapat dilihat pada Gambar 4.4.





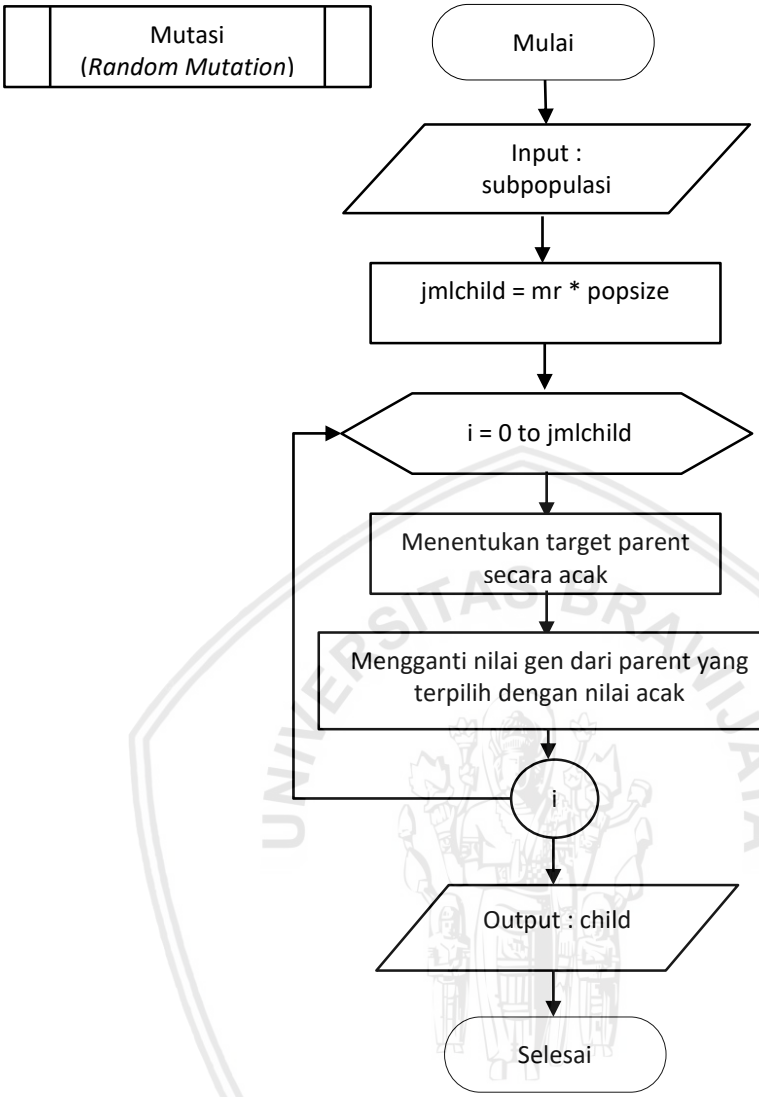
Gambar 4.4 Diagram Alir *Crossover*

4.1.3 Mutasi

Tahap selanjutnya dari reproduksi adalah proses mutasi untuk menghasilkan keturunan (*child*) yang didapatkan dari *parent* dalam setiap subpopulasi. Proses mutasi pada penelitian ini menggunakan dua metode yaitu *random mutation* dan *reciprocal exchange mutation*.

4.1.3.1 Random Mutation

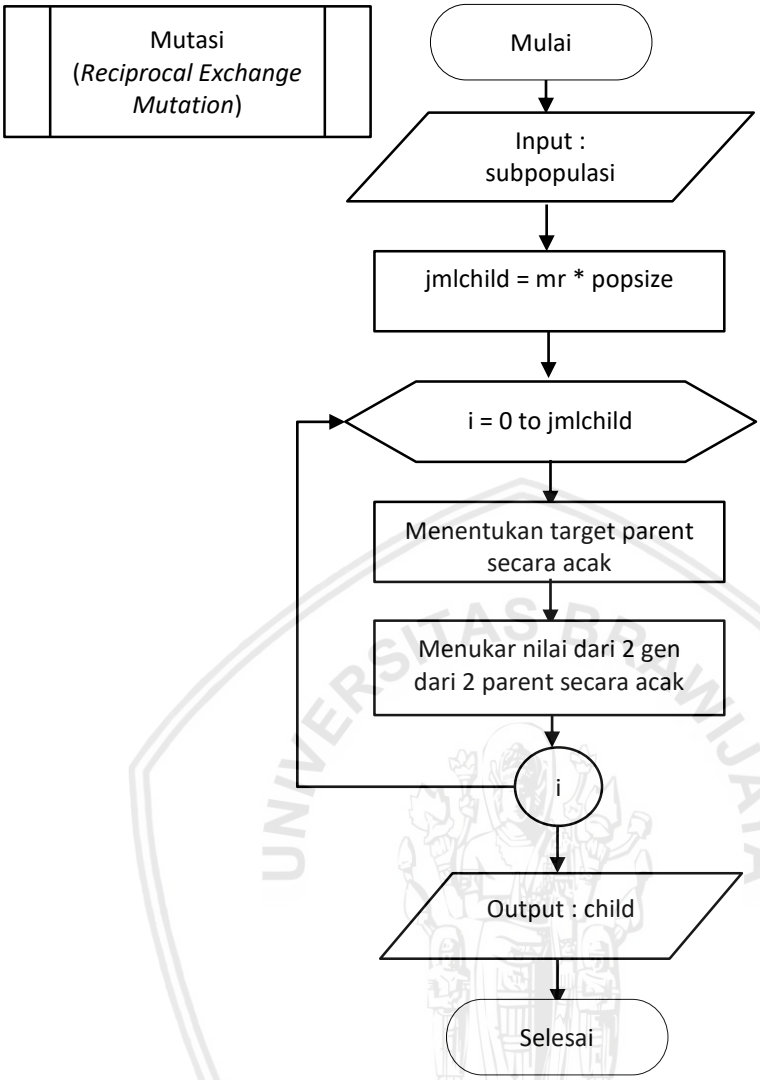
Pada metode *random mutation* diawali dengan menghitung jumlah *child* yang akan dihasilkan dengan menghitung nilai *mr* dan *popsize*, setelah itu menentukan *parent* secara acak. Jika *parent* telah ditentukan, selanjutnya adalah menentukan gen yang dipilih untuk kemudian nilainya diganti secara acak (*random* sehingga menghasilkan *child*. Diagram alir proses mutasi dengan metode *random mutation* dapat dilihat pada Gambar 4.5.



Gambar 4.5 Diagram Alir Random Mutation

4.1.3.2 Reciprocal Exchange Mutation

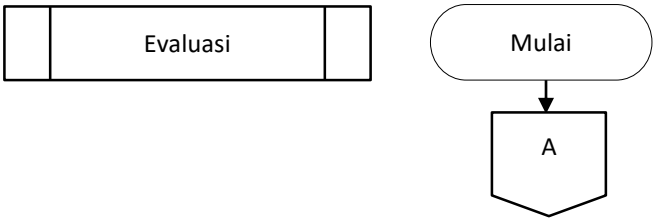
Pada proses mutasi menggunakan metode *reciprocal exchange mutation* dilakukan dengan menghitung nilai *mr* dan *popsize* untuk mengetahui jumlah *child* yang akan didapatkan. Setelah mendapatkan hasilnya, proses selanjutnya adalah mencari *parent* secara acak untuk dilakukan proses mutasi. Jika *parent* telah ditentukan maka tahap selanjutnya adalah menentukan gen yang dipilih untuk ditukar, kemudian dilakukan proses penukaran sehingga menghasilkan *child*. Diagram alir proses mutasi dengan metode *reciprocal exchange mutation* dapat dilihat pada Gambar 4.6.

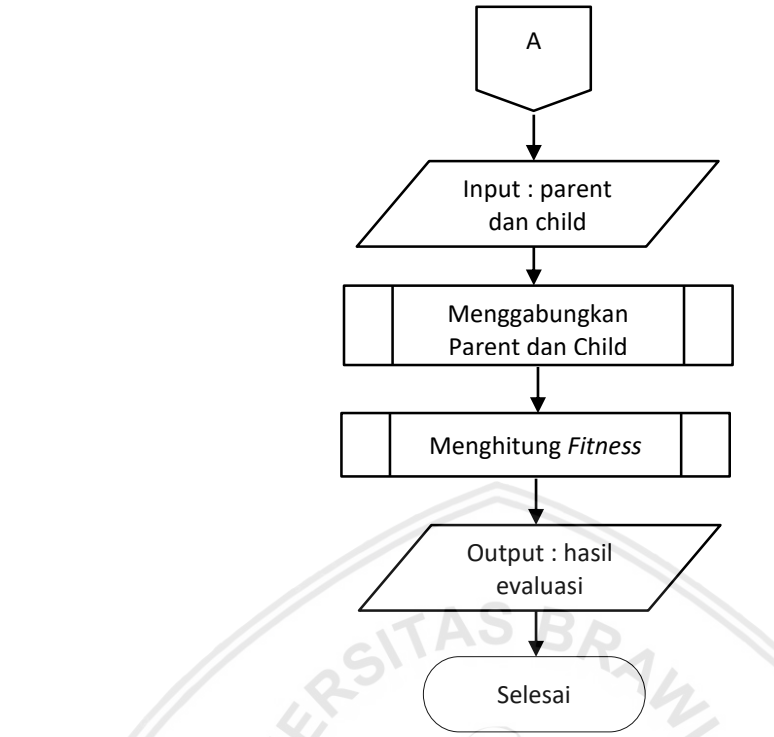


Gambar 4.6 Diagram Alir Reciprocal Exchange Mutation

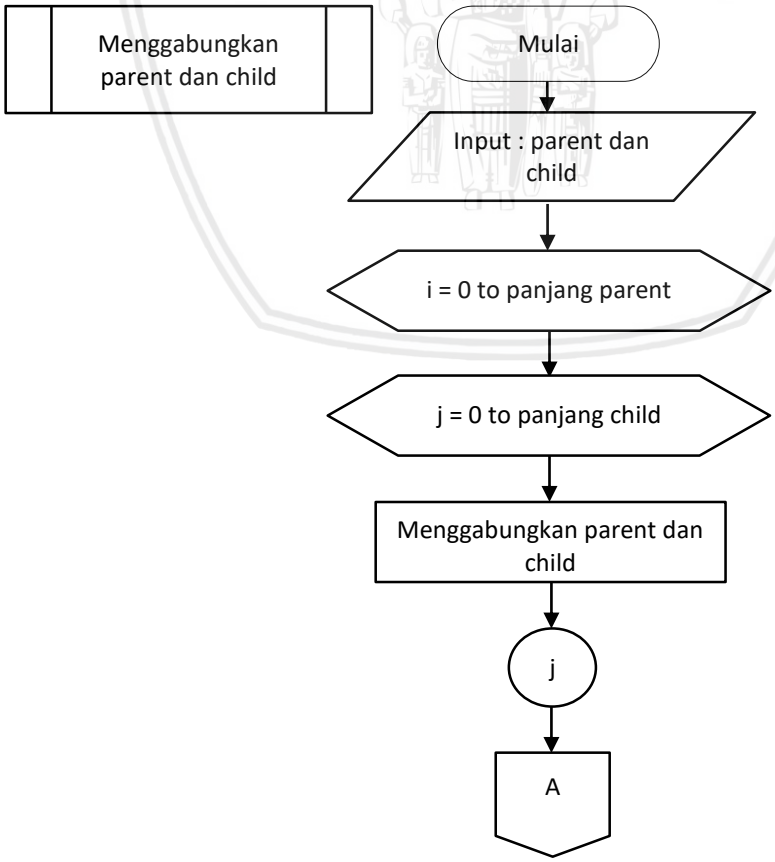
4.1.4 Evaluasi

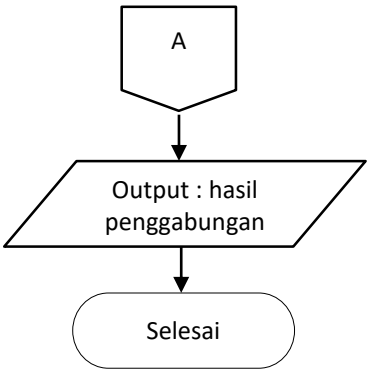
Proses evaluasi digunakan untuk menggabungkan *parent* dan *child* dari proses reproduksi. Pada evaluasi akan memanggil fungsi *crossover* dan fungsi mutasi yang selanjutnya akan digabungkan hasilnya menjadi subpopulasi yang baru. Diagram alir proses untuk menghitung nilai *fitness* dapat dilihat pada Gambar 4.7.




A
Input : parent dan child
Menggabungkan Parent dan Child
Menghitung *Fitness*
Output : hasil evaluasi
Selesai

(a)

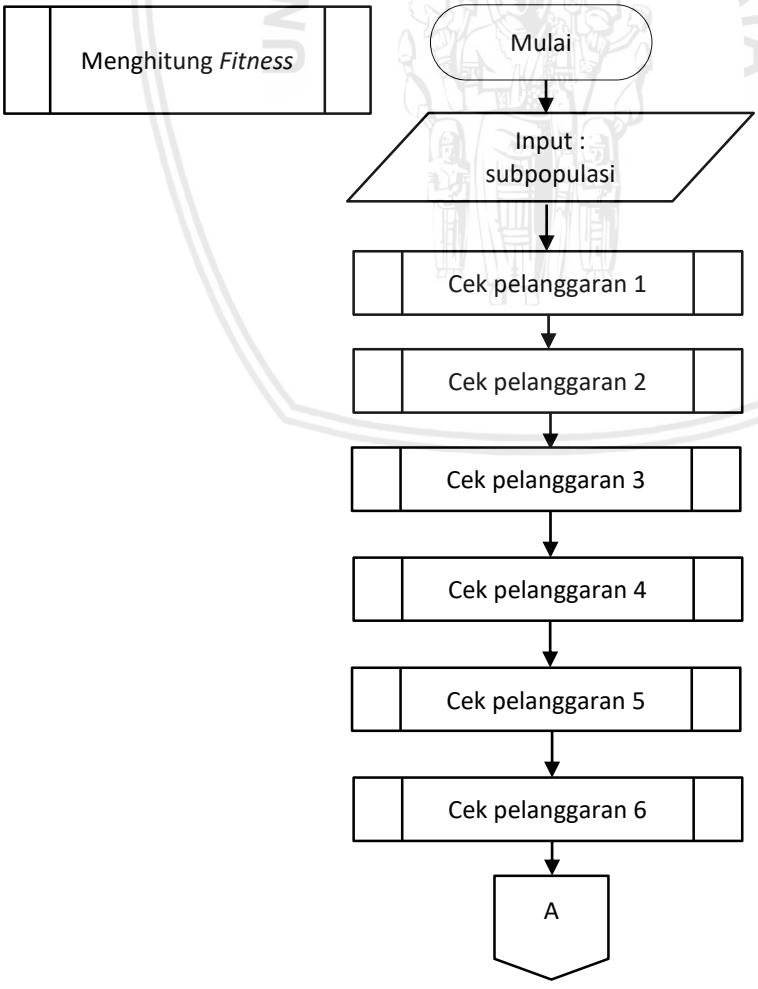

Menggabungkan parent dan child
Mulai
Input : parent dan child
i = 0 to panjang parent
j = 0 to panjang child
Menggabungkan parent dan child
j
A

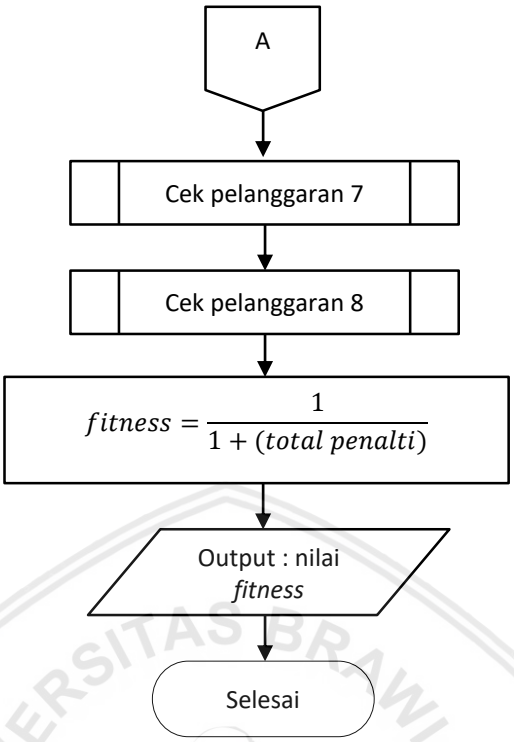


Gambar 4.7 Diagram Alir Evaluasi, (a) Evaluasi, (b) Menggabungkan *Parent* dan *Child*

4.1.5 Menghitung *Fitness*

Setelah melalui tahap reproduksi, tahap selanjutnya adalah menghitung nilai *fitness.* Perhitungan nilai *fitness* dilakukan untuk menjadi pertimbangan dalam menentukan solusi yang optimal. Perhitungan nilai *fitness* dilakukan pada seluruh individu pada setiap subpopulasi. Sebelum nilai *fitness* dihitung, akan dilakukan proses mengecek total pinalti yang didapatkan dari setiap subpopulasi. Diagram alir proses untuk menghitung nilai *fitness* dapat dilihat pada Gambar 4.8.

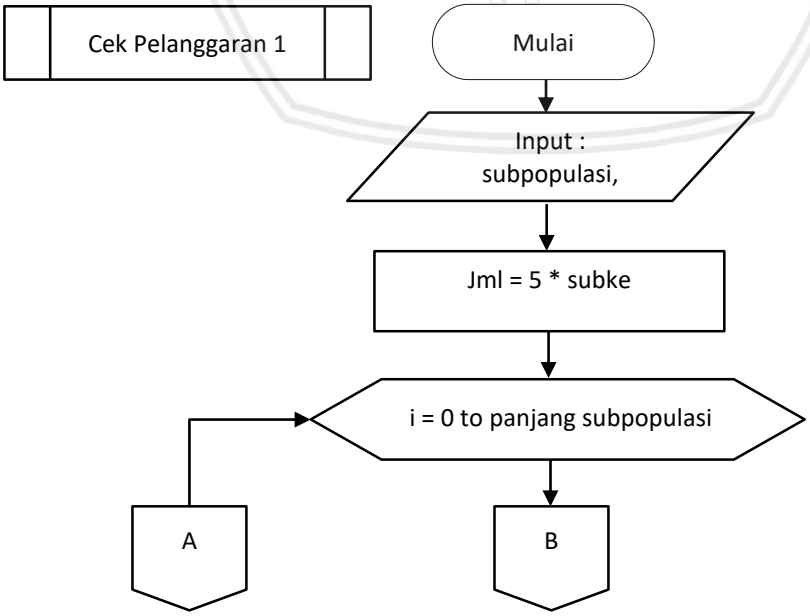


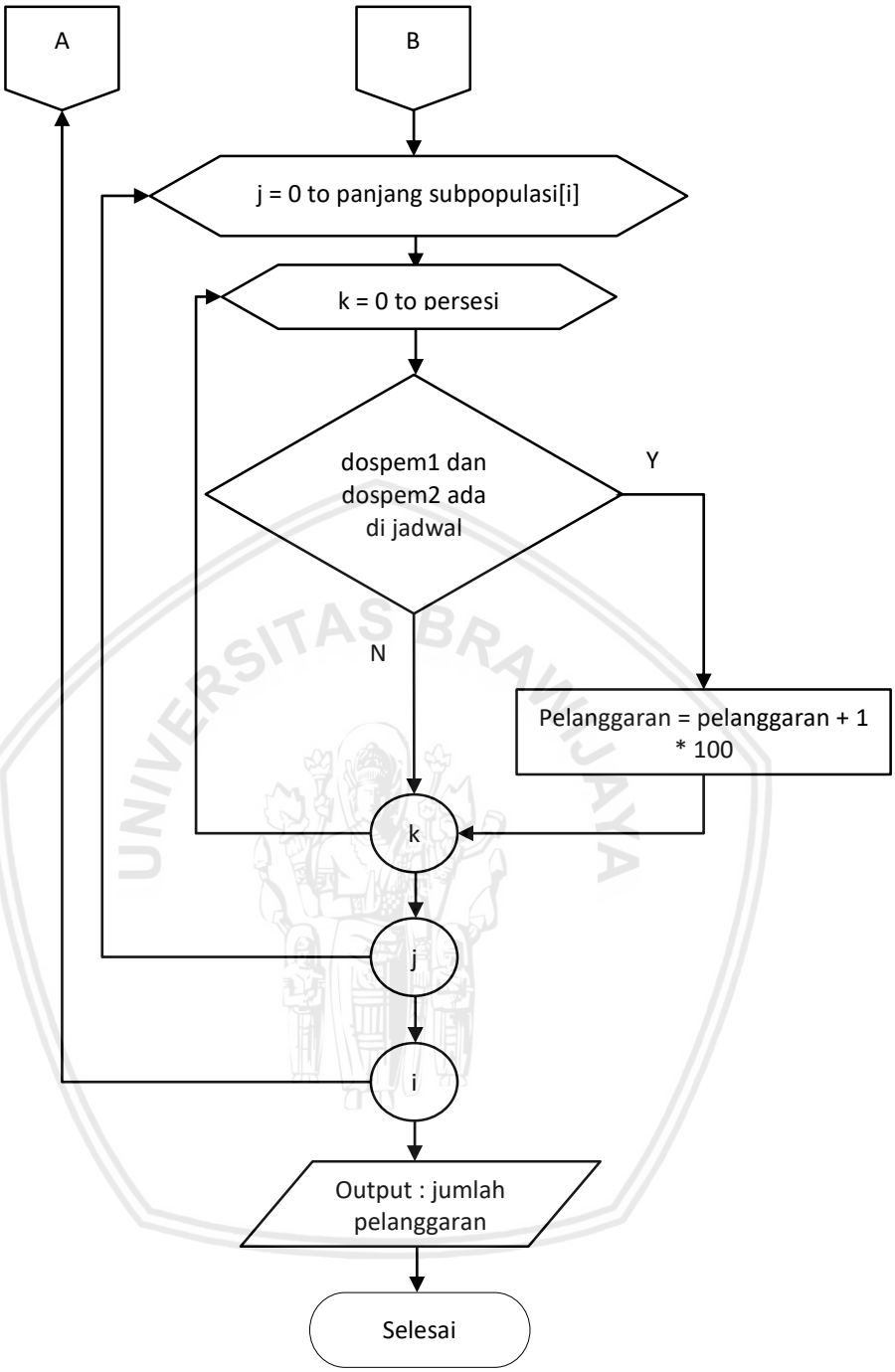


Gambar 4.8 Diagram Alir Menghitung *Fitness*

4.1.5.1 Cek Pelanggaran 1

Pada tahap menghitung *fitness* akan dicek pelanggaran 1 yaitu jadwal mengajar dosen pembimbing bentrok dengan jadwal sidang skripsi. Pelanggaran 1 ini merupakan jenis pelanggaran *hard constraint.* Jika pelanggaran 1 ini dilanggar maka akan bernilai 100 untuk pelanggarannya. Diagram alir proses untuk cek pelanggaran 1 dapat dilihat pada Gambar 4.9.

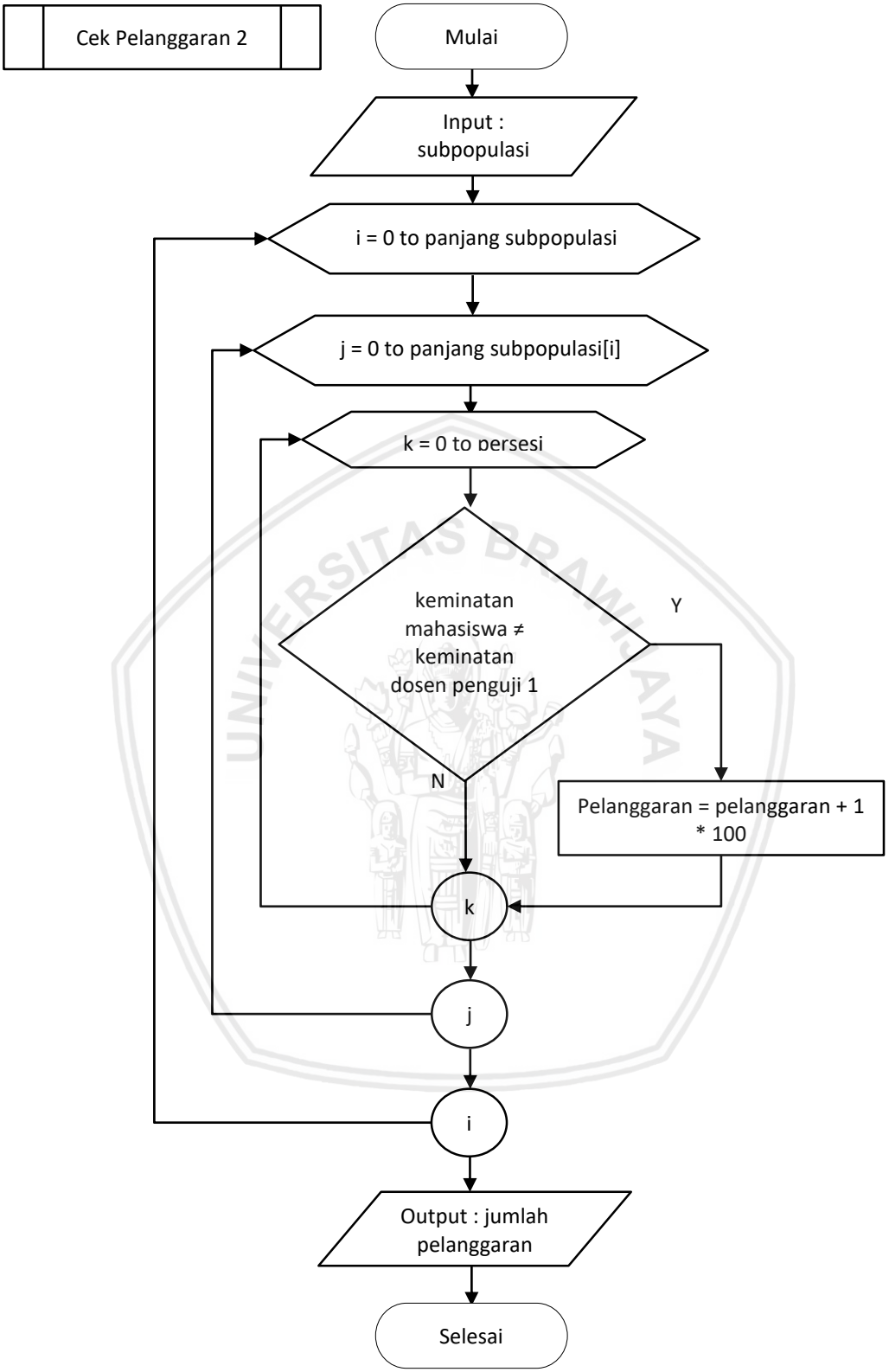




Gambar 4.9 Diagram Alir Cek Pelanggaran 1

4.1.5.2 Cek Pelanggaran 2

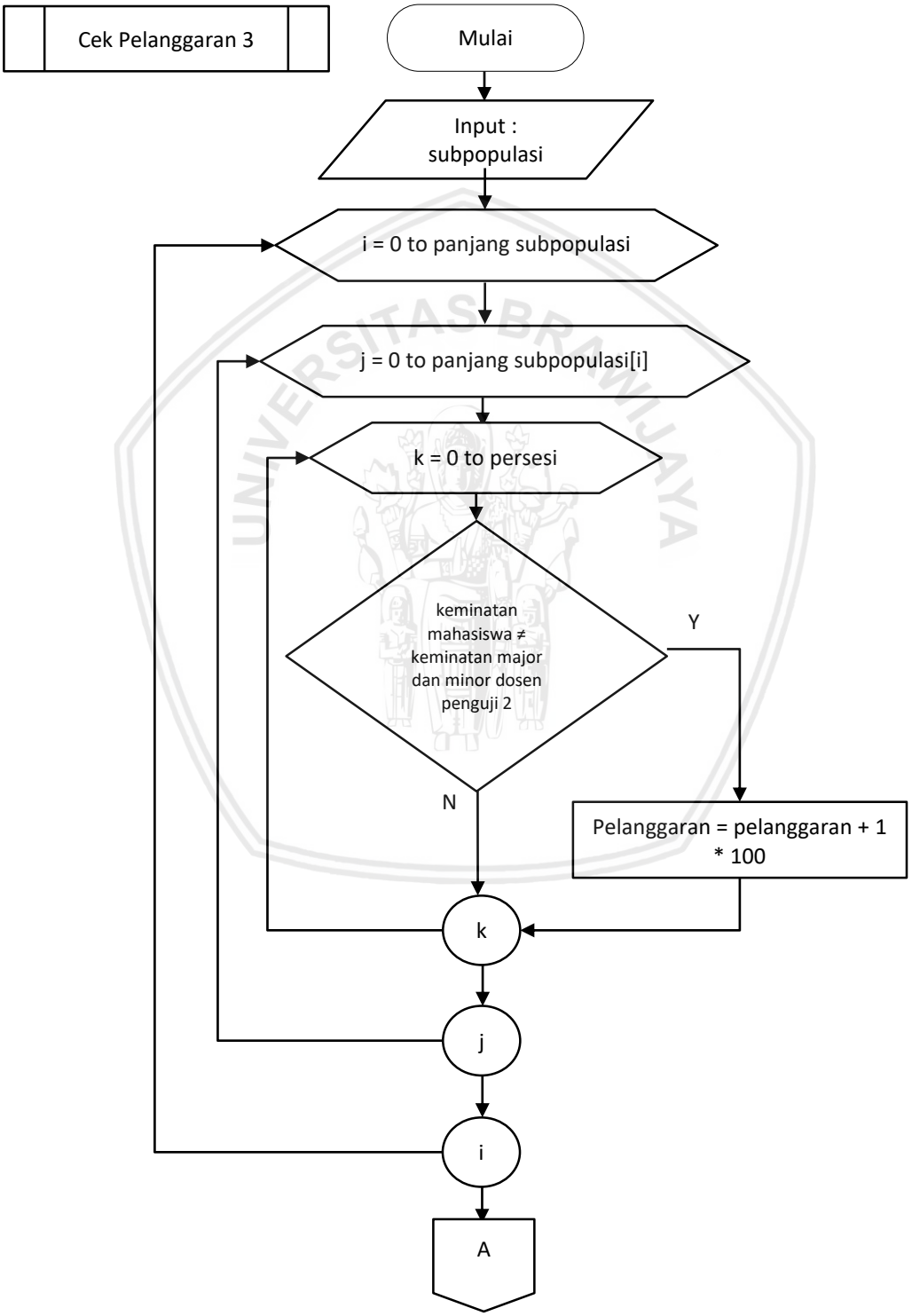
Pada tahap menghitung *fitness* akan dicek pelanggaran 2 yaitu dosen penguji 1 menguji mahasiswa yang berbeda keminatannya dengan major dosen penguji. Pelanggaran 2 ini merupakan jenis pelanggaran *hard constraint.* Jika pelanggaran 2 ini dilanggar maka nilai pelanggarannya adalah 100. Diagram alir proses untuk cek pelanggaran 2 dapat dilihat pada Gambar 4.10.

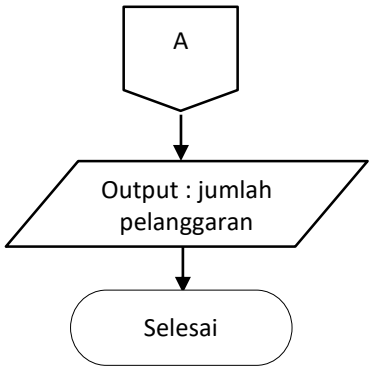


Gambar 4.10 Diagram Alir Cek Pelanggaran 2

4.1.5.3 Cek Pelanggaran 3

Pada tahap menghitung *fitness* akan dicek pelanggaran 3 yaitu dosen penguji 2 menguji mahasiswa yang berbeda keminatannya dengan major dosen penguji 2. Pelanggaran 3 ini merupakan jenis pelanggaran *hard constraint.* Jika pelanggaran 3 ini dilanggar maka nilai pelanggarannya adalah 100. Diagram alir proses untuk cek pelanggaran 3 dapat dilihat pada Gambar 4.11.

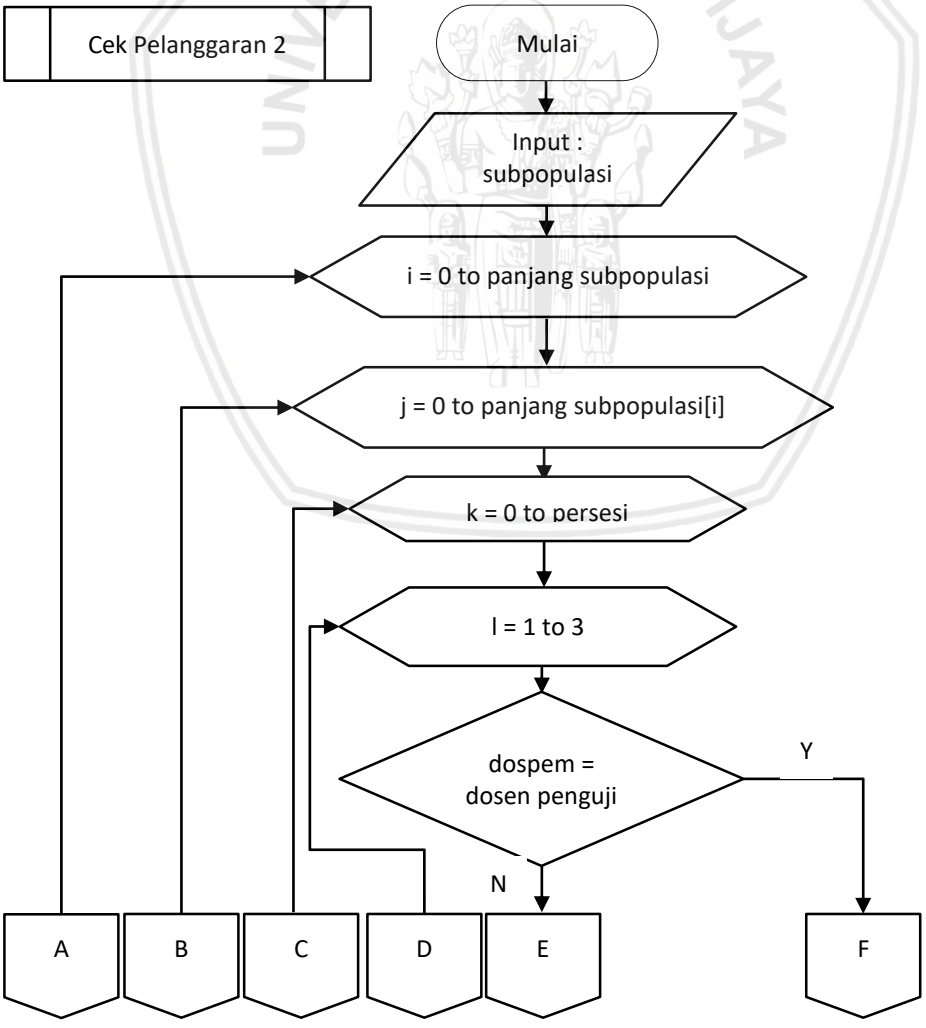


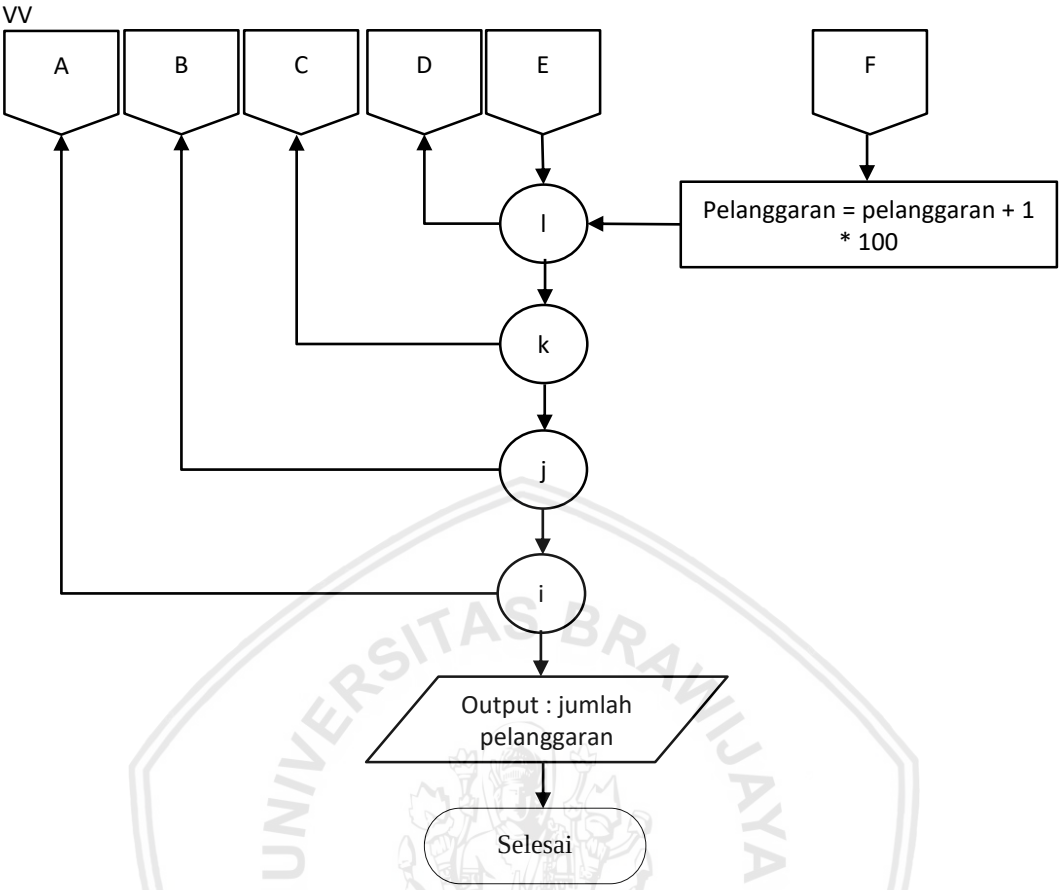


Gambar 4.11 Diagram Alir Cek Pelanggaran 3

4.1.5.4 Cek Pelanggaran 4

Pada tahap menghitung *fitness* akan dicek pelanggaran 4 yaitu dosen pembimbing harus berbeda dengan dosen penguji. Pelanggaran 4 ini merupakan jenis pelanggaran berat (*hard constraint*). Jika pelanggaran 4 ini dilanggar maka nilai pelanggarannya adalah 100. Diagram alir proses untuk cek pelanggaran 4 dapat dilihat pada Gambar 4.12.

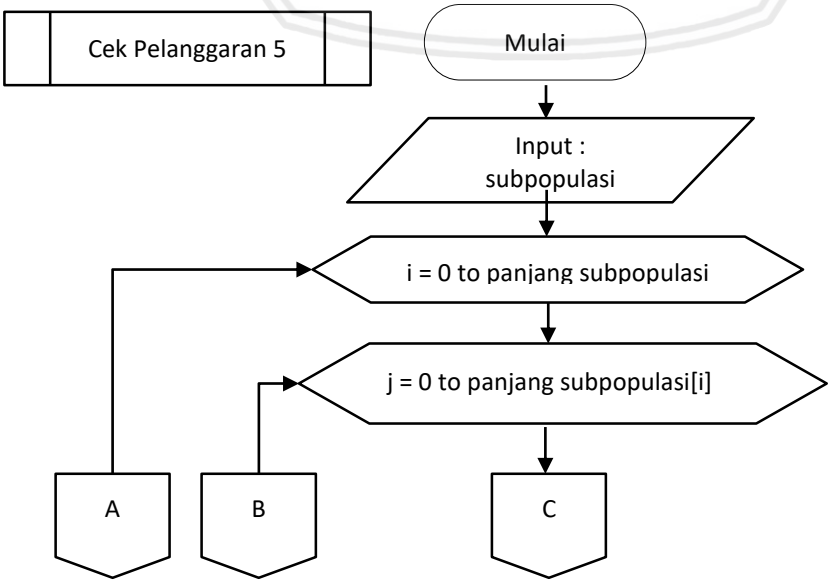


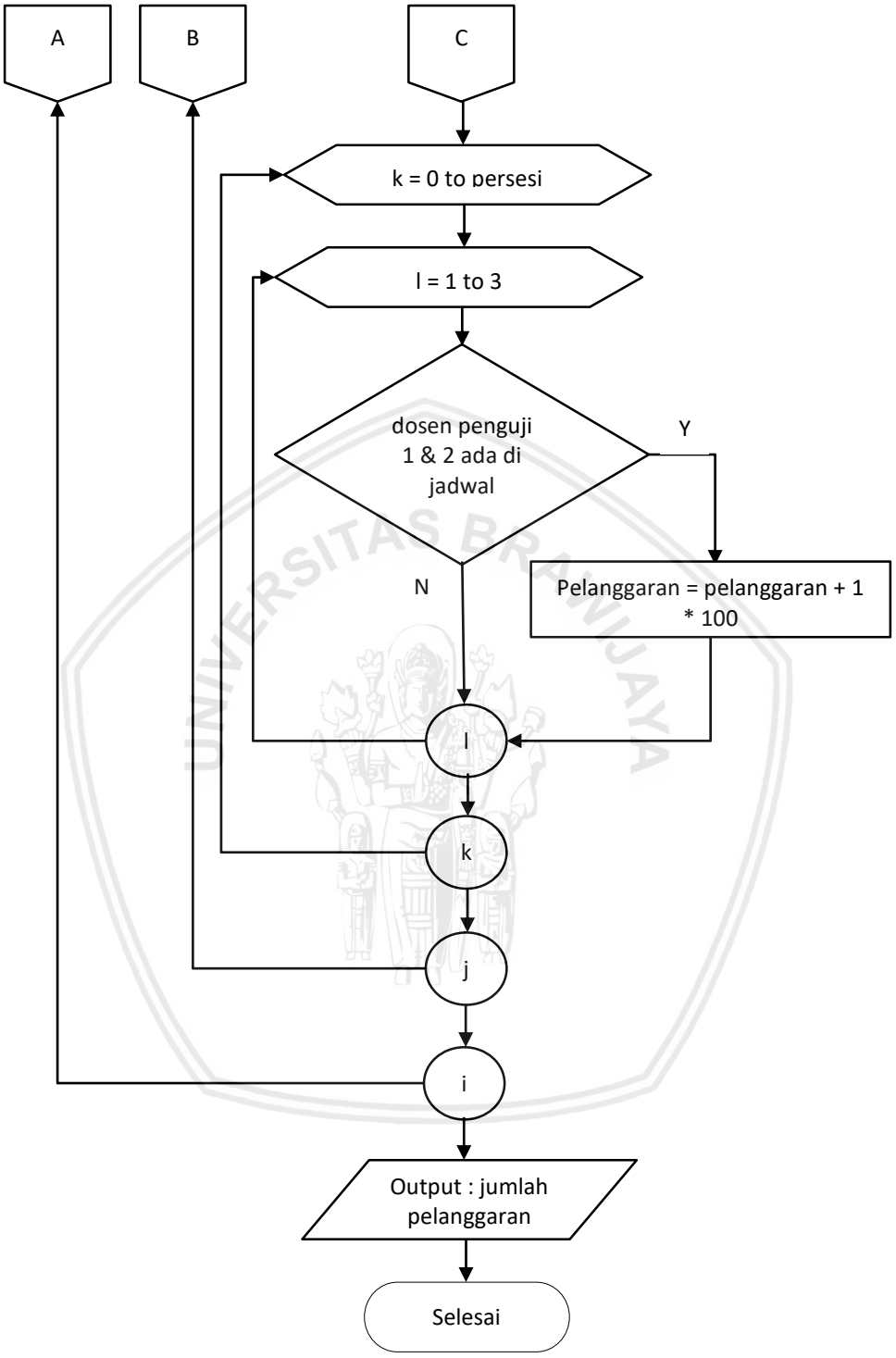


Gambar 4.12 Diagram Alir Cek Pelanggaran 4

4.1.5.5 Cek Pelanggaran 5

Pada tahap menghitung *fitness* akan dicek pelanggaran 5 yaitu jadwal dosen penguji bentrok dengan jadwal sidang. Pelanggaran 5 ini merupakan jenis pelanggaran *hard constraint.* Jika dilanggar maka nilai pelanggarannya adalah 100. Diagram alir proses untuk cek pelanggaran 4 dapat dilihat pada Gambar 4.13.



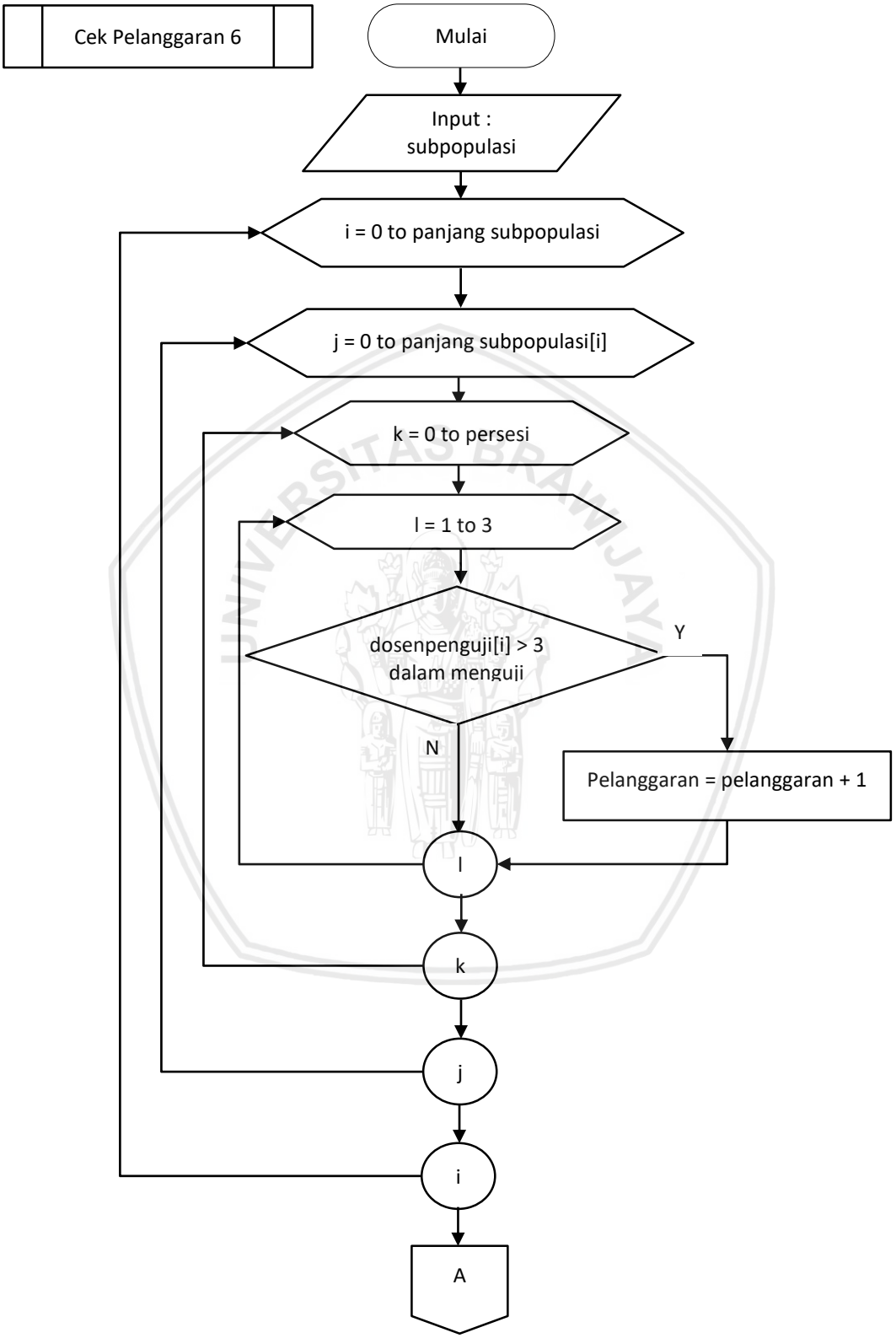


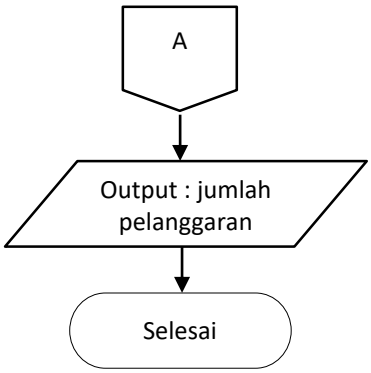
Gambar 4.13 Diagram Alir Cek Pelanggaran 5

4.1.5.6 Cek Pelanggaran 6

Pada tahap menghitung *fitness* akan dicek pelanggaran 6 yaitu dosen penguji menguji lebih dari 2 kali dalam satu hari. Pelanggaran 6 ini merupakan jenis pelanggaran *soft constraint.* Jika pelanggaran 6 ini dilanggar maka nilai

pelanggarannya adalah 1. Diagram alir proses untuk cek pelanggaran 6 dapat dilihat pada Gambar 4.14.

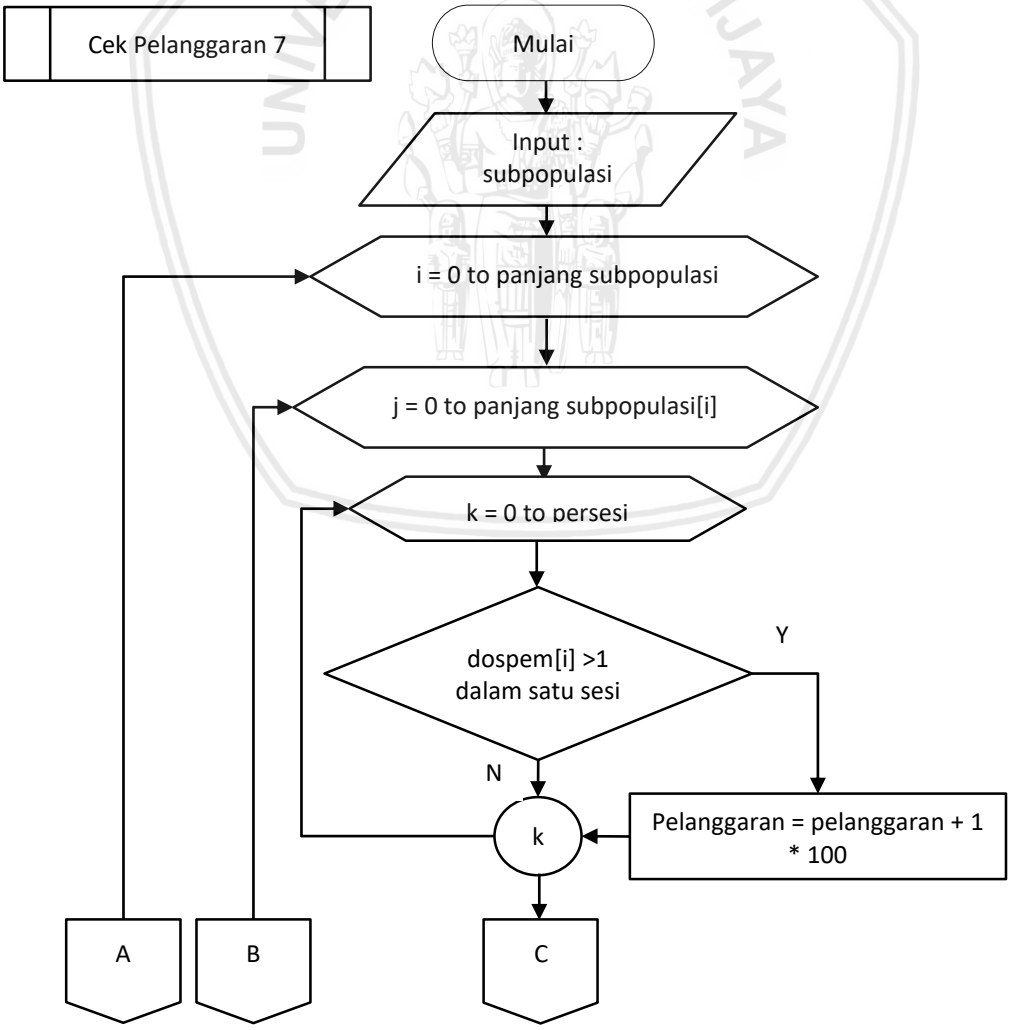


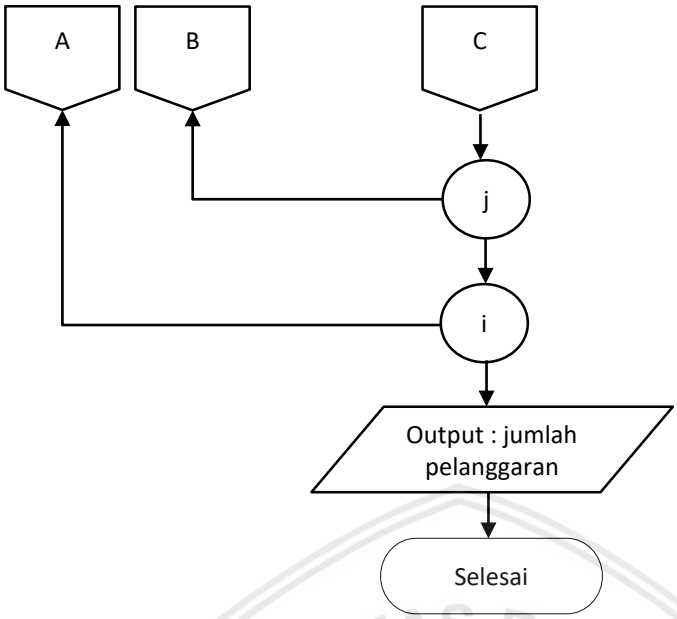


Gambar 4.14 Diagram Alir Cek Pelanggaran 6

4.1.5.7 Cek Pelanggaran 7

Pada tahap menghitung *fitness* akan dicek pelanggaran 7 yaitu dosen pembimbing hadir lebih dari satu kali dalam satu sesi. Pelanggaran 7 ini merupakan jenis pelanggaran *hard constraint.* Jika pelanggaran 7 ini dilanggar maka nilai pelanggarannya adalah 100. Diagram alir proses untuk cek pelanggaran 7 dapat dilihat pada Gambar 4.15.

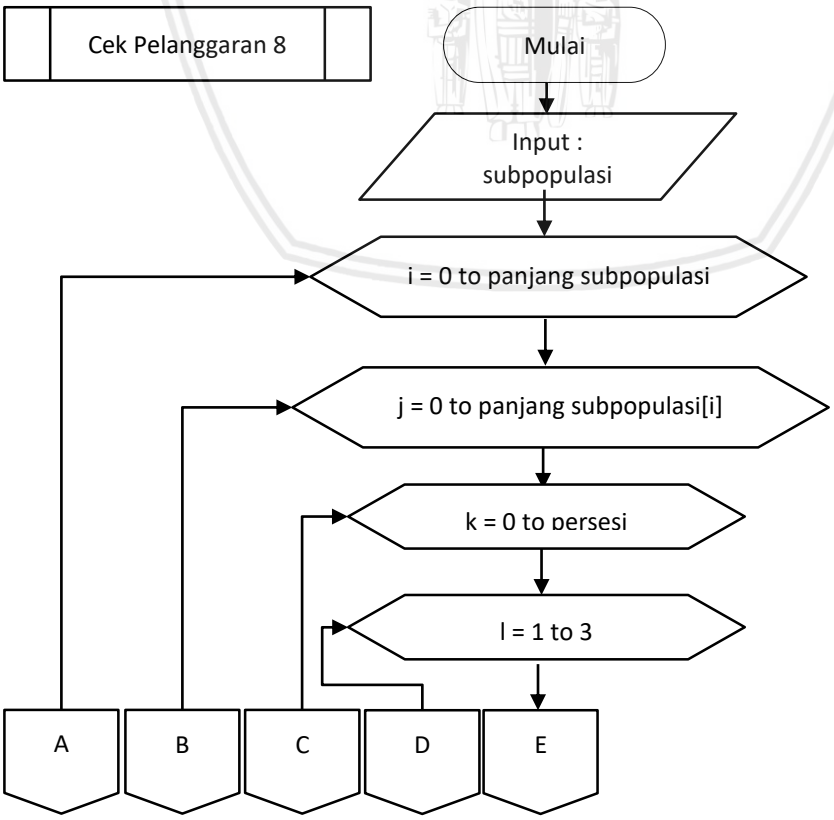


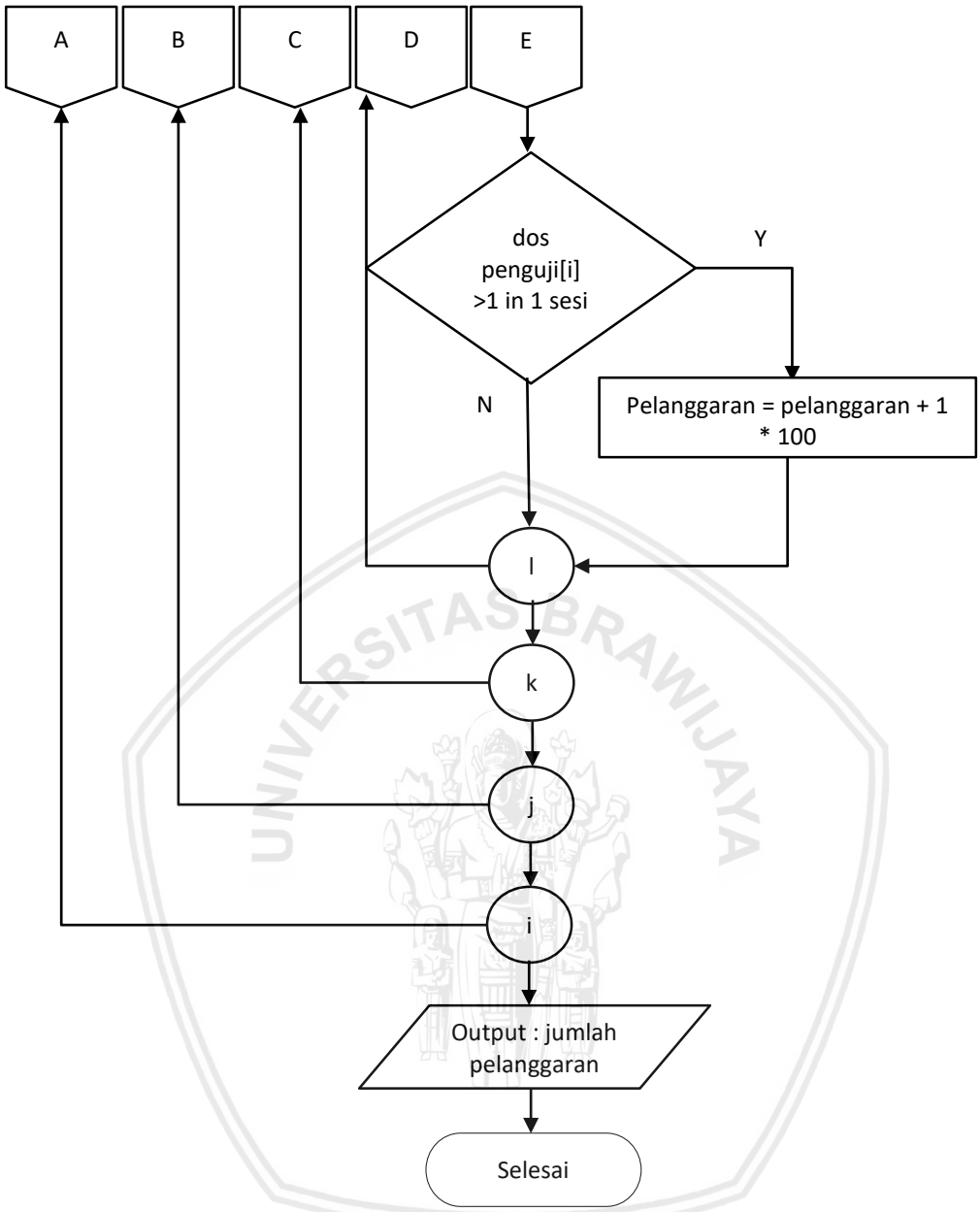


Gambar 4.15 Diagram Alir Cek Pelanggaran 7

4.1.5.8 Cek Pelanggaran 8

Pada tahap menghitung *fitness* akan dicek pelanggaran 8 yaitu dosen penguji hadir lebih dari satu kali dalam satu sesi. Pelanggaran 8 ini merupakan jenis pelanggaran *hard constraint*. Jika pelanggaran 8 ini dilanggar maka nilai pelanggarannya adalah 100. Diagram alir proses untuk cek pelanggaran 8 dapat dilihat pada Gambar 4.16.

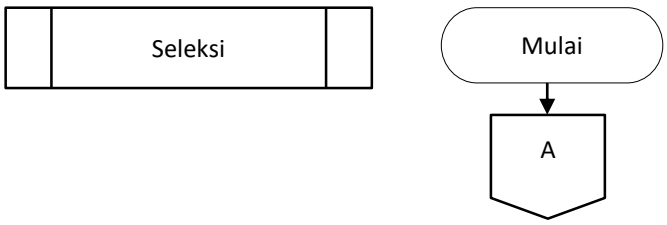


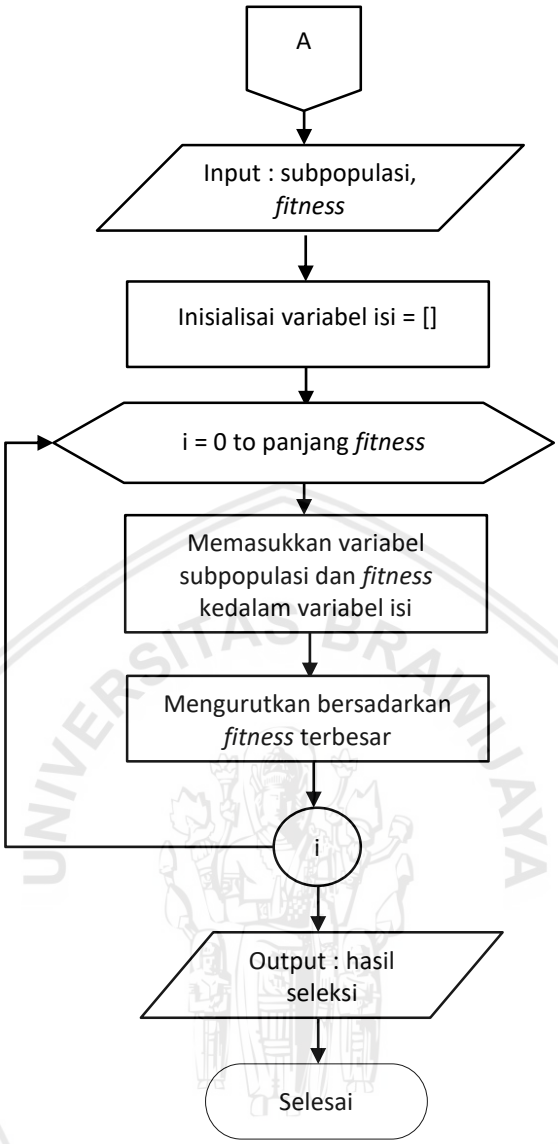


Gambar 4.16 Diagram Alir Cek Pelanggaran 8

4.1.6 Seleksi

Pada tahap seleksi, metode yang digunakan adalah *elitism selection*, dimana setiap subpopulasi akan diurutkan berdasarkan nilai *fitness* tertinggi berdasarkan perhitungan nilai *fitness* pada Persamaan 2.1. Diagram alir proses seleksi dapat dilihat pada Gambar 4.17.

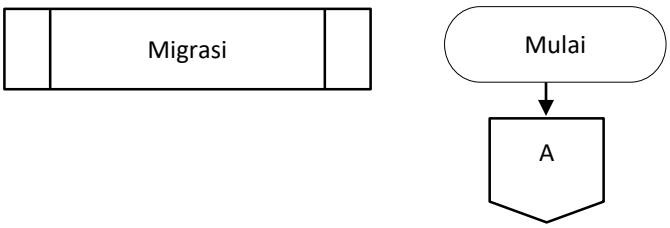


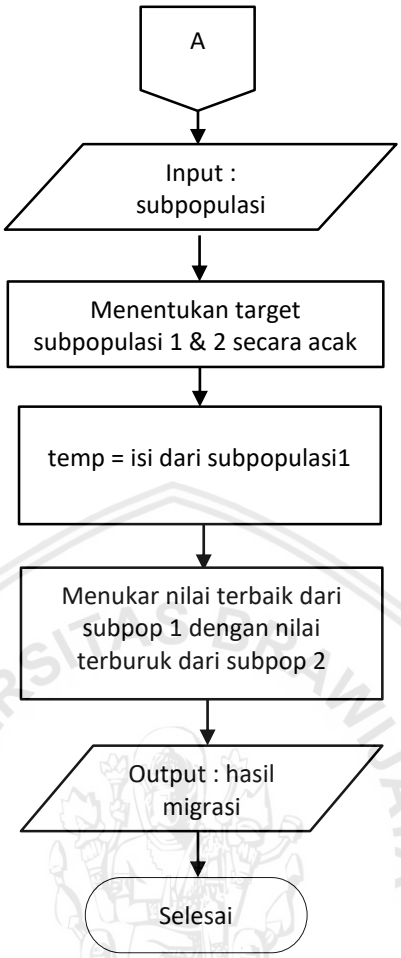


Gambar 4.17 Diagram Alir Seleksi

4.1.7 Migrasi

Tahap terakhir dalam algoritme genetika terdistribusi adalah migrasi yang digunakan untuk menambah keberagaman individu pada setiap subpopulasi. Pada penelitian ini migrasi dilakukan dengan cara menukar individu terbaik dari satu subpopulasi dengan individu terbaik dari populasi lainnya untuk mendapatkan hasil yang optimal. Diagram alir proses migrasi dapat dilihat pada Gambar 4.18.

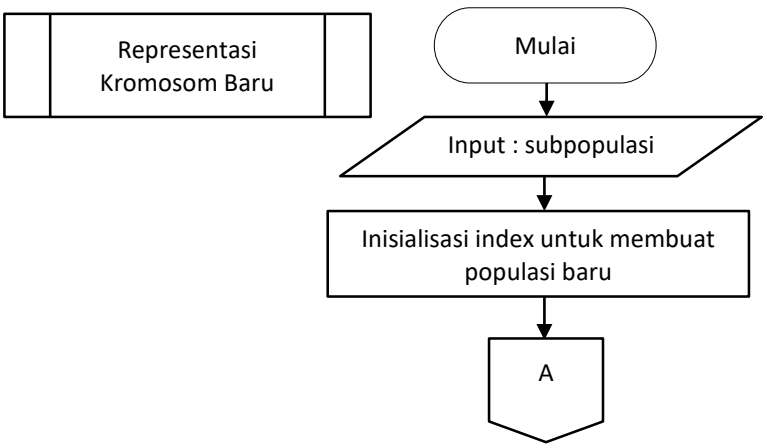


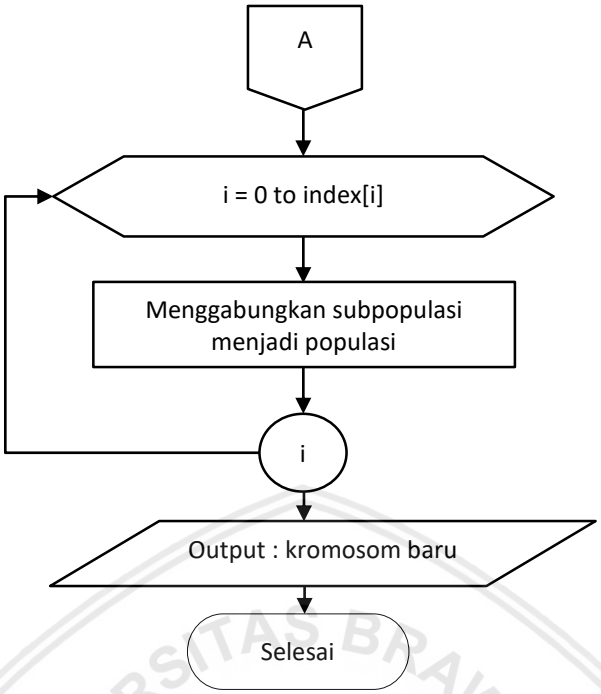


Gambar 4.18 Diagram Alir Migrasi

4.1.8 Representasi Kromosom Baru

Tahap terakhir dari penelitian ini adalah dengan menggabungkan semua subpopulasi hasil migrasi menjadi sebuah populasi baru. Representasi kromosom baru ini akan berlangsung sampai generasi terakhir. Nantinya hasil dari representasi kromosom inilah yang akan dijadikan hasil akhir algoritme genetika terdistribusi. Diagram alir proses representasi kromosom baru dapat dilihat pada Gambar 4.19.





Gambar 4.19 Diagram Alir Representasi Kromosom Baru

4.2 Perhitungan Manual

Perhitungan manual disini berupa contoh penerapan algoritme genetika terdistribusi untuk optimasi penjadwalan sidang skripsi. Pada perhitungan manual ini, akan digunakan 7 data mahasiswa dan 14 data dosen FILKOM UB. Dari 7 data mahasiswa dan 14 data dosen yang dipilih, akan dilakukan proses penerapan algoritme genetika terdistribusi. Selain itu setiap sub-populasi akan memiliki jumlah gen yang sama.

Pada perhitungan manual juga digunakan aturan-aturan yang jika dilanggar akan dikenakan nilai penalti yang nantinya akan digunakan pada perhitungan *fitness*. Aturan penalti tersebut dapat dilihat pada Tabel 4.1.

Tabel 4.1 Aturan Pinalti

No	Aturan	Jenis	Penalti
1	Jadwal mengajar dosen pembimbing bentrok dengan jadwal sidang skripsi	*Hard Constraint*	100
2	Dosen penguji 1 menguji mahasiswa yang berbeda keminatannya dengan major dosen penguji 1	*Hard Constraint*	100
3	Dosen penguji 2 menguji mahasiswa yang berbeda	*Hard Constraint*	100

	keminatannya dengan major dan minor dosen penguji 2		
4	Dosen pembimbing sama dengan dosen penguji	*Hard Constraint*	100
5	Jadwal mengajar dosen penguji bentrok dengan jadwal sidang	*Hard Constraint*	100
6	Dosen penguji menguji lebih dari 3 kali dalam satu hari	*Soft Constraint*	1
7	Dosen pembimbing hadir lebih dari satu dalam satu sesi	*Hard Constraint*	100
8	Dosen penguji hadir lebih dari satu kali dalam satu sesi	*Hard Constraint*	100

4.2.1 Representasi Kromosom

Membentuk popsize dapat dilakukan secara random, untuk perhitungan manual ini dibuat 1 populasi yang akan dipecah menjadi 2 subpopulasi berdasarkan sesi sidang skripsi, sehingga 1 subpopulasi merupakan kemungkinan 1 sesi. Jumlah gen masing-masing kromosom adalah 5 yang merupakan data mahasiswa dan 10 data dosen yang telah dipilih dari data *real*. Nilai kromosom dibangkitkan secara acak sesuai data mahasiswa dan dosen. Sub-populasi pada perhitungan manual ini ada 2, jadi setiap sub-populasi memiliki popsize dan banyak gen yang sama. Contoh bentuk kromosom untuk populasi 1 ada pada Gambar 4.20, untuk kromosom untuk sub-populasi 1 ada pada Gambar 4.21, untuk kromosom untuk sub-populasi 2 ada pada Gambar 4.22.

P1	1			2			3			4			5		
	11	2	5	18	14	6	27	8	19	20	1	12	13	24	25
P2	1			2			3			4			5		
	1	22	23	14	15	26	17	8	9	10	5	21	3	4	2

Gambar 4.20 Kromosom Populasi 1

P1	1			2			3			4			5		
	11	2	5	18	14	6	27	8	19	20	1	12	13	24	25
P2	1			2			3			4			5		
	1	22	23	14	15	26	17	8	9	10	5	21	3	4	2

Gambar 4.21 Kromosom Subpopulasi 1

P1	1			2			3			4			5		
	60	23	21	56	14	25	30	27	29	12	7	20	31	6	3
P2	1			2			3			4			5		
	21	3	17	78	6	13	54	25	2	9	8	3	22	1	9

Gambar 4.22 Kromosom Subpopulasi 2

4.2.2 *Crossover*

Pada *crossover* metode yang digunakan adalah metode *one-cut point* untuk setiap subpopulasi. Pada proses *crossover* untuk mengetahui berapa jumlah anak yang harus dihasilkan, akan digunakan rumus cr x PopSize. Dimana cr atau *crossover rate* merupakan data masukan dan PopSize merupakan ukuran populasi yang digunakan. Setelah jumlah offspring/*child* yang harus dihasilkan sudah diketahui maka dilanjutkan dengan proses *crossover* pada masing-masing sub-populasi. Proses *crossover one-cut point* dimulai dengan memilih 2 individu secara acak dari populasi awal dengan syarat *parent* tidak diperbolehkan sama, lalu menentukan titik potong *crossover* secara acak, selanjutnya menukar gen yang dibatasi titik potong silang dan digabungkan pada *child*. Untuk *crossover* pada sub-populasi 1 akan dijelaskan pada Gambar 4.23 dan 4.24 Untuk *crossover* pada sub-populasi 2 akan dijelaskan pada Gambar 4.25 dan 4.26.

Child = cr x popsize

= 0,9 x 2

= 1,8 dibulatkan menjadi 2 *Cut point*

P1	1			2			3			4			5		
	11	2	5	18	14	6	27	8	19	20	1	12	13	24	25
P2	1			2			3			4			5		
	1	22	23	14	15	26	17	8	9	10	5	21	3	4	2

Gambar 4.23 *Parent Crossover* Subpopulasi 1

C1	1			2			3			4			5		
	11	2	5	18	14	6	17	8	9	10	5	21	3	4	2
C2	1			2			3			4			5		
	1	22	23	14	15	26	27	8	19	20	1	12	13	24	25

Gambar 4.24 *Child Crossover* Subpopulasi 1

P1	1			2			3			4			5		
	60	23	21	56	14	25	30	27	29	12	7	20	31	6	3
P2	1			2			3			4			5		
	21	3	17	78	6	13	54	25	2	9	8	3	22	1	9

cut point

Gambar 4.25 *Parent Crossover* Subpopulasi 2

C1	1			2			3			4			5		
	60	23	21	56	14	25	30	27	29	9	8	3	22	1	9
C2	1			2			3			4			5		
	21	3	17	78	6	13	54	25	2	12	7	20	31	6	3

Gambar 4.26 *Child Crossover* Subpopulasi 2

4.2.3 Mutasi

Pada mutasi, setiap sub-populasi akan secara acak menggunakan metode *reciprocal exchange mutation* atau metode *random mutation*. Untuk mengetahui berapa jumlah anak yang harus dihasilkan, akan digunakan rumus mr x PopSize. Dimana mr atau *mutation rate* merupakan data masukan dan PopSize merupakan ukuran populasi yang digunakan. Setelah jumlah offspring/*child* yang harus dihasilkan sudah diketahui maka dilanjutkan dengan proses mutasi pada masing-masing sub-populasi. Berikut adalah cara untuk menghitung jumlah *child* mutasi :

Child = mr x popsize = 0.5 x 2 = 1

4.2.3.1 Random Mutation

Mutasi dengan menggunakan metode *random mutation* dilakukan dengan memilih secara acak *parent* yang nilainya akan diacak, setelah itu nilai tersebut diganti/diubah secara acak. Untuk mutasi pada sub-populasi 1 akan dijelaskan pada Gambar 4.27 dan 4.28.

P1	1			2			3			4			5		
	11	2	5	18	14	6	27	8	19	20	1	12	13	24	25

Gambar 4.27 *Parent* Mutasi (*Random Mutation)* Subpopulasi 1

C3	1			2			3			4			5		
	12	1	4	18	14	6	27	8	19	20	1	12	13	24	25

Gambar 4.28 *Parent* Mutasi (*Random Mutation*) Subpopulasi 1

4.2.3.2 Reciprocal Exchange Mutation

Mutasi dengan menggunakan metode *reciprocal exchange mutation* hal yang pertama dilakukan adalah memilih acak 2 *parent*, setelah itu memilih memilih acak gen mana yang akan ditukar dan akan dilakukan proses penukaran. Untuk mutasi pada sub-populasi 2 dengan menggunakan metode *reciprocal exchange mutation* akan dijelaskan pada Gambar 4.29 dan 4.30.

P1	1			2			3			4			5		
	60	23	21	56	14	25	30	27	29	12	7	20	31	6	3

Gambar 4.29 *Parent* Mutasi (*Reciprocal Exchange Mutation*) Subpopulasi 2

C3	1			2			3			4			5		
	60	23	21	12	7	20	30	27	29	56	14	25	31	6	3

Gambar 4.30 *Child* Mutasi (*Reciprocal Exchange Mutation*) Subpopulasi 2

4.2.4 Evaluasi

Proses evaluasi akan menggabungkan seluruh individu yaitu *parent* dan *child* yang telah melalui proses reproduksi (*crossover* dan mutasi) ke dalam sub-populasinya masing-masing, sehingga akan terbentuk sub-populasi 1 seperti pada Gambar 4.31 dan sub-populasi 2 seperti pada Gambar 4.32. Setelah menggabungkan individu maka akan di lakukan perhitungan penalti sesuai dengan pelanggaran aturan yang terjadi. Selanjutnya akan dihitung nilai *fitness* dengan menggunakan Persamaan 2.1.

Kromosom																P	F
P1	1			2			3			4			5			5	0,167
	11	2	5	18	14	6	27	8	19	20	1	12	13	24	25		
P2	1			2			3			4			5			6	0,143
	1	22	23	14	15	26	17	8	9	10	5	21	3	4	2		
C1	1			2			3			4			5			5	0,167
	11	2	5	18	14	6	17	8	9	10	5	21	3	4	2		
C2	1			2			3			4			5			4	0,200
	1	22	23	14	15	26	27	8	19	20	1	12	13	24	25		
C3	1			2			3			4			5			7	0,125
	12	1	4	18	14	6	27	8	19	20	1	12	13	24	25		

Gambar 4.31 Hasil Evaluasi Subpopulasi 1

Kromosom																P	F
P1	1			2			3			4			5			6	0,143
	60	23	21	56	14	25	30	27	29	12	7	20	31	6	3		
P2	1			2			3			4			5			3	0,250
	21	3	17	78	6	13	54	25	2	9	8	3	22	1	9		
C1	1			2			3			4			5			5	0,167
	60	23	21	56	14	25	30	27	29	9	8	3	22	1	9		
C2	1			2			3			4			5			6	0,143
	21	3	17	78	6	13	54	25	2	12	7	20	31	6	3		
C3	1			2			3			4			5			6	0,143
	60	23	21	12	7	20	30	27	29	56	14	25	31	6	3		

Gambar 4.32 Hasil Evaluasi Subpopulasi 2

4.2.5 Seleksi

Proses seleksi akan menggunakan metode *elitism selection*, dimana setelah melakukan proses evaluasi, maka nilai *fitness* akan diurutkan berdasarkan nilai *fitness* terbesar. Seleksi pada sub-populasi 1 dapat dilihat pada Gambar 4.33, sedangkan seleksi pada sub-populasi 2 dapat dilihat pada Gambar 4.34.

Kromosom																P	F
C2	1			2			3			4			5			4	0,200
	1	22	23	14	15	26	27	8	19	20	1	12	13	24	25		
P1	1			2			3			4			5			5	0,167
	11	2	5	18	14	6	27	8	19	20	1	12	13	24	25		
C1	1			2			3			4			5			5	0,167
	11	2	5	18	14	6	17	8	9	10	5	21	3	4	2		
P2	1			2			3			4			5			6	0,143
	1	22	23	14	15	26	17	8	9	10	5	21	3	4	2		
C3	1			2			3			4			5			7	0,125
	12	1	4	18	14	6	27	8	19	20	1	12	13	24	25		

Gambar 4.33 Hasil Seleksi Subpopulasi 1

Kromosom																P	F
P2	1			2			3			4			5			3	0,250
	21	3	17	78	6	13	54	25	2	9	8	3	22	1	9		
C1	1			2			3			4			5			5	0,167
	60	23	21	56	14	25	30	27	29	9	8	3	22	1	9		
P1	1			2			3			4			5			6	0,143
	60	23	21	56	14	25	30	27	29	12	7	20	31	6	3		
C2	1			2			3			4			5			6	0,143
	21	3	17	78	6	13	54	25	2	12	7	20	31	6	3		
C3	1			2			3			4			5			6	0,143
	60	23	21	12	7	20	30	27	29	56	14	25	31	6	3		

Gambar 4.34 Hasil Seleksi Subpopulasi 2

4.2.6 Migrasi

Proses migrasi merupakan proses akhir dari proses algoritme genetika terdistribusi. Pada proses migrasi akan dilakukan pertukaran satu individu dari satu sub-populasi dengan sub-populasi lainnya. Pertukaran akan dilakukan dengan menukar individu dengan nilai *fitness* tertinggi dari satu sub-populasi dengan individu dengan nilai *fitness* terendah. Migrasi pada penelitian ini akan diterapkan setiap 10 generasi sekali. Dapat dilihat pada Gambar 4.35 Dan Gambar 4.36 Bahwa individu C2 sebagai individu dengan nilai *fitness* terendah terbaik dari sub-populasi 1 ditukar dengan C3 sebagai individu dengan nilai *fitness* terendah dari sub-populasi 2.

Kromosom																P	F
C2	1			2			3			4			5			4	0,200
	1	22	23	14	15	26	27	8	19	20	1	12	13	24	25		
P1	1			2			3			4			5			5	0,167
	11	2	5	18	14	6	27	8	19	20	1	12	13	24	25		
C1	1			2			3			4			5			5	0,167
	11	2	5	18	14	6	17	8	9	10	5	21	3	4	2		

P2	1			2			3			4			5			6	0,143
	1	22	23	14	15	26	17	8	9	10	5	21	3	4	2		
C3	1			2			3			4			5			7	0,125
	12	1	4	18	14	6	27	8	19	20	1	12	13	24	25		

Gambar 4.35 Migrasi Subpopulasi 1

Kromosom																P	F
P2	1			2			3			4			5			3	0,250
	21	3	17	78	6	13	54	25	2	9	8	3	22	1	9		
C1	1			2			3			4			5			5	0,167
	60	23	21	56	14	25	30	27	29	9	8	3	22	1	9		
P1	1			2			3			4			5			6	0,143
	60	23	21	56	14	25	30	27	29	12	7	20	31	6	3		
C2	1			2			3			4			5			6	0,143
	21	3	17	78	6	13	54	25	2	12	7	20	31	6	3		
C3	1			2			3			4			5			6	0,143
	60	23	21	12	7	20	30	27	29	56	14	25	31	6	3		

Gambar 4.36 Migrasi Subpopulasi 2

Pada proses migrasi akan dilakukan pertukaran satu individu dari satu sub-populasi dengan sub-populasi lainnya. Pertukaran akan dilakukan dengan menukar individu terbaik dari masing-masing sub-populasi. Dapat dilihat pada Gambar 4.35 Dan Gambar 4.36 Bahwa individu P2 sebagai individu terbaik dari sub-populasi 1 ditukar dengan C3 sebagai individu terbaik dari sub-populasi 2.

4.2.7 Representasi Kromosom Baru

Setelah hasil migrasi dari setiap subpopulasi didapatkan, hasil dari migrasi setiap subpopulasi akan digabungkan menjadi satu populasi sehingga akan membentuk representasi krosom baru. Representasi kromosom baru dapat dilihat pada Gambar 4.37.

Kromosom															
P1	1			2			3			4			5		
	1	22	23	14	15	26	27	8	19	20	1	12	13	24	25
	1			2			3			4			5		
	60	23	21	12	7	20	30	27	29	56	14	25	31	6	3

Gambar 4.37 Hasil Algoritme Genetika Terdistribusi

4.3 Perancangan Pengujian

Perancangan pengujian digunakan untuk mengetahui bagaimana pengaruh parameter algoritme genetika terdistribusi dalam optimasi penjadwalan sidang skripsi. Hal ini dilakukan karena dalam algoritme genetika terdistribusi tidak diberikan nilai parameter secara pasti, sehinga perlu dilakukan pengujian untuk

mencari nilai parameter yang paling optimal untuk dapat menghasilkan nilai *fitness* terbaik. Jenis pengujian yang akan dilakukan pada penelitian ini antara lain:

1. Pengujian jumlah ukuran populasi (*popsize*).
2. Pengujian kombinasi antara *crossover rate* (cr) dan *mutation rate* (mr).
3. Pengujian jumlah generasi.

4.3.1 Perancangan Pengujian Jumlah Populasi (*popsize*)

Pengujian jumlah populasi digunakan untuk dapat mengetahui pada jumlah populasi keberapa DGA dapat memberikan hasil yang optimal untuk penjadwalan sidang skripsi. Parameter yang digunakan untuk pengujian populasi :

1. Jumlah ukuran populasi : ditentukan berdasarkan pengujian (5-15)
2. Jumlah ukuran sub-populasi : menggunakan data hari pelaksanaan sidang skripsi (5 subpopulasi).
3. Jumlah Generasi : 50.
4. Cr & Mr : 0,5 dan 0,5.

Tabel 4.2 Perancangan Pengujian Jumlah Populasi

Popsize	Nilai *fitness*					Rata-rata *fitness*
	Percobaan Populasi Ke-					
	1	2	3	4	5	
5						
7						
9						
11						
13						
15						

4.3.2 Perancangan Pengujian Kombinasi *Crossover Rate* dan *Mutation Rate*

Pengujian selanjutnya adalah menguji kombinasi nilai cr dan mr, pengujian ini dilakukan untuk mengetahui kombinasi nilai cr dan mr yang paling optimal untuk penjadwalan sidang skripsi. Untuk nilai cr dan mr dibatasi pada interval 0 sampai 1. Parameter yang digunakan untuk pengujian kombinasi cr dan mr adalah:

1. Jumlah ukuran populasi: menggunakan hasil dari pengujian jumlah populasi.
2. Jumlah ukuran subpopulasi : menggunakan data hari pelaksanaan sidang skripsi (5 subpopulasi).
3. Jumlah Generasi: 50 generasi.
4. Cr : interval 0 sampai 1
5. Mr : interval 1 sampai 0.

Tabel 4.3 Perancangan Pengujian Cr dan Mr

Cr		Nilai *fitness*					Rata-rata *fitness*
		Nilai *Fitness* Percobaan kombinasi cr dan mr ke-					
Cr	*Mr*	1	2	3	4	5	
0	1						
0,1	0,9						
0,2	0,8						
0,3	0,7						
0,4	0,6						
0,5	0,5						
0,6	0,4						
0,7	0,3						
0,8	0,2						
0,9	0,1						

4.3.3 Perancangan Pengujian Jumlah Generasi

Pengujian selanjutnya adalah pengujian jumlah generasi karena dari pengujian generasi dapat diketahui pada generasi keberapa algoritme genetika terdistribusi mendapatkan hasil yang optimal. Parameter yang digunakan untuk pengujian jumlah generasi adalah jumlah ukuran populasi yang diperoleh dari pengujian jumlah ukuran populasi yang dilakukan sebelumnya, jumlah generasi, nilai cr dan nilai mr.

1. Jumlah ukuran populasi : hasil terbaik dari pengujian jumlah populasi.
2. Jumlah ukuran subpopulasi : hari pelaksanaan sidang skripsi (5 subpopulasi).
3. Jumlah Generasi : ditentukan berdasarkan pengujian (100 - 3000).
4. Cr dan Mr : menggunakan hasil dari pengujian jumlah kombinasi cr dan mr

Tabel 4.4 Perancangan Pengujian Jumlah Generasi

Gen	Nilai *fitness*					Rata-rata *Fitness*
	Percobaan Populasi Ke-					
	1	2	3	4	5	
100						
250						
500						
750						
1000						
1250						
1500						
1750						
2000						
2250						
2500						

BAB 5 IMPLEMENTASI

Pada bab ini akan dilakukan implementasi dari perancangan algoritme genetika terdistribusi yang telah dilakukan pada Bab 4. Implementasi dilakukan untuk mengetahui bagaimana penerapan algoritme genetika terdistribusi dalam menyelesaikan permasalahan penjadwalan sidang skripsi agar lebih optimal dalam bentuk program sehingga dapat memberikan hasil seperti yang diharapkan sesuai dengan perancangan telah dilakukan.

Berdasarkan perancangan algoritme yang dilakukan sebelumnya, maka untuk implementasi algoritme genetika terdistribusi akan terdiri dari implementasi representasi kromosom, selanjutnya reproduksi yaitu *crossover* dengan metode *one-cut point,* mutasi dengan metode *random mutation* dan *reciprocal exchange mutation*, evaluasi, menghitung *fitness*, seleksi, migrasi, dan representasi kromosom baru. Implementasi dibuat dengan bahasa pemrograman Python.

5.1 Implementasi Algoritme Representasi Kromosom

Algoritme representasi kromosom digunakan untuk mengisinisalisasikan individu awal secara acak. Pada representasi kromosom akan di bagi menjadi 2, yaitu menginisialisasikan populasi yang setelahnya akan dipecah menjadi beberapa subpopulasi. Kode program untuk algoritme representasi kromosom dijelaskan pada Gambar 5.1 untuk populasi dan Gambar 5.2 untuk sub-populasi.

Algoritme Representasi Kromosom (Populasi)

```
1   def inisialisasi(self):
2           dosen = [i for i in range(len(self.dosen))]
3           mhss = [i for i in range(len(self.mhs))]
4           populasi =
    np.zeros((self.popsize,self.pjgkromosom,self.persesi,3))
5           for i in range(self.popsize):
6               random.shuffle(dosen)
7               for j in range(self.pjgkromosom):
8                   mhs = copy.deepcopy(mhss)
9                   random.shuffle(mhs)
10                  for k in range(self.persesi):
11                      populasi[i,j,k,0] = mhs.pop()
12                      indx1 =
    random.randint(0,len(self.dosen)-1)
13                      indx2 =
14  self.getRandom(indx1,len(self.dosen))
15                      populasi[i,j,k,1] = dosen[indx1]
16                      populasi[i,j,k,2] = dosen[indx2]
            return populasi
```

Gambar 5.1 *Souce Code* Representasi Kromosom (Populasi)

Algortime Representasi Kromosom (Subpopulasi)

```
1   def pecahSubPopulasi(self,pop,jmlpecah):
2           hasil_subpopulasi = []
3           self.jmlsub=jmlpecah
```

```
4           for j in range(0,self.pjgkromosom,jmlpecah):
5               subpop=[]
6               for i in range(self.popsize):
7                   subpop.append(pop[i][j:j+5])
8               hasil_subpopulasi.append(subpop)
9           return np.array(hasil_subpopulasi)
```

Gambar 5.2 *Souce Code* Representasi Kromosom (Sub-populasi)

Proses representasi kromosom pada populasi dimulai dengan baris ke 2-4 untuk menginisialisasikan variabel dosen, mhss, dan populasi untuk dijadikan nilai awal sebuah populasi. Selanjutnya pada baris 5-6 akan dilakukan perulangan untuk *popsize* dengan memberikan nilai acak pada variabel dosen. Baris ke 7-9 dilakukan perulangan untuk pjgkromosom dengan memberikan nilai acak pada variabel mahasiswa. Setelah itu baris ke 11-15 perulangan untuk persesi untuk mengambil nilai dari variabel dosen dan mahasiswa. Selanjutnya nilai tersebut akan di pisahkan dalam index yang berbeda. Pada baris 16 akan mengembalikan nilai variabel populasi.

Pada bagian representasi kromosom sub-populasi diawali pada baris ke 2-3 yaitu menginisialisasikan hasil_subpopulasi dengan isi array dan inisialisasi jmlsub-dengan nilai jmlpecah. Lalu pada baris ke 4-5 akan dilakukan perulangan untuk pjgkromosom dan jmlpecah dengan menginisialisasikan subpop. Baris ke 6-8 dilakukan perulangan untuk *popsize* dengan mengambil nilai dari populasi dan dilakukan pemecahan nilai populasi. Pada baris ke 9 digunakan untuk mengembalikan nilai variabel hasil_subpopulasi.

5.2 Implementasi Algoritme *Crossover*

Algoritme *crossover* dengan menggunakan metode *one-cut point* digunakan untuk menghasilkan populasi baru hasil dari perkawinan silang antara dua populasi. Kode program untuk algoritme *crossover* akan dijelaskan pada Gambar 5.3.

```
Algoritme Crossover
1       def crossover(self,subpopulasi):
2           jml_child = round((self.cr*self.popsize),0)
3           jml_anak = jml_child if jml_child%2==0 else
    jml_child+1
4           hasil = []
5           listparent = []
6           for i in range(int(jml_anak/2)):
7               target_p1 = random.randint(0,len(subpopulasi)-
    1)
8               target_p2 =
    self.getRandom(target_p1,len(subpopulasi))
9               cutoff = random.randint(0,self.persesi-1)
10              targetp1,targetp2 =
    self.cekcross(target_p1,target_p2,listparent)
11              p1 = subpopulasi[target_p1]
12              p2 = subpopulasi[target_p2]
13              c1,c2= self.crossover_ocp(p1,p2,cutoff)
14              hasil.append(c1)
15              hasil.append(c2)
16              listparent.append((targetp1,targetp2))
```

```
17          return np.array(hasil)
```

Gambar 5.3 *Source Code Crossover*

Algoritme *crossover* dimulai pada baris 2-5 dengan menginisialisasikan variabel jml_*child*, jml_anak, hasil, dan list*parent*. Selanjutnya pada baris 6-16 dilakukan perulangan untuk proses *crossover* dengan menentukan 2 target *parent*, setelah mendapatkan *parent* akan di cek apakah *parent* tersebut sudah melakukan proses *crossover* atau belum dengan memanggil method cekcross. Setelah itu dilakukan pertukaran individu antara kedua *parent* tersebut sehingga menghasilkan keturunan (*offspring*). Pada baris 17 akan dilakukan mengembalian nilai ke dalam variabel hasil.

5.3 Implementasi Algoritme Mutasi

Pada agoritme mutasi akan di bagi menjadi 2 metode, yaitu metode *random mutation* dan metode *reciprocal exchange mutation*. Mutasi pada digunakan untuk menambah jumlah kemungkinan populasi hasil yang optimal. Kode program untuk algoritme mutasi dengan metode *random mutation* akan dijelaskan pada Gambar 5.4. Sedangkan algoritme mutasi dengan metode *reciprocal exchange mutation* akan dijelaskan pada Gambar 5.5.

Algoritme Mutasi (*Random Mutation)*

```
1       def mutasi_random(self,subpopulasi):
2           jml_child = int(round((self.mr*self.popsize),0))
3           hasil = []
4           list_parent = []
5           for i in range(jml_child):
6               target_p = random.randint(0,len(subpopulasi)-1)
7               target_hari = random.randint(0,self.jmlsub-1)
8               target_sesi =
    random.randint(0,len(subpopulasi[0][0])-1)
9               subpop_terpilih =
    copy.deepcopy(subpopulasi[target_p])
10              subpop_terpilih[target_hari][target_sesi][0] =
    random.randint(0,len(self.mhs)-1)
11              d1 = random.randint(0,len(self.dosen)-1)
12              subpop_terpilih[target_hari][target_sesi][1]= d1
13              subpop_terpilih[target_hari][target_sesi][2] =
    self.getRandom(d1,len(self.dosen))
14              hasil.append(subpop_terpilih)
15              list_parent.append(target_p)
16          return hasil
```

Gambar 5.4 *Source Code Random Mutation*

Pada metode *random mutation* baris 2-4 digunakan untuk menginisialisasikan variabel jml_*child*, hasil, dan list_*parent*. Setelah itu pada baris 5 dilakukan perulangan proses *random mutation*. Pada baris ke 6-8 akan dipilih target *parent* dan gen yang akan melakukan proses mutasi. Pada baris ke 9-14 gen yang terpilih dari *parent* tersebut akan diganti nilainya dengan nilai secara acak. Setelah itu pada baris 15 akan diambil nilai target_p ke dalam list_parent. Pada baris 16 akan dilakukan mengembalian nilai ke dalam variabel hasil.

Algortime Mutasi (*Reciprocal Exchange Mutation*)	
1	def mutasi_exchange(self,subpopulasi):
2	jml_*child* = int(round((self.mr*self.popsize),0))
3	hasil = []
4	list_*parent* = []
5	for i in range(jml_*child*):
6	target_p = random.randint(0,len(subpopulasi)-1)
7	target_pindah1 = random.randint(0,self.jmlsub-1)
8	target_pindah2 = self.getRandom(target_pindah1,self.jmlsub)
9	pop_terpilih = copy.deepcopy(subpopulasi[target_p])
10	temp = pop_terpilih[target_pindah1]
11	pop_terpilih[target_pindah1] = pop_terpilih[target_pindah2]
12	pop_terpilih[target_pindah2] = temp
13	list_*parent*.append(target_p)
14	hasil.append(pop_terpilih)
15	return np.array(hasil)

Gambar 5.5 *Souce Code Reciprocal Exchange Mutation*

Pada algoritme mutasi dengan metode *reciprocal exchange mutation* baris ke 2-4 dilakukan inisialisasi variabel jml_*child*, hasil, dan list_*parent*. Baris ke 5 dilakukan proses mutasi dengan perulangan sesuai dengan variabel jml_*child*. Pada baris ke 6 akan ditentukan 2 target *parent* yang akan melakukan proses mutasi. Setelah itu pada baris ke 7-12 *parent* yang terpilih akan melakukan pertukaran gen sesuai dengan gen yang terpilih. Baris 13 variabel target_p akan dimasukkan nilainya pada list_parent. Lalu pada baris ke 14 nilai variabel pop_terpilih akan dimasukkan ke dalam hasil. Pada baris 15 akan dilakukan mengembalian nilai ke dalam variabel hasil.

5.4 Implementasi Algoritme Menghitung *Fitness*

Setelah melakukan proses reproduksi, akan dilakukan perhitungan nilai *fitness* yang digunakan untuk mengukur apakah populasi tersebut baik untuk dijadikan solusi. Kode program algoritme menghitung *fitness* akan dijelaskan pada Gambar 5.6.

Algoritme Menghitung *Fitness*	
1	def *fitness*(self,subpopulasi,subke):
2	pelanggaran = [self.pelanggaran1(subpopulasi,subke), self.pelanggaran2(subpopulasi),self.pelanggaran3(subpopulasi), self.pelanggaran4(subpopulasi), self.pelanggaran5(subpopulasi,subke), self.pelanggaran6(subpopulasi),self.pelanggaran7(subpopulasi), self.pelanggaran8(subpopulasi)]
3	const = np.zeros_like(pelanggaran[0])
4	for i in range(len(pelanggaran)):
5	const = const + pelanggaran[i]

```
6           fitness = 1/(1+const)
7           return fitness
```

Gambar 5.6 *Source Code* Menghitung *Fitness*

Penjelasan kode program diatas akan dimulai dari baris 2 untuk memanggil method pelanggaran 1 sampai pelanggaran 8. Setelah itu pada baris ke 3 dilakukan inisialisasi variabel const, lalu pada baris ke 4-6 akan dilakukan perhitungan nilai *fitness* sesuai dengan rumus persamaan menghitung *fitness*. Pada baris 6 akan dilakukan mengembalian nilai ke dalam variabel fitness.

5.4.1 Implementasi Algotime Cek Pelanggaran 1

Pelanggaran 1 adalah jika jadwal mengajar dosen pembimbing bentrok dengan jadwal sidang skripsi maka akan mendapatkan pinalti bernilai 100. Kode program untuk mengecek pelanggaran 1 terdapat pada Gambar 5.7.

```
Algoritme Cek Pelanggaran 1
1   def pelanggaran1(self,subpopulasi,subke):
2   pelanggaran = list()
3           jml=5*subke
4           for i in range(len(subpopulasi)):#untuk setiap
    popsize
5               jumlah_pelanggaran = 0
6               for j in range(len(subpopulasi[i])):#untuk
    setiap sesi
7                   sesi = 'sesi'+str(j+1+jml)
8                   list_dosen= [i for i in
    self.jadwaldosen[sesi]]
9                   for k in range(self.persesi): #untuk satu
    kali sidang setiap sesi
10                      mhs = subpopulasi[i][j][k][0]
11                      dosen1 = self.mhs['Dospem 1'][mhs]
12                      dosen2 = self.mhs['Dospem 2'][mhs]
13                      if dosen1 in list_dosen:
                            jumlah_pelanggaran =
14  jumlah_pelanggaran +1
15                      if dosen2 in list_dosen:
16                          jumlah_pelanggaran =
    jumlah_pelanggaran +1
17              pelanggaran.append(jumlah_pelanggaran*100)
18          return np.array(pelanggaran)
```

Gambar 5.7 *Source Code* Cek Pelanggaran 1

Pada kode program ini, baris ke 2- 3 melakukan inisialisasi variabel pelanggaran dan jml. Setelah itu baris 4-5 dilakukan perulangan untuk subpopulasi dengan inisialisasi jumlah_pelanggaran = 0. Baris ke 6-8 dilakukan perulangan untuk subpopulasi pada index ke i dengan inisialisasi variabel sesi dan list_dosen. Baris ke 9-12 perulangan untuk persesi pada satu kali sidang disetiap sesi dengan inisialisasi variabel mhs, dosen1, dosen2. Pada baris ke 13-16 akan dicek apabila dosen1 dan dosen2 ada pada list_dosen maka variabel jumlah_pelanggaran akan bertambah. Baris ke 17 akan dilakukan pengambilan nilai untuk mengisi variabel jumlah_pelanggaran dan membalikan nilai ke variabel pelanggaran. Pada baris 18 akan dilakukan mengembalian nilai ke dalam variabel pelanggaran.

5.4.2 Implementasi Algoritme Cek Pelanggaran 2

Pelanggaran 2 adalah jika keminatan dosen penguji pertama tidak sama dengan keminatan mahasiswa maka akan mendapatkan pinalti bernilai 100. Kode program untuk mengecek pelanggaran 2 terdapat pada Gambar 5.8.

```
Algoritme Cek Pelanggaran 2
1   def pelanggaran2(self,subpopulasi):
2           pelanggaran=[]
3           for i in range(len(subpopulasi)):
4               jumlahpelanggaran = 0
5               for j in range(len(subpopulasi[i])):
6                   for k in range(self.persesi):
7                       indxmhs = subpopulasi[i][j][k][0]
8                       keminatanmhs =
    self.mhs['Keminatan'][indxmhs]
9                       indxdsn = subpopulasi[i][j][k][1]
10                      keminatandsn = self.dosen['Major'][indxdsn]
11                      if keminatanmhs != keminatandsn:
12                          jumlahpelanggaran=jumlahpelanggaran+1

13              pelanggaran.append(jumlahpelanggaran*100)
14          return np.array(pelanggaran)
```

Gambar 5.8 *Source Code* Cek Pelanggaran 2

Pada baris ke 2 dilakukan inisialisasi variabel pelanggaran. Setelah itu baris 3-4 dilakukan perulangan untuk subpopulasi dengan inisialisasi jumlah_pelanggaran = 0. Pada baris ke 5 dilakukan perulangan untuk subpopulasi pada index ke i. Baris ke 6-10 dilakukan perulangan untuk persesi dengan inisialisasi variabel indxmhs, keminatanmhs, indxdsn, dan keminatandsn. Selanjutnya baris ke 11-12 akan dicek apabila keminatanmhs dan keminatandsn tidak sama maka jumlah pelanggaran akan bertambah. Baris ke 13 akan dilakukan pengambilan nilai untuk mengisi variabel jumlah_pelanggaran dikali 100 dan membalikan nilai ke variabel pelanggaran. Pada baris 14 akan dilakukan mengembalian nilai ke dalam variabel pelanggaran.

5.4.3 Implementasi Algoritme Cek Pelanggaran 3

Pelanggaran 3 adalah jika keminatan dosen penguji kedua tidak sama dengan keminatan mahasiswa maka akan mendapatkan pinalti bernilai 1. Kode program untuk mengecek pelanggaran 3 terdapat pada Gambar 5.9.

```
Algoritme Cek Pelanggaran 3
1      def pelanggaran3(self,subpopulasi):
2           pelanggaran=[]
3           for i in range(len(subpopulasi)):
4               jumlahpelanggaran = 0
5               for j in range(len(subpopulasi[i])):
6                   for k in range(self.persesi):
7                       indxmhs = subpopulasi[i][j][k][0]
8                       keminatanmhs =
    self.mhs['Keminatan'][indxmhs]
9                       indxdsn = subpopulasi[i][j][k][2]
10                      keminatandsn =
    self.dosen['Mayorminor'][indxdsn].split('.')
```

```
11                          if keminatanmhs not in keminatandsn:
12                              jumlahpelanggaran=jumlahpelanggaran+1
13                  pelanggaran.append(jumlahpelanggaran*100)
14              return np.array(pelanggaran)
```

Gambar 5.9 *Source Code* Cek Pelanggaran 3

Pada cek pelanggaran 3 baris ke 2 dilakukan inisialisasi variabel pelanggaran. Setelah itu baris 3-4 dilakukan perulangan untuk subpopulasi dengan inisialisasi jumlah_pelanggaran = 0. Selanjutnya baris ke 5 dilakukan perulangan untuk subpopulasi pada index ke i. Pada baris ke 6 dilakukan perulangan untuk persesi dengan inisialisasi variabel indxmhs, keminatanmhs, indxdsn, dan keminatandsn. Baris ke 7-12 akan dicek apabila keminatanmhs dan keminatandsn tidak sama maka jumlah pelanggaran akan bertambah. Baris ke 13-14 akan dilakukan pengambilan nilai untuk mengisi variabel jumlah_pelanggaran dan membalikan nilai ke variabel pelanggaran.

5.4.4 Implementasi Cek Pelanggaran 4

Pelanggaran 4 adalah jika dosen penguji sama dengan dosen pembimbing maka akan mendapatkan pinalti bernilai 100. Kode program untuk mengecek pelanggaran 4 terdapat pada Gambar 5.10.

```
Algoritme Cek Pelanggaran 4
1       def pelanggaran4(self,subpopulasi):
2           pelanggaran = []
3           for i in range(len(subpopulasi)):
4               jumlahpelanggaran = 0
5               for j in range(len(subpopulasi[i])):
6                   for k in range(self.persesi):
7                       indxmhs = subpopulasi[i][j][k][0]
                        dosenmhs = [self.mhs['Dospem
8   1'][indxmhs],self.mhs['Dospem 2'][indxmhs]]
                        for l in range(1,3):
9                           indxdsn = subpopulasi[i][j][k][l]
10                          namadosen =
11  self.dosen['Dosen'][indxdsn]
12                          if namadosen in dosenmhs:
13                           jumlahpelanggaran=jumlahpelanggaran+1
14              pelanggaran.append(jumlahpelanggaran*100)
15          return np.array(pelanggaran)
```

Gambar 5.10 *Source Code* Cek Pelanggaran 4

Kode program diatas pada baris baris ke 2 dilakukan inisialisasi variabel pelanggaran. Baris 3-4 dilakukan perulangan untuk subpopulasi dengan inisialisasi jumlah_pelanggaran = 0. Selanjutnya baris ke 5 dilakukan perulangan untuk subpopulasi pada index ke i. Pada baris ke 6-8 dilakukan perulangan untuk persesi dengan inisialisasi variabel indxmhs, dosenmhs, indxdsn, dan namadosen. Pada baris ke 9-11 dilakukan perulangan untuk melihat index dari dosen. Baris ke 12-13 akan dicek apabila namadosen ada pada dosenmhs maka jumlah pelanggaran akan bertambah. Baris ke 14-15 akan dilakukan pengambilan nilai untuk mengisi

variabel jumlah_pelanggaran dikali 100 dan membalikan nilai ke variabel pelanggaran.

5.4.5 Implementasi Algoritme Cek Pelanggaran 5

Pelanggaran 5 adalah jika jadwal mengajar dosen penguji bentrok dengan jadwal sidang maka akan mendapatkan pinalti bernilai 100. Kode program untuk mengecek pelanggaran 5 terdapat pada Gambar 5.11.

```
Algoritme Cek Pelanggaran 5
1   def pelanggaran5(self,subpopulasi,subke):
2           pelanggaran = list()
3           jml=5*subke
4           for i in range(len(subpopulasi))
5               jumlah_pelanggaran = 0
6               for j in range(len(subpopulasi[i])):
7                   sesi = 'sesi'+str(j+1+jml)
8                   list_dosen= [i for i in
    self.jadwaldosen[sesi]]
9                   for k in range(self.persesi):
10                      for l in range(1,3):
11                          indexdosen =
    subpopulasi[i][j][k][l]
12                          dosen =
    self.dosen['Dosen'][indexdosen]
13                          if dosen in list_dosen:
14  jumlah_pelanggaran=jumlah_pelanggaran+1
15              pelanggaran.append(jumlah_pelanggaran*100)
16          return np.array(pelanggaran)
```

Gambar 5.11 *Source Code* Cek Pelanggaran 5

Pada cek pelanggaran 5 baris ke 2-3 dilakukan inisialisasi variabel pelanggaran dan jml. Setelah itu baris 4-5 dilakukan perulangan untuk subpopulasi dengan inisialisasi jumlah_pelanggaran = 0. Selanjutnya baris ke 6-8 dilakukan perulangan untuk subpopulasi pada index ke i dengan inisialisasi variabel sesi dan list_dosen. Pada baris ke 9 dilakukan perulangan untuk persesi. Baris ke 10-12 dilakukan perulangan untuk nilai 1 sampai 3 dengan menginisialisasikan variabel indexdosen dan dosen. Baris ke 13-14 akan dicek apabila dosen ada pada list_dosen maka jumlah pelanggaran akan bertambah. Baris ke 15 dan 16 akan dilakukan pengambilan nilai untuk mengisi variabel jumlah_pelanggaran dikali 100 dan membalikan nilai ke variabel pelanggaran.

5.4.6 Implementasi Algoritme Cek Pelanggaran 6

Pelanggaran 6 adalah jika dosen penguji menguji lebih dari 3 kali dalam satu hari maka akan mendapatkan pinalti bernilai 1. Kode program untuk mengecek pelanggaran 6 terdapat pada Gambar 5.12.

```
Algoritme Cek Pelanggaran 6
1       def pelanggaran6(self,subpopulasi):
2           pelanggaran=[]
3           for i in range(len(subpopulasi)):
4               dsn = [0 for i in range(len(self.dosen))]
5               for j in range(len(subpopulasi[i])):
```

```
6                   for k in range(self.persesi):
7                       for l in range(1,3):
8                           dospeng = subpopulasi[i][j][k][l]
9                           dsn[int(dospeng)] =
    dsn[int(dospeng)] + 1
10              pelanggaran.append(self.sum_pelanggaran6(dsn))
11          return np.array(pelanggaran)
12      def sum_pelanggaran6(self,dosen):
13          jumlahpelanggaran = 0
14          for i in dosen:
15              if dosen[i]>3:
16                  jumlahpelanggaran=jumlahpelanggaran+1
17          return jumlahpelanggaran
```

Gambar 5.12 *Source Code* Cek Pelanggaran 6

Pada kode program cek pelanggaran 6 baris ke 2 dilakukan inisialisasi variabel pelanggaran. Setelah itu baris 3-4 dilakukan perulangan untuk subpopulasi dengan inisialisasi variabel dsn. Selanjutnya baris ke 5 dilakukan perulangan untuk subpopulasi pada index ke i. Pada baris ke 6 dilakukan perulangan untuk persesi. Baris ke 7-11 dilakukan perulanga untuk nilai 1 sampai 3 dengan inisialisasi variabel dospeng dan dsn, selanjutnya akan dipanggil method sum_pelanggaran6 untuk mendapatkan nilai hasil pelanggaran dan mengembalikan nilai ke variabel pelanggaran. Pada baris 12-17 merupakan method sum_pelanggaran6 dengan perulangan pada variabel dosen. Jika dosen dengan index ke i lebih dari 3 maka jumlahpelanggaran akan bertambah.

5.4.7 Implementasi Algoritme Cek Pelanggaran 7

Pelanggaran 7 adalah jika dosen pembimbing hadir lebih dari satu dalam satu sesi maka akan mendapatkan pinalti bernilai 100. Kode program untuk mengecek pelanggaran 7 terdapat pada Gambar 5.13.

```
Algoritme Cek Pelanggaran 7
1    def pelanggaran7(self,subpopulasi):
2           list_dosen = [i for i in self.dosen['Dosen']]
3           pelanggaran=[]
4           for i in range(len(subpopulasi)):
5               jml_pelanggaran = 0
6               for j in range(len(subpopulasi[i])):
7                   dsn = [0 for i in range(len(list_dosen))]
8                   for k in range(self.persesi):
9                       indxmhs = subpopulasi[i][j][k][0]
10                      dospem1 = self.mhs['Dospem 1'][indxmhs]
11                      index_dospem1 =
    list_dosen.index(dospem1)
12                      dsn[int(index_dospem1)] =
    dsn[int(index_dospem1)]+1
13                      dospem2 = self.mhs['Dospem 2'][indxmhs]
14                      try:
15                          index_dospem2 =
    list_dosen.index(dospem2)
16                          dsn[int(index_dospem2)] =
    dsn[int(index_dospem2)]+1
17                      except:
18                          pass
```

```
19                  jml_pelanggaran =
    jml_pelanggaran+np.sum(np.where(np.array(dsn)>1,1,0))#self.
    sum_pelanggaran78(dsn)
20              pelanggaran.append(jml_pelanggaran*100)
21          return np.array(pelanggaran)
```

Gambar 5.13 *Source Code* Cek Pelanggaran 7

Cek pelanggaran 7 pada baris ke 2-3 dilakukan inisialisasi variabel list_dosen dan pelanggaran. Baris 4-5 dilakukan perulangan untuk subpopulasi dengan inisialisasi variabel jml_pelanggaran = 0. Selanjutnya baris ke 6-7 dilakukan perulangan untuk subpopulasi pada index ke I dengan menginisialisasikan variabel dsn. Pada baris ke 8-13 dilakukan perulangan untuk persesi dengan inisialisasi variabel indxmhs, dospem1, index_dospem1, dsn, jml_pelanggaran, dan dospem2. Baris ke 14-18 dilakukan apabila dosen pembimbing kedua memiliki nilai maka akan melakukan perintah untuk mengecek jadwal. Baris ke 19-21 akan dilakukan pengambilan nilai untuk mengisi variabel jml_pelanggaran dikali 100 dan membalikan nilai ke variabel pelanggaran.

5.4.8 Implementasi Algoritme Cek Pelanggaran 8

Pelanggaran 8 adalah jika dosen penguji hadir lebih dari satu dalam satu sesi maka akan mendapatkan pinalti bernilai 100. Kode program untuk mengecek pelanggaran 8 terdapat pada Gambar 5.14.

```
Algoritme Cek Pelanggaran 8
1       def pelanggaran8(self,subpopulasi):
2           pelanggaran=[]
3           for i in range(len(subpopulasi)):
4               jml_pelanggaran = 0
5               for j in range(len(subpopulasi[i])):
6                   dsn = [0 for i in range(len(self.dosen))]
7                   for k in range(self.persesi):
8                       for l in range(1,3):
9                           dospem = subpopulasi[i][j][k][l]
10                          dsn[int(dospem)] = dsn[int(dospem)]+1
11                  jml_pelanggaran =
    jml_pelanggaran+self.sum_pelanggaran78(dsn)
12              pelanggaran.append(jml_pelanggaran*100)
13          return np.array(pelanggaran)
```

Gambar 5.14 *Source Code* Cek Pelanggaran 8

Kode program diatas pada baris ke 2 dilakukan inisialisasi variabel pelanggaran. Selanjutnya baris 3-4 dilakukan perulangan untuk subpopulasi dengan inisialisasi variabel jml_pelanggaran = 0. Pada baris ke 5-6 dilakukan perulangan untuk subpopulasi pada index ke i dengan menginisialisasikan variabel dsn. Baris ke 7 dilakukan perulangan untuk persesi, pada baris ke 8-11 dilakukan perulangan untuk nilai 1 sampai 3 dengan inisialisasi variabel dospem, dsn, dan jml_pelanggaran. Baris ke 12-13 akan dilakukan pengambilan nilai untuk mengisi variabel jml_pelanggaran dikali 100 dan membalikan nilai ke variabel pelanggaran.

5.5 Implementasi Algoritme Seleksi

Setelah melakukan perhitungan *fitness* dengan mengecek semua pelanggarannya, maka akan dilakukan proses seleksi dengan metode *elitism selection*. Pada algoritme seleksi akan dilakukan mengurutan nilai *fitness* dari yang paling besar ke nilai *fitness* yang paling kecil. Kode program untuk implementasi algoritme seleksi terdapat pada Gambar 5.15.

```
Algoritme Seleksi
1       def seleksi(self,subpop,fitness):
2           isi = []
3           for i in range(len(fitness)):
4               isi.append((fitness[i],subpop[i]))
5           hasilsort = sorted(isi,key=lambda x:x[0],reverse=True)
6           isiKembali=[]
7           for i in range(self.popsize):
8               isiKembali.append(hasilsort[i][1])
9           return isiKembali
```

Gambar 5.15 *Source Code* Seleksi

Kode program implementasi algoritme seleksi pada baris ke 2 yaitu menginisialisasikan variabel isi. Setelah itu, baris ke 3-6 melakukan perulangan untuk panjang variabel *fitness* dengan mengambil nilai dari variabel isi, inisialisasi variabel hasilsort untuk mengurutkan nilai *fitness*, dan inisialisasi variabel isiKembali. Pada baris ke 7-8 digunakan perulangan untuk popsize dengan mengambil nilai dari variabel isiKembali. Baris ke 9 digunakan untuk mengembalikan nilai ke variabel isiKembali.

5.6 Implementasi Algoritme Migrasi

Algoritme migrasi digunakan untuk menambah variasi individu dengan menukar individu dengan nilai *fitness* terbaik dari subpopulasi satu dengan inidividu dari subpopulasi lain dengan nilai *fitness* terendah. Kode program untuk implementasi algoritme migrasi terdapat pada Gambar 5.16.

```
Algoritme Migrasi
1       def migrasi(self,subpopulasi):
2           subpop1 = random.randint(0,len(subpopulasi)-1)
3           subpop2 = self.getRandom(subpop1,len(subpopulasi)-1)
4           temp = subpopulasi[subpop1][0]
5           subpopulasi[subpop1][0] =
    subpopulasi[subpop2][len(subpopulasi[subpop2])-1]
6           subpopulasi[subpop2][0] = temp
7           return subpopulasi
```

Gambar 5.16 *Source Code* Migrasi

Pada kode program migrasi diatas, baris ke 2-3 digunakan untuk inisialisasi variabel subpop1 dan subpop2 dengan memilih subpopulasi secara acak. Baris 4-6 digunakan untuk mengambil nilai dari subpop1 dan subpop2, setelah itu akan dipindahkan nilainya dari masing-masing subpopulasi. Baris ke 7 akan mengambalikan nilai ke variabel subpopulasi.

5.6.1 Implementasi Algoritme Representasi Kromosom Baru

Setelah melakukan migrasi pada sub-populasi maka selanjutnya akan dilakukan proses representasi kromosom baru, dimana semua subpopulasi akan digabungkan menjadi satu populasi yang nantinya akan di proses melalui proses algoritme genetika terdistribusi kembali. Setelah kromosom baru di representasikan, maka semua populasi akan melakukan perhitungan nilai *fitness* kembali untuk mendapatkan hasil yang optimal. Kode program untuk algoritme representasi kromosom baru terdapat pada Gambar 5.17.

Algoritme Representasi Kromosom Baru	
1	`def jadikanPopulasi(self,subpop):`
2	`index = subpop.shape`
3	`populasi =[]`
4	`for i in range(index[1]):`
5	`isi= np.concatenate((subpop[0][i],subpop[1][i],subpop[2][i], subpop[3][i],subpop[4][i]),axis=0)`
6	`populasi.append(isi)`
7	`return populasi`

Gambar 5.17 Representasi Kromosom Baru

Kode program algoritme diatas pada baris ke 2-3 dilakukan inisialisasi variabel index dan populasi. Bari ke 4-5 merupakan perulangan untuk variabel index pada index ke 1 dengan inisialisasi variabel isi untuk mengambil nilai dari masing-masing subpopulasi. Pada baris ke 6-7 digunakan untuk mengambil nilai dari variabel isi dan mengembalikan nilai ke variabel populasi.

BAB 6 PENGUJIAN

Pada Bab ini akan dibahas mengenai hasil pengujian pada optimasi penjadwalan sidang skripsi menggunakan algoritme genetika terdistribusi. Sebelumnya pada Bab 4 telah dilakukan perancangan pengujian, setelah di implementasikan pada Bab 5, maka pada Bab 6 ini akan dilihat parameter algortime genetika terdistribusi yang mana yang paling optimal untuk penjadwalan sidang skripsi.

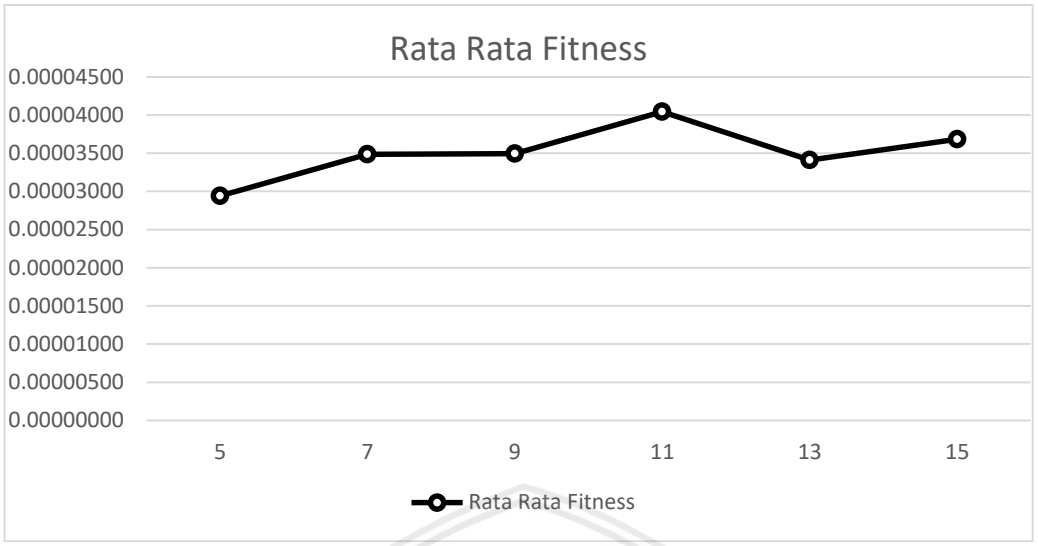
6.1 Pengujian Jumlah Populasi

Pada pengujian jumlah populasi yang sebelumnya telah dirancang pada Bab 4, akan digunakan jumlah ukuran sub-populasi sebanyak 5 sub-populasi, 50 generasi, 0,5 untuk cr, dan 0,5 untuk mr. Pengujian ini hanya dilakukan sebanyak 50 generasi karena akan membuat waktu komputasi yang lebih efektif. Pengujian ini akan dilakukan dengan menggunakan jumlah ukuran populasi rentang 5 sampai 15 dengan kenaikan 2 perpopulasi. Pengujian ini akan dilakukan sebanyak 5 kali untuk melihat jumlah ukuran populasi mana yang optimal untuk mengatasi masalah penjadwalan sidang skripsi. Tabel hasil pengujian jumlah populasi dapat dilihat pada Tabel 6.1.

Tabel 6.1 Pengujian Jumlah Populasi

Popsize	Nilai *fitness*					Rata-rata *fitness*
	Percobaan Populasi Ke-					
	1	2	3	4	5	
5	0,00001853	0,00002840	0,00002836	0,00002549	0,00002785	0,00002941
7	0,00001943	0,00002876	0,00002395	0,00003048	0,00002953	0,00003486
9	0,00003386	0,00002879	0,00002905	0,00002752	0,00002920	0,00003494
11	0,00003296	0,00003320	0,00003183	0,00003388	0,00002921	0,00004045
13	0,00002853	0,00002839	0,00003445	0,00002708	0,00002454	0,00003413
15	0,00002838	0,00003660	0,00003223	0,00003490	0,00002720	0,00003685

Tabel 6.1 menunjukkan hasil dari pengujian jumlah populasi yang selanjutnya akan dibuat sebuah grafik yang menampilkan hasil pengujiannya. Dari grafik tersebut dapat terlihat dengan mudah pada populasi keberapa algoritme genetika terdistribusi memiliki hasil yang maksimal untuk mendapatkan hasil yang optimal. Grafik hasil pengujian jumlah populasi terdapat pada Gambar 6.1.



Gambar 6.1 Grafik Hasil Pengujian Jumlah Populasi

Pada Gambar 6.1 dapat dilihat pada jumlah populasi 5 rata-rata *fitness* yang dihasilkan cukup besar yaitu 0,00002941, kemudian pada populasi 7 mengalami kenikan menjadi 0,00003486, selanjutnya pada jumlah ukuran populasi 9 dengan rata-rata *fitness* sebesar 0,00003494. Pada populasi sejumlah 11 rata-rata *fitness* kembali naik dengan nilai 0,00004045, akan tetapi mengalami penurunan hingga nilai *fitness* rata-rata sebesar 0,00003413. Pada populasi ke 15 rata-rata *fitness* mengalami kenaikan kembali dengan nilai 0,00003685. Pada pengujian populasi, jika semakin besar jumlah populasi maka semakin lama waktu komputasinya. Dari pengujian jumlah populasi yang telah dilakukan, dapat diketahui bahwa jumlah populasi 11 memiliki nilai rata-rata *fitness* terbesar dengan nilai 0,00004045 dan jumlah populasi 5 menunjukkan nilai rata-rata *fitness* paling kecil dengan nilai 0,00002941. Maka, 11 populasi merupakan jumlah populasi optimal untuk masalah optimasi penjadwalan sidang skripsi, populasi optimal ini selanjutnya akan digunakan sebagai parameter berikutnya untuk pengujian generasi dan crmr.

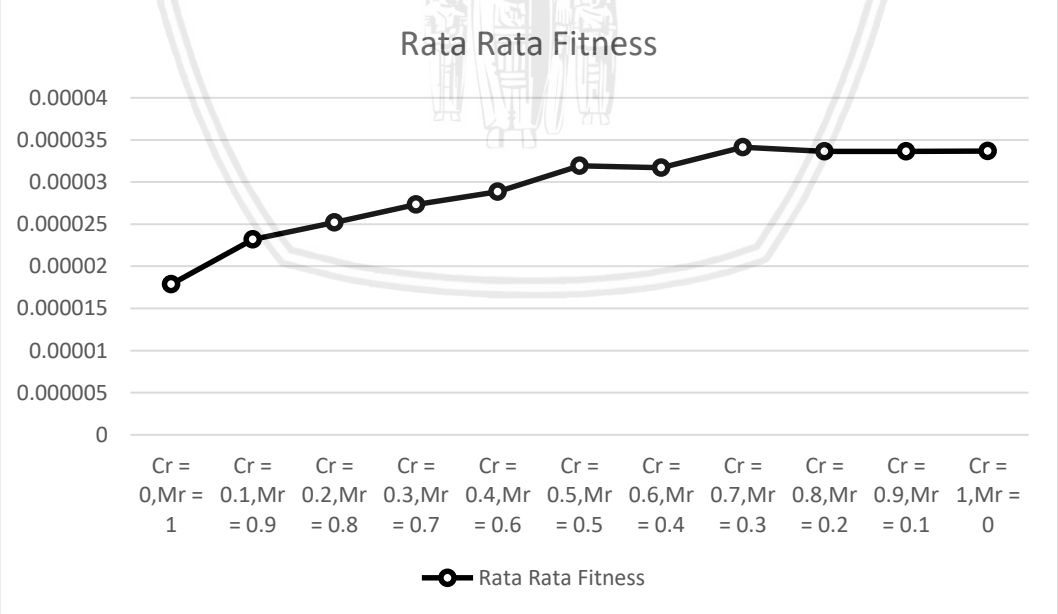
6.2 Pengujian Kombinasi *Crossover Rate* dan *Mutation Rate*

Pada bab 4 telah dirancang pengujian kombinasi cr dan mr, pada bab ini akan dilakukan pengujian dengan menggunakan kombinasi rentang cr 0 sampai 1 dan rentang mr 1 sampai 0. Pada pengujian ini akan digunakan jumlah ukuran populasi yang optimal berdasarkan pengujian sebelumnya yaitu sebanyak 11 populasi, untuk subpopulasi menggunakan jumlah hari pada penjadwalan sidang menjadi 5 subpopulasi, dan menggunakan generasi yang optimal pada pengujian sebelumnya yaitu 50 generasi. Pengujian untuk setiap kombinasi cr dan mr akan diuji sebanyak 5 kali untuk melihat hasil kombinasi yang benar-benar optimal. Hasil dari pengujian kombinasi cr dan mr ada pada Tabel 6.2.

Tabel 6.2 Pengujian Cr dan Mr

Cr		Nilai *fitness*					Rata-rata *fitness*
		Nilai *Fitness* Percobaan kombinasi cr dan mr ke-					
Cr	*Mr*	1	2	3	4	5	
0	1	0,00001875	0,00001744	0,00001845	0,00001726	0,00001741	0,00001786
0,1	0,9	0,00002270	0,00002285	0,00002269	0,00002205	0,00002561	0,00002318
0,2	0,8	0,00002528	0,00002405	0,00002593	0,00002504	0,00002562	0,00002519
0,3	0,7	0,00002603	0,00002829	0,00002828	0,00002738	0,00002663	0,00002732
0,4	0,6	0,00002837	0,00002846	0,00002993	0,00002792	0,00002956	0,00002885
0,5	0,5	0,00002988	0,00003056	0,00003035	0,00003506	0,00003385	0,00003194
0,6	0,4	0,00003255	0,00002905	0,00003276	0,00003093	0,00003321	0,00003170
0,7	0,3	0,00003729	0,00003591	0,00003204	0,00003387	0,00003150	0,00003412
0,8	0,2	0,00003285	0,00003662	0,00003073	0,00003264	0,00003542	0,00003365
0,9	0,1	0,00003963	0,00003423	0,00002281	0,00003622	0,00003533	0,00003365
1	0	0,00002620	0,00003406	0,00003455	0,00003844	0,00003515	0,00003368

Tabel 6.2 telah menunjukkan hasil dari pengujian kombinasi nilai cr dan mr. Selain ditunjukkan oleh tabel akan dibuat sebuah grafik yang menampilkan hasil pengujian kombinasi cr dan mr. Pada grafik berikut akan terlihat kombinasi cr dan mr dengan berapa yang dapat memberikan hasil yang optimal untuk masalah penjadwalan sidang skripsi. Grafik hasil pengujian untuk kombinasi cr dan mr terdapat pada Gambar 6.2.



Gambar 6.2 Grafik Hasil Pengujian Kombinasi Cr dan Mr

Dari Gambar 6.2 dapat diketahui nilai rata-rata *fitness* dari setiap kombinasi cr dan mr, dengan rentang 1 sampai 0 untuk cr dan rentang 0 sampai 1 untuk mr. Pada kombinasi cr 0 dan mr 1 rata-rata nilai *fitness* sebesar 0,00001786,

setelah itu pada kombinasi cr 0,1 dan mr 0,9 mengalami peningkatan menjadi 0,00002318. Setelah itu rata-rata nilai *fitness* terus mengalami peningkatan, seperti pada kombinasi cr 0,2 dan 0,8 sebesar 0,00002519, kombinasi cr 0,3 dan mr 0,7 sebesar 0,00002732. Lalu masih pengalami peningkatan saat kombinasi cr 0,4 dan mr 0,6 menjadi 0,00002885. Peningkatan yang pesat terjadi pada kombinasi cr 0,5 dan mr 0,5 dengan nilai 0,00003194, dan mengalami sedikit penurunan pada kombinasi cr 0,6 dan mr 0,4 dengan nilai 0,00003170. Setelah itu rata-rata nilai *fitness* telah mengalami konvergen atau perubahan yang tidak memiliki arti seperti pada kombinasi cr 0,7 dan mr 0,3 sebesar 0,00003412, kombinasi cr 0,8 dan mr 0,2 sebesar 0,00003365, kombinasi cr 0,9 dan mr 0,1 sebesar 0,00003365, dan kombinasi cr 1 dan mr 0 dengan nilai 0,00003368. Pada grafik diatas dapat terlihat bahwa pada kombinasi nilai cr 0,7 dan mr 0,3 merupakan kombinasi dengan rata-rata nilai *fitness* terbesar yaitu 0,00003412. Sehingga kombinasi nilai cr 0,7 dan mr 0,3 menjadi nilai yang optimal untuk masalah penjadwalan sidang skripsi.

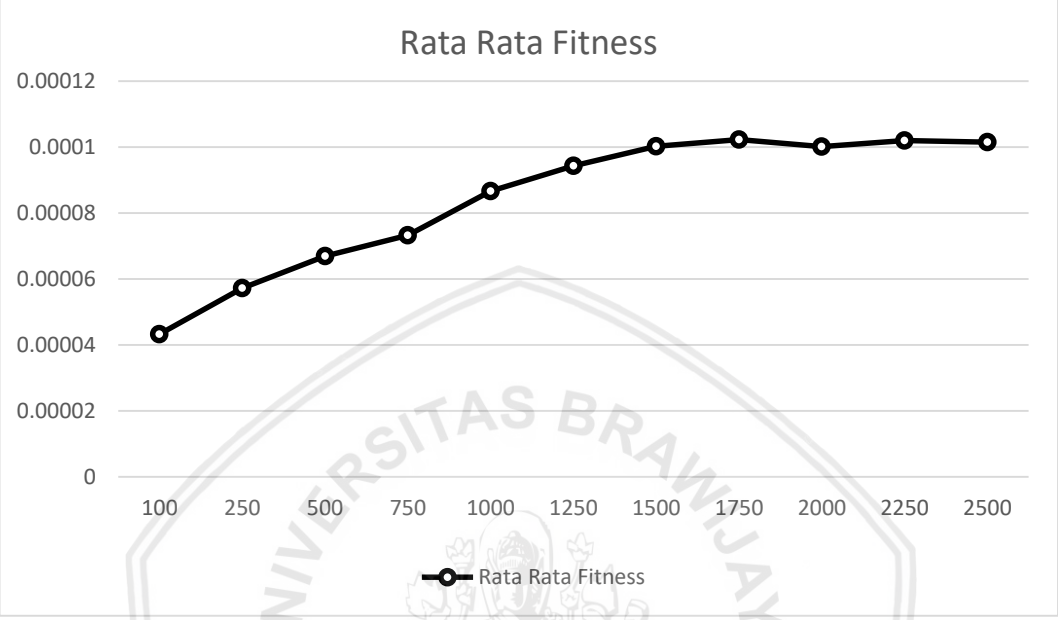
6.3 Pengujian Jumlah Generasi

Pada pengujian ini akan digunakan jumlah ukuran populasi yang optimal pada pengujian sebelumnya yaitu 11 populasi, jumlah ukuran sub-populasi sebanyak 5, nilai cr 0,7, dan nilai mr 0,3. Jumlah generasi akan ditentukan pada rentang 100 sampai 2500. Jumlah generasi ditentukan dengan rentang tersebut karena pada rentang tersebut dapat terlihat perbedaan rata-rata nilai *fitness* yang signifikan. Pengujian ini akan dilakukan sebanyak 5 kali untuk melihat rata-rata nilai *fitness* terbaik dari jumlah generasi. Tabel pengujian jumlah generasi dapat dilihat pada Tabel 6.3.

Tabel 6.3 Pengujian Jumlah Generasi

gen	Nilai *fitness*					Rata-rata *Fitness*
	Percobaan Populasi Ke-					
	1	2	3	4	5	
100	0,00004219	0,00004182	0,00004255	0,00004608	0,00004385	0,00004330
250	0,00006060	0,00005492	0,00004974	0,00005617	0,00006493	0,00005727
500	0,00007352	0,00006097	0,00006022	0,00006022	0,00007999	0,00006698
750	0,00007873	0,00006579	0,00006659	0,00006620	0,00008928	0,00007332
1000	0,00008546	0,00009090	0,00007511	0,00007997	0,00010203	0,00008670
1250	0,00009008	0,00008770	0,00009803	0,00008471	0,00011110	0,00009432
1500	0,00009258	0,00010525	0,00009338	0,00009801	0,00011235	0,00010031
1750	0,00009433	0,00009801	0,00009607	0,00010413	0,00011903	0,00010232
2000	0,00009523	0,00008332	0,00009795	0,00010407	0,00012047	0,00010021
2250	0,00008127	0,00010988	0,00012655	0,00010524	0,00008690	0,00010197
2500	0,00008472	0,00011493	0,00011362	0,00010750	0,00008690	0,00010154

Pada tabel 6.3 telah menunjukkan hasil dari pengujian jumlah generasi yang selanjutnya akan dibuat sebuah grafik yang menampilkan hasil pengujiannya. Dengan grafik berikut akan terlihat pada generasi keberapa algoritme genetika terdistribusi memberikan hasil yang optimal untuk masalah penjadwalan sidang skripsi. Grafik hasil pengujian jumlah generasi terdapat pada Gambar 6.3.



Gambar 6.3 Grafik Hasil Pengujian Jumlah Generasi

Pada Gambar 6.3 diketahui pada generasi ke-100 rata-rata nilai *fitness* yang dihasilkan yaitu 0,00004330, lalu mengalami kenaikan terus menerus seperti pada generasi ke-250 dengan rata-rata nilai *fitness* sebesar 0,00005727, pada generasi ke-500 dengan rata-rata nilai *fitness* sebesar 0,00006698, pada generasi ke-750 dengan rata-rata nilai *fitness* sebesar 0,00007332, generasi ke-1000 dengan rata-rata nilai *fitness* sebesar 0,00008670, generasi ke-1250 dengan rata-rata nilai *fitness* sebesar 0,00009432, pada generasi ke-1500 dengan rata-rata nilai *fitness* sebesar 0,00010031, pada generasi ke-1750 dengan rata-rata nilai *fitness* sebesar 0,00010232. Akan tetapi mengalami sedikit penurunan pada generasi ke-2000 dengan rata-rata nilai *fitness* 0,00010021, dan mengalami sedikit kenaikan pada generasi ke-2250 dengan rata-rata nilai *fitness* 0,00010197. Setelah itu mengalami sedikit penurunan lagi pada generasi ke-2500 dengan rata-rata nilai *fitness* sebesar 0,00010154. Dari pengujian jumlah generasi yang telah dilakukan, dapat diketahui bahwa pada generasi ke-1750 merupakan generasi dengan nilai rata-rata *fitness* terbesar dengan nilai 0,00010232, sehingga generasi ke-1750 merupakan generasi yang optimal untuk masalah optimasi penjadwalan sidang skripsi.

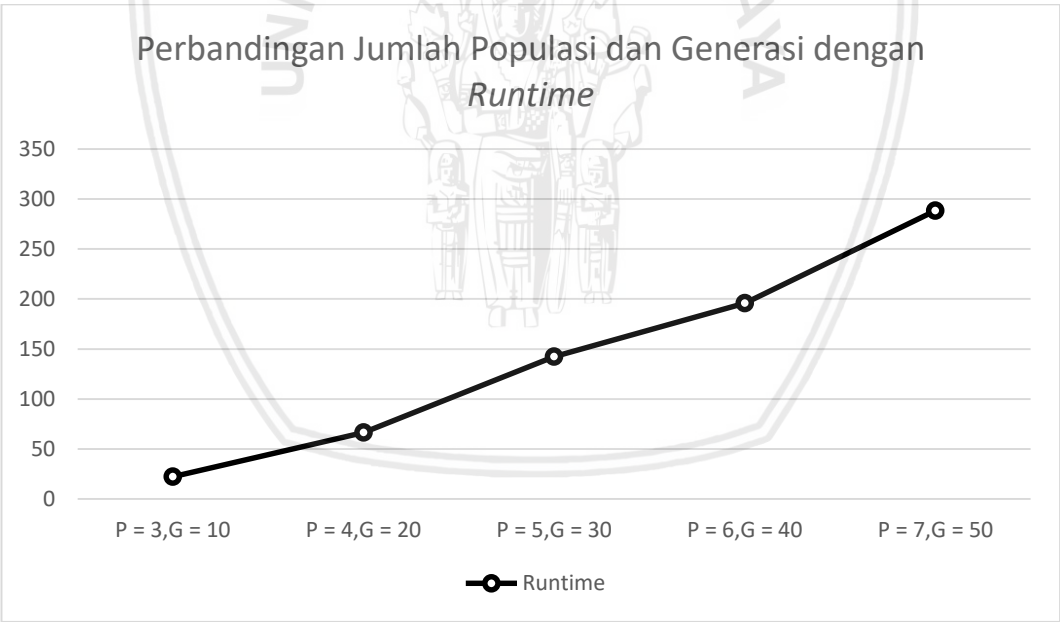
6.4 Analisis Hasil Pengujian

Berdasarkan pada pengujian parameter algoritme genetika terdistribusi yang sudah dilakukan yaitu pengujian jumlah populasi, pengujian jumlah generasi, dan pengujian kombinasi *crossover rate* dan mutation *rate*, telah didapatkan hasil

optimal dengan melihat dari besarnya rata-rata nilai *fitness* pada setiap pengujian parameter.

Pada pengujian jumlah populasi didapatkan hasil optimal yaitu 11 populasi, kemudian 11 populasi digunakan sebagai parameter populasi untuk pengujian kombinasi *crossover rate* (cr) dan *mutation rate* (mr) sehingga didapatkan kombinasi *crossover rate* (cr) dan *mutation rate* (mr) yang optimal adalah 0,7 untuk *crossover rate* (cr) dan 0,3 untuk *mutation rate* (mr). Setelah itu, hasil yang optimal dari pengujian tersebut akan digunakan untuk menguji jumlah generasi. Pada pengujian jumlah generasi telah didapatkan hasil yang optimal yaitu 1750 generasi.

Dari pengujian tersebut, dapat diketahui bahwa parameter algoritme genetika terdistribusi sangat mempengaruhi hasil untuk menentukan jadwal sidang skripsi yang optimal. Semua parameter algoritme genetika terdistribusi memiliki peran penting dalam memberikan hasil dengan menjaga keragaman variasi individu dan kemampuan eksplorasi daerah solusi untuk memberikan hasil yang optimal. Akan tetapi pada pengujian ini, jika semakin banyak jumlah populasi dan generasi, maka semakin luas daerah pencarian solusi, akan tetapi waktu komputasi menjadi semakin lama. Berikut diberikan grafik untuk perbandingan jumlah populasi dan generasi dengan waktu komputasi (*runtime*) dapat dilihat pada Gambar 6.4.



Gambar 6.4 Grafik Perbandingan Jumlah Populasi dan Generasi dengan *Runtime*

BAB 7 KESIMPULAN DAN SARAN

Bab ini menjelaskan bagaimana kesimpulan dari penelitian yang sudah dilakukan untuk menentukan jadwal yang optimal pada penjadwalan sidang skripsi, serta saran untuk penelitian berikutnya.

7.1 Kesimpulan

Berdasarkan pengujian yang dilakukan, penulis dapat menyimpulkan bagaimana pengaruh struktur dan struktur operator pada parameter algoritme genetika terdistribusi dalam menangani masalah penjadwalan sidang skripsi yang optimal adalah sebagai berikut:

1. Struktur algoritme genetika terdistribusi untuk masalah penjadwalan sidang skripsi telah diterapkan pada penelitian ini dengan merepresentasikan data mahasiswa dan dosen menjadi bilangan *real* agar lebih mudah dalam merepresentasikan kromosom pada proses inisialisasi. Selanjutnya populasi akan dipecah menjadi beberapa subpopulasi, dan akan melalui tahap reproduksi untuk menghasilkan keturunan. Setelah itu dilakukan evaluasi untuk menghitung nilai *fitness*. Jika nilai *fitness* semakin besar maka semakin baik untuk dijadikan solusi. Berikutnya dilakukan seleksi untuk memilih individu yang akan dipertahankan untuk generasi selanjutnya. Tahap terakhir adalah migrasi untuk menambah keberagaman individu. Setelah semua proses algoritme genetika terdistribusi selesai maka akan merepresentasikan kromosom baru dengan menggabungkan kembali subpopulasi menjadi satu populasi.
2. Struktur operator algoritme genetika terdistribusi pada penelitian ini menggunakan metode *reciprocal exchange mutation* dan *random mutation* pada proses mutasi, metode *one-cut point* pada proses *crossover* dan metode *elitism selection* pada proses seleksi. Melakukan proses migrasi setiap 10 generasi dengan menukar nilai *fitness* tertinggi dari satu sub-populasi dengan nilai *fitness* terendah dari sub-populasi lain.
3. Nilai parameter terbesar dari hasil pengujian untuk jumlah populasi sebesar 11, *crossover rate* 0,7 mutation *rate* 0,3, jumlah sub populasi 5, dan jumlah generasi sebanyak 1750 dengan rata-rata nilai *fitness* sebesar 0.00010232.

7.2 Saran

Pada penelitian selanjutnya dapat dikembangkan dengan menggunakan struktur algoritme genetika terdistribusi yang berbeda dan menggunakan struktur operator-operator algoritme genetika terdistribusi yang berbeda. Seperti menggunakan atau menambahkan metode yang berbeda untuk proses *crossover*, mutasi, seleksi, dan migrasi. Hal tersebut digunakan agar penelitian selanjutnya mendapatkan hasil yang optimal.

DAFTAR REFERENSI

Damayanti, L. and Cholissodin, I., 2018. Optimasi Penjadwalan Bimbingan Skripsi Menggunakan Algoritme Genetika (Studi Kasus : Fakultas Ilmu Komputer Universitas Brawijaya). 2(9), pp.3370–3375.

Defersha, F.M. and Chen, M., 2010. A parallel genetic algorithm for a flexible job-shop scheduling problem with sequence dependent setups. *International Journal of Advanced Manufacturing Technology*, 49(1–4), pp.263–279.

Đumić, M., Šišejković, D., Čorić, R. and Jakobović, D., 2018. Evolving priority rules for resource constrained project scheduling problem with genetic programming. *Future Generation Computer Systems*, 86, pp.211–221.

FILKOM, 2018. *Jumlah Mahasiswa Pertahun*. [online] Available at: <http://ti.filkom.ub.ac.id/page/read/number -of-students-per-year/04c7f1bd0d22e9/> [Accessed 5 Mar. 2019].

FILKOM UB, 2018. *Data Dosen*. [online] Available at: <http://filkom.ub.ac.id/page/read/lecturer/461acea> [Accessed 5 Mar. 2019].

Haupt, R.L. and Haupt, S.E., 2004. *PRACTICAL GENETIC ALGORITHMS*.

Indra, Z. and Subanar, 2014. Optimasi Biaya Distribusi Rantai Pasok Tiga Tingkat dengan Menggunakan Algoritma Genetika Adaptif dan Terdistribusi. 8(2).

Jain, a, Jain, D. and Chande, D., 2010. Formulation of Genetic Algorithm to Generate Good Quality Course Timetable. *International Journal of Innovation, …*, [online] 1(3), pp.248–251. Available at: <http://ijimt.org/papers/46-M431.pdf>.

Khoirul L.M.A., O., Widodo, A.W. and Setiawan, B.D., 2017. Optimasi Penjadwalan Mata Pelajaran Menggunakan Metode Tabu Search (Studi Kasus: SMKN 2 Singosari). *International Clinical Psychopharmacology*, 22(6), pp.338–347.

Mahmudy, W.F., 2009. Optimasi Fungsi Tak Berkendala Menggunakan Algoritma Genetika Terdistribusi dengan Pengkodean Real. *Seminar Nasional Basic Science FMIPA, Universitas Brawijaya*, IV(August), pp.1–5.

Mahmudy, W.F., 2015. *Dasar-Dasar Algoritma Evolusi*. *Universitas Brawijaya*.

Michalewicz, Z., 1996. *Algorithms + Data Structures = Evolution Programs (3ed)*. *Artificial Intelligence in Medicine*.

Raissa, T., Widodo, A.W. and Setiawan, B.D., 2017. Penentuan Portofolio Saham Optimal Menggunakan Algoritma Genetika Terdistribusi. 1(1), pp.63–68.

Sharma, A. and Mehta, A., 2013. International Journal of Advanced Research in Computer Science and Software Engineering Review Paper of Various Selection Methods in Genetic Algorithm Types of Selection Method. 3(7), pp.1476–1479.

Suryaningrum, D.A., Ratnawati, D.E. and Setiawan, B.D., 2017. Prediksi Waktu Panen Tebu Menggunakan Gabungan Metode Backpropagation dan Algoritma Genetika. 1(11), pp.1443–1450.

Ula, Y.N., Ratnawati, D.E. and Wicaksono, S.A., 2018. Penjadwalan Dinas Pegawai Menggunakan Algoritma Genetika pada PT . Kereta Api Indonesia (KAI) DAOP 7 Stasiun Besar Kediri. *Pengembangan Teknologi Informasi dan Ilmu Komputer*, 2(8), pp.2473–2479.

Vladimir I, L., J A, B., Gagarin, S., A, T.A. and World, T., 2002. The Application of the Distributed Genetic Algorithm to the Decision of the Packing in Containers Problem. pp.0–4.

Yang, S. and Jat, S.N., 2011. Genetic algorithms with guided and local search strategies for university course timetabling. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 41(1), pp.93–106.

Yussof, S. and Razali, R.A., 2012. An Investigation on the Effect of Migration Strategy on Parallel GA-Based Shortest Path Routing Algorithm. 2012(May), pp.93–100.

Yussof, S., Razali, R.A., See, O.H., Ghapar, A.A. and Md Din, M., 2009. A coarse-grained parallel genetic algorithm with migration for shortest path routing problem. *2009 11th IEEE International Conference on High Performance Computing and Communications, HPCC 2009*, (May 2014), pp.615–621.