TUGAS AKHIR – IS184853

OPTIMASI PENJADWALAN STATIK ADMISI PASIEN MENGGUNAKAN METODE VARIABLE NEIGHBORHOOD SEARCH

OPTIMIZATION OF STATIC PATIENT ADMISSION SCHEDULING USING VARIABLE NEIGHBORHOOD SEARCH

VARIAN ELBERT
NRP 0521 16 4000 0063

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

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

TUGAS AKHIR – IS184853

OPTIMASI PENJADWALAN STATIK ADMISI PASIEN MENGGUNAKAN METODE VARIABLE NEIGHBORHOOD SEARCH

VARIAN ELBERT
NRP 0521 16 4000 0063

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

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

"Halaman ini sengaja dikosongkan"

FINAL PROJECT – IS184853

OPTIMIZATION OF STATIC PATIENT ADMISSION SCHEDULING USING VARIABLE NEIGHBORHOOD SEARCH

VARIAN ELBERT
NRP 0521 16 4000 0063

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

DEPARTEMEN SISTEM INFORMASI
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

"Halaman ini sengaja dikosongkan"

LEMBAR PENGESAHAN

OPTIMASI PENJADWALAN STATIK ADMISI PASIEN MENGGUNAKAN METODE VARIABLE NEIGHBORHOOD SEARCH

TUGAS AKHIR

Disusun Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer (S.Kom)
pada
Departemen Sistem Informasi
Fakultas Teknologi Elektro dan Informatika Cerdas
(ELECTICS)
Institut Teknologi Sepuluh Nopember

Oleh:

VARIAN ELBERT
NRP 05211640000063

Surabaya, 14 Agustus 2020

KEPALA DEPARTEMEN SISTEM INFORMASI

Dr. Mudjahidin ST., MT.
NIP 197010102003121001

“Halaman ini sengaja dikosongkan”

LEMBAR PERSETUJUAN

OPTIMASI PENJADWALAN STATIK ADMISI PASIEN MENGGUNAKAN METODE VARIABLE NEIGHBORHOOD SEARCH

TUGAS AKHIR

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

Oleh:

VARIAN ELBERT
NRP 0521 16 4000 0063

Disetujui Tim Penguji: Tanggal Ujian: 3 Juli 2020
Periode Wisuda: September 2020

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

Edwin Riksakomara, S.Kom., MT. **(Penguji I)**

Faizal Mahananto, S.Kom., M.Eng., Ph.D. **(Penguji II)**

“Halaman ini sengaja dikosongkan”

OPTIMASI PENJADWALAN STATIK ADMISI PASIEN MENGGUNAKAN METODE VARIABLE NEIGHBORHOOD SEARCH

Nama Mahasiswa : Varian Elbert
NRP : 0521 16 4000 0063
Departemen : Sistem Informasi FTEIC-ITS
Pembimbing : Ahmad Muklason, S.Kom., M.Sc., Ph.D.

ABSTRAK

Perencanaan dan manajemen sumber daya merupakan aspek penting dalam keberlangsungan operasional perusahaan. Dengan manajemen yang baik, perusahaan dapat mencapai target sekaligus menekan biaya operasional yang dikeluarkan. Hal serupa juga dilakukan oleh rumah sakit. Berbagai tantangan terkait sumber daya yang dialami rumah sakit diantara lain penjadwalan perawat, penjadwalan operasi pasien, dan penjadwalan appointment pasien.

Permasalahan yang diangkat dalam hal ini adalah penjadwalan admisi pasien. Penjadwalan admisi pasien atau yang dikenal sebagai Patient Admission Scheduling Problem (PASP) adalah masalah penjadwalan pasien dengan mempertimbangkan preferensi serta kebutuhan pasien, ketersediaan kasur, serta efisiensi dan utilisasi sumber daya. Permasalahan ini sangatlah relevan terutama untuk rumah sakit berukuran besar. Jumlah ruangan, spesialis, serta fasilitas yang ada membuat penjadwalan secara manua sangat sulit. Hal ini ditambah dengan preferensi, kebutuhan serta durasi tinggal pasien yang sangat beragam.

Penjadwalan admisi pasien bisa saja dilakukan hanya sekedar sebagai fungsi administratif belaka. Artinya, admisi pasien dilakukan langsung ketika pasien mendaftar, dan hanya memperhatikan occupancy rate dari rumah sakit; apakah ada kasur dan ruangan yang tersedia atau tidak. Akan tetapi, hal ini akan berdampak pada efisiensi operasional rumah sakit,

efisiensi alokasi sumber daya, serta kepuasan pasien. Sebagai contoh, pasien yang tidak memerlukan ruangan sendiri tetapi dialokasikan ke single room. Alhasil pasien mungkin harus dipindahkan ke ruangan lain ketika terdapat pasien baru yang perlu dikarantina.

Berbagai metode baik heuristik maupun exact telah dikemukan. Salah satunya yakni penggunaan metode integer programming. Untuk search space yang luas, metode ini membutuhkan waktu komputasi yang sangat lama. Untuk itu, metode Variable Neighborhood Search (VNS) digunakan, yang notabene merupakan algoritma metaheuristik.

Dengan metode VNS, jadwal admisi pasien yang efisen dapat diperoleh dengan waktu komputasi yang relatif lebih cepat bila dibandingkan dengan metode integer programming. Optimasi dapat dihentikan kapan saja dan tetap memberikan solusi yang feasible. Sebagai pembanding, algoritma optimasi dengan pemilihan neighborhood secara acak juga digunakan. Dengan menggunakan waktu eksekusi 30 detik, metode pemilihan acak di semua percobaan menghasilkan rata-rata penalti 1440,7, sedangkan metode VNS di semua percobaan menghasilkan rata-rata penalti 1398,2 (-2,95%).

Kata Kunci: optimasi, algoritma variable neighborhood search, patient admission scheduling

OPTIMIZATION OF STATIC PATIENT ADMISSION SCHEDULING USING VARIABLE NEIGHBORHOOD SEARCH

Nama Mahasiswa	**: Varian Elbert**
NRP	**: 0521 16 4000 0063**
Departemen	**: Sistem Informasi FTEIC-ITS**
Pembimbing	**: Ahmad Muklason, S.Kom., M.Sc., Ph.D.**

ABSTRACT

Resource planning and management is crucial in maintaining company's operation. A good management practice helps the company to reach their target while at the same time, maintaining their operational cost at minimum. The same concept also applies to a hospital. A couple challenges that hospital faces includes nurse rostering, surgery scheduling, and patient appointment scheduling.

The problem that the writer want to discuss is patient admission scheduling problem, abbreviated as PASP. PASP concerns scheduling a patient while taking into account their needs, preferences, the hospital's bed availability, and also the said hospital's resource allocation efficiency and utilization level.This can become even more problematic if the hospital is huge.The amount of room, supported specialism, along with the facility makes it difficult or even impractical for the scheduling to be done manually. Not to mention the patient's differing needs, preferences, and admission duration.

In actuality, scheduling patient admission can be done as a mere administrative function. In this case, admission is done directly as the patient registers and it considers only the current occupancy rate. In other words, it only checks whether there is a room available for the admission or not. This may affect the operational efficiency, resource efficiency, and patient's satisfaction level. For instance, a single room assigned to a patient that does not necessarily need or perhaps prefer a

single room. If say, there are another patient that requires a quarantine, then the other patient might need to be moved to another room.
Multiple varying exact and heuristic-based method was tested and used in the past. One of it is integer programming. For a problem with a very large search space, this takes a lot of time to compute. For this reason, the writer chooses to use Variable Neighborhood Search, which is a metaheuristic algorithm With VNS, an effiicient solution can be formed in a relatively quicker computation time compared to integer programming. The writer hopes that an altenative optimal solution can be found with a reasonable amount of time. An optimization method using random neighborhood selection is also considered. With execution time of 30 seconds, random selection yields in all trial yields average penalty of 1440.7, while VNS in all trial yields average penalty of 1398.2 (-2.95%).

Keywords: optimization, variable neighborhood search algorithm, patient admission scheduling

SURAT PERNYATAAN BEBAS PLAGIARISME

Saya yang bertandatangan di bawah ini:

Nama	: Varian Elbert
NRP	: 05211640000063
Tempat/Tanggal lahir	: Jakarta, 20 November 1998
Fakultas/Departemen	: FTEIC / Departemen Sistem Informasi
Nomor Telp/Hp/email	: varianelbert.f@gmail.com

Dengan ini menyatakan dengan sesungguhnya bahwa tugas akhir saya yang berjudul

OPTIMASI PENJADWALAN STATIK ADMISI PASIEN MENGGUNAKAN METODE VARIABLE NEIGHBORHOOD SEARCH

Bebas Dari Plagiarisme Dan Bukan Hasil Karya Orang Lain.

Apabila dikemudian hari ditemukan seluruh atau sebagian penelitian/makalah/tugas akhir tersebut terdapat indikasi plagiarisme, maka saya bersedia menerima sanksi sesuai peraturan dan ketentuan yang berlaku.

Demikian surat pernyataan ini saya buat dengan sesungguhnya dan untuk dipergunakan sebagaimana mestinya.

Jakarta, 4 Agustus 2020

VARIAN ELBERT
NRP. 05211640000063

"Halaman ini sengaja dikosongkan"

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena hanya dengan berkat dan rahmatnya penulis dapat menyelesaikan tugas akhir yang berjudul :

OPTIMASI PENJADWALAN STATIK ADMISI PASIEN MENGGUNAKAN METODE VARIABLE NEIGHBORHOOD SEARCH

Sebagai salah satu syarat kelulusan di Departemen Sistem Informasi, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember, Surabaya.

Penulis mengucapkan terimakasih kepada semua pihak yang telah membantu penulis menyelesaikan tugas akhir ini, baik secara langsung maupun tidak langsung. Beberapa diantaranya yakni:

1. Orang tua serta saudara penulis yang berada di Jakarta, yang terus memberikan dukungan dan motivasi bagi penulis untuk menyelesaikan tugas akhir serta pendidikan sarjana.
2. Keluarga besar orang tua penulis yang berada di Surabaya dan Sidoarjo, yang telah memberikan penulis tempat tinggal sementara semasa perkuliahan, serta melengkapi kebutuhan makan dan minum penulis.
3. Bapak Ahmad Muklason, S.Kom., M.Sc., Ph.D. selaku Kepala Lab RDIB sekaligus dosen pembimbing yang gigih membantu penulis dalam mengerjakan tugas akhir ini, terutama di masa pandemik ini. Beliau telah memberikan pencerahan ketika penulis mengalami permasalahan atau kebingungan di saat pengerjaan tugas akhir ini.
4. Bapak Edwin Riksakomara, S.Kom., M.T. dan Bapak Faizal Mahananto, S.Kom., M.Eng., Ph.D. selaku dosen penguji yang telah memberikan kritik dan saran untuk menyempurnkan tugas akhir ini.

5. Bapak Radityo Prasetianto Wibowo, S.Kom., M.Kom. selaku dosen wali, yang membantu memberikan pencerahan di awal masa perkuliahan, serta tidak mempersulit pilihan mata kuliah yang diambil penulis semasa perkuliahan.
6. Kepala Departemen Sistem Informasi, Kepala Program Studi S1, serta seluruh dosen pengajar beserta staf administrasi dan staf-staf lain yang telah mebantu penulis semasa perkuliahan, baik melalui pemberian informasi, bantuan administrasi, pengarahan, hingga menjaga kendaraan mahasiswa termasuk kendaraan penulis.
7. Teman-teman ARTEMIS yang menerima penulis apa adanya dan yang telah membantu penulis serta berjuang bersama penulis semasa perkuliahan.
8. Pihak-pihak lain yang juga membantu penulis tetapi belum penulis tuliskan di atas.

Penulis menyadari bahwa penulisan tugas akhir ini tidak sempurna. Untuk itu, penulis terbuka atas kritik dan saran yang dapat membantu penulis untuk dapat membuat tulisan atau artiel ilmiah yang lebih baik di masa mendatang. Penulis mengucapkan terimakasih kepada pembaca, dan semoga tugas akhir ini dapat menjadi manfaat bagi pembaca.

Jakarta, 4 Agustus 2020

Penulis

DAFTAR ISI

"Halaman ini sengaja dikosongkan"

DAFTAR TABEL

"Halaman ini sengaja dikosongkan"

DAFTAR GAMBAR

DAFTAR PERSAMAAN

"Halaman ini sengaja dikosongkan"

DAFTAR KODE

BAB I
PENDAHULUAN

Dalam bab ini dijelaskan mengenai identifikasi permasalahan penelitian meliputi latar belakang masalah, rumusan masalah, batasan masalah, tujuan tugas akhir, manfaat tugas akhir, dan relevansi terhadap pengerjaan tugas akhir. Lewat uraian ini, diharapkan gambaran umum permasalahan serta tujuan pembuatan tugas akhir ini dapat dimengerti.

1.1. Latar Belakang

Setiap perusahaan memiliki sumber daya yang terbatas. Sumber daya tersebut baik berupa tenaga kerja, material, mesin, waktu, uang, dan lainnya. Oleh karena itu, perusahaan perlu melakukan perencanaan dan manajemen sumber daya supaya tujuan dapat dicapai secara efisien.Walaupun demikian, perencanaan dan manajemen sumber daya terbukti menjadi masalah yang cukup kompleks. Banyaknya jumlah sumber daya dengan batasan yang beragam mempersulit perusahaan untuk memanajemen sumber daya yang dimilikinya secara efisien. Dalam survei pada tahun 2007 yang dilakukan oleh *Computing Technology Industry Association* (CompTIA), 18% dari ribuan manajer proyek menyalahkan buruknya perencanaan sumber daya sebagai alasan kedua kegagalan proyek.

Selain perusahaan, rumah sakit juga diharuskan merencanakan dan memanajemen sumber dayanya secara efisien untuk menekan pengeluaran. Hal ini menjadi sangat penting mengingat secara umum terdapat *gap* yang besar antara kapasitas rumah sakit dengan permintaan yang ada [1]. Kesenjangan ini menjadi lebih besar terutama di rumah sakit milik negara. Di Yunani misalnya, berdasarkan survei yang dilakukan di [2], pasien dengan menggunakan jaminan kesehatan nasional harus menunggu dari 2 hingga 6 bulan untuk melakukan operasi non darurat (*elective surgery*).

Salah satu perencanaan yang dilakukan oleh rumah sakit adalah penjadwalan perawat, atau secara formal dikenal sebagai *Nurse Rostering Problem* (NSP). Setiap perawat akan dijadwalan ke shift dan lokasi tertentu sesuai dengan keahlian dan preferensinya. NSP mudah diselesaikan ketika jumlah perawat dan lokasinya relatif kecil. Namun ketika jumlah perawatnya sangat banyak, dengan lokasi, keahlian, serta departemen yang beragam, masalah ini menjadi sulit diselesaikan.

Untuk rumah sakit berukuran relatif besar, masalah manajemen sumber daya ini diperparah dengan admisi pasien yang memiliki kebutuhan, spesifikasi, dan preferensi yang juga beragam. Di level strategis, perencanaan kapasitas rumah sakit menjadi sangat krusial, mengingat rumah sakit harus sebisa mungkin memenuhi kebutuhan admisi pasien sembari meminimalkan *delay* [1]. Permasalahan *delay* ini menjadi isu yang sering dialami negara-negara berkembang, bahkan beberapa negara maju sekalipun [2]. Di level operasional, masalah ini secara formal dikenal sebagai *Patient Admission Scheduling Problem* (PASP). Pada kenyataannya, masih banyak rumah sakit yang langsung melakukan admisi pasien tanpa menjadwalkannya dengan matang. Dalam survei yang dilakukan oleh Gammel & Van Dierdonck terhadap 83 rumah sakit di Belgia, hanya satu pertiga yang menjadwalkan admisi pasien [3]. Padahal, dengan melakukan penjadwalan admisi pasien, rumah sakit dapat memaksimalkan utilisasi sumber daya, mengestimasi *length-of-stay* (LOS), serta memenuhi kebutuhan dan bila memang memungkinkan, memenuhi preferensi pasien. Kenyataannya, banyak rumah sakit yang sekedar menggunakan model penjadwalan admisi *First Come, First Served*, dimana admisi dilakukan sesuai dengan urutan pendaftaran atau kedatangan pasien.

Metode *exact* seperti *Integer Programming* membutuhkan waktu yang sangat lama untuk menyelesaikan PASP dengan jumlah sumber daya dan pasien yang sangat banyak dan beragam. Untuk itu, digunakan metode heuristik untuk

menyelesaikan PASP. Metode heuristik dapat menyelesaikan masalah yang rumit dengan waktu komputasi yang relatif tidak lama, walaupun keoptimalannya tidak dapat sepenuhnya dijamin. Metode *Variable Neighborhood Search* (VNS) menjanjikan solusi yang relatif optimal dengan parameter yang relatif sedikit [4].

1.2. Perumusan Masalah

Rumusan masalah yang akan diselesaikan dalam tugas akhir ini adalah:

1. Perolehan alternatif solusi terhadap *Patient Admission Scheduling Problem* (PASP).
2. Desain algoritma *Variable Neighborhood Search* untuk memperoleh solusi PASP yang lebih baik

1.3. Batasan Tugas Akhir

Beberapa batas yang diberlakukan dalam Tugas Akhir ini adalah:

1. Dataset yang digunakan berasal dari Katholieke Universiteit Leuven (KU Leuven). Dataset ini merupakan pengembangan dari dataset yang digunakan oleh Demeester et al. dalam *A hybrid tabu search algorithm for automatically assigning patients to beds* [5]. Dataset merupakan dataset sintetik, yang dibuat sesuai dengan kondisi nyata lewat wawancara dengan pegawai admisi pasien berpengalaman.
2. *Patient Admission Scheduling Problem* yang akan diselesaikan menggunakan model matematis yang meninjau pada [6] dengan beberapa modifikasi. Dalam hal ini, pasien yang dipertimbangkan dalam PASP hanya memiliki satu kebutuhan spesialisme.
3. Bahasa pemrogaman yang digunakan adalah Java.

1.4. Tujuan Tugas Akhir

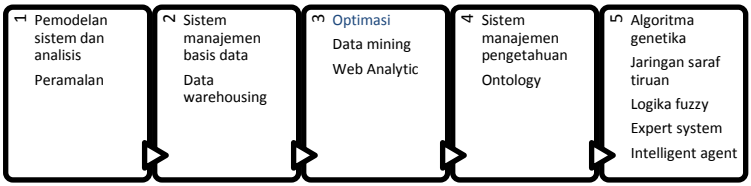
Tujuan dari tugas akhir ini adalah mengimplementasikan algoritma VNS untuk memperoleh solusi alternatif yang lebih baik untuk *Patient Admission Scheduling Problem*..

1.5. Manfaat Tugas Akhir

Manfaat yang diharapkan dari tugas akhir ini adalah perolehan solusi terhadap permasalahan *static PASP* dengan kualitas yang lebih baik. Selain itu, diharapkan solusi yang dihasilkan dapat menjadi basis perbandingan untuk penelitian-penelitian ke depan terkait PASP dengan dataset yang sama.

1.6. Relevansi

Topik tugas akhir ini berkaitan dengan bidang Optimasi Kombinatorik dan Heuristik di laboratorium Rekayasa Data dan Inteligensi Bisnis (RDIB) Departemen Sistem Informasi, Fakultas Teknologi Elektro dan Informatika Cerdas.



Gambar 1.1 Relevansi Topik dengan Bidang Kajian Laboratorium RDIB

BAB II
DASAR TEORI DAN TINJAUAN PUSTAKA

Bab ini menjelaskan beberapa teori yang digunakan sebagai dasar pengerjaan Tugas Akhir. Selain itu, bab ini juga menjelaskan tinjauan singkat tehadap beberapa literatur terkait.

2.1. Dasar Teori

Bab ini menjelaskan dasar teori yang menjadi acuan tugas akhir

2.1.1 Perencanaan Kapasitas Rumah Sakit

Perencanaan kapasitas merupakan hal yang penting dilakukan di berbagai industri, tidak terkecuali rumah sakit. Hal ini dikarenakan jumlah sumber daya yang dimiliki rumah sakit sangat terbatas. Tidak hanya itu, spesialisme yang dimiliki masing-masing sumber daya juga menjadi tantangan dalam merencanakan kapatas rumah sakit. Secara umum, rumah sakit umum atau pubik memiliki lebih banyak permintaan dibandingkan dengan kapasitas yang tersedia. Oleh karena itu, dilakukan peramalan serta manajemen kapasitas supaya permintaan dapat dipenuhi dengan menjaga kepuasan dan kepentingan pasien.

Terdapat dua hal yang perlu diperhatikan dalam melakukan perencanaan kapasitas rumah sakit [1]. Pertama, keputusan terkait *flow* pasien yang terdiri dari jumlah tahunan atau bulanan pasien yang dapat diterima dan dirawat per grup spesialis atau patologi. Kedua, keputusan terkait sumber daya yang terdiri dari kebutuhan kapasitas per masing-masing spesialis atau grup patologi di rumah sakit. Untuk itu, seringkali rumah sakit kemudian dianalogikan menjadi sebuah gudang. Di dalam sebuah gudang raksasa ini terdapat banyak kasur. Tujuan objektif dari perencanaan kapasitas adalah untuk menentukan berapa banyak kasur yang tersedia (*available*) dan berapa banyak kasur yang dijadikan sebagai cadangan (*slack*). *Slack*

capacity ini selanjutnya digunakan untuk mengakomodasi pasien dengan waktu kedatanganan yang tidak pasti.

Pendekatan berbasis gudang atau *warehouse* pun memiliki banyak masalah, walaupun perspektif ini sangat lazim digunakan di banyak rumah sakit di berbagai negara. Pasalnya, salah satu faktor masalah yang menjadi cikal bakal lamanya *delay* admisi pasien adalah ketidakcukupan kapasitas rumah sakit, yang dalam hal ini adalah kasur. Dalam prakteknya, lambatnya *turnover* seringkali tidak semata-mata dikarenakan oleh komplikasi pasien yang berujung pada perpanjangan *Length-of-Stay* (LOS), tetapi juga dikarenakan penundaan atau *delay* perawatan yang dialami [2]. Faktor penundaan atau *delay* ini tidak dipertimbangkan dalam pendekatan berbasis gudang atau *warehouse*.

Untuk mengatasi kelemahan pendekatan berbasis *warehouse*, diusulkan pendekatan baru berbasis *lean* [2]. Pendekatan ini tidak hanya mempertimbangkan jumlah kasur belaka, tetapi juga mempertimbangkan *patient flow* dalam satu rumah sakit [7]. Perencanaan kapasitas dalam perspektif ini ditujukan untuk memperbaiki dan meningkatkan *patient flow*, yang berujung pada *turnover* yang lebih cepat. Dengan demikian, lebih banyak pasien yang dapat diakomodasi dan dilayani oleh rumah sakit.

2.1.2 *Red-Blue Transportation Problem*

Red-Blue Transportation Problem (RBTP) adalah bentuk generalisasi dari *transportation problem* yang diusulkan oleh Vancroonenburg et al. [8]. Pada *transportation problem*, terdapat grafik *bipartite* dengan kumpulan *supply node* S dan kumpulan *demand node* D. Diberikan *cost Cij* untuk masing-masing *edge* yang menghubungkan *supply node* dengan *demand node*. Permasalahan yang ingin dipecahkan adalah mencari konfigurasi pengiriman dari *supply node* ke *demand node* sedemikian rupa sehingga meminimalkan total *cost*.

Dalam usulan Vancroonenburg et al., *supply* node pada *transportation problem* diasosiasikan dengan warna. Untuk kasus *transportation problem*, setiap *supply node* diberikan warna yang sama, karena memang tidak ada batasan yang menghalangi *demand node* untuk menerima suplai dari *supply node* tertentu. Tetapi dalam kasus-kasus tertentu, *demand node* tidak bisa menerima suplai dari semua *supply node*. Beberapa contohnya seperti pada manajemen fans olahraga sepak bola, dimana fans dari tim yang berlawanan tidak boleh ditempatkan di kereta yang sama untuk mencegah *hooliganism* [9].

Tabel 2.1 Notasi Matematis dalam RBTP

Notasi	Deskripsi
S	Kumpulan *supply nodes*
D	Kumpulan *demand nodes*
S_j	Kumpulan *supply nodes* yang boleh mensuplai *demand node* $j \in D$
D_i	Kumpulan *demand nodes* yang boleh disuplai *supply node* $i \in S$
a_i	Jumlah suplai *node* $i \in S$
b_j	Jumlah permintaan *node* $j \in D$
$R\ (B)$	Kumpulan *red(blue) supply nodes*, $S = R \cup B$
$R_j(B_j)$	Kumpulan *red(blue) supply nodes* yang boleh mensuplai *demand node* $j \in D$
C_{ij}	Harga atau *cost* pengiriman satu unit suplai dari *supply node i* ke *demand node j*

$X_{ij} =$ jumlah suplai yang diberikan $node\ i$ ke $node\ j$

$$Y_{ij} = \begin{cases} 1 \text{ jika } node\ i \text{ mensuplai } node\ j \\ 0 \text{ jika } node\ i \text{ tidak mensuplai } node\ j \end{cases}$$

Formulasi RBTP seperti dibawah ini:

$$\text{Min} \sum_{i \in S} \sum_{j \in Di} C_{ij}.X_{ij}$$

Persamaan 2.1 Fungsi Tujuan RBTP

subject to:

$$\sum_{j \in Di} X_{ij} = a_i \quad \forall i \in S$$

Persamaan 2.2 Batasan Suplai RBTP

$$\sum_{i \in Sj} X_{ij} = b_j \quad \forall j \in D$$

Persamaan 2.3 Batasan Demand RBTP

$$X_{ij} \leq \min(a_i, b_j) \; . Y_{ij} \quad \forall i \in S, j \in D_i$$

Persamaan 2.4 Batasan Alokasi Maksimal RBTP

$$Y_{ij} + Y_{kj} \leq 1 \quad \forall i \in B, k \in R, j \in D_i \cap D_k$$

Persamaan 2.5 Batasan Warna RBTP

$$X_{ij} \in \mathbb{N}, Y_{ij} \in \{0,1\} \quad \forall i \in S, j \in D_i$$

Persamaan 2.6 Batasan Range Nilai RBTP

Fungsi objektifnya adalah untuk meminimalkan *total cost*. Batasan pada Persamaan 2.2 Batasan Suplai RBTPPersamaan 2.2 digunakan untuk memastikan bahwa *supply node i* hanya mengirimkan suplainya ke *demand note Di*. Batasan pada Persamaan 2.3 memastikan bawah *demand node j* hanya menerima suplai dari *supply node Sj*, Batasan pada Persamaan

2.4 memastikan bahwa *Xij* tidak akan memiliki nilai ketika tidak ada alokasi *Yij*. Ketika *Yij* bernilai 1, maka *Xij* akan memiliki nilai *ai* (ketika *ai* > *bj*), atau *bj* (ketika *ai* < *bj)*. Batasan pada Persamaan 2.5 memastikan bahwa tidak ada *supply node* dengan warna berbeda yang mensuplai *demand node j*. Batasan pada Persamaan 2.6 menyatakan bahwa *Xij* memiliki nilai berupa bilangan natural, dan *Yij* bernilai 0 atau 1 saja.

2.1.3 Admisi Pasien

Berbeda dengan perencanaan kapasitas rumah sakit, admisi pasien langsung berbicara tentang pasien mana yang akan diadmisi beserta kapan dan di kasur yang mana. Perencanaan kapasitas berada di level strategis / *tactical*, sedangkan admisi pasien berada di level operasional [5]. Secara umum, pasien dapat dikategorikan menjadi *inpatient* dan *outpatient* [10]. *Inpatient* akan menginap di rumah sakit, sedangkan *outpatient* hanya akan menghabiskan beberapa jam di rumah sakit. *Inpatient* selanjutnya akan dibagi menjadi:

- *Emergency Inpatient*
- *Elective Inpatient*
- *Admitted Inpatient*

Penjadwalan *emergency inpatient* secara umum lebih sulit dilakukan, dikarenakan kedatangannya (*arrival time*) yang tidak terduga. Faktor ketidakpastian ini bahkan menjadikan penjadwalan pasien di unit gawat darurat (UGD) sebuah golongan masalah tersendiri [11]. Di sisi lain, rumah sakit secara umum memiliki lebih banyak kebebasan dalam menjadwalkan admisi *elective patient*. Kebebasan inilah yang menjadikan rumah sakit harus sebisa mungkin memaksimalkan sumber daya, efisensi operasional, dan disaat bersamaan mengutamakan kenyamanan dan kebutuhan pasien [12].

Terdapat dua kondisi yang harus dipenuhi supaya penjadwalan admisi pasien dapat dilakukan [5]. Pertama, penjadwal harus mempunyai akses informasi ketersediaan sumber daya. Dalam konteks ini, sumber daya dapat berupa kasur, spesialis, serta alat-alat medis terkait. Kedua, penjadwal harus mempunyai kewenangan terbataas untuk mengubah tanggal admisi. Pada kenyataannya ,seringkali penjadwalan hanya membawa fungsi administratif; penjadwal hanya sekedar menjadwalkan admisi pasien sesuai kebutuhan dan preferensi pasien, tanpa memikirkan efisiensi operasional dan utilisasi sumber dayanya [3].

Selanjutnya, penjadwalan dilakukan berbasis pada tiga variabel, yakni admisi *emergency patient*, *length-of-stay* (LOS), dan kebutuhan pasien. Untuk admisi *emergency patient*, secara umum rumah sakit akan menyediakan *slack capacity*. Kemudian untuk faktor LOS, secara umum rumah sakit akan mengestimasikan LOS dari pasien, sesuai dengan kondisi pasien. Terakhir, kebutuhan pasien tidak hanya berbicara mengenai peralatan mmedis yang perlu disediakan, tetapi juga mengenai servis apa saja yang pasien perlukan.

Masalah penjadwalan admisi pasien ini kemudian diformalkan ke dalam *Patient Admission Scheduling Problem* (PASP) oleh Demeester et al. [5]. Secara umum PASP dapat dibagi menjadi *static PASP* dan *dynamic PASP*. Dalam varian statis seperti yang dikemukakan oleh Demeester et al., tanggal admisi dan *discharge date* pasien diasumsikan sudah diketahui dan ditentukan. Varian penjadwalan statik ini dalam dunia riset operasi juga sering dikenal dengan sebutan *offline scheduling*. Dalam *offline scheduling*, penjadwalan dilakukan dengan informasi terhadap sistem secara penuh diketahui sebelumnya [13]. Lain dengan *online scheduling*, dimana penjadwalan dilakukan dengan informasi atau data yang diperoleh pada waktu penjadwalan dilakukan. Hal ini berujung pada penjadwalan yang mungkin dapat berubah ketika informasi baru diketahui [13]. Dalam kasus PASP misalnya,

menggunakan model *online scheduling* berarti pasien dapat di-*discharge* lebih cepat atau lebih lama bila dibandingkan dengan LOS, dan disaat bersamaan pasien baru mungkin tiba-tiba muncul, dan algoritma harus menentukan kapan pasien baru ini akan diadmisikan.

Dynamic PASP atau disebut juga *PASP under uncertainty* seperti yang diusulkan dalam [14], mempunyai tingkat kompleksitas yang jauh lebih tinggi bila dibandingkan dengan *static PASP*. Hal ini dikarenakan jadwal yang dibut akan secara berkesinambungan berubah, sesuai dengan munculnya pasien baru yang perlu diadmisi. Oleh karena itu, *dynamic PASP* dikategorikan ke dalam *predictive-reactive scheduling* [15].

Di sisi lain, Rosito et al. dalam penelitiannya menyebut *static PASP* sebagai *Patient-to-Bed Assignment Problem* (PABP), karena *static PASP* yang dipionirkan oleh Demeester et al. hanya sekedar mengalokasikan pasien *elective* dengan tanggal admisi dan tanggal *discharge* diketahui. Dia berpendapat bahwa PASP seharusnya menyinggung tidak hanya pasien *elective*, tetapi juga pasien *emergency*. Menurutnya, admisi tidak hanya berbicara tentang alokasi kasur ke pasien. Walaupun demikian, penjadwalan khusus untuk pasien *emergency* memang dapat dilakukan secara terpisah, seperti pada [11]. PASP menurutnya pun memiliki *multiple objective*, tidak seperti pada PABP yang secara umum hanya memiliki satu tujuan objektif.

Dalam tugas akhir ini, *static PASP* klasik seperti yang dipionirkan oleh Demeester et al. akan digunakan. Seperti yang sudah dijelaskan sebelumnya, PASP model statik ini dikhusukan untuk *elective inpatient.* Dalam kasus ini, pasien yang kedatangannya sifatnya darurat (*emergency inpatient*) tidak dipertimbangkan. Rumah sakit secara umum memiliki kebebasan yang lebih banyak dalam melakukan penjadwalan pasien elektif, karena beberapa faktor seperti kondisi medis pasien yang sifatnya tidak darurat atau *life-threatening*. Dengan kata lain, pihak rumah sakit akan menentukan tanggal admisi

serta tanggal *discharge* pasien. Beberapa admisi yang dapat dijadwalkan secara statik yakni operasi atau bedah elektif. Terutama di rumah sakit umum, dimana pasien menggunakan jaminan sosial yang disediakan negara. Hal ini dikarenakan jumlah pasien yang membutuhkan bantuan medis lebih banyak, sehingga memang terdapat keterbatasan alokasi sumber daya di rumah sakit. Salah satu penelitian terkait yakni [16] dimana penelitian dilakukan terhadap pasien bedah elektif di bedah sentral Rumah Sakit Dharmais. Selain itu, penelitian di Yunani yang dilakukan oleh Tsitsakis et al. [2] juga membahas tentang penjadwalan pasien bedah elektif di rumah sakit publik di Yunani.

Beberapa terminologi pada *static PASP* yakni:

- Pasien
  Pasien dalam hal ini khusus menandakan *elective inpatient*, dimana setiap pasien memiliki preferensi, kebutuhan spesialis, serta *expected length-of-stay* (LOS). Dalam hal ini LOS hanya berupa estimasi, sehingga memang dalam prakteknya seringkali LOS estimasi berbeda dengan LOS pada kenyataan.

- Departemen
  Departemen dalam PASP yang diformalisasikan oleh Demeester et al. merujuk pada suatu unit dalam rumah sakit yang memiliki keahlian atau spesialisme khusus di bidang tertentu. Bidang tersebut misalnya terkait perawatan kanker, *maternity* (terkait kelahiran), *burn center* (terkait luka bakar), dan lain-lain. Departemen dalam hal ini berbeda dengan poliklinik, yang notabene mengurus *outpatient* atau dengan kata lain khusus untuk rawat jalan. Departemen di sini khusus untuk rawat inap atau *inpatient*, sehingga memang karena keterbatasan sumber daya misalnya, suatu departemen dapat menerima pasien dengan patologi yang bukan merupakan bidang utamanya. Tentunya hal ini juga terbatas pada pilihan

prioritas departemen tersebut. Misalnya, ada kemungkinan pasien dengan patologi penyakit saraf diadmisikan ke departemen yang mengurus perawatan kanker karena kekurangan kasur.

- *Planning Horizon*
  Planning horizon dalam hal ini menandakan total timeslot atau durasi yang dipertimbangkan dalam penjadwalan.

- *Specialism*
  Spesialis atau *specialism* mengindikasikan kemampuan spesifik departemen di rumah sakit. Setiap pasien akan memiliki kebutuhan spesialisme yang berbeda-beda

- Ruangan
  Ruangan atau *room* memiliki properti yang membedakannya dengan ruangan lain. Misalnya, satu ruangan memiliki fasilitas tabung oksigen, sedangkan ruangan lain tidak memilikinya. Beberapa spesifikasi lain seperti jenis kelamin pasien yang diperbolehkan, serta ukuran ruangan.

- Transfer
  Dalam penjadwalan, pasien mungkin dipindahkan dari suatu ruangan atau kasur ke ruangan atau kasur lain. Transfer pasien mungkin dilakukan ketika pasien lain membutuhkan ruangan atau kasur yang ditempati pasien pertama.

Secara umum, PASP memang dapat memiliki variasi dalam variabel atau batasan yang dilibatkan. Seperti misalnya, mungkin rumah sakit tidak melakukan pembatasan jenis kelamin per ruangan, atau terdapat jenis ruangan yang berbeda dari sisi kualitas dan harga (ruangan umum, VIP, VVIP) seperti yang sering dilakukan oleh rumah sakit swasta, dan lain sebagainya.

PASP dikategorikan ke dalam *Red Blue Transportation Problem* (RBTP) oleh Vancroonenburg et al.. Bahkan, bentuk generalisasi dari *transporation problem* ini pada awalnya memang dibuat setelah Vancroonenburg melihat bagaimana PASP memiliki kemiripan dengan *transportation problem* [8]. Perbedaannya hanya pada pengalokasian kamar di dalam satu ruangan yang dalam kasus ini adalah *demand node*, hanya untuk pasien (*supply* node) dengan jenis kelamin yang sama. Jenis kelamin ini dapat dilambangkan dengan warna merah dan biru seperti pada RBTP. Ketika satu kamar sudah terisi oleh pasien perempuan, maka pasien selanjutnya di kamar itu juga harus perempuan. Dalam penelitian tersebut, Vancroonenburg juga memaparkan bagaimana permasalahan ini memiliki tingkat kesulitan *NP-hard*. Ia memaparkan bagaimana *Exact-3-Cover* (X3C) yang *NP-complete* dapat direduksi menjadi RBTP, *PARTITION* yang juga *NP-complete* dapat direduksi menjadi RBTP, dan bagaimana RBTP dengan *demand node* (kamar) berjumlah tetap adalah *polynomialy solvable*. Tabel berikut adalah daftar variabel yang digunakan dalam PASP statik.

Tabel 2.2 Variabel dalam PASP Statik

Notasi	Deskripsi
ap_p	Umur pasien p
ax_{dt}	Batas bawah umur di departemen dt
ay_{dt}	Batas atas umur di departemen dt
B	Kumpulan kasur, dengan $b \in B$
$bs_{p,b,t}$	Alokasi kasur *b* untuk pasien *p* pada timeslot *t*
d_p	Durasi tinggal (*Length-of-Stay*) pasien
DT	Kumpulan departemen, dengan $dt \in DT$
e_r	Ukuran atau kapasitas ruangan *r*, dimana nilainya 1 jika *single room*, 2 jika *twin room*, 4 jika *ward*
ep_p	Preferensi ukuran ruangan pasien p. Idealnya $ep_p = e_r$
$f_{r,t}$	Jumlah pasien perempuan ($g_p = 0$) di ruangan r pada timeslot *t*

	$f_{r,t} = \sum_{p \in P} (1 - g_p).rs_{p,r,t}$
g_p	Jenis kelamin pasien; bernilai 0 jika pasien *p* adalah perempuan, atau 1 jika pasien *p* adalah laki-laki
$m_{r,t}$	Jumlah pasien laki-laki ($g_p = 1$) di ruangan *r* pada timeslot *t* $m_{r,t} = \sum_{p \in P} g_p.rs_{p,r,t}$
O	Kumpulan *Room properties* atau fasilitas rumah sakit, dengan $o \in O$.
OA_r	Kumpulan *Room properties* yang tersedia di ruangan *r*
OP_p	Kumpulan *Room properties* yang diinginkan pasien *p*. Idealnya pasien *p* dirawat di ruangan *r*, sehingga $OP_p \subseteq OA_r$
OR_p	Kumpulan *Room properties* yang dibutuhkan pasien *p*. Idealnya pasien *p* dirawat di ruangan *r*, sehingga $OR_p \subseteq OA_r$
P	Kumpulan pasien, dengan $p \in P$
PS_p	Periode admisi atau kumpulan waktu/timeslot tinggal pasien
R	Kumpulan ruangan, dengan $r \in R$.
RDT_{dt}	Kumpulan ruangan yang dimiliki departemen *dt*
$rp_{sc,p,r}$	Penalti ketika pasien *p* ditempatkan di ruangan *r*. *sc* menandakan penalty untuk *soft constraint* mana yang diperhitungkan (SC3, SC4, SC5, SC6, SC7, atau SC8)
S	Kumpulan spesialisme, dengan $s \in S$
sp_p	Spesialisme yang dibutuhkan pasien *p*. Idealnya pasien *p* dirawat di ruangan *p* dan di departemen *dt*, sehingga $sp_p \in SR_r \wedge sp_p \in SDT_{dt}$
SDT_{dt}	Kumpulan prioritas spesialisme di departemen *dt*
SR_r	Kumpulan prioritas spesialisme di ruangan *r*

T	Kumpulan timeslot (*planning horizon*), dengan $t \in T$
$t_{s,p}$	Timeslot/waktu admisi pasien

Variabel Keputusan:

$$rs_{p,r,t} = \begin{cases} 1 \text{ jika pasien } p \text{ ditempatkan di ruangan } r \text{ pada timeslot } t, \\ 0 \text{ jika tidak} \end{cases}$$

$$tr_{p,t} = \begin{cases} 1 \text{ jika pasien } p \text{ berada di ruangan berbeda pada timeslot } t-1, \\ 0 \text{ jika tidak} \end{cases}$$

Hard constraint:

HC1: Jumlah pasien yang dialokasikan ke suatu ruangan pada satu timeslot terbatas pada kapasitas ruangan tersebut. Bentuk matematisnya yakni:

$$\sum_{p \in P} rs_{p,r,t} \leq e_r, \forall r \in R, \forall t \in T$$

Persamaan 2.7 Batasan Kapasitas Ruangan

Setiap penempatan pasien *p* pada ruangan *r* di timeslot *t* tidak boleh melebihi kapasitas ruangan *r* (e_r).

HC2: Tanggal admisi dan *discharge* pasien adalah tetap. Bentuk matematisnya yakni:

$$\sum_{p \in P} PS_p = \{t \in T \mid t \geq t_{s,p} \ \wedge \ t < t_{s,p} + \ d_p\}, \forall p \in P$$

Persamaan 2.8 Batasan Periode Admisi

Untuk setiap pasien *p*, periode admisinya (PS_p) yakni dimulai dari tanggal admisi pasien tersebut ($t_{s,p}$), hingga tanggal *discharge* pasien tersebut. Tanggal *discharge* dihitung dari

tanggal admisi ($t_{s,p}$) ditambah *length-of-stay* (d_p). Dengan demikian, pasien harus dijadwalkan sesuai dengan tanggal admisi dan tanggal *discharge* pasien tersebut.

HC3: Pasien harus dialokasikan ke ruangan untuk setiap timeslot dalam *length-of-stay* pasien. Bentuk matematisnya yakni:

$$\sum_{r \in R} rs_{p,r,t} = 1, \forall p \in P, \forall t \in PS_p$$

Persamaan 2.9 Batasan Waktu Alokasi

Pasien *p* harus dijadwalkan ke satu dan hanya satu ruangan *r* untuk semua timeslot *t* pada periode admisi (PS_p) pasien tersebut. Selain membatasi penjadwalan supaya hanya dilakukan dalam periode admisi pasien, batasan ini juga secara tidak langsung mencegah adanya duplikasi, dimana pasien dijadwalkan ke lebih dari satu ruangan dalam timeslot yang sama.

Soft Constraint:

SC1: Pasien-pasien dalam satu ruangan dan timeslot memiliki jenis kelamin yang sama. Bentuk matematisnya yakni:

$$GP = \sum_{r \in R} \sum_{t \in T} \min(f_{r,t}, m_{r,t})$$

Persamaan 2.10 Biaya Pelanggaran Batasan Jenis Kelamin Ruangan

Perhitungan jumlah pelanggaran SC1 diperoleh dengan melihat pada setiap ruangan *r*, dimana nilainya sama dengan jumlah denominasi pasien paling sedikit. $f_{r,t}$ adalah jumlah pasien perempuan, dan $m_{r,t}$ adalah jumlah pasien laki-laki.

SC2: Pasien tidak ditransfer ke ruangan lain selama periode *length-of-stay* pasien tersebut. Bentuk matematisnya yakni:

$$TR = \sum_{p \in P} \sum_{t \in T} tr_{p,t}$$

Persamaan 2.11 Biaya Transfer

Jumlah pelanggaran SC2 dihitung dengan melihat pada jumlah transfer yang terjadi ($tr_{p,t}$).

SC3: Pasien ditempatkan di departemen yang memenuhi kebutuhan patologi pasien. Bentuk matematisnya yakni:

$$DSP = \sum_{p} \sum_{r} \sum_{t} rs_{p,r,t}.rp_{sc3,p,r}$$

dengan

$$rp_{sc3,p,r} = \begin{cases} 0 \text{ jika } sp_p \in SDT_{dt}, \text{dengan } RDT_{dt} \ni r \\ 1 \text{ jika tidak} \end{cases}$$

Persamaan 2.12 Biaya Pelanggaran Spesialisme Departemen

Jumlah pelanggaran SC3 dihitung dengan meninjau pada setiap pasien p yang pada waktu atau timeslot t ditempatkan di ruangan *r*, dimana ruangan *r* merupakan anggota dari kumpulan ruangan (RDT_{dt}) departemen *dt*, dan spesialisme pasien (sp_p) tidak terdapat dalam kumpulan spesialisme prioritas (SDT_{dt}) departemen tersebut.

SC4: Pasien ditempatkan di ruangan yang memenuhi kebutuhan fasilitas pasien. Bentuk matematisnya yakni:

$$PRP = \sum_{p}\sum_{r}\sum_{t} rs_{p,r,t}.rp_{sc4,p,r}$$

dengan

$$rp_{sc4,p,r} = \begin{cases} 0 \text{ jika } OR_p \subseteq OA_r \\ 1 \text{ untuk setiap } o \in OR_p \wedge o \notin OA_r \end{cases}$$

Persamaan 2.13 Biaya Pelanggaran Kebutuhan Properti Ruangan Pasien

Jumlah pelanggaran SC4 dihitung dengan meninjau pada setiap pasien p yang pada waktu atau timeslot t ditempatkan di ruangan r, dimana ada fasilitas atau properti o yang terdapat pada kumpulan fasilitas atau properti yang dibutuhkan pasien (OR_p), tetapi tidak terdapat pada kumpulan fasilitas atau properti yang tersedia dalam ruangan (OA_r).

SC5: Pasien ditempatkan di ruangan yang memenuhi preferensi fasilitas pasien. Bentuk matematisnya yakni:

$$PPP = \sum_{p}\sum_{r}\sum_{t} rs_{p,r,t}.rp_{sc5,p,r}$$

dengan

$$rp_{sc5,p,r} = \begin{cases} 0 \text{ jika } OP_p \subseteq OA_r \\ 1 \text{ untuk setiap } o \in OP_p \wedge o \notin OA_r \end{cases}$$

Persamaan 2.14 Biaya Pelanggaran Preferensi Properti Ruangan Pasien

Jumlah pelanggaran SC5 dihitung dengan meninjau pada setiap pasien p yang pada waktu atau timeslot t ditempatkan di ruangan r, dimana ada fasilitas atau properti o yang terdapat pada kumpulan fasilitas atau properti yang merupakan

preferensi pasien (OP_p), tetapi tidak terdapat pada kumpulan fasilitas atau properti yang tersedia dalam ruangan (OA_r).

SC6: Pasien ditempatkan di ruangan yang memenuhi kebutuhan spesialisme patologi pasien. Bentuk matematisnya yakni:

$$RSP = \sum_{p} \sum_{r} \sum_{t} rs_{p,r,t} . rp_{sc6,p,r}$$

dengan

$$rp_{sc6,p,r} = \begin{cases} 0 \text{ jika } sp_p \in SR_r \\ 1 \text{ jika tidak} \end{cases}$$

Persamaan 2.15 Biaya Pelanggaran Spesialisme Ruangan

Jumlah pelanggaran SC6 dihitung dengan meninjau pada setiap pasien *p* yang pada waktu atau timeslot *t* ditempatkan di ruangan *r*, dimana spesialisme pasien tersebut (sp_p) tidak terdapat pada kumpulan spesialisme yang diprioritaskan (SR_r) di ruangan tersebut.

SC7: Pasien ditempatkan di ruangan yang memenuhi preferensi ruangan pasien

$$RSPP = \sum_{p} \sum_{r} \sum_{t} rs_{p,r,t} . rp_{sc7,p,r}$$

dengan

$$rp_{sc7,p,r} = \begin{cases} 0 \text{ jika } ep_p = e_r \\ 1 \text{ jika tidak} \end{cases}$$

Persamaan 2.16 Biaya Pelanggaran Preferensi Ukuran Ruangan Pasien

Jumlah pelanggaran SC7 dihitung dengan meninjau pada setiap pasien *p* yang pada waktu atau timeslot *t* ditempatkan di

ruangan *r*, dimana preferensi ukuran ruangan pasien tersebut (ep_p) berbeda dengan ukuran ruangannya (e_r).

SC8: Pasien ditempatkan di departemen yang mendukung umur pasien

$$DAP = \sum_{p} \sum_{r} \sum_{t} rs_{p,r,t} . rp_{sc8,p,r}$$

dengan

$$rp_{sc8,p,r} = \begin{cases} 0 \text{ jika } ap_p \geq ax_{dt} \wedge ap_p \leq ay_{dt}, \text{dengan } RDT_{dt} \ni r \\ 1 \text{ jika tidak} \end{cases}$$

Persamaan 2.17 Biaya Pelanggaran Batasan Umur Departemen

Jumlah pelanggaran SC7 dihitung dengan meninjau pada setiap pasien *p* yang pada waktu atau timeslot *t* ditempatkan di ruangan *r*, dimana ruangan *r* merupakan anggota dari kumpulan ruangan (RDT_{dt}) departemen *dt*, dan umur pasien tersebut (ap_p) lebih kecil daripada batas bawah umur pasien di departemen (ax_{dt}) atau lebih besar daripada batas atas umur pasien di departemen (ay_{dt}).

Kemudian, pembobotan dilakukan terhadap *soft constraint*.

Tabel 2.3 Bobot per Batasan

Batasan	Variabel	Bobot
SC1	w_g	5,0
SC2	w_t	11,0
SC3	w_{ds}	1,0
SC4	w_{pr}	5,0
SC5	w_{pp}	2,0
SC6	w_{rs}	1,0
SC7	w_{rsp}	0,8
SC8	w_{da}	10,0

Nilai bobot ini tentunya bisa bervariasi pada kasus yang berbeda. Contohnya, ketika rumah sakit mengutamakan pemenuhan kebutuhan serta preferensi pasien dibanding dengan menghindari transfer, maka bobot SC3 dan SC4 dapat dinaikan, dan bobot SC2 (transfer) diturunkan.

Fungsi tujuan:

$$\text{Min } w_g.GP + w_t.TR + w_{ds}.DSP + w_{pr}.PRP + w_{pp}.PPP + w_{rs}.RSP + w_{rsp}.RSPP + w_{da}.DAP$$

Persamaan 2.18 Fungsi Tujuan PASP

2.1.4 *Benchmark Dataset*

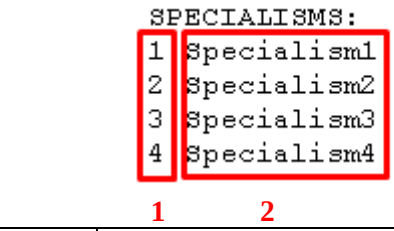
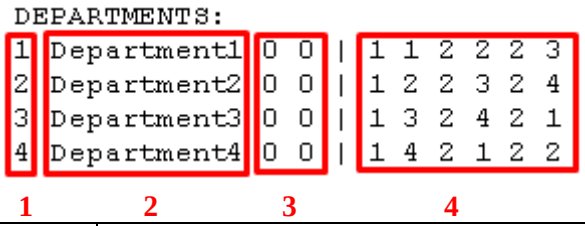
Dataset terdiri dari *Rooms, Roomproperties, Beds, Departments, Specialisms,*dan *Patients.*. Terdapat 7 *instances*, dimana *instance* pertama (*instance 0*) digunakan oleh Demeester et al. dalam [5]. *Instance* pertama dengan 6 *instance* baru selanjutnya digunakan oleh Bilgin et al. dalam [6].

Tabel 2.4 Karakteristik Benchmark Instances

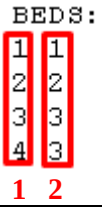
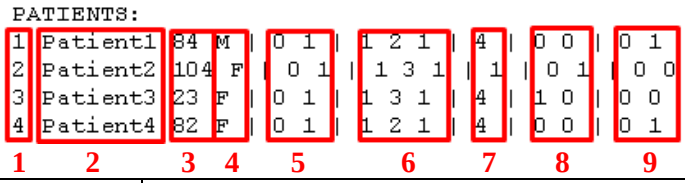
Jumlah Properti	*Benchmark Instances*						
	0	1	2	3	4	5	6
Departemen	6	4	6	5	6	4	4
Pasien	660	693	778	757	782	631	726
Ruangan	150	98	151	131	155	102	104
Kasur	447	286	465	395	471	325	313
Spesialisme	6	4	6	5	6	4	4

Setiap *instance* memiliki 2 *roomproperties*, dan *planning horizon* sepanjang 14 hari. Berikut adalah tabel deskripsi dataset.

Tabel 2.5 Deskripsi Dataset

Kolom	Deskripsi
Spesialisme (*Spesialisms*)	
	
1	ID Spesialisme
2	Nama Spesialisme
Departemen (*Departments*)	
	
1	ID Departemen
2	Nama Departemen
3	*Range* umur pasien yang dapat diterima departemen Nilai 0 pada kedua digit menandakan departemen menerima pasien dengan umur berapapun
4	Pilihan spesialisme Digit ganjil menandakan urutan pilihan Digit genap menandakan ID spesialisme yang dimaksud dalam urutan pilihan di digit sebelumnya Contoh pada baris pertama: Departement1 memiliki pilihan pertama Spesialism1 dan urutan pilihan kedua Spesialism2 dan Spesialism3
Properti ruangan (*Roomproperties*)	

<table>
<tr><td colspan="2">ROOMPROPERTIES:
1 telemetry
2 oxygen
1 2</td></tr>
<tr><td>1</td><td>ID roomproperties</td></tr>
<tr><td>2</td><td>Nama roomproperties</td></tr>
<tr><td colspan="2">Ruangan (Rooms)</td></tr>
<tr><td colspan="2">ROOMS:
1 | 11 | 1 | 1 | D | 1 1 3 2 1 3 | 1 1
2 | 12 | 1 | 1 | D | 1 1 1 2 2 3 | 1 0
3 | 13 | 2 | 1 | D | 1 1 1 2 1 3 | 0 1
4 | 14 | 2 | 1 | D | 1 1 3 2 1 3 | 0 1
1 2 3 4 5 6 7</td></tr>
<tr><td>1</td><td>ID Ruangan</td></tr>
<tr><td>2</td><td>Nama Ruangan</td></tr>
<tr><td>3</td><td>Kapasitas Ruangan
Kolom ini juga menandakan tipe ruangan
1 = single room
2 = twin room
3 = ward (4-bed room)</td></tr>
<tr><td>4</td><td>Departemen dimana ruangan berada</td></tr>
<tr><td>5</td><td>Jenis kelamin pasien yang dapat diterima ruangan
D = Dependent. Jenis kelamin pasien yang dapat diterima tergantung pada pasien pertama yang diterima di ruangan</td></tr>
<tr><td>6</td><td>Spesialisme ruangan
Digit ganjil menandakan urutan pilihan
Digit genap menandakan ID spesialisme yang dimaksud dalam urutan pilihan di digit sebelumnya
Contoh pada baris bertama:
Ruangan 11 memiliki pilihan pertama Spesialism1 dan Spesialism3, dan pilihan ketiga Spesialism2</td></tr>
</table>

7	Kelengkapan *roomproperties* Jumlah digit sesuai dengan jumlah *roomproperties* di rumah sakit Nilai 1/0 menandakan tersedia/tidak tersedia Indeks atau posisi digit menandakan *roomproperties* Contoh pada baris pertama: Ruangan 11 memiliki *roomproperties* pertama (*telemetry*)
Kasur (*Beds*)	
	
1	ID kasur
2	ID ruangan dimana kasur berada
Pasien (*Patients*)	
	
1	ID pasien
2	Nama pasien
3	Umur pasien
4	Jenis kelamin pasien
5	Digit pertama: tanggal admisi pasien Digit kedua: *discharge date* pasien Digit pertama – Digit kedua: *length of stay* pasien
6	Spesialisme yang dibutuhkan pasien Digit pertama menandakan jumlah spesialisme yang dibutuhkan

	Digit genap menandakan spesialisme yang dibutuhkan Digit ganjil (terkecuali digit pertama) menandakan durasi yang dibutuhkan untuk spesialisme pada digit sebelumnya Contohnya untuk pasien pertama, pasien memiliki kebutuhan 1 spesialisme, dengan spesifikasi Spesialism2 untuk durasi 1 hari
7	Preferensi ruangan 1 = *Single Room* 2 = *Twin Room* 3 = *Ward*
8	Kebutuhan *roomproperties* Jumlah digit sesuai dengan jumlah *roomproperties* Nilai 1/0 menandakan butuh/tidak butuh Indeks/posisi digit menandakan *roomproperties* Contoh pada baris pertama: Patient1 tidak membutuhkan *telemetry* dan *oxygen*
9	Preferensi *roomproperties* Jumlah digit sesuai dengan jumlah *roomproperties* Nilai 1/0 menandakan preferensi pasien Indeks/posisi digit menandakan *roomproperties* Contoh pada baris pertama: Patient1 memiliki preferensi *roomproperties oxygen*

2.1.5 Algoritma *Variable Neighborhood Search*

Algoritma *Variable Neighborhood Search* (VNS) pertama kali diperkenalkan oleh Hansen pada tahun 1997 [6]. Algoritma ini didasarkan pada algoritma *local search*, dimana dicari solusi terbaik pada suatu daerah lokal di dalam *search space*. Dengan

metode *local search*, seringkali algoritma hanya berhasil menemukan *local optimum*. *Local optimum* merupakan solusi terbaik di suatu daerah lokal di *search space*. Untuk menghindari hal ini, metode VNS akan melakukan *neighborhood* secara deterministik setelah solusi terbaik ditemukan. *Local optimum* pada suatu *neighborhood* tidak selalu merupakan *local optimum* pada *neighborhood* lain. Konsep inilah yang memebedakan algoritma VNS dengan algoritma *local search* sederhana. Dengan berpindah *neighborhood*, VNS dapat menemukan *global optimum* yang sebenarnya adalah *local minimum* dari semua kemungkinan *neighborhood*. Konsep selanjutnya yang digunakan adalah bahwa *local minimum* seringkali memberikan informasi terkait *global minimum*. Sebagai contoh, mungkin keduanya memiliki nilai yang sama dibeberapa variabel. Tetapi, algoritma tidak mengetahui mana variabel tersebut. Oleh karena itu, dilakukan operasi di *neighborhood* sekitar *local optimum* untuk memperoleh solusi yang lebih baik.

Tidak hanya dibidang penjadwalan, VNS juga digunakan di banyak bidang lain seperti desain jaringan, analisis kluster, biologi, geometri, dan desain telekomunikasi [16]. Hal ini dikarenakan kemudahan implementasi VNS ditambah dengan parameternya yang relatif sedikit. Seperti yang disebutkan oleh Gendreau dan Potvin, ketika metode yang lebih sederhana memberikan hasil yang kompetitif, maka metode yang lebih sederhanalah yang harus digunakan. Konsep seperti ini didasarkan pada *occam's razor*, dimana penjelasan yang paling sederhana adalah penjelasan yang paling benar. Dalam kasus ini, seringkali metode *population-based* seperti algoritma genetik yang didasarkan pada analogi, sulit menjelaskan mengapa terjadi peningkatan kualitas solusi ketika parameter diubah. Hal ini dikarenakan penamaan parameter disesuaikan dengan analogi yang diambil, misalnya pada algoritma genetika terdapat parameter *crossver probability* dan *mutation probability*.

Konsep *neighborhood change* pada VNS dilakukan dengan terlebih dahulu menentukan *neighborhood structure* k.

Algorithm 1 Neighborhood change

```
Function NeighborhoodChange (x, x', k)
1  if f(x') < f(x) then
2      x ← x' // Make a move
3      k ← 1 // Initial neighborhood
   else
4      k ← k + 1 // Next neighborhood
return x, k
```

Kode 2.1 Pseudocode Fungsi Perpindahan Neighborhood

Ketika solusi baru $f(x')$ lebih kecil daripada solusi $f(x)$, maka dilakukan perpindahan dimana titik x diberi nilai titik x'. Solusi ini pun menjadi solusi terbaik untuk sementara. Jikalau ternyata solusi $f(x)$ lebih baik dibanding solusi baru $f(x')$, maka nilai *k* akan ditambah 1, yang menandakan perpindahan *neighborhood*. Perpindahan secara deterministic dilakukan dengan *variable neighborhood descent* (VND). Pencarian solusi terbaik akan dilakukan sampai nilai k_{max} tercapai. Solusi terbaik dari kemungkinan *neighborhood structure* tersebut pun sudah diperoleh. Dalam hal ini, "*descend*" menandakan solusi yang terus turun menuju *global optimum*.

Algorithm 2 Variable neighborhood descent

```
Function VND (x, k_max)
1  k ← 1
2  repeat
3      x' ← arg min_{y∈N_k(x)} f(y) // Find the best neighbor in N_k(x)
4      x, k ← NeighborhoodChange (x, x', k) // Change neighborhood
   until k = k_max
return x
```

Kode 2.2 Pseudocode Algoritma VND

Dalam implementasinya, VND dapat menggunakan *first descent heuristic*, atau *steepest descent heuristic*. Dengan heuristik *steepest descent* seperti pada *pseudocode* VND di atas,

perpindahan dilakukan ke titik terbaik dalam *search space*. Algoritma akan mengecek dan memperhitungkan semua *direct neighbor*, lalu kemudian perpindahan dilakukan ke titik dengan penurunan paling drastic (*steepest descent*). Implementasi lain dari VND yakni menggunakan heuristic *first descent*, perpindahan dilakukan seketika saat titik dalam *search space* yang memiliki penurunan nilai penalti atau *fitness function* ditemukan pertama kali.

Berbeda dengan VND, metode *reduced* VNS merupakan pendekatan yang besifat stokastik.

Algorithm 3 Reduced VNS

```
Function RVNS(x, k_max, t_max)
1  repeat
2      k ← 1
3      repeat
4          x' ← Shake(x, k)
5          x, k ← NeighborhoodChange (x, x', k)
       until k = k_max
6      t ← CpuTime ()
   until t > t_max
return x
```

Kode 2.3 Pseudocode Algoritma Reduced VNS

"*Descent*" tidak secara eksplisit dilakukan, melainkan titik baru pada *feasible region* akan dipilih secara acak. Titik atau nilai tersebut kemudian dibandingkan dengan titik atau nilai solusi yang sekarang dimiliki. Proses ini dilakukan hingga t_{max} tercapai. Fungsi *shake(x,k)* dalam hal ini digunakan untuk mengacak dan memperoleh titik baru.

Algorithm 4 Shaking function

Function Shake(x, k)
1 $w \leftarrow [1+\text{Rand}(0,1) \times |\mathcal{N}_k(x)|]$
2 $x' \leftarrow x^w$
return x'

Kode 2.4 Pseudocode Fungsi Shaking

Dalam hal ini, *reduced VNS* tidak melakukan *local search* deterministik seperti yang dilakukan VND. *Reduced VNS* umumnya digunakan ketika masalah yang dihadapi memiliki *search space* yang luas, sehingga pencarian menggunakan metode deterministic menjadi sangat mahal.

Seperti pada [16], metode VNS secara umum memiliki kesuluruhan 11 aspek metaheuristik yang baik, yakni:

1. *Simplicity*: metaheuristik berbasis pada prinsip yang sederhana dan mudah dipahami
2. *Precision*: metaheuristik dapat diformulasikan ke dalam bentuk matematis, terbebas dari analogi yang digunakan sebagai dasar atau inspirasi pembuatan metodenya
3. *Coherence*: semua langkah d alam heuristik dapat mengikuti prinsip metaheuristik
4. *Effectiveness*: heuristik untuk masalah tertentu harus bisa setidaknya memberikan solusi optimal atau *near-optimal* untuk kebanyakan *benchmark instances*
5. *Efficiency*: heuristik untuk masalah tertentu dapat menemukan solusi yang optimal, *near-optimal*, atau bahkan lebih baik dengan waktu komputasi *moderate*
6. *Robustness*: performa dari metaheuristik harus konsisten terhadap banyak variasi *instance*
7. *User friendliness*: metaheuristik diekspresikan dengan jelas, mudah dimengerti serta mudah digunakan/diimplementasikan.
8. *Innovation*: metaheuristik memberikan kontribusi inovasi baik dari sisi efisiensi maupun sisi efektivitas

9. *Generality*: metaheuristik memberikan hasil solusi yang relatif baik untuk varian permasalahan ang berbeda-beda
10. *Interactivity*: pengguna metaheuristik dapat dengah mudah memasukan pengetahuannya untuk meningkatkan kualitas solusi
11. *Multiplicity*: metaheuristic dapat memberikan banayak solusi optimal atau *near-optimal*, yang mana pengguna dapat memilih

Dikarenakan aspek tersebut, VNS sangat relevan digunakan hingga sekarang. Bahkan, terdapat konferensi tahunan khusus VNS yakni *International Conference on Variable Neighborhood Search* (ICVNS). Dalam implementasinya, VNS sering dimodifikasi atau dikustomisasi untuk menyelesaikan masalah-masalah spesifik. Beberapa contoh modifikasinya yakni *Variable Neighborhood Decomposition Search*, dimana pencarian dilakukan di *search space* yang merupakan masalah utama yang telah di dekomposisi menjadi masalah-masalah yang lebih kecil. Selain itu, penggabungan metode pencarian deterministik dan pencarian stokastik juga sering dilakukan (*general VNS*).

2.2. Tinjauan Pustaka

Penelitian sebelumnya dilakukan oleh Demeester et al. [5], dimana masalah penjadwalan admisi pasien diformalkan ke dalam *Patient Admission Scheduling Problem* (PASP). Demeester melakukan penelitian ini setelah melihat bagaimana penelitian sebelumnya oleh Van Dierdonck [3] menunjukkan adanya jarak yang signifikan antara praktek penjadwalan admisi dengan yang kebanyakan rumah sakit lakukan di kenyataan. Demeester berpendapat bahwa penjadwalan yang dilakukan secara tersentralisir dan dengan memerhatikan aspek efisiensi operasional serta kebutuhan dan preferensi pasien sangatlah krusial mengingat keterbatasan sumber daya dan biaya di dunia *healthcare*. Penelitiannya berfokus pada aspek operasional penjadwalan, tidak pada aspek taktis dan strategis seperti yang kebanyakan penelitian sebelumnya telusuri [17].

Lewat penelitiannya, Demeester menawarkan solusi penjadwalan dengan algoritma Tabu Search. Demeester bersama Bilgin et al. kemudian menawarkan solusi *hyper-heuristic* untuk memecahkan PASP dan *nurse rostering problem* [6].

Setelah penelitian Demeester, optimasi serta penjadwalan di dunia *healthcare* mulai menjadi topik yang hangat dibicarakan di dunia riset operasi. Range et al. memberikan solusi baru dengan metode *column generation*, dimana PASP dipecah menjadi *master problem* dan *pricing problem* [18]. Kemudian Bastos et al. memberikan solusi menggunakan *exact* pada tahun 2016 [12]. Boudali et al. melakukan penelitian serupa tetapi hanya berfokus kepada admisi pasien di laboratorium darurat [11]. Solusi yang ditawarkan adalah dengan menggunakan metode *harmony search.*

Ceschia dan Schaerf pada tahun 2012 memiliki argumen bahwa model *static* yang diperkenalkan oleh Demeester et al. belum begitu representatif dengan dunia nyata. Oleh karena itu, mereka memperkenalkan model *dynamic PASP* dimana terdapat aspek ketidakpastian waktu kedatangan dan *discharge date* pasien. Model ini selanjutnya diperluas oleh Guido et al. pada tahun 2017 [19]. Model dinamis ini digunakan oleh Lusby et al. dalam penelitiannya pada tahun 2016 [20]. Lusby menawarkan metode *adaptive large neighborhood search* untuk memberikan solusi terhadap permasalahan *dynamic PASP*.

Selanjutnya, metode *Variable Neighborhood Search* (VNS) diperkenalkan oleh Hansen dan Mladenovic pada tahun 1997 [4]. Implementasinya yang sederhana dan tidak berbasis pada analogi menyebabkan VNS sering dipakai di berbagai bidang dan cabang ilmu, seperti penjadwalan, optimasi, biologi, geometri, *graph theory*, dan lannya. Hal ini juga didasarkan pada keraguan komunitas riset operasi pada metode-metode berbasis analogi, yang seringkali memiliki kontribusi di bidang

riset operasi yang sangat *marginal*. Hal ini ditunjukkan pada penelitian yang dilakukan oleh Sorensen pada tahun 2012 [21], dimana ia mengkritisi metode-metode baru yang menggunakan analogi seperti *Intelligent Water Drop Algorithm, Bat Algorithm, Harmony Search Algorithm, Frog Leaping Algorithm*, dan lainnya.

VNS pun dalam implementasinya memiliki metode yang berbeda-beda, beberapa diantaranya seperti *reduced VNS, skewed VNS, Variable Neighborhood Descent*, dan lainnya. Beberapa penelitian yan menggunakan VNS yakni penyelesaian *capacitated clustering problem* [22], penyelesaian *vehicle routing problem* [23] [24]. VNS juga dapat digunakan sebagai *hyperheuristics*, seperti pada [25].

“Halaman ini sengaja dikosongkan”

BAB III
METODOLOGI PENELITIAN

Pada bab ini dijelaskan sistematis pengerjaan tugas akhir

Gambar 3.1 Tahapan Pengerjaan Tugas Akhir

3.1. Pemahaman Masalah

Pada tahap ini, masalah yang dihadapi yakni *static PASP* dipahami baik dari segi tujuan maupun batasan-batasan terkait. Permasalahan ini kemudian dijadikan basis pengerjaan dimana hasil yang diberikan harus merupakan solusi dari permasalahan *static PASP*. Setelah itu, dilakukan pemahaman model matematis yang terdiri variabel keputusan, fungsi objektif, serta batasan-batasannya sesuai dengan [5] dan bagaimana hal ini akan berdampak pada desain algoritma. Terakhir, dilakukan pemahaman lebih lanjut *benchmark dataset* yang digunakan dan bagaimana dataset mempengaruhi algoritma yang digunakan.

3.2. Studi Literatur

Pada tahap ini, dilakukan studi literatur untuk meninjau bagaimana berbagai penelitian menghadapi *static PASP* dengan caranya masing-masing, dan bagaimana formulasi permasalahannya sering bervariasi antar penelitian. Selanjutnya, literatur terkait implementasi VNS beserta metode

implementasinya dipahami untuk memberikan gambaran bagaimana implementasi VNS yang baik untuk kasus yang dihadapi.

3.3. Desain dan Implementasi Algoritma

Pada tahap ini, algoritma VNS disusun untuk menyelesaikan *static PASP*. Implementasi dilakukan dengan menggunakan Bahasa pemrogaman Java.

3.4. Pengujian Algoritma

Algoritma diujikan untuk memecahkan semua *benchmark instances* yang tersedia menggunakan parameter yang ditentukan.

3.5. Analisis dan Penarikan Kesimpulan

Pada tahap ini, dilakukan analisis solusi yang diperoleh berupa jadwal serta penalti yang diperoleh per solusi. Analisis dilakukan dengan membandingkan penalti serta *running time* yang diperoleh. Analisis dilakukan hanya dengan membandingkan hasil yang diperoleh di sampel, dan tidak menggunakan uji statistik. Hal ini dikarenakan uji statistik seperti *f-test* dan *t-test* dalam kasus ini memerlukan jumlah sampel yang lebih banyak. Kesimpulan dibuat dengan melihat solusi terbaik yang dihasilkan per *benchmark instances*.

3.6. Pengerjaan Laporan dan Dokumentasi

Pada tahap ini, dilakukan dokumentasi tahapan yang sudah dilakukan dalam bentuk laporan. Laporan ini akan terdiri dari:

1. Pendahuluan, dimana latar belakang serta tujuan pembuatan tugas akhir dipaparkan
2. Dasar Teori & Tinjauan Pustaka, dimana pada bagian ini dijelaskan teori yang menjadi basis pengerjaan tugas akhir, serta referensi-referensi yang digunakan.

3. Metodologi, dimana pada tahap ini dijelaskan alur pengerjaan tugas akhir
4. Perancangan, dimana pada tahap ini dilakukan perancangan model mateimatis yang nantinya akan dipecahkan, beserta metode pembuatan algoritmanya
5. Implementasi, dimana pada tahap ini dilakukan implementasi algoritma yang sebelumnya sudah dirancang
6. Hasil dan Pembahasan, dimana pada tahap ini dilakukan uji coba algoritma yang telah dibuat untuk menemukan solusi penjadwalan terbaik per *benchmark instance*. Selanjutnya juga akan dilakukan pembahasan mengenai hasil yang diperoleh beserta bagaimana *parameter tuning* dapat meningkatkan kualitas solusi.
7. Kesimpulan dan Saran, dimana akan dipaparkan kesimpulan yang diperoleh beserta saran.

“Halaman ini sengaja dikosongkan”

BAB IV
PERANCANGAN

Pada bab ini dijelaskan pemahaman terkait *instance-instance* dari permasalahan yang dihadapi, serta desain kelas dan algoritma yang akan dibangun untuk menyelesaikan masalah PASP.

4.1. Pemahaman *Benchmark Instance*

Untuk lebih memahami perilaku *benchmark* instances, dilakukan analisis statistik kemasing-masing *benchmark instance*. Beberapa statistik yan digunakan yakni rata-rata *length-of-stay* pasien, dan tingkat *bed occupancy* per *benchmark instance*. *Instance 1*, *Instance 2*, hingga *Instance 6* memiliki pasien dengan *length-of-stay* sepanjang 0 hari.

Tabel 4.1 Jumlah Pasien per Instance

Instance	Jumlah Pasien	
	Semua LOS	LOS = 0
0	660	0
1	693	41
2	778	23
3	757	49
4	782	36
5	631	44
6	726	41

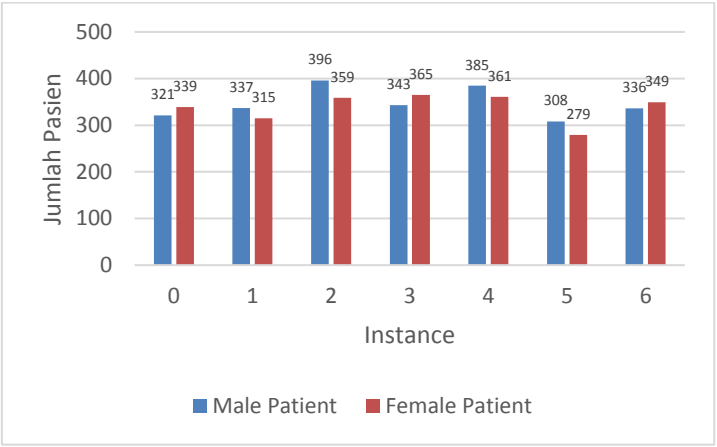
Instance 4 memiliki keseluruhan jumlah pasie tertinggi (782), dengan *instance 0* memiliki keseluruhan jumlah pasien terendah (660). Tanpa memperhitungkan pasien dengan 0 LOS, *instance 2* memiliki jumlah pasien terbanyak (755).

Tabel 4.2 Rata-rata LOS Pasien per Instance

Instance	*Average Patient LOS*	
	Semua LOS	Tanpa LOS = 0
0	5,26	5,26
1	3,79	4,03
2	5,65	5,82
3	4,58	4,90
4	5,04	5,28
5	3,84	4,13
6	4,22	4,47

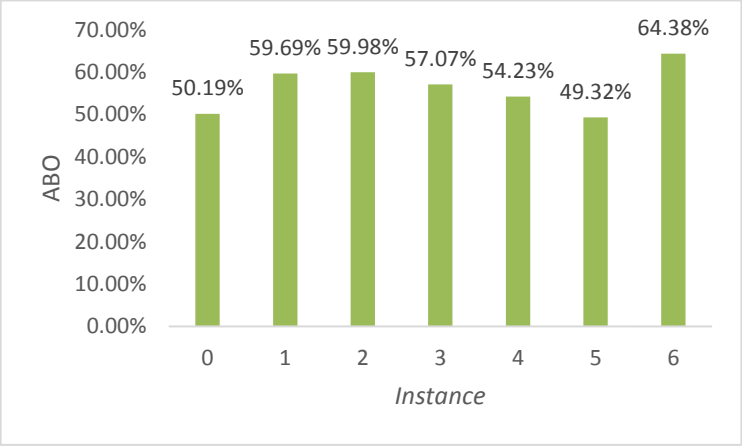
Instance 2 dan *Instance 1* berturut-turut memiliki rata-rata LOS tertinggi dan rata-rata LOS terendah, baik dengan maupun tanpa mempertimbangkan pasien dengan 0 LOS.

Untuk memperhitungkan distribusi jenis kelamin pasien dan rata-rata *bed occupancy*, hanya pasien dengan LOS > 0 yang dipertimbangkan. Pasien dengan LOS = 0 berarti pasien tersebut di-*discharge* di hari itu, sehingga tidak perlu dijadwalkan.



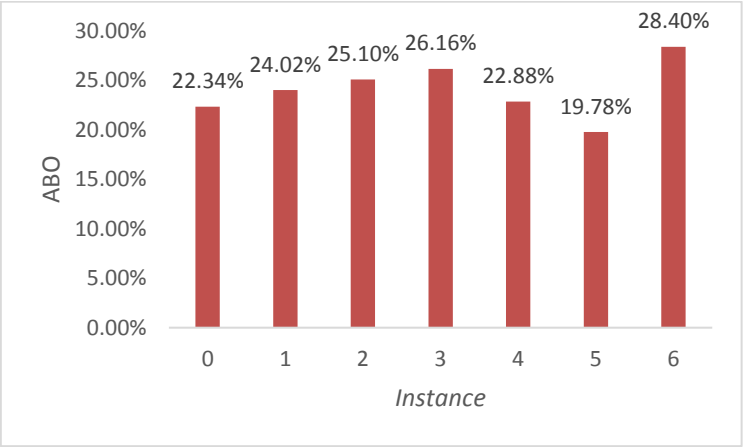
Gambar 4.1 Bar Chart Distribusi Jenis Kelamin Pasien per Instance

Setiap *Instance* memiliki distribusi jenis kelamin yang relatif merata antara pasien laki-laki dengan pasien perempuan. Distribusi tercondong (*skewed*) terdapat pada *instance 5*, dengan persentase distribusi pasien berjenis kelamin laki-laki sebanyak 52,47% (308), dan pasien berjenis kelamin perempuan sebanyak 47,53% (279). Distribusi paling merata terdapat pada *instance 6*, dengan persentase distribusi pasien berjenis kelamin laki-laki sebanyak 49,05% (336) dan pasien berjenis kelamin perempuan sebanyak 50,95% (349).



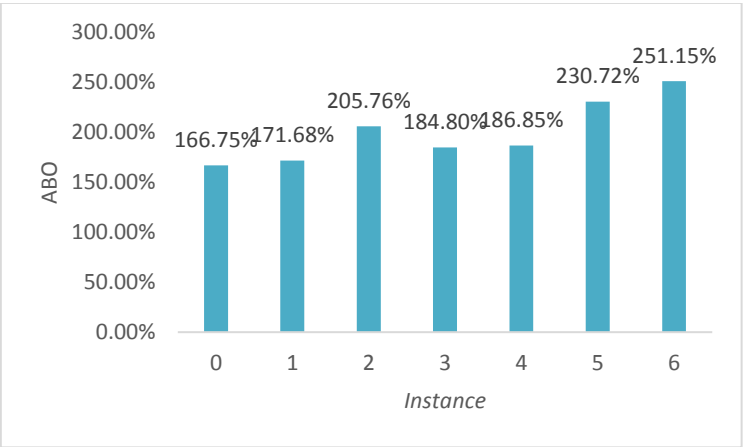
Gambar 4.2 Bar Chart Rata-rata Bed Occupancy (ABO) per Instance

Rata-rata *bed occupancy* didapat dengan memperhitungkan jumlah pasien yang diadmisi dan jumlah kasur yang tersedia di setiap timeslot / hari di dalam *planning horizon*. *Instance* 6 dan *instance 5* berturut-turut memiliki ABO tertinggi (64,38%) dan ABO terendah (49,32%).



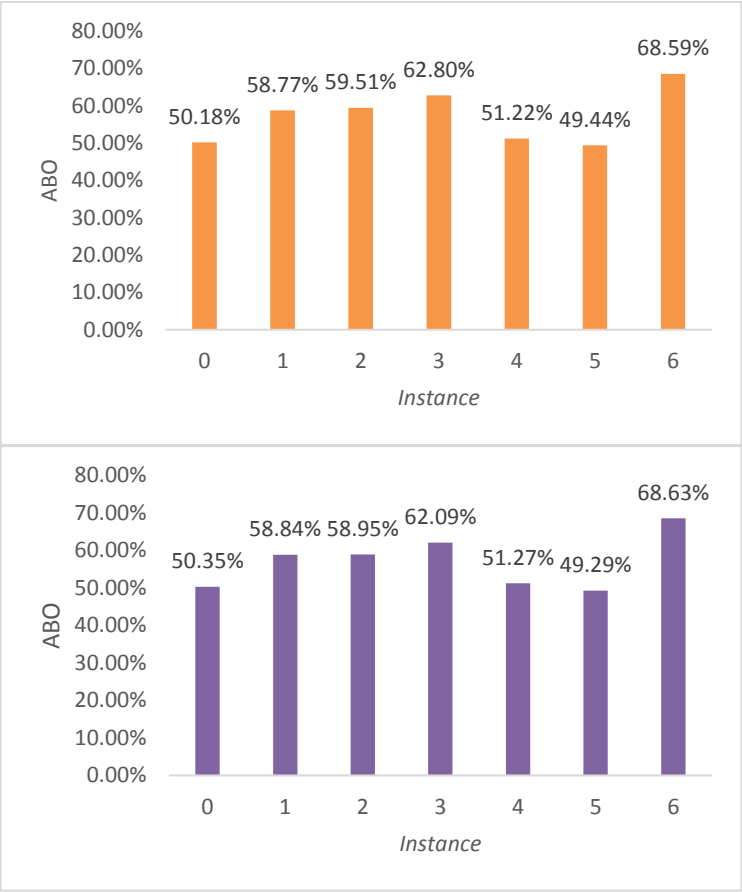
Gambar 4.3 Bar Chart Rata-rata Bed Occupancy per Instance, ditinjau dari Specialism

Dengan meninjau pada jumlah pasien per spesialisme dan jumlah yang mendukung spesialisme tersebut sebagai prioritas pertama, ditemukan bahwa *instance 6* dan *instance 5* berturut-turut memiliki ABO tertinggi (28,40%) dan ABO terendah (19,78%).



Gambar 4.4 Bar Chart Rata-rata Bed Occupancy per Instance, ditinjau dari Room Size Pref

Dengan meninjau pada preferensi ruangan pasien, ditemukan bahwa *instance 6* dan *instance 0* berturut-turut memiliki ABO tertinggi (251,15%), dan ABO terendah (166,75%).



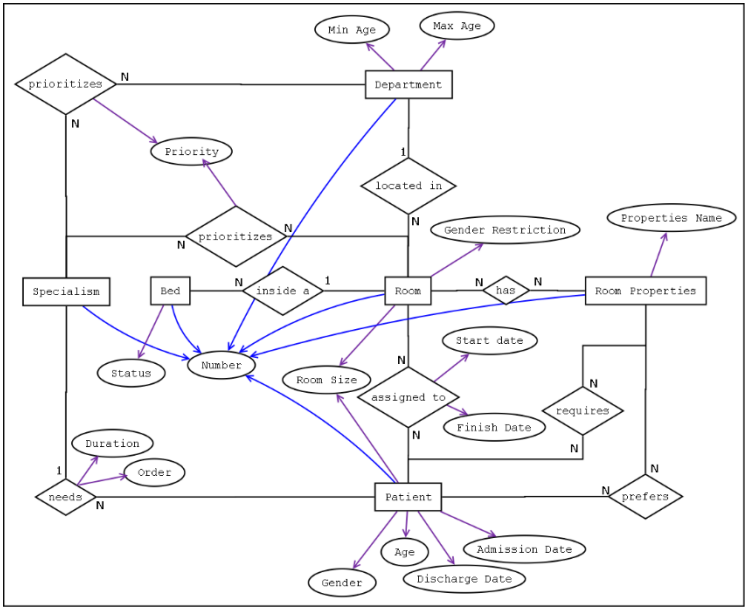
Gambar 4.5 Bar Chart ABO per Instance, ditinjau dari Room Prop Pref (atas), dan ditinjau dari Room Prop Req (bawah)

Ditinjau dari *room properties preferences*, hasilnya tidak berbeda jauh dengan ABO yang ditinjau dari *room properties requirements*. *Instance 6* dan *instance 5* berturut-turut memiliki ABO tertinggi dan ABO terendah.Berdasarkan visualisasi di

atas, faktor preferensi ukuran atau tipe ruangan adalah faktor yang paling sulit dipenuhi, dengan nilai ABO minimum 166,75%. Faktor spesialisme adalah faktor yang paling mudah dipenuhi, dengan nilai ABO maksimum 28,40%.

4.2. Desain *Entity Relationship Diagram*

Entity Relationship Diagram (ERD) dalam kasus ini dibuat untuk mempermudah visualisasi hubungan konseptual dan *logical* dari variabel-variabel yang terdapat pada PASP. Entitas yang ada melambangkan kelas yang akan dibuat dan diimplementasikan lewat Java, dan bukan sebagai tabel dalam *database*. ERD kemudian dijadikan basis dalam pembuatan diagram kelas.



Gambar 4.6 Entity Relationship Diagram PASP

Terdapat 6 entitas yakni *Room, Patient, Room properties, Department, Bed,*dan *Specialism*. Entitas dilambangkan dengan

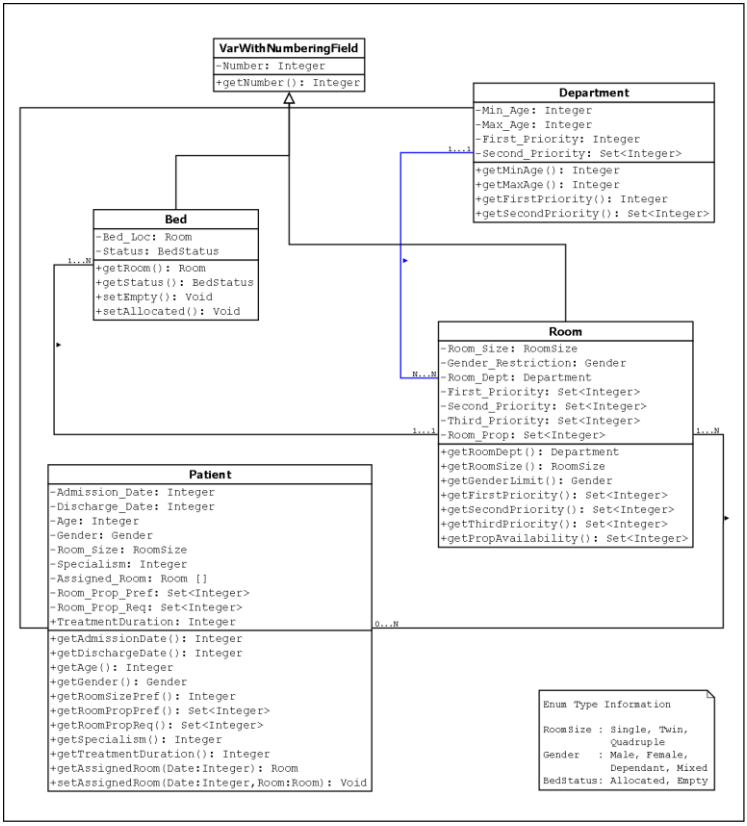
persegi panjang. Selanjutnya, setiap entitas memiliki minimal 1 atribut, yang dilambangkan dengan bentuk oval. Bentuk bujur sangkar melambangkan relasi yang yang menghubungkan masing-masing entitas.

- Departemen memprioritaskan spesialisme (N:N)
- Kasur terdapat dalam ruangan (N:1)
- Ruangan berlokasi di departemen (N:1)
- Ruangan memiliki properti (N: N)
- Ruangan memprioritaskan spesialisme (N:N)
- Pasien ditempatkan di ruangan (N:N)
- Pasien mempunyai preferensi properti (N:N)
- Pasien membutuhkan properti (N:N)
- Pasien membutuhkan spesialisme (N:1)

Atribut *number* dimiliki oleh semua entitas. Atribut *room size* dimiliki oleh pasien yang menandakan sebagai preferensi pasien, dan juga dimiliki oleh ruangan. Dalam admisinya, pasien dapat ditempatkan di ruangan yang berbeda-beda pada waktu yang berbeda. Oleh karena itu, terdapat atribut *start date* dan *finish date* pada relasi pasien-ruangan. Hal yang sama juga berlaku pada kebutuhan spesialisme pasien. Departemen dan ruangan memiliki prioritas spesialisme yang beragam. Oleh karena itu, relasi *prioritizes* pada departemen-spesialisme dan pada ruangan-spesialisme memiliki atribut *priority* yang menandakan prioritas ke berapa spesialisme ini di ruangan atau departemen itu. Entitas ruangan dan pasien sama-sama memiliki atribut *gender*, tetapi dalam implementasinya pasien hanya memiliki 2 jenis *gender* (laki-laki, perempuan), sedangkan atribut *gender* pada ruangan yang menandakan jenis kelamin yang dapat diterima di ruangan tersebut memiliki 4 jenis *gender* (laki-laki, perempuan, *dependent* (bergantung ke pasien pertama), campuran). Oleh karena itu, kedua entitas tersebut memiliki atribut *gender* masing-masing.

4.3. Desain *Class Diagram*

Setelah ERD dibuat, tahap selanjutnya adalah mendesain *Class Diagram*. Dalam hal ini, setiap entitas dan atributnya diubah menjadi kelas. Relasi dalam ERD dapat diubah menjadi kelas baru, atau dilebur ke dalam kelas-kelas yang terhubung.



Gambar 4.7 Class Diagram PASP

Beberapa aspek desain yang berubah dalam transisi ERD ke *Class Diagram* dapat dilihat di tabel berikut.

Tabel 4.3 Perbandingan Desain ERD dengan Desain Class Diagram

Desain ERD	Desain *Class Diagram*
Atribut *Number*	
Atribut *number* dimiliki oleh semua entitas pada ERD.	Atribut *Number* diubah menjadi variabel *number* dan kelas generalisasi dibuat. Semua kelas lain mewarisi kelas generalisasi tersebut.
Relasi *Patient-Specialism*	
Relasi dilambangkan dengan *needs*, dimana terdapat atribut *order* dan *duration*	PASP mengimplementasikan satu kebutuhan *specialism*, sehingga hanya atribut *duration* yang disimpan. Relasi ini dilebur ke dalam kelas *Patient*, dimana terdapat variabel *specialism* dan *treatment_duration*.
Relasi *Department-Specialism*	
Relasi dilambangkan dengan *prioritizes*, dimana terdapat atribut *priority*	Relasi dilebur ke dalam kelas *department*, dan dibagi menjadi variabel *first_priority* yang berisi satu spesialisme utama, dan variabel *second_priority* yang berisi spesialisme-spesialime prioritas kedua.
Relasi *Room-Specialism*	
Relasi dilambangkan dengan *prioritizes*, dimana terdapat atribut *priority*	Relasi dilebur ke dalam kelas *room*, dan dibagi menjadi variabel *first_priority*, *second_priority*, dan *third_priority*. Setiap kategori prioritas dapat memiliki nol atau lebih spesialisme.
Relasi *Room-Room Properties*	
Relasi dilambangkan dengan *has*.	Relasi dilebur ke dalam kelas *Room*, dimana variabelnya disimpan dalam bentuk Set. Set tersebut berisi *room properties*

	yang di dukung ruangan tersebut.
Relasi *Patient-Room Properties*	
Relasi dilambangkan dengan *prefers* dan *requires*, sesuai dengan kebutuhan dan preferensi pasien	Relasi dilebur ke dalam kelas *Patient*, dimana terdapat Set *room_prop_req* dan *room_prop_pref* yang berisi kebutuhan dan preferensi properti pasien.
Relasi *Bed-Room*	
Relasi dilambangkan dengan *inside a*.	Relasi dilebur ke dalam kelas *Bed*, dengan variabel *bed_loc* menyimpan *room* dimana *bed* tersebut berada.
Relasi *Patient-Room*	
Relasi dilambangkan dengan *assigned to*, yang memiliki atribut *start_date* dan *finish_date*.	Relasi dilebur ke dalam kelas *Patient*, dimana variabel *assigned_room* berupa *array room* yang memiliki panjang sesuai dengan LOS pasien (tanggal *discharge* – tanggal admisi)
Relasi *Room-Department*	
Relasi dilambangkan dengan *located in*.	Relasi dilebur ke dalam kelas *Room*, dimana variabel *room_dept* menyimpan departemen tempat ruangan tersebut berada.

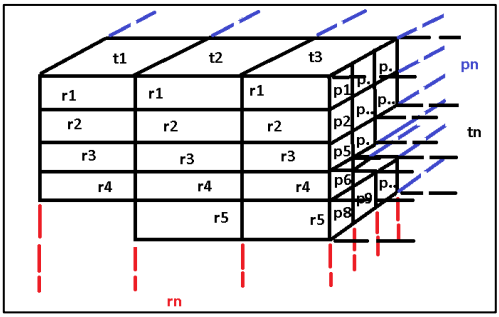
Entitas *Room Properties* dan *Specialism* pun tidak dikonversi menjadi kelas, karena sudah dilebur ke dalam kelas lain. Sesuai dengan prinsip enkapsulasi, semua variabel di setiap kelas memiliki aksesibilitas *private*, dengan metode *getter* dan *setter* yang digunakan untuk mengakses variabel tersebut dari kelas lain. Walaupun demikian, kebanyakan variabel sifatnya *final*, sehingga tidak memiliki metode *setter*. Beberapa relasi antara kelas memiliki *participation* dan *multiplicity* yakni:

- Satu *Room* memiliki nol hingga banyak *Patient*. Satu *Patient* memiliki minimal satu hingga banyak *Room*.
- Satu *Room* memiliki minimal satu hingga banyak *Bed*. Satu *Bed* memiliki satu dan hanya satu *Room.*
- Satu *Room* memiliki satu dan hanya satu *Department*. Satu *Department* memiliki minimal satu hingga banyak *Room*.

Desain *Class Diagram* ini digunakan menjadi basis dalam mengimplementasikan PASP dalam *Java* yang sifatnya memang *object-oriented*.

4.4. Desain Objek Penyimpan Solusi

Solusi yang akan dibuat menggunaan pendekatan ruangan-pasien, dan bukan kasur-pasien. Hal ini dikarenakan setiap ruangan memiliki atribut seperi ukuran ruangan, properti yang tersedia, dan spesialisme yang diprioritaskan. Semua atribut tersebut akan dimiliki oleh setiap kasur yang ada dalam ruangan itu. Dengan kata lain, setiap kasur dalam satu ruangan sifatnya adalah identik satu sama lain. Oleh karena itu, pencarian solusi dilakukan di level pasangan ruangan-pasien. Nantinya, pemilihan kasur untuk pasien dapat dilakukan secara acak.



Gambar 4.8 Ilustrasi Objek yang Menyimpan Solusi

Sebuah blok solusi memiliki panjang sesuai dengan *planning horizon T*. Tinggi dari blok solusi sesuai dengan jumlah ruangan terbanyak yang terisi pasien dalam timeslot tertentu. Lebar dari

blok solusi sesuai dengan jumlah pasien terbanyak yang ditempatkan di ruangan tertentu. Pendekatan yang dilakukan dalam hal ini adalah pendekatan dinamis, dimana lebar dan tinggi blok solusi disesuaikan dengan secara berurutan jumlah pasien terbanyak dan jumlah ruangan terbanyak. Pendekatan lain yang dapat digunakan adalah pendekatan statik, dimana lebar blok solusi mengikuti jumlah total pasien pada *instance* yang digunakan, dan tinggi blok solusi mengikuti jumlah total ruangan yang dapat diisi pada *instance* yang digunakan. Pendekatan blok solusi berukuran dinamis tentunya akan menghemat jumlah iterasi, walaupun memang disisi lain implementasinya mungkin akan jadi lebih sulit.

4.5. Desain Algoritma Pembangkit Solusi Awal

Untuk membangun solusi awal, digunakan algoritma berbasis pada heuristik *best-fit*. Dalam hal ini, kamar atau ruangan yang dialokasikan ke pasien adalah kamar yang paling sesuai dengan kebutuhan pasien tersebut.

```
1   define unassigned_patient
2   define currently_assigned_patient
3   define newly_assigned_patient
4   define newly_discharged_patient
5   define available_rooms
6
7   for each timeslot in planning horizon
8           if timeslot > 0
9                   for each patient in currently_assigned_patient
10                          if patient's discharge date is the current timeslot
11                                  increase patient's room capacity
12                                  if patient's room capacity == 1
13                                          add patient's room to available_rooms
14                                  add patient to newly_discharged_patient
15                          else
16                                  add patient to the same room in the current timeslot
17                  remove all patients in newly_discharged_patient from currently_assigned_patient
18                  clear newly_discharged_patient
19          for each patient in unassigned_patient
20                  if patient's admission date is the current timeslot
21                          sort available_rooms according to patient's needs & preference
22                          assign the first room in the sorted available_rooms to patient
23                          add patient to newly_assigned_patient
24                          if the room's gender limit is dependant
25                                  set the room's gender limit according to patient's gender
26                          decrease the room's capacity
27                          if the room capacity == 0
28                                  remove the room from available_rooms
29          remove all patients in newly_assigned_patient from unassigned_patient
30          add all patients in newly_assigned_patient to currently_assigned_patient
31          clear newly_assigned_patient
```

Kode 4.1 Pseudocode Heuristik Best-Fit untuk PASP

Baris ke-8 hingga ke-18 digunakan untuk mengecek tanggal *discharge* dari setiap pasien yang sedang dalam admisi. Jikalau pasien di-*discharge* pada hari ini, kapasitas ruangan pasien tersebut akan ditambahkan. Jikalau tidak, pasien dijadwalkan di ruangan yang sama pada hari ini. Setiap pasien yang di-*discharge* hari ini kemudian akan dihapus dari data pasien yang sedang dalam admisi. Baris ke-19 hingga ke-31 digunakan untuk menjadwalkan pasien yang belum dijadwalkan. Jikalau pasien diadmisi di hari ini, maka pasien akan dijadwalkan ke ruangan pertama dari daftar ruangan yang tersedia yang sebelumnya sudah disortir mengikuti kebutuhan dan preferensi pasien. Jikalau ruangan tersebut memiliki batasan jenis kelamin *dependant*, maka batasan jenis kelamin tersebut akan diganti sesuai dengan pasien yang dijadwalkan sebelumnya. Selanjutnya, jikalau ruangan tersebut penuh, ruangan akan

dihapus dari daftar ruangan yang tersedia. Semua pasien yang dijadwalkan di hari ini kemudian dihapus dari daftar pasien yang belum dijadwalkan, dan ditambahkan ke daftar pasien yang sedang dalam admisi. Desain ini memastikan bahwa pasien hanya dijadwalkan ke satu ruangan di setiap periode admisinya, pasien hanya dijadwalkan di sepanjang periode admisinya, dan pasien tidak dijadwalkan ke ruangan yang sudah penuh.

Penyortiran daftar ruangan dilakukan dengan meninjau pada kebutuhan properti ruangan pasien, preferensi properti ruangan pasien, jenis kelamin pasien, umur pasien, kebutuhan spesialisme pasien, dan preferensi ukuran ruangan pasien.

Urutan prioritas untuk setiap kategori sebagai berikut:

- Sortir jenis kelamin (*gender*)
  1) *Room* dengan *gender DEPENDENT*
  2) *Room* dengan *gender* berlawanan dengan *gender Patient*
- Sortir properti ruangan (*room properties*)
  1) *Room* yang memiliki semua *room properties* yang dibutuhkan atau diinginkan *Patient*
  2) *Room* yang memiliki beberapa (sebagian besar) *room properties* yang dibutuhkan atau diinginkan *Patient*
  3) *Room* yang memiliki beberapa (sebagian kecil) *room properties* yang dibutuhkan atau diinginkan *Patient*
  4) *Room* yang tidak memiliki semua *room properties* yang dibutuhkan atau diinginkan *Patient*
- Sortir preferensi ukuran ruangan (*room size*)
  1) *Room* yang memiliki *room size* sesuai dengan r*oom size* yang diinginkan *Patient*
  2) *Room* yang memiliki *room size* tidak sesuai dengan *room size* yang diinginkan *Patient*
- Sortir spesialisme (*specialism*) untuk *Room*

1) *Room* dengan *specialism* yang dibutuhkan *Patient* berada pada prioritas pertama
2) *Room* dengan *specialism* yang dibutuhkan *Patient* berada pada prioritas kedua
3) *Room* dengan *specialism* yang dbutuhkan *Patient* berada pada prioritas ketiga
4) *Room* yang tidak memprioritaskan *specialism* yang dibutuhkan *Patient*

- Sortir spesialisme (*specialism*) untuk *Room Department*
  1) *Room* dengan *Department* dimana *specialism* yang dibutuhkan *Patient* berada pada prioritas pertama
  2) *Room* dengan *Department* dimana *specialism* yang dibutuhkan *Patient* berada pada prioritas kedua
  3) *Room* dengan *Department* yang tidak memprioritaskan *specialism* yang dibutuhkan *Patient*
- Sortir umur (*age)*
  1) *Room* dengan *Department* dimana *age* yang dimiliki *Patient* termasuk dalam jangkauan umur yang diterima
  2) *Room* dengan *Department* dimana *age* yang dimiliki *Patient* diluar jangkauan umur yang diterima

Urutan kategori mana yang akan disortir terlebih dahulu hingga kategori mana yang disortir terakhir adalah:

1) Umur pasien
2) Kebutuhan spesialisme pasien (ruangan)
3) Kebutuhan properti ruangan pasien
4) Jenis kelamin pasien
5) Preferensi properti ruangan pasien
6) Kebutuhan spesialisme pasien (departemen)
7) Preferensi ukuran ruangan pasien

Urutan ini dapat berubah di implementasi, tergantung dari hasil yang diperoleh.

4.6. Evaluasi *Hard Constraint* dan *Fitness Function*

Perhitungan *fitness function* dilakukan untuk mengukur kualitas solusi yang diperoleh. Desain algoritma optimasi dan algoritma pembangkit solusi awal akan selalu menghasilkan solusi yang memenuhi *hard constraint*. Hal ini dilakukan dengan memastikan bahwa penjadwalan pasien dilakukan mengikuti tanggal admisi dan tanggal *discharge* pasien. Selain itu, setiap ruangan dialokasikan mengikuti kapasitas ruangan tersebut. Begitu juga dengan desain alokasi kasur – pasien yang juga dapat dilakukan hanya ketika kasur memiliki status kosong (*empty*). Khusus untuk penalti spesialisme ruangan pasien, terdapat *multiplier* yang disesuaikan dengan prioritas spesialisme ruangan tersebut. Artinya, penalti akan dikali 1x jika spesialisme pasien merupakan prioritas kedua ruangan, dan dikali 2x jika spesialisme pasien berada pada prioritas ketiga atau tidak diprioritaskan sama sekali oleh ruangan tersebut.

```
1   define starting penalty = 0 for each penalty type
2   define multiplier = 0
3
4   for each timeslot in the solution block
5           for each room inside the timeslot
6                   for each patient inside the room
7                           if patient's room size preference < actual room size
8                                   room size penalty++
9                           for each patient's room properties preference
10                                  if room properties is not available inside the room
11                                          room prop pref penalty++
12                          for each patient's room properties requirement
13                                  if room properties is not available inside the room
14                                          room prop req penalty++
15                          if patient's specialism is not prioritized by the room's department
16                                   room dept spec penalty++
17                          if patient's admission date < current timeslot
18                                  if patient is not assigned to the same room in the previous timeslot
19                                          transfer penalty++
20                          if patient's specialism is not the room's first priority
21                                  multiplier++
22                                  if patient's specialism is not the room's second priority
23                                          multiplier++
24                                  room spec penalty += multiplier
25                                  multiplier = 0
26                          if room's gender != MIXED
27                                  if patient's gender != room's gender
28                                          gender penalty++
29                          if patient's age < room's department min age
30                                  if patient's age > room's department max age
31                                          age penalty++
32  multiply each penalty with its weight
33  sum the penalty
```

Kode 4.2 Pseudocode Perhitungan Penalti atau Fitness Function

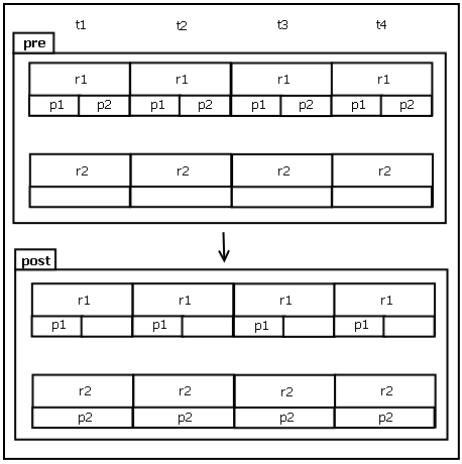
Setiap hasil kategori penalti yang sudah dikalikan dengan bobotnya kemudian akan ditambahkan untuk memperoleh penalti total.

4.7. Desain Algoritma Optimasi

Dalam penelitian ini, digunakan 2 algoritma optimasi yang berbeda. Perbedaannya hanya terletak pada bagaimana pemilihan *neighborhood* di setiap iterasi dilakukan. Algoritma pertama menggunakan perpindahan *neighborhood* yang mengacu pada hirarki (VNS). Algoritma kedua menggunakan perpindahan *neighborhood* secara acak.

4.7.1. *Neighborhood*

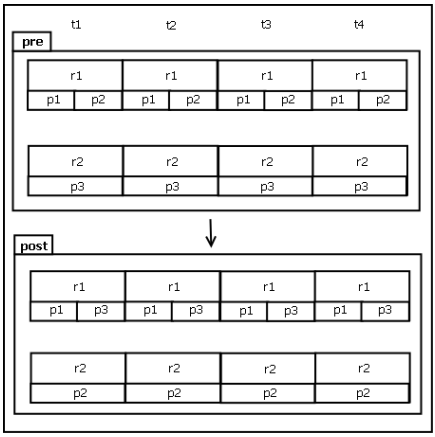
Desain awal menggunakan empat *neighborhood*, yakni *complete move*, *complete swap*, *partial move*, dan *partial swap*.



Gambar 4.9 Ilustrasi Complete Move

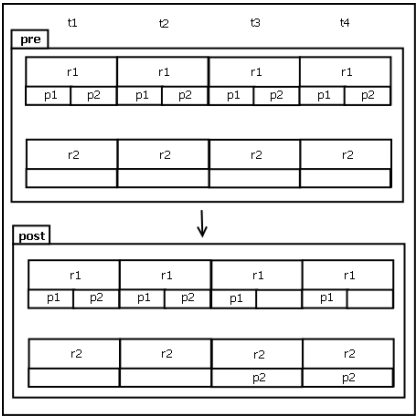
Pasien p_2 yang dijadwalkan di ruangan r_1 selama periode admisinya, dipindahkan ke ruangan r_2 secara penuh. Ini merupakan *complete move*, dimana perpindahan pasien dilakukan ke ruangan lain yang tersedia sepanjang waktu admisi pasien tersebut.

Pada *complete swap*, perpindahan dilakukan dengan menukar alokasi ruangan dua pasien yang berbeda. Pada Gambar 4.10, pasien p_2 di ruangan r_1 dipindahkan ke ruangan r_2, dan pasien p_3 di ruangan r_2 dipindahkan ke ruangan r_1. Perpindahan yang dilakukan tidak harus simetris, atau dengan kata lain kedua pasien memiliki tanggal admisi dan tanggal *discharge* yang persis sama. Perpindahan dapat dilakukan secara asimetris, dengan syarat ruangan yang dituju harus tidak penuh. Sebagai contoh, jikalau pasien p_3 pada Gambar 4.10 dijadwalkan di ruangan r_2 dari timeslot t_1 hingga t_5, maka *complete swap* tidak dapat dilakukan jika pada timeslot t_5 ruangan tersebut penuh.



Gambar 4.10 Ilustrasi Complete Swap

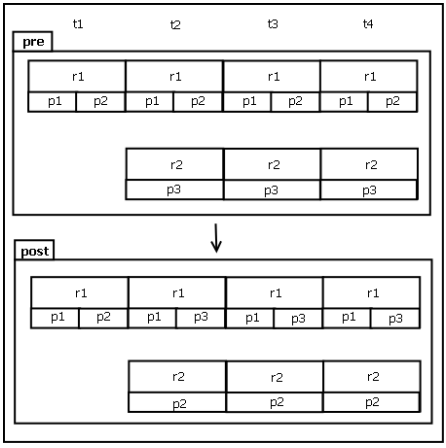
Pada *partial move*, pasien dipindahkan ke ruangan lain hanya untuk sebagian periode admisinya. Perpindahan ini tentunya akan menimbulkan *transfer cost*, namun tetap berpeluang menurunkan nilai *fitness function* jika memang alokasi di ruangan sebelumnya menghasilkan penalti yang tinggi. Pada Gambar 4.11, pasien p_2 dipindahkan dari ruangan r_1 ke ruangan r_2 hanya pada periode t_3 hingga t_4.



Gambar 4.11 Ilustrasi Partial Move

Neighborhood ini memiliki peluang untuk melakukan *complete move*, karena pemilihan tanggal transfer yang sifatnya diacak. Dalam kasus ini, tanggal transfer nilainya sama dengan tanggal admisi pasien.

Pada *partial swap*, dua pasien akan ditukar alokasi ruangannya selama periode tertentu. Pasien pertama yang dipilih akan selalu dipindahkan ke ruangan pasien kedua dari suatu titik diantara periode admisi pasien tersebut, sampai ke akhir periode admisinya. Sedangkan untuk pasien kedua, dipindahkan ke ruangan pasien pertama mulai dari awal periode admisi, hingga ke titik tertentu. Pada Gambar 4.12, pasien r_2 pada periode t_2 hingga t_4 dipindahkan dari ruangan r_1 ke ruangan r_2. Pasien r_3 pada periode yang sama, dipindahkan dari ruangan r_2 ke ruangan r_1. Dalam kasus dimana tanggal *discharge* pasien r_3 lebih besar daripada tanggal *discharge* pasien r_2, maka pada periode t selanjutnya pasien r_3 akan dikembalikan ke ruangan r_2. Desain memang sengaja dibatasi seperti pada penjelasan sebelumnya, untuk mempermudah implementasi. Dalam hal ini, pasien pertama selalu dipindahkan dari satu titik timeslot hingga ke periode akhir admisi, dan pasien kedua selalu dipindahkan dari periode awal admisi hingga ke titik timeslot tertentu.



Gambar 4.12 Ilustrasi Partial Swap

Ada kemungkinan bahwa tanggal *discharge* pasien kedua sama dengan tanggal *discharge* pasien pertama, atau tanggal admisi pasien pertama sama dengan tanggal admisi pasien kedua. Dalam kasus ini, *transfer cost* hanya akan ditimbulkan satu kali. Kemungkinan lain yang dapat terjadi yakni dilakukannya *complete move*, ketika pasien kedua memiliki tanggal admisi dan tanggal *discharge* yang sama dengan pasien pertama.

4.7.2. Algoritma *VNS* dan *Random Selection*

Algoritma VNS yang akan digunakan pada penelitian ini adalah varian *reduced VNS*, dimana algoritma ini tidak melakukan pencarian deterministik. Tetapi, pencarian dilakukan secara stokastik lewat fungsi *shaking*.

Algoritma *Local Search* memilih *neighborhood* secara acak di setiap iterasinya. Fungsi *shake* pada algoritma VNS juga digunakan dalam algoritma ini. Perbedaannya hanya terletak pada *neighborhood* yang dipilih tidak berdasarkan pada urutan atau hirarki tertentu, melainkan dipilih secara acak. Penggunaan *if-statement* pad baris ke-10 hingga 11 sama dengan baris ke-1 dan 2 pada fungsi *NeighborhoodChange* di bab 2.1.5.

```
1   define compute_time
2   define neighbor_pool
3
4   max_time = time + compute_time
5   x = initial solution
6
7   while time < max_time
8           neighbor= randomly get neighborfrom neighbor_pool
9           x' = shake(x, neighbor)
10          if f(x') < f(x)
11                  x = x'
```

Kode 4.3 Pseudocode Pencarian dengan Random Neighborhood Selection

Evaluasi dilakukan di setiap akhir pergerakan, dimana pengecekan dilakukan dengan prinsip *forward-checking*. Evaluasi dilakukan sebelum pergerakan dilakukan, kemudian evaluasi kedua dilakukan setelah pergerakan dilakukan. Jikalau, penalti yang dihasilkan setelah pergerakan lebih baik, maka akan dilakukan *backtracking*.

4.7.3. Parameter dan *Stopping Mechnanism*

Algoritma Optimasi yang digunakan memerlukan parameter waktu komputasi serta *neighborhood* yang digunakan. Waktu komputasi ini yang juga menjadi *stopping mechanism*. Dalam penelitian ini, beberapa parameter waktu komputasi yang akan digunakan yakni 30 detik, 400 detik, dan 3600 detik. Penambahan waktu eksekusi atau komputasi ini dimaksudkan untuk mengetahui bagaimana performa algoritma di waktu yang singkat. Selain itu, hal ini juga dilakukan untuk melihat apakah peningkatan waktu eksekusi akan menghasilkan solusi yang lebih baik dan apakah algoritma *stuck* di titik tertentu (*local optimum*). Selain itu, waktu komputasi 4810 detik juga digunakan, dimana waktu tersebut diperoleh menggunakan program *benchmark* dari *2nd International Timetabling Competition* (ITC) dikali 10, seperti pada [6]. Program *benchmark* ini memang sering digunakan di berbagai penelitian optimasi diluar ITC, sehingga hasil yang didapat di satu penelitian dapat dibandingkan secara adil dengan hasil penelitian lain. Program *benchmark* ini akan memberikan waktu eksekusi yang sesuai dengan performa perangkat keras yang digunakan. Namun demikian, seringkali penelitian menggunakan waktu yang lebih rendah karena beberapa alasan, seperti algoritma yang dibuat sudah jauh mengalahkan performa algoritma sebelumnya dengan waktu yang lebih singkat, atau memang algoritma sudah terjebak pada *local optimum*.

BAB V
IMPLEMENTASI

Pada bab ini dijelaskan proses implementasi kelas dan algoritma yang digunakan untuk membangun solusi awal serta algoritma yang digunakan untuk melakukan optimasi solusi. Implementasi dilakukan menggunakan bahasa pemrogaman Java.

5.1. Realisasi *Class Diagram*

Semua kelas diimplementasikan sesuai dengan desainnya. Walaupun demikian, terdapat beberapa variabel tambahan, kelas tambahan, dan juga beberapa tipe data atau objek yang berubah. Kelas baru dibuat hanya untuk menyimpan berbagai *enum* yang dibuat. Desain menggunakan *enum* dalam hal ini digunakan supaya membuat kode lebih *readable* dan mudah diingat. Tanpa menggunakan *enum*, implementasi tentunya dapat menggunakan *int*, yang membutuhkan translasi dari level ke bentuk angka, atau menggunakan *String*.

Beberapa implementasi yang berubah-ubah yakni objek penyimpanan properti ruangan di kelas *Room* dan kelas *Patient*. Implementasi awal menggunakan *List*, dengan posisi digit melambangkan properti ruangan yang mana, dan nilai biner untuk menandakan apakah properti ini tersedia atau tidak untuk konteks ruangan, dan dibutuhkan, diinginkan, atau tidak untuk konteks pasien. Implementasi yang sama memang digunakan pada setiap file *instance*. Namun demikian, hal ini mempersulit implementasi kelas *comparator*, dimana implementasi awal yang memang tidak f leksibel. Jikalau jumlah properti ruangan ditambah, maka terdapat urutan perbandingan baru yang mungkin belum ter-*cover*.

Sebagai contoh, ketika terdapat 2 properti ruangan, maka urutan prioritasnya untuk pasien yang membutuhkan semua properti tersebut yakni:

1. Sepenuhnya terpenuhi (2 properti ruangan)
2. Sebagian terpenuhi (1 properti ruangan)
3. Tidak terpenuhi (0 properti ruangan).

Ketika terdapat 4 properti ruangan, maka urutan prioritasna untuk pasien yang membutuhkan semua properti tersebut menjadi:

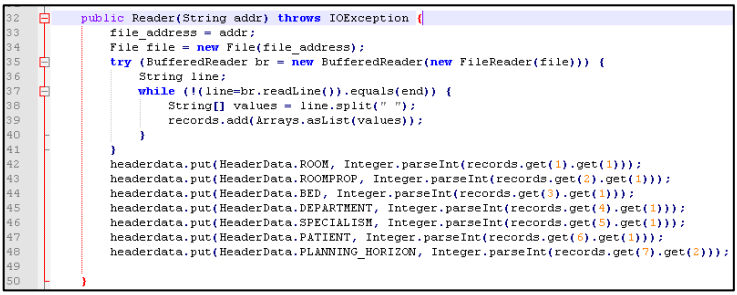
1. Sepenuhnya terpenuhi (4 properti ruangan)
2. Sebagian besar terpenuhi (3 properti ruangan)
3. Sebagian terpenuhi (2 properti ruangan)
4. Sebagian kecil terpenuhi (1 properti ruangan)
5. Tidak terpenuhi (0 properti ruangan)

Untuk itu, implementasi objek penyimpanan properti ruangan diganti menggunakan *Set*. Dalam hal ini, posisi sudah tidak lagi menandakan properti mana yang dituju. Tetapi, properti diidentifikasi langsung menggunakan nilai digit atau angkanya. Setiap *set* hanya menyimpan properti ruangan yang memang tersedia di ruangan itu, atau dibutuhkan pasien. Dengan begitu, ketika melakukan perbandingan ruangan, *comparator* cukup mencari ruangan mana yang memiliki paling banyak properti ruangan yang dibutuhkan pasien.

Beberapa objek baru juga ditambahkan di kelas pasien. Beberapa diantaranya yakni metode *setAssignedRoom()* tanpa parameter *date*, yang artinya ruangan pasien akan diubah sepnuhnya. Metode ini ditambahkan supaya dalam pergerakan yang sifatnya *complete*, *loop* ke setiap timeslot pasien tidak perlu diketik berkali-kali, melainkan cukup di kelas *Patient* saja. Selain itu, ditambahkan *array candidate_room*, beserta metode *getter* dan *setter*-nya. Variabel ini digunakan untuk menyimpan ruangan yang menjadi calon ruangan baru bagi pasien dalam iterasi optimasi.

5.2. Implementasi *Deserializer*

Implementasi *deserializer* dilakukan di kelas *Reader*. Blok kode pada Kode 5.1 digunakan untuk membaca setiap baris data pada *instance*.



```
32      public Reader(String addr) throws IOException {
33          file_address = addr;
34          File file = new File(file_address);
35          try (BufferedReader br = new BufferedReader(new FileReader(file))) {
36              String line;
37              while (!(line=br.readLine()).equals(end)) {
38                  String[] values = line.split(" ");
39                  records.add(Arrays.asList(values));
40              }
41          }
42          headerdata.put(HeaderData.ROOM, Integer.parseInt(records.get(1).get(1)));
43          headerdata.put(HeaderData.ROOMPROP, Integer.parseInt(records.get(2).get(1)));
44          headerdata.put(HeaderData.BED, Integer.parseInt(records.get(3).get(1)));
45          headerdata.put(HeaderData.DEPARTMENT, Integer.parseInt(records.get(4).get(1)));
46          headerdata.put(HeaderData.SPECIALISM, Integer.parseInt(records.get(5).get(1)));
47          headerdata.put(HeaderData.PATIENT, Integer.parseInt(records.get(6).get(1)));
48          headerdata.put(HeaderData.PLANNING_HORIZON, Integer.parseInt(records.get(7).get(2)));
49
50      }
```

Kode 5.1 Blok Kode untuk Membaca Data per Baris

Data dibaca per baris, kemudian setiap data per baris tersebut akan di *split* menggunakan *delimiter* spasi, dan disimpan ke dalam *List* bernama *records*. Data pada *header* di setiap *instance* yang merupakan jumlah nilai per variabel, disimpan ke dalam *EnumMap headerdata*. Data ini digunakan untuk memperoleh baris mana yang mendeskripsikan setiap variabel. Selanjutnya, terdapat metode *deserializeData()* yang fungsinya membuat *array integer* 2 dimensi khusus per variabel. Parameter yang dibutuhkan adalah *enum HeaderData* yang disimpan di kelas *Utility*. *Enum* ini dibuat untuk mengidentifikasi variabel apa yang dituju, yang dalam kasus ini yakni departemen, ruangan, pasien, dan kasur. Parameter tersebut kemudian digunakan untuk mengidentifikasi *key* mana yang akan dipanggil pada *EnumMap headerdata*. Nilai yang akan dikembalikan adalah nilai jumlah baris data yang merupakan variabel tujuan (*key*). Nilai ini digunakan dalam perhitungan untuk memperoleh index atau baris awal setiap data. Nilai awal *index* adalah 13 + jumlah spesialisme.

Tabel 5.1 Index atau Posisi Baris Awal Setiap Variabel

Variabel	Indeks Awal
Departemen	*index*
Ruangan	*index* + 4 + jumlah departemen + jumlah properti ruangan
Kasur	*index* + 6 + jumlah departemen + jumlah properti ruangan + jumlah ruangan
Patien	*index* + 8 + jumlah departemen + jumlah properti ruangan + jumlah ruangan + jumlah kasur

Setiap data tersebut kemudian disimpan ke dalam *array integer* 2 dimensi. Beberapa nilai seperti jenis kelamin dan ukuran ruangan dalam *instance* menggunakan simbol karakter, sehingga perlu ditranslasikan ke dalam bentuk angka terlebih dahulu. Setelah itu, setiap baris dari *array* tersebut akan digunakan sebagai parameter untuk memanggil *constructor* dari kelas objek yang sesuai. Objek yang dibuat akan disimpan di dalam *List*, dengan pengecualian objek *bed* yang disimpan ke dalam *Set*. Hal ini dilakukan karena dalam optimasi dan pembangkitan solusi awal, beberapa objek akan di-*sort*, dan *set* tidak mendukung operasi *sort*.

5.3. Implementasi Objek Penyimpan Solusi

Implementasi blok solusi dilakukan menggunakan implementasi *LinkedHashMap* dari *Map*.

```
Map<Integer, Map<Room, List<Patient>>> solution = new LinkedHashMap<>();
```

Kode 5.2 Deklarasi dan Instansiasi Objek Solusi

Implementasi ini dipilih dikarenakan *Map* dibutuhkan dalam keadaan yang urut dari tanggal atau timeslot pertama hingga ke tanggal atau timeslot terakhir dalam *planning horizon*. Kondisi urut tersebut terutama penting saat melakukan operasi *stream()*. Kondisi urut tersebut juga dapat dicapai dengan menggunakan implementasi *TreeSet*, namun *LinkedHashMap* dipilih karena memang iterasi untuk melakukan *insertion* awal dimulai dari timeslot pertama hingga ke timeslot terakhir. Hal ini tentunya

memiliki konsekuensi dimana untuk mendapatkan nilai *value* dengan menggunakan *key* yakni timeslot yang berada pada posisi tengah, akan memakan waktu yang lebih banyak. Namun demikian, hal tersebut tidak akan memiliki implikasi negatif yang sangat besar mengingat panjang *planning horizon* yang relatif pendek (14 hari atau timeslot dalam kasus ini).

5.4. Implementasi Algoritma Pembangkit Solusi Awal

Implementasi akhir yang dilakukan memiliki beberapa perbedaan bila dibandingkan dengan desain awal yang dibuat. Beberapa hirarki urutan dalam metode *comparator* juga mengalami perubahan setelah program *validator* yang disediakan dipelajari lebih lanjut. Beberapa tambahan yakni penyortiran pasien sebelum algoritma pembangkit solusi awal dijalankan.

```
Collections.sort(unassigned_patient, new Comparator<Patient>() {
    @Override
    public int compare (Patient p1, Patient p2) {
        return p1.getAdmissionDate() - p2.getAdmissionDate();
    }
}
        .thenComparing(new FieldComparator(ComparatorField.PATIENT_ROOMPROP))
        .thenComparing(new FieldComparator(ComparatorField.PATIENT_ROOMSIZE))
);
```

Kode 5.3 Blok Kode untuk Menyortir Daftar Pasien

Kelas *FieldComparator* dibuat dengan mengimplementasikan *Comparator<Object>*. Implementasi dilakukan menggunakan *Objek* yang sifatnya umum, karena kelas ini akan digunakan untuk melakukan komparasi untuk objek *Room* dan juga untuk objek *Patient*. Selain itu melakukan *overriding* secara langsung di *compare()* berarti perbandingan hanya dapat melibatkan dua objek yang dibandingkan tersebut. Perbandingan antar objek *Room* di saat solusi awal dibuat, juga membutuhkan nilai yang dimiliki oleh objek *Patient*. Untuk dapat mengirimkan nilai tersebut, maka dibuat kelas *FieldComparator*, dengan nilai yang berada di objek *Patient* dijadikan parameter di *constructor*. Beberapa komparasi yang mengalami perubahan seperti pada implementasi *validator* yakni komparasi

departemen ruangan, dimana perbedaan antara prioritas pertama dan prioritas kedua dihilangkan. Selain itu, komparasi ukuran ruangan juga berubah, dimana urutan prioritasnya menjadi:

1. Ruangan yang memiliki ukuran sesuai preferensi pasien
2. Ruangan yang memiliki ukuran lebih kecil daripada preferensi pasien (diutamakan ruangan dengan ukuran paling dekat dengan preferensi ukuran ruangan pasien)
3. Ruangan yang memiliki ukuran lebih besar daripada preferensi pasien (diutamakan ruangan dengan ukuran paling dekat dengan preferensi ukuran ruangan pasien).

Untuk membedakan komparasi objek *Patient* dengan objek *Room*, digunakan fungsi *instanceof* untuk mengecek apakah objek adalah *instance* dari *Patient* atau *Room*. Selanjutnya, pembangunan solusi awal dilakukan menggunakan *stream()* untuk menggantikan iterasi tradisional.

```
Collections.sort(available_rooms, new FieldComparator(ComparatorField.DEPT_AGE, patient.getAge())
        .thenComparing(new FieldComparator(ComparatorField.ROOM_PRIORITY, patient.getSpecialism()))
        .thenComparing(new FieldComparator(ComparatorField.ROOM_PROP, patient.getRoomPropReq()))
        .thenComparing(new FieldComparator(ComparatorField.ROOM_GENDER, patient.getGender(), current_date))
        .thenComparing(new FieldComparator(ComparatorField.ROOM_PROP, patient.getRoomPropPref()))
        .thenComparing(new FieldComparator(ComparatorField.DEPT_SPEC, patient.getSpecialism()))
        .thenComparing(new FieldComparator(ComparatorField.ROOM_SIZE, patient.getRoomSizePref()))
);
```

Kode 5.4 Blok Kode Penyortiran Ruangan

Urutan komparasi yang digunakan sama dengan desain. Urutan ini memang ditentukan secara arbitrari, dan beberapa percobaan menggunakan urutan yang berbeda tidak menghasilkan perubahan penalti yang signifikan. Dua kandidat ruangan dipilih dengan ruangan pertama adalah ruangan yang sudah diisi oleh pasien dan ruangan kedua yang belom diisi pasien sama sekali. Jikalau ruangan kedua memenuhi lebih banyak preferensi dan kebutuhan pasien, maka ruangan kedua yang akan dipilih. Hal ini dilakukan supaya ruangan yang sudah diisi pasien diutamakan terlebih dahulu. Hasil penjadwalan akan disimpan di dalam blok solusi yang sudah dibuat.

5.5. Implementasi *Serializer*

Implementasi kode untuk men-*write* solusi ke file dilakukan di kelas *Writer*. Implementasi ini dilakukan hanya untuk melakukan cek hasil menggunakan program *validator* yang disediakan. Format yang digunakan untuk program *validator* adalah *patient-bed*, sehingga solusi yang sudah dibuat harus dikonversi terlebih dahulu dari format *date-room-patient*.

Metode *convertRoomToBedAssignment()* diawali dengan melakukan iterasi ke setiap timeslot, ruangan, dan pasien pada blok solusi. Hasil konversi disimpan di dalam objek *TreeMap* yang akan mengurutkan *map* berdasarkan *key* yakni *Patient*. Hal ini dilakukan karena iterasi yang dilakukan di blok solusi tidak akan menjamin *insertion* pasien yang urut ke dalam blok solusi yang baru, dan urutan tersebut diperlukan dalam file untuk mengidentifikasi pasien yang dituju.

```
Map<Patient, List<Bed>> bed_alloc = new TreeMap<>(new Comparator<Patient>() {
    @Override
    public int compare(Patient p1, Patient p2) {
        return p1.getNumber()-p2.getNumber();
    }
});
```

Kode 5.5 Deklarasi dan Instansiasi Objek Solusi dengan Format Patient-Bed

Selanjutnya, iterasi dilakukan untuk memulai konversi dari format *date-room-patient* ke format *patient-bed*.

```
if (day.getKey()>0 && solution.get(day.getKey()-1).containsKey(room.getKey())
        && solution.get(day.getKey()-1).get(room.getKey()).contains(patient)) {
    Bed yesterday_bed = bed_alloc.get(patient).get(bed_alloc.get(patient).size()-1);
    bed_alloc.get(patient).add(yesterday_bed);
}
else {
    Bed bed = room_beds.get(room.getKey())
                .stream()
                .filter(b -> b.getStatus() == BedStatus.EMPTY)
                .findFirst()
                .get();
    if (!bed_alloc.containsKey(patient)) {
        List <Bed> temp = new ArrayList<>();
        temp.add(bed);
        bed_alloc.put(patient, temp);
    }
    else
        bed_alloc.get(patient).add(bed);

    if (patient.getDischargeDate()>0)
        bed.setAllocated();
}
```

Kode 5.6 Penggalan Kode untuk Konversi Format

Kode diatas digunakan dalam iterasi untuk membuat peta solusi baru dengan objek *patient* sebagai *key* dan *value* berupa *list* yang berisi ruangan-ruangan pasien tersebut. Blok *if* pertama dilakukan untuk menyalin ruangan jikalau pasien sudah pernah diadmisi tidak berpindah ruangan dari timeslot atau tanggal sebelumnya. Jikalau tidak, maka akan dicari kasur pertama yang tidak terisi pasien di ruangan tersebut. Pencarian tersebut dilakukan menggunakan peta *room_beds*. Walaupun objek *bed* sudah menyimpan ruangan terkait, dibuat peta *room_beds* sehingga pencarian cukup menggunakan *key* berupa ruangan dan tidak melakukan iterasi ke seluruh kasur yang ada.

```
bed_alloc.entrySet().forEach(r -> {

    List<Bed> lb = r.getValue();
    int len = lb.size();
    Bed last_bed = lb.get(len-1);
    if (r.getKey().getDischargeDate() >= day.getKey()) {
        if(r.getKey().getDischargeDate() == day.getKey() && r.getKey().getDischargeDate()>0) {
            last_bed.setEmpty();
        }
        else if (!solution.get(day.getKey()).containsKey(last_bed.getRoom())
                || !solution.get(day.getKey()).get(last_bed.getRoom()).contains(r.getKey())) {
            last_bed.setEmpty();
        }
    }

});
```

Kode 5.7 Penggalan Kode untuk Mengecek Tanggal Disharge Pasien

Kode diatas digunakan untuk mengecek apakah pasien di-*discharge* hari ini, atau apakah pasien ditransfer ke ruangan lain. Jikalau ya, maka objek *bed* yang terhubung akan dijadikan

empty statusnya. Setelah dikonversi, maka blok solusi akan di-*write* ke dalam file teks yang diberikan format arbitrari .sol.

```
File file = new File(this.file_address);
if (!file.createNewFile())
        new PrintWriter(file).close();

BufferedWriter bw = new BufferedWriter(new FileWriter(file));
try {
    room_alloc.entrySet()
       .forEach(patient -> {
           String pat = String.valueOf(patient.getKey().getNumber()).concat(" ");
           String rlist = patient.getValue().stream()
                   .map(bed -> String.valueOf(bed.getNumber()))
                   .collect(Collectors.joining(" "));
           try {
               bw.write(pat.concat(rlist));
               bw.newLine();
           }
           catch (IOException ioe) {
               ioe.printStackTrace();
           }
       });
}
finally {
    bw.close();
}
```

Kode 5.8 Blok Kode untuk Menyimpan Solusi ke Dalam Teks

Jikalau file dengan nama yang sama sudah ada, maka file tersebut akan dihapus terlebih dahulu. Selanjutnya, dilakukan iterasi ke setiap pasien dalam blok solusi. Setiap daftar ruangan yang ada digabungkan dengan tanda spasi sebagai *delimiter*, kemudian di-*write* menjadi baris baru. Dalam format ini, pasien diidentifikasi dengan posisi barisnya, sehingga memang dalam metode konversi ke format ini sudah dilakukan penyortiran berdasarkan nomor pasien.

5.6. Implementasi Validator

Implementasi *validator* dilakukan di kelas *Validator*. Fungsi *calculateCost()* dibuat untuk menghitung nilai penalti atau *fitness function* dari solusi yang diperoleh. Iterasi dilakukan ke setiap pasien di setiap ruangan dan di setiap timeslot pada blok solusi.

```
if (patient.getRoomSizePref().ordinal() < room.getKey().getRoomSize().ordinal()) {
    room_size_pref_cost++;
}
```

Kode 5.9 Pengecekan Penalti Ukuran Ruangan

Pengecekan pertama dilakukan ke biaya atau penalti pelanggaran preferensi ukuran ruangan pasien. Fungsi *getRoomSizePref()* dan *getRoomSize()* mengembalikan tipe data *RoomSize* yang merupakan *enum*. Fungsi *ordinal* mengembalikan nilai urutan *enum* saat deklarasi *enum* dilakukan, dan deklarasi *RoomSize* dilakukan urut dari ukuran ruangan terkecil ke ukuran ruangan terbesar. Dalam hal ini, memang terdapat perubahan di desain dimana desain dan fungsi matematis sebelumnya menyatakan bahwa penalti ditimbulkan ketika ukuran ruangan dengan preferensi ukuran ruangannya tidak sama. Pada kenyataannya, penalti hanya ditimbulkan ketika pasien ditempatkan di ruangan dengan ukuran yang lebih besar dibandingkan dengan preferensi pasien tersebut.

```
if (entry.getKey() < patient.getAdmissionDate() + patient.getTreatmentDuration()) {
    if (!patient.getRoomPropPref().isEmpty()) {
        if (!room.getKey().getPropAvailability().isEmpty()){
            for (Integer rprop : patient.getRoomPropPref()) {
                if (!room.getKey().getPropAvailability().contains(rprop)) {
                    insertIntoMap(RoomPropViol,patient, room.getKey());
                    room_prop_pref_cost++;
                }
            }
        }
        else {
            insertIntoMap(RoomPropViol,patient, room.getKey());
            room_prop_pref_cost += patient.getRoomPropPref().size();
        }
    }
}
```

Kode 5.10 Pengecekan Penalti Preferensi Properti Ruangan Pasien

Selanjutnya, pengecekan dilakukan ke preferensi properti ruangan pasien. Kode yang serupa juga digunakan untuk mengecek kebutuhan properti ruangan pasien. Fungsi *insertIntoMap()* digunakan diawal pembuatan *validator* untuk membandingkan penempatan pasien di ruangan mana yang menghasilkan nilai penalti yang berbeda dengan hasil pengecekan menggunakan program *validator*.

```
//Room Department Specialism's Choice/Priority
if (!room.getKey().getDepartment().getPriority().contains(patient.getSpecialism()))
    room_dept_spec_cost++;

//Room Department Age Policy
Department rd = room.getKey().getDepartment();
if (!(rd.getMinAge() == rd.getMaxAge() && rd.getMinAge() == 0)) {
    if (patient.getAge() < rd.getMinAge() && patient.getAge() > rd.getMaxAge())
        room_dept_age_cost++;
}
```

Kode 5.11 Pengecekan Penalti Spesialisme dan Batasan Umur Departemen

Selanjutnya, pengecekan dilakukan ke berturut-turut spesialisme prioritas departemen dan batasan umur pasien di departemen. Pada data departemen di setiap *instance*, setiap departemen memiliki 1 spesialisme utama dan 2 spesialisme prioritas kedua. Hal ini memberikan implikasi bahwa departemen yang memprioritaskan spesialisme di urutan pertama akan diutamakan.Pada kenyataannya, penalti tidak akan ditimbulkan selama pasien ditempatkan di departemen yang memprioritaskan spesialisme tersebut tanpa memperhatikan apakah spesialisme tersebut diutamakan atau berada pada prioritas kedua.

```
int current_date = entry.getKey();
if (current_date > 0 && patient.getAdmissionDate() < current_date) {
    if (solution.get(current_date - 1).containsKey(room.getKey())) {
        if (!solution.get(current_date - 1).get(room.getKey()).contains(patient))
            patient_transfer_cost++;
    }
    else
        patient_transfer_cost++;
}
```

Kode 5.12 Pengecekan Penalti Transfer

Pengecekan selanjutnya dilakukan ke transfer. Pengecekan dilakukan dengan melihat apakah pasien di tengah admisi ditempatkan di ruangan yang berbeda dengan ruangan di admisi timeslot atau hari sebelumnya.

```
if ((entry.getKey() < patient.getAdmissionDate() + patient.getTreatmentDuration()) ) {
    Room rm = room.getKey();
    int spec = patient.getSpecialism();
    double cost;
    if (rm.getFirstPriority().contains(spec)) {
        cost = 0.0;}
    else if (rm.getSecondPriority().contains(spec)) {
        cost = 1.0;}
    else
        cost = 2.0;
    room_spec_cost += cost;
}
```

Kode 5.13 Pengecekan Spesialisme Ruangan

Selanjutnya dilakukan pengecekan spesialisme yang diprioritaskan ruangan. Variabel *cost* merupakan *multiplier* di desain.

```
if (patient.getGender() != gend ) {
    insertIntoMap(RoomGenderViol,patient, room.getKey());
    room_gender_cost++;
}
```

Kode 5.14 Pengecekan Batasan Jenis Kelamin Ruangan

Pengecekan terakhir dilakukan ke batasan jenis kelamin ruangan. Sebelum masuk ke iterasi pasien di dalam ruangan, variabel *gend* dibuat terlebih dahulu, dimana batasan jenis kelamin akan diperhitungkan dengan melihat pada distribusi jenis kelamin pasien di ruangan. Desain pertama menentukan batasan jenis kelamin hanya dengan melihat pada jenis kelamin pasien pertama yang ditempatkan. Setelah melihat pada implementasi program *validator*, ditemukan bahwa jenis kelamin ditentukan setiap harinya dengan melihat distribusi pasien. Jikalau distribusinya seimbang, maka akan meninjau pada pasien dengan tanggal *discharge* terlama. Selanjutnya, setiap hasil penalti akan dikali dengan bobotnya masing-masing, kemudian ditambahkan untuk memberikan penalti total. Nilai bobot per penalti disimpan di kelas *Weight*, dan dapat diidentifikasi dan diperoleh menggunakan *enum weight.*

5.7. Implementasi Algoritma Optimasi

Tahap ini menjelaskan tentang implementasi algoritma optimasi. Algoritma optimasi diimplementasikan di kelas *Optimizer*.

5.7.1. *Neighborhood*

Untuk permasalahan dengan *search space* yang cukup besar, pencarian yang tidak terarah sama sekali menjadi sangat mahal. Dalam hal ini, pencarian yang dilakukan dilakukan sepenuhnya secara acak. Dalam konteks penelitian ini, berarti pasien dipilih secara acak dan dipindahkan ke ruangan lain yang juga dipilih secara acak, atau ditukar dengan ruangan pasien lain yang dipilih secara acak. Hal ini akan menimbulkan banyak gerakan yang tidak *feasible*, dan tentunya akan membuang iterasi seandainya algoritma berbasis pada jumlah iterasi dan bukan waktu komputasi. Dalam implementasi ini, pergerakan yang dilakukan dipastikan untuk memenuhi *hard constraint*, sehingga algoritma hanya perlu membandingkan kualitas solusi. Untuk mencapai hal tersebut, dibuat daftar kandidat ruangan yang tersedia yang dapat menerima pasien untuk pergerakan *complete move* dan *partial move*. Untuk *complete swap* dan *partial swap*, dibuat daftar kandidat pasien yang jadwal ruangannya dapat ditukar dengan pasien yang dipilih. Hal ini memang akan menambahkan *overhead* di program, tetapi dapat memberikan hasil akhir yang lebih baik terutama ketika waktu komputasi yang diizinkan sangat pendek.

Implementasi lain yang juga berbeda dengan desain adalah pengecekan dan evaluasi beserta metode *backtracking* yang diimplementasikan di setiap *neighborhood*. Dengan demikian, algoritma VNS dan *random selection* di setiap iterasi cukup memanggil *neighborhood* lalu mengecek apakah penalti mengalami penurunan atau tidak. *Backtracking* dilakukan dengan menyalin blok solusi terlebih dahulu sebelum mengalami perubahan. Jikalau penalti setelah pergerakan

dilakukan lebih buruk, maka blok solusi dijadikan kembali ke salinan blok solusi yang sebelumnya sudah dibuat.

Selain itu, terdapat beberapa *neighborhood* tambahan yang dibuat, seperti *deterministic move*, *deterministic swap*, dan *partial swap 2*. Untuk *deterministic move*, pemilihan pasien yang akan dipindahkan tetap dilakukan secara acak, seperti pada pencarian stokastik. Namun, evaluasi dilakukan terhadap setiap perpindahan ke ruangan berbeda yang tersedia. Perpindahan yang dipilih adalah perpindahan yang menghasilkan penurunan penalti terbaik. Hal yang sama juga dilakukan untuk *deterministic swap*. Untuk *partial swap 2*, implementasi yang membedakan dengan *partial swap* adalah pada *partial swap 2,* pergerakan dilakukan ketika pasien kedua memiliki tanggal *discharge* lebih pendek atau sama dengan tanggal *discharge* pasien pertama. Artinya *neighborhood* ini memungkinkan maksimal 1x transfer yang terjadi. Jikalau pasien kedua memiliki tanggal *discharge* yang lebih pendek daripada pasien pertama, maka pasien pertama tetap akan dipindahkan ke ruangan kedua sampai periode akhir. Tentunya hal juga disertakan dengan pengecekan ruangan apakah masih tersedia atau tidak.

5.7.2. Algoritma *VNS* dan *Random Selection*

Seperti yang dijelaskan di subbab sebelumnya, implementasi evaluasi penalti dan *backtracking* dilakukan di setiap *neighborhood*. Sehingga, algoritma optimasi VNS dan *random selection* cukup memanggil fungsi terkait *neighborhood* tersebut. Untuk VNS, perubahan *neighborhood* dilakukan dengan mengecek apakah penalti yang didapat lebih baik. Jikalau ya, maka *neighborhood* tidak dipindah.

```
for (long stop = System.nanoTime() + TimeUnit.MILLISECONDS.toNanos(exec_time) ; stop > System.nanoTime();) {
    random = (int)Math.floor(Math.random()*5);
    switch (random) {
        case 0:
            solution = optimizeHC1(solution, weights, 100, pat_list, room_list, plan_hor);
            break;
        case 1:
            solution = optimizeHC2(solution, weights, 100, pat_list, room_list, plan_hor);
            break;
        case 2:
            solution = optimizeHC3(solution, weights, 100, pt_list, room_list, plan_hor);
            break;
        case 3:
            solution = optimizeHC4(solution, weights, 100, pt_list, room_list, plan_hor);
            break;
        case 4:
            solution = optimizeHC41(solution, weights, 100, pt_list, room_list, plan_hor);
            break;
    }

}
```

Kode 5.15 Blok Kode Utama untuk Random Selection

Untuk pemilihan acak, dalam setiap iterasi akan dipilih *neighborhood* secara acak. Untuk VNS, perubahan *neighborhood* dilakukan jika *neighborhood* pada hirarki di atasnya tidak memperoleh penalti yang lebih baik. Jikalau *neighborhood* manapun di dalam hirarki menghasilkan penalti yang lebih baik, maka pada iterasi selanjutnya eksekusi dilakukan kembali dari *neighborhood* teratas di hirarki yang ada. Hirarki disusun dengan melihat pada performa setiap *neighborhood*.

```
for (long stop = System.nanoTime() + TimeUnit.MILLISECONDS.toNanos(exec_time) ; stop > System.nanoTime();) {
    double prev_pen = pen;
    switch (neighbor) {
        case 0:
            solution = optimizeHC1(solution, weights, 100, pat_list, room_list, plan_hor);
            break;
        case 1:
            solution = optimizeHC2(solution, weights, 100, pat_list, room_list, plan_hor);
            break;
        case 2:
            solution = optimizeHC3(solution, weights, 100, pt_list, room_list, plan_hor);
            break;
        case 3:
            solution = optimizeHC4(solution, weights, 100, pt_list, room_list, plan_hor);
            break;
        case 4:
            solution = optimizeHC41(solution, weights, 100, pt_list, room_list, plan_hor);
            break;
    }
    if (pen >= prev_pen) {
        if (neighbor < 2) {
            neighbor++; }
        else
            neighbor = 0;
    }
    else
        neighbor = 0;

}
```

Kode 5.16 Blok Kode Utama untuk VNS

Berbagai *pseudocode* algoritma *reduced VNS* menggunakan 2 iterasi dimana iterasi pertama merupakan iterasi waktu, dan di dalamnya terdapat iterasi per *neighborhood*. Dengan

implementasi demikian, ada kemungkinan iterasi tetap melakukan pencarian sampai ke *neighborhood* terakhir walaupun waktu komputasi sudah melewati batas. Sehingga, implementasi yang dibuat hanya memiliki satu iterasi, yakni iterasi berbasis pada waktu. Jikalau semua *neighborhood* sudah diiterasi dan tidak diperoleh penalti yang lebih baik, maka *neighbor* kembali di-*reset* ke 0 (blok merah).

5.7.3. Parameter dan *Stopping Mechanism*

Waktu awal eksekusi diperoleh menggunakan *System.nanoTime()*, kemudian parameter waktu eksekusi diimplementasikan menggunakan *Integer* yang menyimpan nilai *milisecond*. Konfigurasi *neighborhood* yang digunakan langsung di-*hardcode* di dalam iterasi dan tidak dijadikan parameter dalam implementasi.

BAB VI
HASIL DAN PEMBAHASAN

Pada bab ini dijelaskan hasil dan pembahasan yang didapatkan dari tugas akhir ini. Penulisan *neighborhood* disingkat sesuai menggunakan inisialnya.

6.1. Lingkungan Uji Coba

Lingkungan uji coba digunakan untuk melakukan eksekusi algoritma serta menunjang proses analisis dan penyusunan dokumen. Spesifikasi dari perangkat keras yang digunakan ditunjukkan pada Tabel 6.1.

Tabel 6.1 Spesifikasi Perangkat Keras

Perangkat Keras	Spesifikasi
Jenis	Laptop
Processor	Intel Celeron N2840 (2 Cores / 2 Threads) 2.16GHz (turbo 2.58GHz)
RAM	8192 MB
HDD Space	500GB

Selain itu, perangkat lunak yang digunakan dalam menyelesaikan tugas akhir dapat dilihat pada Tabel 6.2.

Tabel 6.2 Perangkat Lunak Yang Digunakan

Perangkat Lunak	Fungsi
Windows 8.1	Sistem Operasi
RStudio 1.1.456 (R 3.5.1)	Membuat visualisi trajektori *fitness function*
Apache NetBeans IDE 11.2 (JDK 13.0.1)	Mengimplementasikan dan menjalankan algoritma
Lightshot	Membuat dan mengatur *screenshot*
Dia	Membuat ERD & *Class Diagram*

Microsoft Office Excel 2016	Merekap dan analisis data
Microsoft Office Word 2016	Membuat laporan

6.2. Hasil Pembangkitan Solusi Awal

Dengan menggunakan heuristik *Best Fit*. Didapat hasil sebagai berikut.

Tabel 6.3 Penalti dari Solusi Awal

No	Penalti per Percobaan			
	1	2	3	4
0	4530,6	3105,4	2082,2	2000,4
1	3532,6	2498,4	1316,0	1327,6
2	5673,0	4482,8	2633,0	2488,2
3	4119,6	3045,4	1933,6	1847,8
4	5603,0	4756,0	2848,6	2831,4
5	3000,6	1617,0	965,2	914,0
6	3933,8	2859,0	1698,8	1685,0

Pada percobaan 1, digunakan metode *First Come First Served* (FCFS), dimana pasien dijadwalkan mengikuti urutan penulisan di *instance* terkait. Metode ini berhasil memberikan solusi yang *feasibile*, walaupun memang memiliki kualitas solusi yang relatif buruk.

Pada percobaan 2, dilakukan modifikasi pada metode FCFS yang digunakan. Prinsip FCFS memang harus dipenuhi dalam kasus ini, mengingat pasien dijadwalkan mengikuti tanggal admisinya, dan urutan penulisan di *instance* terkait memang sudah urut sesuai dengan tanggal admisi pasien. Namun demikian, jumlah pasien yang tinggi dengan *planning horizon* yang relatif singkat (14 hari untuk semua *instance*) berarti terdapat banyak pasien yang memiliki tanggal admisi yang sama. Sebelum algoritma dijalankan, daftar pasien terlebih dahulu di sortir. Penyortiran disini mengutamakan pasien dengan kebutuhan dan preferensi properti ruangan yang paling

banyak, kemudian disusul dengan pasien dengan preferensi ukuran ruangan terbesar. Tentunya penyortiran tetap mematuhi tanggal admisi masing-masing pasien.

Pada percobaan 3, dilakukan beberapa modifikasi di metode komparasi. Modifikasi ini dilakukan setelah perilaku program evaluasi yang disediakan di [5] dipelajari lebih lanjut. Beberapa perilaku yang ditemukan yakni:

- Penentuan batasan jenis kelamin untuk ruangan yang memiliki batasan jenis kelamin awal *DEPENDENT*. Hal ini tidak semata-mata ditentukan dengan melihat pasien mana yang diadmisi lebih awal diruangan tersebut, tetapi ditentukan dengan mempertimbangkan jumlah pasien terbanyak di ruangan tersebut. Jikalau seri, maka jenis kelamin ditentukan dengan meninjau pada pasien mana yang memiliki sisa LOS terbanyak, atau tanggal *discharge* terjauh.
- Pengaturan sortir preferensi ukuran ruangan pasien. Sebelumnya, implementasi dilakukan dengan mengutamakan ruangan yang memiliki ukuran ruangan sama dengan preferensi pasien. Ruangan lain dengan ukuran ruangan yang berbeda merupakan prioritas kedua. Implementasi tersebut dilakukan dengan menimbang pada *soft constraint*. Pada kenyataannya, tidak ada penalti yang ditimbulkan jika pasien dijadwalkan ke ruangan yang berukuran lebih kecil daripada preferensi pasien tersebut. Hal ini tentunya mengubah metode komparasi, dimana prioritas pertamanya menjadi ruangan dengan ukuran sama dengan atau lebih kecil daripada preferensi pasien, dan prioritas kedua menjadi ruangan dengan ukuran lebih besar daripada preferensi pasien. Pada model mateimatisnya, $ep_p = e_r$ berubah menjadi $ep_p \geq e_r$
- Pengaturan sortir spesialisme ruangan. Dalam implementasi, ruangan yang memiliki spesialisme di prioritas ketiga, akan diutamakan dibanding dengan ruangan yang tidak memprioritaskan spesialisme yang

sama. Dalam program *validator*, ternyata pengali bobot yang ditimbulkan dengan menempatkan pasien di salah satu ruangan tersebut bernilai sama (*multiplier* = 2). Implementasi komparasi pun diubah mengikuti perilaku ini.

- Pengaturan sortir spesialisme departemen ruangan. Dalam implementasi, ruangan yang berada di departemen yang memiliki spesialisme di prioritas pertama berada di posisi yang lebih tinggi dibanding dengan ruangan yang berada di departemen dengan spesialisme yang sama berada pada prioritas kedua. Dalam program *validator*, kedua prioritas tersebut sama-sama tidak menghasilkan penalti. Penalti hanya ditimbulkan jikalau pasien ditempatkan di ruangan dimana departemennya tidak memprioritaskan spesialisme tersebut. Implementasi komparasi pun diubah mengikuti perilaku ini.

Pada percobaan 4, dilakukan *filter* terhadap daftar pasien yang akan dijadwalkan. Pasien yang memiliki tanggal admisi dan tanggal *discharge* yang sama (LOS = 0) akan dibuang dari daftar tersebut. Hal ini dilakukan setelah diketahui bahwa program *validator* yang disediakan di [5] tidak menghasilkan *error* atau penalti ketika pasien-pasien tersebut tidak dijadwalkan. Perilaku yang sama juga ditekankan di [26]. Hal ini mungkin dikarenakan:

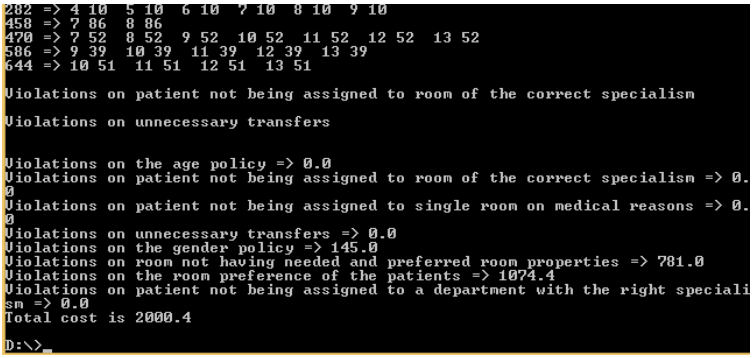
- Pasien tersebut adalah pasien dari penjadwalan di periode *planning horizon* sebelumnya yang di-*discharge* di periode ini. Hal ini masuk akal mengingat kebanyakan pasien dengan 0 LOS kebanyakan berada di periode awal *planning horizon*.
- Pasien tersebut adalah *outpatient* yang tidak membutuhkan rawat inap. Oleh karena itu, pasien tersebut hanya membutuhkan fasilitas rumah sakit untuk sementara.

Teori kedua sempat dicobakan, dimana dua pasien dengan satu diantaranya memiliki 0 LOS dijadwalkan di ruangan yang sama

dengan kapasitas penuh. Hal ini tidak menimbulkan pelanggaran *hard constraint* ketika dicek menggunakan program *validator*. Hal ini bisa dibilang masuk akal, mengingat deskripsi waktu dalam konteks penjadwalan ini adalah hari atau *timeslot* di setiap *planning horizon*, dan tidak ada definisi tepat apakah pasien tersebut dijadwalkan di jam tertentu atau periode tertentu (pagi, siang, malam). Artinya, bisa saja pasien pertama dengan 0 LOS dijadwalkan pagi dan di-*discharge* pada malam hari, dengan pasien kedua kemudian dijadwalkan di ruangan yang sama di malam hari. Walaupun demikian, percobaan ini pada akhirnya dilakukan dengan membuang semua pasien dengan 0 LOS, karena memang dari sisi penalti tidak menimbulkan penalti tambahan.

6.2.1. Validasi Solusi

Seperti yang sudah dijelaskan di tahap desain, Algoritma pembangkit solusi awal memastikan bahwa setiap pasien dijadwalkan di waktu admisinya dan di-*discharge* sesuai dengan tanggal *discharge* pasien tersebut. Setiap kali pasien ditambahkan ke ruangan tertentu, kapasitas ruangan tersebut akan dikurangi. Ketika kapasitas ruangan mencapai 0, maka ruangan tersebut akan dihapus dari daftar ruangan yang tersedia. Hal ini memastikan bahwa semua *hard constraint* dipenuhi secara *implicit*. Untuk memastikan, pengecekan *explicit* menggunakan metode juga telah diimplementasikan dan dijalankan. Solusi juga sudah dikonversi ke dalam bentuk relasi pasien-kamar untuk kemudian dicek menggunakan program *validator*.



Gambar 6.1 Sampel Bukti Solusi Awal Lulus Validasi - Instance 0

Jikalau melanggar *hard constraint*, *validator* akan menghasilkan *exception*. Hasil lengkap validasi dapat dilihat pada Lampiran A.

6.2.2. Evaluasi *Fitness Function*

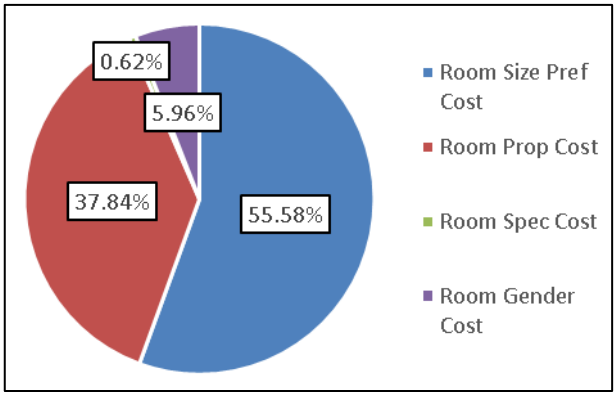
Perhitungan penalti / *fitness function* dilakukan menggunakan implementasi *validator* yang sebelumnya telah dibuat. Hasil yang diperoleh sudah sesuai dengan hasil perhitungan menggunakan program *validator* (lihat Lampiran A).

Tabel 6.4 Distribusi Penalti masing-masing Instance

Jenis Penalti	*Instance*						
	0	1	2	3	4	5	6
Room Size	1074,4	841,6	1371,2	996,8	1254,4	744	996
Room Prop	781	486	972	684	1340	170	522
Dept Spec	0	0	0	0	0	0	0
Dept Age	0	0	0	0	0	0	0
Transfer	0	0	0	0	0	0	0
Room Spec	0	0	0	22	2	0	57
Gender	145	0	145	145	235	0	110
Total	2000,4	1327,6	2488,2	1847,8	2831,4	914	1685

Algoritma pembangkit solusi awal menempatkan pasien hanya di satu ruangan selama periode admisinya, sehingga memang tidak ada penalti transfer yang ditimbulkan. Selain itu, semua *instance* dapat menerima pasien dengan umur manapun, sehingga tidak ada penalti umur yang ditimbulkan. Spesialisme yang dapat didukung oleh banyak departemen juga menghasilkan solusi tanpa penalti spesialisme departemen. Hasil penalti yang didapat juga selaras dengan statistik yang diperoleh pada subbab 4.1. di BAB IV, dimana ABO tertinggi ada pada ABO per *room size preference*, disusul dengan ABO per *room prop*.

Dengan menjumlahkan semua penalti dari semua *instance*, diperoleh informasi bahwa penalti ukuran ruangan mendominasi total penalti (55,58%), disusul dengan penalti properti ruangan (37,84%) yang merupakan gabungan dari penalti kebutuhan properti ruangan dan penalti preferensi properti ruangan. Distribusi total penalti dapat dilihat pada Gambar 6.2.



Gambar 6.2 Pie Chart Distribusi Total Penalti

6.3. Hasil Uji Coba Awal Optimasi - Satu *Neighborhood*

Optimasi awal dilakukan melihat performa setiap *neighborhood*. Uji coba awal dilakukan selama 30 detik dan

sebanyak 10 kali. Tabel 6.5 menampilkan statistik rata-rata, batas bawah (*lower bound*), dan batas atas (*upper bound*) untuk *complete move* dan *complete swap*.

Tabel 6.5 Statistik Uji Coba Awal - Complete Move & Complete Swap

No	*Complete Move*			*Complete Swap*		
	Mean	*LB*	*UB*	*Mean*	*LB*	*UB*
0	1521,6	1494,6	1574,0	1676,2	1623,0	1704,0
1	1207,4	1194,6	1216,0	1159,9	1125,0	1175,8
2	2016,7	1967,6	2062,6	2151,5	2076,2	2183,0
3	1445,3	1385,2	1500,4	1506,9	1480,4	1532,4
4	2240,3	2179,2	2289.,2	2307,5	2253,2	2353,6
5	829,6	823,6	838,8	839,1	820,8	847,4
6	1465,4	1414,8	1487,4	1302,8	1246,8	1358,4

Complete move memberikan rata-rata penalti terbaik untuk 4 dari 6 *instance*. *Complete swap* memberikan rata-rata penalti terbaik untuk *Instance 1* dan *instance 6*. Tren yang sama juga terjadi di batas bawah dan batas atas, dengan pengeculian pada *instance 5* yang memiliki batas bawah terbaik dihasilkan oleh *complete swap*.

Untuk *partial move* dan *partial swap*, dilakukan beberapa percobaan dengan mengganti nilai LOS yang akan di-*filter*. Pemilihan dilakukan dengan melihat pada perubahan nilai penalti antara penalti awal dengan penalti rata-rata masing-masing *filter*.

Tabel 6.6 Perbandingan Filter LOS untuk Partial Move & Partial Swap

No	*Penalty Changes*							
	Partial Move				Partial Swap			
	No Filter	LOS > 1	LOS > 2	LOS > 3	No Filter	LOS > 1	LOS > 2	LOS > 3
0	320,5	306,5	358,5	350,6	132,0	147,4	144,7	145,8
1	105,6	103,5	98,8	90,7	71,9	70,8	61,3	56,8

2	290,7	297,8	326,7	311,6	117,9	126,4	104,6	127,6
3	321,0	310,7	322,7	316,2	101,9	105,9	90,9	93,9
4	391,8	415,6	435,8	442,5	125,9	126,6	118,9	130,7
5	66,4	74,1	69,7	58,5	38,0	33,9	30,7	25,8
6	194,4	194,0	192,7	174,4	120,8	113,7	113,5	94,4

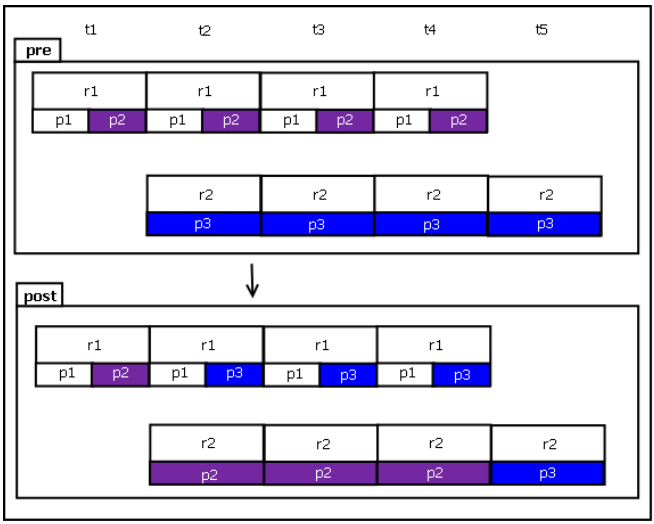
Berdasarkan hasil perbandingan filter, LOS > 2 dipilih untuk digunakan di uji coba selanjutnya. Filter LOS >2 menghasilkan penurunan penalti terbaik untuk 3 *instance* menggunakan *partial move*. Dalam hal ini, memang hasil *partial move* lebih diutamakan mengingat penurunan penalti yang dihasilkan lebih baik daripada *partial swap*. Tanpa menggunakan filter, penurunan penalti terbaik diperoleh untuk 3 *instance* menggunakan *partial swap*. Walaupun demikian, hal ini juga berarti bahwa lebih banyak pergerakan *complete* yang dilakukan, mengingat semakin pendek LOS pasien maka semakin besar kemungkinan pergerakan *complete* dilakukan dan bukan *partial*.

Tabel 6.7 Statistik Uji Coba Awal - Partial Move & Partial Swap

No	*Partial Move*			*Partial Swap*		
	Mean	*LB*	*UB*	*Mean*	*LB*	*UB*
0	1641,9	1609,6	1675,8	1855,7	1807,4	1888,6
1	1228,8	1220,2	1248,2	1266,3	1253,6	1280,0
2	2161,5	2127,8	2246,2	2383,6	2349,6	2439,6
3	1525,1	1491,2	1556,8	1756,9	1744,0	1776,4
4	2395,6	2326,6	2474,8	2712,5	2673,4	2741,0
5	844,3	841,2	849,6	883,3	879,8	892,2
6	1492,3	1432,4	1540,8	1571,5	1546,4	1599,0

Partial move memberikan hasil penalti yang jauh lebih signifikan di semua *instance* bila dibandingkan dengan *partial swap*. Batas bawah dari *partial swap* untuk setiap *instance* masih berada di atas batas atas dari *partial move*. Perilaku ini

masuk akal, mengingat *partial swap* membutuhkan penurunan penalti penukaran ruangan pasien yang lebih besar daripada biaya transfer yang ditimbulkan. Selain itu, *partial swap* memiliki kemungkinan untuk menghasilkan tidak hanya satu transfer, tetapi juga dua transfer. Hal ini terjadi jika tanggal *discharge* pasien kedua lebih besar dibanding tanggal *discharge* pasien pertama. Alhasil, pasien pertama dipindahkan ke ruangan pasien kedua hingga ke akhir admisinya, dan pasien kedua dijadwalkan di ruangan pasien pertama untuk awal admisinya, kemudian dipindahkan ke ruangan pasien kedua hingga akhir admisinya. Implementasi serupa juga dilakukan di [26].



Gambar 6.3 Contoh Partial Swap yang Menghasilkan 2x Transfer

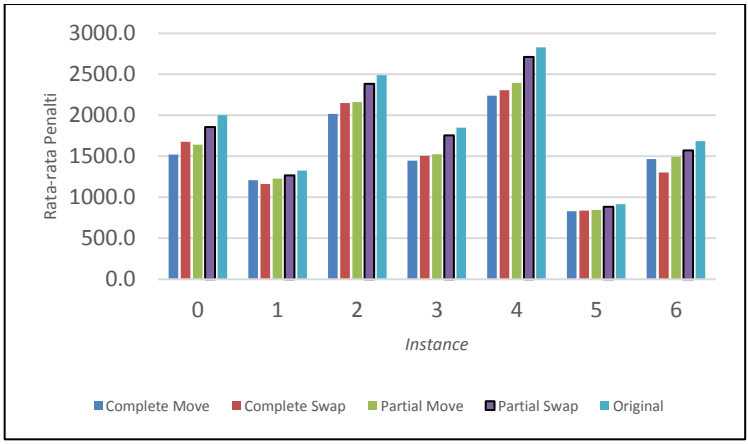
6.3.1. Validasi Solusi

Solusi yang dihasilkan selalu *feasible*, mengingat implementasi *neighborhood* yang memastikan bahwa setiap gerakan selalu memenuhi *hard constraint*. Selain itu, hasil juga sudah dicek menggunakan metode *checkForAllocationRequirements()* dan *checkForAllocationCapacity()*. Metode yang digunakan untuk melakukan konversi solusi ke format yang diterima program

validator (*convertRoomToBedAssignment()*) juga akan men-*throw exception* jikalau terdapat alokasi ruangan yang melebihi kapasitas ruangan tersebut.

6.3.2. Evaluasi *Fitness Function*

Dengan membandingkan hasil yang diperoleh dari berbagai *neighborhood*, *complete move* memberikan hasil terbaik, disusul dengan *complete swap*, *partial move*, dan *partial swap*. *Partial move* menghasilkan hasil yang lebih baik bila dibandingkan dengan *complete swap* hanya untuk *instance 1*.



Gambar 6.4 Bar Chart Perbandingan Performa Neighborhood

Complete swap menghasilkan penurunan rata-rata penalti yang paling rendah, dengan penurunan penalti 4% - 7%. *Complete move* di sisi lain, menghasilkan penurunan rata-rata penalti sebanyak 9% - 23%.

Tabel 6.8 Persentase Penurunan Rata-rata Penalti

No	*Neighborhood*			
	CM	CS	PM	PS
0	-23,94%	-16,21%	-17,92%	-7,23%
1	-9,05%	-12,63%	-7,45%	-4,62%

2	-18,95%	-13,53%	-13,13%	-4,20%
3	-21,78%	-18,45%	-17,47%	-4,92%
4	-20,88%	-18,50%	-15,39%	-4,20%
5	-9,23%	-8,19%	-7,62%	-3,36%
6	-13,03%	-22,68%	-11,44%	-6,73%

Untuk hasil penalti di semua *run*, dapat dilihat pada LAMPIRAN B.

6.4. Hasil Uji Coba 1: Eksekusi Selama 30S

Skenario ini dilakukan menggunakan gabungan *neighborhood* dan dengan menggunakan dua metode pemilihan *neighborhood* yang berbeda; *random selection* dan *VNS*. Uji coba dilakukan sebanyak 10 kali per konfigurasi *neighborhood*. Untuk VNS, hanya beberapa konfigurasi yang diujicobakan, dikarenakan jumlah permutasi dari keempat *neighborhood* yang cukup banyak. Setiap iterasi memberikan kesempatan bagi *neighborhood* untuk melakukan pergerakan selama 1 detik. *Neighborhood* yang digunakan yakni *Complete Move* (CM), *Complete Swap* (CS), *Partial Move* (PM), dan *Partial Swap* (PS).

6.4.1. Dua *Neighborhood*

Optimasi dilakukan menggunakan semua kemungkinan kombinasi dua *neighborhood*. Dengan menggunakan pemilihan *neighborhood* secara acak, kombinasi CM dan CS memberikan hasil penurunan penalti terbaik di semua *instance*.

Tabel 6.9 Rata-rata Penalti per Kombinasi 2 Neighborhood – Random Selection

No	*Mean*					
	CM,CS	CM,PM	CM,PS	CS,PM	CS,PS	PM,PS
0	1373,7	1471,4	1439,0	1607,4	1783,0	1683,7
1	996,7	1075,4	1031,4	1076,1	1175,8	1153,5

2	1741,9	1868,1	1860,2	1961,4	2187,4	2155,5
3	1247,9	1321,0	1336,9	1369,1	1564,7	1538,4
4	1943,9	2076,6	2117,0	2195,6	2459,6	2364,2
5	723,8	758,4	759,0	755,2	807,7	824,3
6	1107,2	1200,5	1212,8	1234,8	1350,0	1383,5

Selanjutnya, kombinasi CM dan PM memberikan hasil yang lebih baik di 4 *instance* bila dibandingkan dengan kombinasi CM dan PS. Kombinasi CS dan PM menghasilkan penurunan penalti yang lebih tinggi di *instance* 1-6 bila dibandingkan dengan CM. Kombinasi CS-PS dan kombinasi PM-PS disisi lain, menghasilkan penalti yang relatif lebih buruk bila dibandingkan dengan CM di kebanyakan *instance*. Terlihat bahwa *neighborhood* CM sangat dominan, dimana kombinasi yang melibatkan CM menghasilkan penurunan penalti yang lebih signifikan.

Tabel 6.10 Rata-rata Penalti per Permutasi 2 Neighborhood – VNS

No	*Mean*					
	CM,CS	CM,PM	CM,PS	CS,PM	CS,PS	PM,PS
0	1376,7	1396,7	1378,8	1631,5	1742,2	1708,6
1	1001,2	1024,5	1017,9	1059,8	1140,4	1151,4
2	1737,2	1771,6	1757,9	2005,9	2087,2	2105,0
3	1243,6	1283,3	1260,2	1398,4	1468,0	1482,3
4	1996,9	2018,1	2008,2	2198,6	2289,5	2334,9
5	730,9	737,5	743,8	752,7	790,4	823,6
6	1133,8	1142,2	1138,7	1261,4	1270,5	1367,1

Selanjutnya, hirarki pada VNS ditentukan dengan meninjau pada performa setiap *neighrborhood* (CM>CS>PM>PS). Dengan menggunakan VNS, permutasi CM dan CS tetap memberikan penurunan penalti terbaik. Namun demikian, permutasi CM dan PM serta permutasi CM dan PS mengalami penurunan penalti yang lebih signifikan bila dibandingkan

dengan kombinasi yang sama menggunakan pemilihan acak. Kebanyakan *instance* bahkan memiliki perbedaan yang *marginal* bila dibandingkan dengan permutasi CM dan CS. Hal ini kembali membuktikan bahwa CM merupakan *neighborhood* yang paling dominan. Penurunan penalti juga terjadi di kebanyakan *instance* pada permutasi CS dan PS serta permutasi PM dan PS ketika dibandingkan dengan versi pemilihan acaknya. Hal ini membuktikan bahwa CS dan PM memiliki performa yang lebih baik bila dibandingkan dengan PS. Dengan membandingkan perolehan penalti yang didapat pada konfigurasi *neighborhood* yang sama, metode VNS memberikan penurunan penalti sebanyak 2% dibandingkan dengan penalti yang diperoleh menggunakan metode pemilihan acak.

6.4.2. Tiga *Neighborhood*

Optimasi dilakukan dengan menggunakan kombinasi tiga *neighborhood*. Semua kombinasi diujicobakan di metode pemilihan acak, namun tidak semua permutasi yang ada dicobakan di metode VNS.

Tabel 6.11 Rata-rata Penalti per 3 Kombinasi Neighborhood – Random Selection

No	*Mean*			
	CM,CS,PM	CM,CS,PS	CM,PM,PS	CS,PM,PS
0	1438,2	1470,1	1509,7	1670,6
1	1031,8	1022,3	1073,3	1093,8
2	1798,5	1823,0	1927,8	2061,9
3	1258,9	1272,3	1366,2	1459,7
4	2032,8	2084,5	2168,9	2313,2
5	748,6	738,7	770,6	776,7
6	1166,5	1208,1	1235,9	1315,5

Kombinasi CM, CS, dan PM menghasilkan penurunan penalti terbaik untuk 4 *instance*, dengan 2 *instance* lain memiliki nilai

penalti terbaik yang dihasilkan oleh kombinasi CM, CS, dan PS. *Neighborhood* CM tetap memiliki dampak yang besar. Hal ini terlihat pada konfigurasi CS, PM dan PS yang menghasilkan penurunan penalti terburuk. Walaupun demikian, dengan menggunakan konfigurasi CM yang juga menekankan *swap* (PS dan CS), terdapat penurunan penalti yang lebih baik di *instance 1* dan *instance 5* bila dibandingkan dengan konfigurasi CM, CS, dan PM.

Tabel 6.12 Rata-rata Penalti per 3 Permutasi Neighborhood – VNS

No	*Mean*			
	CM,CS,PM	CM,CS,PS	CM,PM,PS	CS,PM,PS
0	1433,8	1376,4	1384,1	1653,3
1	1030,0	1000,1	1039,8	1085,0
2	1810,5	1738,0	1783,4	2010,5
3	1285,7	1237,9	1264,1	1425,2
4	2016,2	1951,0	1992,6	2233,4
5	741,6	730,3	735,4	761,8
6	1159,3	1135,2	1150,5	1246,2

Dengan menggunakan urutan hirarki yang sama (CM>CS>PM>PS), terdapat penurunan rata-rata penalti di semua *instance* di konfigurasi CM,CS,PS, konfigurasi CM,PM, PS, dan konfigurasi CS, PM, PS. Selain itu, penurunan rata-rata penalti juga terdapat pada 5 dari 7 *instance* dengan menggunakan CM, CS, dan PM. Selain itu, penurunan penalti terbaik diperoleh menggunakan konfigurasi CM, CS, dan PS, berbeda dengan hasil yang diperoleh menggunakan pemilihan acak, dimana konfigurasi yang sama hanya memperoleh penurunan penalti terbaik di *instance1* dan *instance 5*. Dengan membandingkan perolehan penalti yang didapat pada konfigurasi *neighborhood* yang sama, metode VNS memberikan penurunan penalti sebanyak 3,32% dibandingkan dengan penalti yang diperoleh menggunakan metode pemilihan acak.

6.4.3. Empat *Neighborhood*

Optimasi dilakukan dengan menggunakan keempat *neighborhood*. VNS yang diujicobakan menggunakan hirarki CM>CS>PM>PS. Dikarenakan jumlah konfigurasi yang sedikit, analisis yang dilakukan juga memperhatikan batas atas dan batas bawah.

Tabel 6.13 Rata-rata Penalti menggunakan 4 Neighborhood

No	*Random Selection*			VNS		
	Mean	*LB*	*UB*	*Mean*	*LB*	*UB*
0	1511,6	1415,4	1640	1386	1322	1438
1	1027,5	970,8	1069,4	1006,4	969,2	1031,6
2	1880,6	1813,4	1913,4	1761,3	1680,6	1807,6
3	1334,1	1291,6	1391,8	1242,7	1203,4	1276
4	2105,4	2024,8	2189,8	2014,5	1931	2143,2
5	749,9	726,6	772,8	734,7	714,4	752,4
6	1193,1	1118,4	1263,6	1135,2	1105,6	1159,6

VNS memberikan rata-rata penalti, batas bawah, dan batas atas yang lebih baik bila dibandingkan dengan pemilihan acak. Walaupun demikian, beberapa *instance* yang menggunakan pemilihan acak memiliki batas bawah dan rata-rata yang relatif dekat bahkan lebih baik dibanding dengan batas atas VNS. Hail ini menandakan bahwa perbedaan jangkauan nilai antar kedua metode belum begitu signifikan. Dengan membandingkan perolehan penalti yang didapat pada konfigurasi *neighborhood* yang sama, metode VNS memberikan penurunan penalti sebanyak 4,96% dibandingkan dengan penalti yang diperoleh menggunakan metode pemilihan acak.

6.4.4. Validasi Solusi

Validasi solusi di uji coba ini dilakukan untuk memastikan bahwa pergerakan yang dihasilkan oleh setiap *neighborhood* sudah sesuai dan *compatible* dengan *neighborhood* lainnya.

Validasi dilakukan hanya terhadap konfigurasi 4 *neighborhood*, karena jikalau dengan solusi yang dihasilkan sudah memenuhi *hard constraint*, maka konfigurasi *neighborhood* yang lebih sedikit sudah pasti sudah memenuhi *hard constraint*. Solusi yang dperoleh sudah terbukti tidak melanggar *hard constraint*.

6.4.5. Evaluasi Keseluruhan

Evaluasi dilakukan dengan membandingkan hasil terbaik yang diperoleh di konfigurasi *neighborhood* dengan jumlah yang berbeda. Selain itu, pada uji coba ini dan uji coba selanjutnya, *acceptance criterion* yang digunakan dubah, dari *f(x') < f(x)* mennjadi *f(x') <= f(x)*. Hal ini dilakukan untuk membantu pergerakan keluar dari *local optimum*. Uji coba singkat juga sempat dilakukan, dan menunjukan bahwa penalti yang didapat lebih baik. Dengan melihat pada rata-rata total di semua percobaan, metode pemilihan acak menghasilkan rata-rata 1440,7, dan metode VNS menghasilkan rata-rata penalti 1398,2 (-2,95%).

Tabel 6.14 Rata-rata Penalti per Konfigurasi Terbaik - Random Selection

No	*Mean*			
	CM	CM,CS	CM,CS,PM	CM,CS,PM,PS
0	1521,6	1373,7	1439,0	1607,4
1	1207,4	996,7	1031,4	1076,1
2	2016,7	1741,9	1860,2	1961,4
3	1445,3	1248,0	1336,9	1369,1
4	2240,3	1943,2	2117,0	2195,6
5	829,6	723,8	759,0	755,2
6	1465,4	1107,2	1212,8	1234,8

Dengan menggunakan pemilihan acak, kombinasi CM dan CS memberikan hasil penalti terbaik. Kombinasi CM, CS, PM memberikan hasil yang lebih buruk dibandingkan dengan kombinasi CM dan CS, begitu juga dengan CM, CS, PM, dan

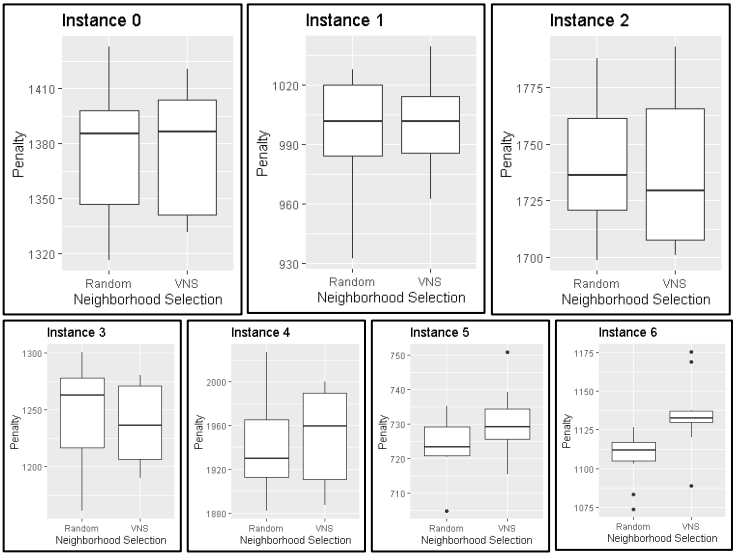
PS. Hal ini memang masuk akal, mengingat *neighborhood* CM adalah *neighborhood* paling dominan. Sehingga, semakin banyak *neighborhood* yang digunakan, semakin kecil kemungkinan *neighborhood* CM untuk terpilih di setiap iterasinya. Walaupun demikian, dibutuhkan pergerakan *swap* untuk dapat lebih lanjut menurunkan penalti.

Tabel 6.15 Rata-rata Penalti per Konfigurasi Terbaik - VNS

No	*Mean*			
	CM*	CM,CS	CM,CS,PS	CM,CS,PM,PS
0	1521,6	1376,7	1376,4	1386,0
1	1207,4	1001,2	1000,1	1006,4
2	2016,7	1737,2	1738,0	1761,3
3	1445,3	1243,6	1237,9	1242,2
4	2240,3	1996,9	1951,0	2014,5
5	829,6	730,9	730,3	734,7
6	1465,4	1133,8	1135,2	1135,2

Dengan menggunakan VNS, 4 dari 6 *instance* memiliki penalti terbaik yang dihasilkan oleh konfigurasi CM, CS, dan PS. *Instance 2* dan *instance 6* di sisi lain, memiliki penalti terbaik yang dihasilkan oleh konfigurasi CM dan CS. Untuk konfigurasi *neighborhood* > 2, VNS menghasilkan penalti yang relatif lebih baik jika dibandingkan *random selection*, dengan perolehan yang lebih baik bila dibandingkan dengan CM. Hal ini dikarenakan VNS yang mengutamakan *neighborhood* sesuai urutan hirarkinya, sehingga pemilihan *neighborhood* dengan urutan teratas relatif yang lebih condong *(skewed*). Hal ini berarti CM memiliki lebih banyak kesempatan untuk melakukan pergerakan, sekalipun jumlah *neighborhood*-nya cukup banyak. Jikalau CM gagal memperoleh penalti yang lebih baik, maka *neighborhood* dengan hirarki dibawahnya akan dipilih. Hasil yang diperoleh menggunakan VNS konfigurasi CM, CS, dan PS adalah hasil terbaik hanya di *instance 2* dan *instance 3* ketika dibandingkan dengan hasil

yang diperoleh menggunakan pemilihan acak dengan konfigurasi CM dan CS. Dengan demikian, untuk waktu eksekusi 30 detik, penggunaan pemilihan acak dengan konfigurasi CM dan CS memberikan hasil terbaik. Selanjutnya, dilakukan perbandingan hasil yang didapat pada kombinasi *neighborhood* terbaik pada metode pemilihan acak dengan kombinasi *neighborhood* terbaik pada metode VNS. Pada pemilihan acak, kombinasi *neighborhood* terbaik adalah CM-CS, dengan rata-rata penurunan penalti sebanyak 29,31% dari penalti solusi awal. Sedangkan pada VNS, kombinasi *neighborhood* terbaik adalah CM-CS-PS, dengan rata-rata penurunan penalti sebanyak 28,98% dari penalti solusi awal.



Gambar 6.5 Boxplot Perbandingan Penalti Metode Pemilihan Acak CM-CS dengan Penalti Metode VNS CM-CS-PS

Sesuai dengan hasil yang didapat sebelumnya, metode VNS berhasil memberikan rata-rata penurunan penalti ketika dibandingkan dengan metode pemilihan acak yang menggunakan kombinasi *neighborhood* yang sama. Selain itu, semakin banyak *neighborhood* yang digunakan, semakin besar

penurunan penalti yang diperoleh. Namun, dengan membandingkan antara kombinasi *neigborhood* terbaik, metode pemilihan acak tetap memberikan rata-rata penurunan penalti yang lebih baik. Pada *instance* 0, 2 dan 4, metode VNS memiliki jangkauan nilai yang relatif lebih tinggi. *Instance* 4, 5 dan 6 memiliki rata-rata penalti yang lebih tinggi pada metode VNS, sedangkan pada *instance* 2 dan 3, VNS menghasilkan rata-rata penalti yang lebih rendah. Hasil lengkap dapat dilihat di LAMPIRAN C.

6.5. Hasil Uji Coba 2: Eksekusi Selama 400S

Skenario dilakukan menggunakan beberapa konfigurasi terbaik yang dihasilkan oleh uji coba 30 detik. Setiap *neighborhood* diberikan kesempatan untuk melakukan pergerakan selama 0,1 detik, sehingga uji coba bersifat lebih acak. Uji coba dilakukan sebanyak 3 kali per konfigurasi *neighborhood*. *Neighborhood* yang digunakan yakni *Complete Move* (CM), *Complete Swap* (CS), *Partial Move* (PM), dan *Partial Swap* (PS).

Tabel 6.16 Rata-rata Penalti per Berbagai Kombinasi Neighborhood – Random Selection

No	*Mean*			
	CM	CM,CS	CM,CS,PM	CM,CS,PM,PS
0	1049,7	989,1	978,9	991,6
1	853,5	803,1	769,6	788,5
2	1464,1	1310,9	1307,3	1316
3	1024,8	955,1	948,6	957,5
4	1635,5	1406,0	1434,1	1410,2
5	696,3	674,9	675,2	674,9
6	963,2	911,8	913,2	926,7

Dengan menggunakan pemilihan acak, konfigurasi CM, CS, dan PM memberikan penalti terbaik di 4 *instance*, dengan 2 *instance* lain memiliki penalti terbaik menggunakan konfigurasi CM dan CS, dan satu *instance* lain yakni *instance 5* memilki

hasil penalti terbaik oleh konfigurasi CM-CS dan konfigurasi CM-CS-PM-PS. Penggunaan kombinasi *neighborhood* memberikan hasil yang lebih baik bila dibandingkan dengan penggunaan hanya satu *neighborhood* (CM). Perbedaan nilai penalti antar *neighborhood* juga menjadi lebih kecil bila dibandingkan dengan perbedaan pada uji coba sebelumnya.

Tabel 6.17 Rata-rata Penalti per Berbagai Kombinasi Neighborhood – VNS

No	*Mean*			
	CM*	CM,CS	CM,CS,PM	CM,CS,PM,PS
0	1049,7	980,4	1011,2	1031,5
1	853,5	788,7	800,2	797,9
2	1464,1	1316,7	1342,6	1347,9
3	1024,8	961,1	967,1	985
4	1635,5	1444,6	1488,5	1501,5
5	696,3	672,5	673,6	675,5
6	963,2	917,2	931,1	926,2

Uji coba VNS dalam tahap ini menggunakan hirarki CM>CS>PM>PS. Kebanyakan percobaan memberikan hasil yang lebih buruk bila dibandingkan dengan pemilihan acak. Konfigurasi CM dan CS memberikan hasil penalti terbaik di semua *instance*. Walaupun demikian, hanya *instance 5* yang menghasilkan penalti terbaik bila dibandingkan dengan penalti-penalti terbaik yang dihasilkan oleh metode pemiilihan acak. Pada kombinasi 2 *neighborhood*, VNS menghasilkan rata-rata kenaikan penalti sebanyak 0,2%. Pada kombinasi 3 *neighborhood* dan 4 *neighborhood*, VNS berturut-turut menghasilkan rata-rata kenaikan penalti sebanyak 2,49% dan 2,43%. Hal ini menunjukkan bahwa dengan menggunakan kombinasi *neighborhood* yang sama, penggunaan metode pemilihan acak menghasilkan penurunan penalti relatif lebih baik daripada VNS. Dalam hal ini, *neighborhood* yang diutamakan dalam konfigurasi VNS mulai sulit memperoleh

gerakan yang dapat menurunkan penalti lebih lanjut, dan pemanggilan *neighborhood* di hirarki bagian bawah yang juga dapat menjadi lebih mahal dibandingkan dengan pemilihan acak. Hasil yang lengkap dapat dilihat di LAMPIRAN D.

6.6. Hasil Uji Coba 3: Eksekusi Selama 3600S

Skenario uji coba dijalankan selama 3600 detik, menggunakan konfigurasi yang sama yang digunakan dipercobaan 400 detik. Selain keempat *neighborhood* yang diujicobakan dalam skenario sebelumnya, *neighborhood* baru juga diujicobakan di tahap ini. *Neighborhood* tersebut adalah *Deterministic Move* (DM), *Deterministic Swap* (DS), dan *Partial Swap 2* (PS2). Pemilihan pasien di *neighborhood* DM dan DS tetap dilakukan secara acak, namun pergerakan dilakukan secara deterministik menggunakan heuristik *steepest-descent*. PS2 memilki implementasi yang sedikit berbeda, dimana biaya transfer dipastikan hanya ditimbulkan maksimal 1x. Uji coba dilakukan satu kali per konfigurasi *neighborhood*. Urutan hirarki VNS dalam percobaan ini ditentukan dengan meninjau pada distribusi penaltinya di model pemilihan acak. Selain itu, setiap kombinasi *neighborhood* dijalankan sebanyak satu kali.

6.6.1. Dua *Neighborhood*

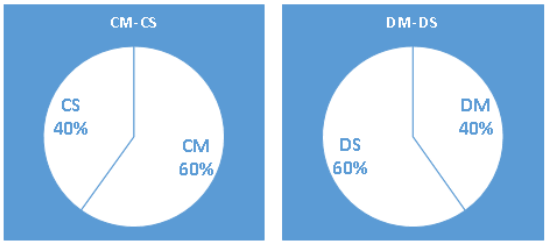
Uji coba dilakukan menggunakan dua kombinasi *neighborhood*, yakni kombinasi CM dan CS dan kombinasi DM dan DS.

Tabel 6.18 Perolehan Penalti dengan Kombinasi 2 Neighborhood

No	Penalti			
	Random Selection		*VNS*	
	CM, CS	DM, DS	CM, CS	DM, DS
0	921,6	919,2	894,8	935,2
1	739,6	724	727,2	721,2
2	1236,4	1241,8	1264	1237
3	882,4	919,4	852,2	902

4	1314,6	1303,2	1318	1295,8
5	661,6	655,2	672	664
6	876,8	871,2	874,6	862

VNS menghasilkan 5 dari 7 *instance* dengan nilai penalti terbaik, dengan 2 *instance* dihasilkan oleh konfigurasi CM dan CS, dan 3 *instance* dihasilkan oleh konfigurasi DM dan DS. Dua *instance* lain memiliki nilai penalti terbaik yang dihasilkan oleh masing-masing kombinasi CM, CS dan kombinasi DM, DS menggunakan pemilihan acak. Kebanyakan *instance* memiliki peningkatan kualitas penalti ketika menggunakan VNS. Untuk urutan hirarki VNS, diperoleh dengan melihat distribusi penalti per *neighborhood* (lihat Gambar 6.6).



Gambar 6.6 Distribusi Total Penalti per Neighborhood – Random Selection (2N)

Dengan menggunakan pencarian stokastik, *move* mendominasi penurunan penalti. Sebaliknya, *swap* mendominasi penurunan penalti ketika pencarian stokastik digunakan.

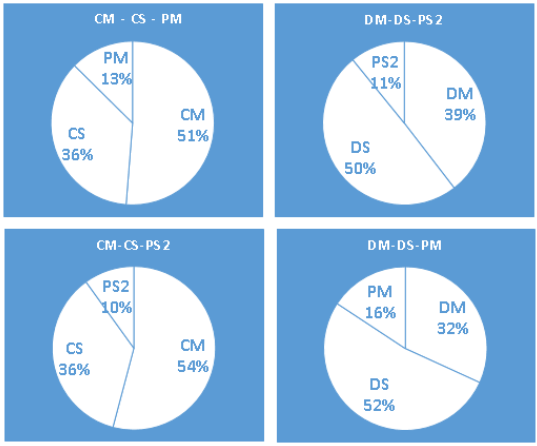
6.6.2. Tiga *Neighborhood*

Uji coba dilakukan dengan menggunakan menggunakan kombinasi *neighborhood* CM, CS, PM, PS, dan juga *neighborhood* baru DM, DS, dan PS2. Tidak semua kombinasi dan permutasi diujicobakan di tahap ini.

Tabel 6.19 Perolehan Penalti dengan Kombinasi 3 Neighborhood

No	Penalti							
	Random Selection				*VNS*			
	CM,CS,PM	CM,CS,PS2	DM,DS,PM	DM,DS,PS2	CM,CS,PM	CM,CS,PS2	DM,DS,PM	DM,DS,PS2
0	918,2	898	908,4	937,2	908,2	918,6	894,4	907,4
1	747,2	710	713,6	723,2	728,8	733,2	733,2	714
2	1233,8	1239,6	1225,2	1242,4	1223,6	1265	1236,2	1231
3	913,4	913	882,6	917	916,8	866,4	901	903
4	1335,4	1336,8	1380,8	1321	1343	1354,4	1313,8	1336,4
5	664,8	660	661,6	668	660,8	663,2	662,4	666,4
6	880,2	878,6	874,2	872,6	884,2	881,4	866,6	876,6

Dengan menggunakan kombinasi 3 *neighborhood*, hasil penalti terbaik untuk 5 *instance* diperoleh oleh VNS. Kombinasi DM, DS, PM memberikan hasil terbaik untuk 3 *instance*. Tidak terdapat perbedaan yang signifikan antara penggunaan VNS dengan pemilihan acak, dengan keduanya sama-sama memberikan 28 hasil terbaik bila dibandingkan dengan konfigurasi yang sama tetapi dengan pemilihan *neighborhood* yang berbeda. Urutan hirarki yang digunakan di VNS mengikuti Gambar 6.7.



Gambar 6.7 Distribusi Total Penalti per Neighborhood - Random Selection (3N)

Dengan pemilihan acak, *neighborhood* CM mendominasi pada konfigurasi CM,CS, PM dan konfigurasi CM,CS, PS2, dan *neighborhood* DS mendominasi pada konfigurasi DM,DS, PM, dan DM, DS, PS2. Pergerakan *move* mendominasi pada kombinasi *neighborhood* yang menggunakan versi pencarian stokastik, dengan pergerakan *swap* mendominasi pada kombinasi *neighborhood* yang menggunakan versi pencarian deterministik. Selain itu, terlihat bahwa PM memiliki kontribusi penurunan penalti yang lebih tinggi bila dibandingkan dengan PS2.

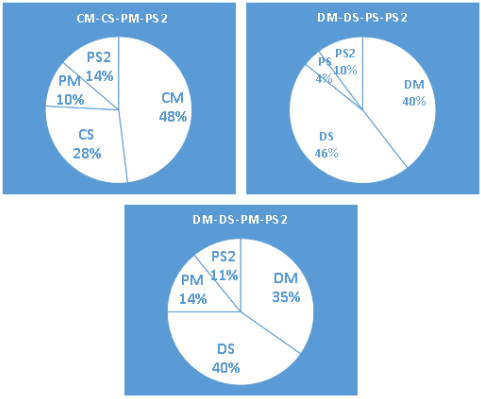
6.6.3. Empat *Neighborhood*

Uji coba optimasi dilakukan menggunakan kombinasi 4 *neighborhood*. Tidak semua kombinasi *neighborhood* yang ada diujicobakan.

Tabel 6.20 Perolehan Penalti dengan Kombinasi 4 Neighborhood - Random Selection

No	Penalti				
	CM,CS,PM,PS	CM,CS,PM,PS2	CM,CS,PS,PS2	DM,DS,PM,PS2	DM,DS,PS,PS2
0	906,4	914	900,2	938,4	902,4
1	745,6	724	716,4	720,8	737,6
2	1258,6	1246,8	1216,4	1246,2	1239,2
3	914,8	931,8	886,2	914,2	900,6
4	1311,2	1366,8	1317	1388,2	1344,2
5	669,6	672,8	670,4	663,2	665,6
6	892,8	868,8	881,8	873,6	880,4

Kombinasi CM, CS, PS, dan PS2 menghasilkan penalti terbaik di 4 *instance* pertama, dengan 3 *instance* lain memiliki penalti terbaik yang berturut-turut dihasilkan oleh kombinasi CM,CS,PM,PS, kombinasi DM,DS,PM,PS2, dan kombinasi CM,CS,PM,PS2. Penentuan hirarki pada VNS mengacu pada Gambar 6.8, dengan pengecualian pada konfigurasi CM,CS,PM, dan PS yang langsung menggunakan urutan CM>CS>PM>PS sesuai dengan hasil di percobaan awal. Urutan pada konfigurasi CM,CS,PS, dan PS2 yakni CM>CS>PS2>PS, karena memang performa PS2 yang lebih baik dibanding dengan performa PS.



Gambar 6.8 Distribusi Total Penalti per Neighborhood – Random Selection (4N)

Selanjutnya, hasil uji coba VNS dengan menggunakan kombinasi *neighborhood* yang sama dapat dilihat di tabel berikut.

Tabel 6.21 Perolehan Penalti dengan Kombinasi 4 Neighborhood - VNS

No	Penalti				
	CM,CS,PM,PS	CM,CS,PM,PS2	CM,CS,PS,PS2	DM,DS,PM,PS2	DM,DS,PS,PS2
0	941,6	898,8	919	915,8	906,6
1	733,2	752,2	740,6	736,8	703,2
2	1255	1264,4	1224,6	1250,2	1243,6
3	897,6	923,2	915,4	917,4	908,6
4	1378,4	1331,4	1331,8	1357,8	1342,2
5	668,8	666,4	666,4	662,4	668,8
6	902,2	877,6	884,6	875,6	871,2

Dengan menggunakan VNS, perolehan penalti terbaik tersebar di konfigurasi yang berbeda-beda. Bila dibandingkan dengan *random selection*, versi VNS, hanya 14 dari total 35 uji coba yang mengalami peningkatan kualitas solusi.

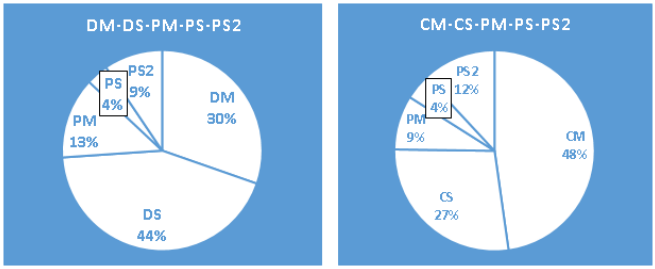
6.6.4. Lima *Neighborhood*

Uji coba pada tahap ini dijalankan menggunakan konfigurasi 5 *neighborhood*. Hasil yang diperoleh dapat dilihat pada Tabel 6.22.

Tabel 6.22 Perolehan Penalti dengan Kombinasi 5 Neighborhood

No	Penalti			
	Random Selection		VNS	
	CM,CS, PM,PS, DS2	DM,DS, PM,PS, DS2	CM,CS, PM,PS, DS2	DM,DS, PM,PS, DS2
0	938,6	920,2	937,4	938,6
1	722,4	714	751,4	723,8
2	1260,8	1240,2	1266,8	1239,6
3	919,2	918	923	944,6
4	1357,4	1359,8	1369,6	1346,2
5	663,2	654,4	659,2	666,4
6	879,6	875	876,2	872,8

Urutan hirarki VNS mengacu pada Gambar 6.9. Dengan menggunakan 5 *neighborhood*, kombinasi DM,DS,PM,PS, dan PS2 dengan pemilihan acak menghasilkan solusi terbaik di 4 *instance*. Tiga *instance* lain memiliki solusi terbaik yang dihasilkan oleh kombinasi yang sama dengan VNS. Dengan menggunakan kombinasi 5 *neighborhood*, kombinasi DM-DS memberikan hasil penalti yang lebih baik bila dibandingkan dengan konfigurasi CM-CS. Selain itu, pnggunaan VNS hanya berhasil menurunkan penalti di 6 dari 14 percobaan.



Gambar 6.9 Distribusi Total Penalti Per Neighborhood - Random Selection (5N)

6.6.5. Evaluasi Keseluruhan

Evaluasi dilakukan dengan melihat pada keseluruhan hasil yang diperoleh pada uji coba dengan durasi 3600 detik.

Tabel 6.23 Perbandingan Perolehan Penalti pada Kombinasi Neighborhood Terbaik di Pemilihan Acak dengan Kombinasi Neighborhood Terbaik di VNS

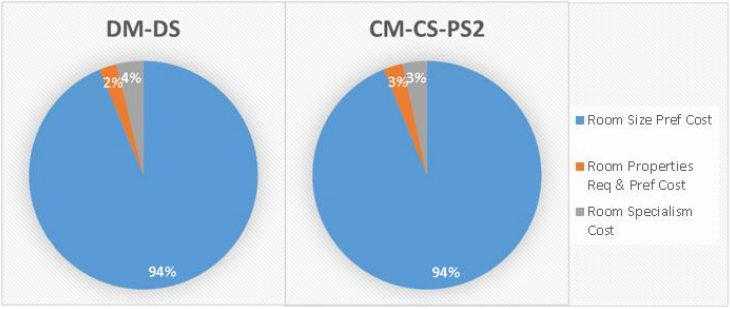
No	*Rand* - CM, CS, PS, PS2	VNS - CM, CS
0	900,2	894,8
1	716,4	727,2
2	1216,4	1264
3	886,2	852,2
4	1317	1318
5	670,4	672
6	881,8	874,6

Kombinasi *neighborhood* CM,CS,PS, dan PS2 dengan pemilihan acak memberikan rata-rata penurunan penalti sebanyak 47,43% dari penalti solusi awal, dengan kombinasi *neighborhood* CM dan CS dengan VNS memberikan rata-rata penurunan penalti sebanyak 47,37% dari penalti solusi awal. Perbedaan yang diperoleh relatif tidak signifikan, sehingga memang tidak terdapat bukti yang kuat bahwa salah satu metode lebih baik dari pada metode yang lain.

Tabel 6.24 Perolehan Penalti Terbaik dalam Uji Coba 3600 detik

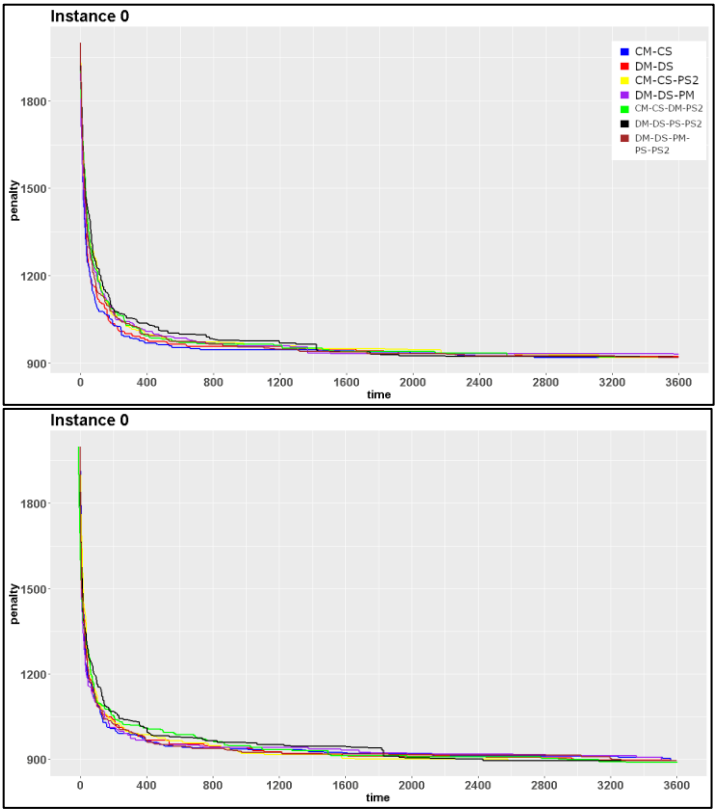
No	*Random Selection*			VNS		
	Neighborhood	Pen	%	*Neighborhood*	Pen	%
0	CM,CS,PS2	898	55,11	DM,DS,PM	894,4	55,29
1	CM,CS,PS2	710	46,52	DM,DS,PS, PS2	703,2	47,03
2	CM,CS,PS,PS2	1216,4	51,11	CM,CS,PM	1223,6	50,82
3	CM,CS	882,2	52,26	CM,CS	852,2	53,88
4	DM,DS	1303,2	53,97	DM,DS	1295,8	54,23
5	DM,DS,PM,PS, PS2	654,4	28,40	CM,CS,PM,PS, PS2	659,2	27,88
6	CM,CS,PM, PS2	868,8	48,44	DM,DS	862	48,84

Algoritma VNS menghasilkan solusi terbaik di 5 *instance*. Bila dibandingkan dengan perolehan penalti dari solusi awal, *random selection* berhasil menurunkan penalti sebanyak 28,4% hingga 55,11%, sedangkan VNS berhasil menurunkan penalti sebanyak 27,84% hingga 55,29%. Distribusi penalti setelah optimasi dilakukan selama 3600 detik diperoleh hanya pada 2 sampel, yakni VNS DM-DS dan *random* CM-CS-PS2.



Gambar 6.10 Pie Chart Distribusi Total Penalti

Setelah dioptimasi selama 3600 detik, distribusi total penalti menjadi sangat didominasi oleh penalti pelanggaran preferensi ukuran ruangan pasien. Hal ini memang wajar, mengingat penalti tersebut memiliki bobot paling kecil, yakni 0,8. Penalti pelanggaran batasan jenis kelamin yang kontribusinya cukup signifikan pada solusi awal, sekarang sudah tidak dihasilkan oleh solusi. Trajektori penurunan penalti divisualisasikan dengan *instance* 0 untuk konfigurasi CM-CS, DM-DS, CM-CS-PS2, DM-DS-PM, CM-CS-PM-PS2, DM-DS-PS-PS2, dan DM-DS-PM-PS-PS2.



Gambar 6.11 Trajektori Penalti Random Selection (atas) dan VNS (bawah) untuk Instance 0

Penalti mengalami penurunan yang signifikan di 400 detik pertama, dengan penurunan yang relatif sedikit dari detik 2400 hingga waktu komputasi selesai. Dengan waktu komputasi 3600 detik, perbedaan nilai penalti antara metode VNS dengan metode pemilihan acak tidak memiliki perbedaan yang signifikan. Grafik trajektori keduanya pun terlihat relatif serupa. Selain itu, terlihat bahwa pada penggunaan *neighborhood* yang deterministik menurunkan penalti lebih cepat pada awal eksekusi optimasi. Selanjutnya batas bawah dari setiap *instance* dibandingkan dengan hasil batas bawah yang diperoleh di penelitian lain. Penelitian tersebut

menggunakan waktu eksekusi yang berbeda-beda, sesuai dengan atau lebih kecil dari waktu eksekusi yang diijinkan program *benchmark* di perangkat lunak yang digunakan.

Tabel 6.25 Perbandingan dan Persetanse Perolehan Batas Bawah Penalti dengan Penelitian Lain - 1

No	Hasil	Demeester et al.		Bilgin et al.		Ceschia et al.	
0	894,4	1446,4	61,8%	815,2	-8,9%		-
1	703,2		-	672,8	-4,3%	659,2	-6,3%
2	1216,4		-	1174,2	-3,5%	1143,6	-6,0%
3	852,2		-	797,8	-6,4%	776,6	-8,9%
4	1295,8		-	1219,6	-5,9%	1176,0	-9,2%
5	654,4		-	634,4	-3,1%	625,6	-4,4%
6	862		-	818,6	-5,0%	801,2	-7,1%

Batas bawah penalti yang diperoleh pada *instance 0* memiliki perbedaan yang sangat signifikan dengan batas bawah penalti yang diperoleh di penelitian oleh Demeester et al. Dengan membandingkan batas bawah penalti di penelitian Ceschia et al. [26] yang menggunakan *simulated annealing* dan probabilitas *neighborhood*, batas bawah penalti yang diperoleh 4,4% hingga 9,2% lebih buruk, Sedangkan pada penelitian Bilgin et al [6] yang menggunakan *hyper-heuristics*, batas bawah penalti yang diperoleh 3,1% hingga 8,9% lebih buruk.

Tabel 6.26 Perbandingan dan Persentase Perolehan Batas Bawah Penalti dengan Penelitian Lain - 2

No	Hasil	Range et al.		Bastos et al.	
0	894,4		-		-
1	703,2	654,4	-6,94%	651,2	-7,39%
2	1216,4	1130,4	-7,07%	1128	-7.27%
3	852,2	768,2	-9,86%	761,6	-10,63%
4	1295,8	1179	-9,01%	1151,6	-11,13%
5	654,4	624	-4,65%	624	-4,65%
6	862	792,7	-8,04%	789,4	-8,42%

Penelitian Range et al. [18] yang menggunakan pendekatan *column generation* dengan *constraint aggregation* memperoleh batas bawah penalti yang 4,65% hingga 9,86% lebih baik dari

batas bawah penalti pada penelitian ini. Penelitian Bastos et al. [12] yang menggunakan metode *mixed integer programming* memberikan hasil batas bawah penalti 4,65% hingga 11,13% lebih baik daripada batas bawah penalti yang diperoleh di penelitian ini.

Waktu komputasi yang digunakan oleh Demeester et al. adalah 4900 detik, Bilgin et al. ±3000 detik, Ceschia et al. ±400 detik, Range et al. 150-500 detik, dan Bastos et al. 3398-86400 detik. Metode *exact* yang digunakan oleh Bastos et al. memperoleh banyak penalti terbaik, tetapi waktu komputasi yang digunakan untuk kebanyakan *instance* relatif sangat lama (tidak dapat ditentukan mengingat metodenya *exact*), dengan waktu tercepat 3398 detik hanya untuk *instance 5*.

6.7. Hasil Uji Coba 4: Eksekusi Selama 4810S

Uji coba awal dijalankan menggunakan kombinasi CM,CS dan PS2 dengan pemilihan acak untuk melihat apakah terjadi penurunan nilai penalti bila dibandingkan dengan eksekusi 3600 detik.

Tabel 6.27 Perbandingan Perolehan Penalti Konfigurasi CM-CS-PS2

Waktu	0	1	2	3	4	5	6
4800S	913,4	726,6	1232,8	913	1297,2	665,6	874
3600S	898	710	1239,6	913	1336,8	660	878,6
%	1,71	2,34	-0,55	0,00	-2,96	0,85	-0,52

Keseluruhan perolehan penalti yang didapat tidak mengalami penurunan yang signifikan bila dibandingkan dengan eksekusi selama 3600 detik. *Instance* 2,4, dan 6 berturut-turut mengalami penurunan penalti sebanyak 0f,55%, 2,96%, dan 0,52%. Sedangkan *instance* 0, 1, dan 5 berturut-turut mengalami kenaikan penalti sebanyak 1,71%, 2,34%, dan 0,85%. Oleh karena penurunan yang tidak terlalu signifikan, percobaan lebih lanjut menggunakan kombinasi neighborhood lain tidak dilakukan. Perlu ditekankan juga bahwa percobaan untuk waktu 3600 detik dan 4800 detik memang hanya dijalankan 1 kali,

sehingga mungkin terdapat variasi yang berbeda ketika percobaan diulang atau sampel diperbesar. Hasil uji coba 3600 detik adalah hasil yang digunakan untuk dibandingkan dengan penelitian lain, karena memang terlihat pada waktu eksekusi 4800 detik algoritma sudah mengalami *stuck* (penalti yang diperoleh tidak mengalami penurunan yang signifikan).

BAB VII
KESIMPULAN DAN SARAN

7.1. Kesimpulan

Dari keseluruan pengerjaan Tugas Akhir, dibuat beberapa kesimpulan sebagai berikut:

a) Tugas Akhir ini telah berhasil mengimpelementasikan algoritma VNS untuk mengoptimasi solusi awal PASP, dengan penurunan penalti sebanyak 27,84% hingga 55,29%. Dengan meninjau pada beberapa sampel kombinasi *neighborhood*, solusi akhir yang dihasilkan memiliki pelanggaran di batasan preferensi ukuran ruangan pasien (SC7) dengan kontribusi yang signifikan (94%), dengan 6% lainnya dihasilkan oleh pelanggaran di batasan kebutuhan properti pasien (SC4), preferensi properti pasien (SC5), dan kebutuhan spesialisme pasien (SC6).

b) Hasil uji coba menunjukkan bahwa VNS memberikan kualitas solusi yang sedikit lebih baik ketika parameter waktu eksekusi yang digunakan pendek, dengan rata-rata penurunan penalti sebanyak 2% hingga 4,96% daripada penalti metode pemilihan acak pada waktu eksekusi 30 detik dan dengan kombinasi *neighborhood* yang sama. Dengan menggunakan waktu komputasi yang lebih lama, belum terdapat bukti yang kuat untuk menunjukkan bahwa VNS memberikan kualitas solusi yang lebih baik. Selain itu, penelitian ini belum mampu menghasilkan solusi yang sama dengan atau lebih baik daripada solusi terbaik yang diketahui, dimana solusi terbaik memiliki penalti 4,65% hingga 11,13% lebih rendah.

7.2. Saran

Saran yang diberikan berdasarkan hasil penelitian yang dilakukan untuk penelitian selanjutnya antara lain:

a) Peningkatan jumlah sampel dan penggunaan metode uji statistik. Dalam tugas akhir ini, perbandingan hanya semata-mata dilakukan dengan melihat sampel. Perbandingan yang baik dapat menggunakan metode statistik *f-test* dan *t-test* untuk melihat konfigurasi mana yang paling signifikan, dengan jumlah sampel yang lebih banyak.

b) Tidak semua kemungkinan permutasi *neighborhood* diujicobakan dalam tugas akhir ini. Ada kemungkinan permutasi *neighborhood* yang dapat menghasilkan penalti yang lebih baik.

c) Optimasi kode untuk meningkatkan efisien waktu eksekusi program. Beberapa diantaranya seperti metode *backtracking*, yang diimplementasikan dengan melakukan *deep copy* terhadap blok solusi sebelum mengalami perubahan. Walaupun memiliki waktu komputasi linear, *backtracking* dapat dilakukan lebih cepat dengan hanya mengubah bagian dari blok solusi yang berubah.

DAFTAR PUSTAKA

[1] S. Sharifi and K. Saberi, "Hospital Capacity Planning in Hospital Management: An Overview," *Indian Journal of Fundamental and Applied Life Sciences*, vol. 4, no. 2, pp. 512-521, 2014.

[2] C. Tsitsakis, P. Polychronidou and K. Anastasios, "The Problem of Capacity Management in Greek Public Hospitals," in *The Economics of Balkan and Eastern Europe Countries in the Changing World*, Athena, 2017.

[3] P. Gemmel and R. Van Dierdonck, "Admission scheduling in acute care hospitals: does the practice fit with the theory?," *International Journal of Operations and Production Management*, vol. 19, no. 9, pp. 863-878, 1999.

[4] P. Hansen and N. Mladenovic, "Variable Neighborhood Search," *Computers and Operation Research*, vol. 24, no. 11, pp. 1097-1100, 1997.

[5] P. Demeester, W. Souffriau, P. De Causmaecker and G. Vanden Berghe, "A Hybrid Tabu Search Algorithm for Automatically Assigning Patients To Beds," *Artificial Intelligence in Medicine*, vol. 48, no. 1, pp. 61-70, 2010.

[6] B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg and G. Vanden Berghe, "One Hyper-heuristic Approach to Two Timetabling Problems in Healthcare," *Journal of Heuristics*, vol. 18, no. 3, pp. 401-434, 2012.

[7] B. Rechel, S. Wright, J. Barlow and M. McKee, "Hospital Capacity Planning: from Measuring Stocks to Modelling Flows," *Bulletin of World Health Organization*, vol. 88, no. 8, pp. 632-636, 2010.

[8] W. Vancroonenburg, F. D. Croce, D. Goossens and F. C. Spieksma, "The Red-Blue Transportation Problem," *European Journal of Operational Research*, vol. 237, no. 2, pp. 814-823, 2014.

[9] J. A. M. Schreuder, "Combinatorial Aspects of Construction of Competition Dutch Professional Football Leagues,"

Discrete Applied Mathematics, vol. 26, pp. 109-119, 1992.

[10] New York University Langone Health, "Inpatient, Outpatient or Observation," 2013. [Online]. Available: https://nyulangone.org/files/Hospital-Visits-IP-OP-OBS-2013-revised.pdf. [Accessed 2 11 2019].

[11] I. Boudali and N. Mokhtar, "Harmony Search Approach for Patient Scheduling in Emergency Laboratories," in *2nd Global Conference on Artifical Intelligence*, Berlin, 2016.

[12] L. S. Bastos, J. F. Marchesi, S. Hamacher and J. L. Fleck, "A Mixed Integer Programming Approach to the Patient Admision Scheduling Problem," *European Journal of Operational Research*, vol. 273, no. 3, pp. 831-840, 2019.

[13] G. Fohler, "How Different are Offline and Online Scheduling?," Kaiserslautern, 2011.

[14] S. Ceschia and A. Schaerf, "Modeling and Solving the Dynamic Patient Admission Scheduling Problem Under Uncertainty," *Artifical Intelligence in Medicine*, vol. 56, no. 3, pp. 199-205, 2012.

[15] D. Ouelhadj and S. Petrovic, "Survey of Dynamic Scheduling in Manufacturing Systems," *Journal of Scheduling*, vol. 12, no. 4, pp. 417-431, 2008.

[16] M. Gendreau and J.-Y. Potvin, Handbook of Metaheuristics 2nd Edition, Springer, 2010.

[17] V. Smith Daniels, S. Schweikhart and D. Smith Daniels, "Capacity Management in Health Care Services," *Decision Sciences*, vol. 19, no. 4, pp. 889-919, 1988.

[18] T. M. Range, R. M. Lusby and J. Larsen, "A Column Generation Approach for Solving the Patient Admission Scheduling Problem," *European Journal of Operational Research*, vol. 235, no. 1, pp. 252-264, 2014.

[19] R. Guido, V. Sollina and D. Conforti, "Offline Patient Admission Scheduling Problems," in *Springer Proceedings in Mathematics & Statistics*, 2017.

[20] R. M. Lusby, M. Schwierz, M. T. Range and J. Larsen, "An Adaptive Large Neighborhood Search Procedure Applied to the Dynamic Patient Admission Scheduling Problem,"

Artificial Intelligence in Medicine, vol. 74, pp. 21-31, 2016.

[21] K. Sorensen, "Metaheuristics-The Metaphor Exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 1-16, 2013.

[22] J. Brimberg, N. Mladenovic, R. Todosijevic and D. Urosevic, "Solving the Capacitated Clustering Problem with Varible Neighborhood Search," *Annals of Operation Research*, vol. 272, no. 1, pp. 289-321, 2019.

[23] M. Affi, H. Derbel and B. Jarboui, "Variable Neighborood Search Algorithm for the Green Vehicle Routing Problem," *International Journal of Industrial Engineerinng Computations*, vol. 9, no. 1, pp. 195-204, 2017.

[24] M. Amous, S. Toumi, B. Jarboui and M. Eddaly, "A Variable Neighborhood Search Algorithm for the Capacitated Vehicle Routing Problem," *Electronic Notes in Discrete Mathematics*, vol. 58, no. 1, pp. 231-238, 2017.

[25] N. D. Angresti, A. Muklason and A. Djunaidy, "Hyper-heuristics untuk Penyelesaian Masalah Optimasi Lintas Domain dengan Seleksi Heuristik bedasarkan Variable Neighborhood Search," *Khazanah Informatika*, vol. 5 , no. 1, 2019.

[26] S. Ceschia and A. Schaerf, "Local Search and Lower Bounds for The Patient Admission Scheduling Problem," *Computers & Operations Research*, vol. 38, pp. 1452-1463, 2011.

[27] A. N. Fitri, "Analisis Waktu Tunggu Operasi Elektif Pasien Rawat Inap Di Instalasi Bedah Sentral Rumah Sakit Kanker Dharmais Tahun 2014," Universitas Indonesia, Jakarta, 2014.

"Halaman ini sengaja dikosongkan"

BIODATA PENULIS

Penulis memiliki nama lengkap Varian Elbert, lahir di Jakarta pada tanggal 24 November 1998. Penulis lulus taman kanak-kanak dan sekolah tingkat dasar di SD K Lemuel II daerah Jakarta Barat, lalu melanjutkan pendidikan di SMP dan SMA Regina Pacis. Setelah lulus SMA pada tahun 2016, penulis mendaftar dan diterima sebagai mahasiswa di Institut Teknologi Sepuluh Nopember (ITS) Surabaya lewat jalur SBMPTN.

Penulis pernah berpartisipasi di beberapa lomba atau kejuaraan tingkat daerah semasa duduk di bangku sekolah. Beberapa diantaranya seperti lomba lari di regional Jakarta, penyisihan Olimpiade Sains Nasional tahap awal, dan lain-lain. Semasa perkuliahan, penulis lebih banyak menghabiskan waktu untuk untuk kegiatan-kegiatan individual. Walaupun demikian, penulis juga tetap berpartisipasi di berbagai forum komunikasi ilmiah baik di tingkat regional hingga ke tingkat nasional. Salah satunya yakni *Ministry of Finance Festival 2019*.

Pada tahun 2019, penulis mengambil tugas akhir terkait optimasi pada laboratorium Rekayasa Digital dan Inteligensi Bisnis (RDIB) di jurusan Sistem Informasi ITS. Untuk keperluan penelitian, penulis dapat dihubungi lewat email varianelbert@tutanota.com.

“Halaman ini sengaja dikosongkan”

LAMPIRAN A:
HASIL VALIDASI SOLUSI AWAL MENGGUNAKAN PROGRAM VALIDATOR

Lampiran ini merupakan hasil validasi solusi awal menggunakan program *validator*. Penalti hanya akan dikalkulasi jika tidak terdapat pelanggaran *hard constraint*. Merujuk pada Tabel A.1.

Tabel A.1 Hasil Validasi Solusi Awal

Hasil Validasi
Instance 0
Violations on the age policy => 0.0 Violations on patient not being assigned to room of the correct specialism => 0.0 Violations on patient not being assigned to single room on medical reasons => 0.0 Violations on unnecessary transfers => 0.0 Violations on the gender policy => 145.0 Violations on room not having needed and preferred room properties => 781.0 Violations on the room preference of the patients => 1074.4 Violations on patient not being assigned to a department with the right specialism => 0.0 Total cost is 2000.4
Instance 1
Violations on the age policy => 0.0 Violations on patient not being assigned to room of the correct specialism => 0.0 Violations on patient not being assigned to single room on medical reasons => 0.0 Violations on unnecessary transfers => 0.0 Violations on the gender policy => 10.0 Violations on room not having needed and preferred room properties => 496.0 Violations on the room preference of the patients => 843.2 Violations on patient not being assigned to a department with the right specialism => 0.0 Total cost is 1349.2
Instance 2
Violations on the age policy => 0.0 Violations on patient not being assigned to room of the correct specialism => 0.0 Violations on patient not being assigned to single room on medical reasons => 0.0 Violations on unnecessary transfers => 0.0 Violations on the gender policy => 145.0 Violations on room not having needed and preferred room properties => 972.0 Violations on the room preference of the patients => 1371.2 Violations on patient not being assigned to a department with the right specialism => 0.0 Total cost is 2488.2
Instance 3
Violations on the age policy => 0.0 Violations on patient not being assigned to room of the correct specialism => 22.0 Violations on patient not being assigned to single room on medical reasons => 0.0 Violations on unnecessary transfers => 0.0 Violations on the gender policy => 145.0 Violations on room not having needed and preferred room properties => 684.0 Violations on the room preference of the patients => 996.8 Violations on patient not being assigned to a department with the right specialism => 0.0 Total cost is 1847.8
Instance 4

A-2

Hasil Validasi
Violations on the age policy => 0.0 Violations on patient not being assigned to room of the correct specialism => 2.0 Violations on patient not being assigned to single room on medical reasons => 0.0 Violations on unnecessary transfers => 0.0 Violations on the gender policy => 235.0 Violations on room not having needed and preferred room properties => 1340.0 Violations on the room preference of the patients => 1254.4 Violations on patient not being assigned to a department with the right specialism => 0.0 Total cost is 2831.4
Instance 5
Violations on the age policy => 0.0 Violations on patient not being assigned to room of the correct specialism => 0.0 Violations on patient not being assigned to single room on medical reasons => 0.0 Violations on unnecessary transfers => 0.0 Violations on the gender policy => 0.0 Violations on room not having needed and preferred room properties => 170.0 Violations on the room preference of the patients => 744.0 Violations on patient not being assigned to a department with the right specialism => 0.0 Total cost is 914.0
Instance 6
Violations on the age policy => 0.0 Violations on patient not being assigned to room of the correct specialism => 57.0 Violations on patient not being assigned to single room on medical reasons => 0.0 Violations on unnecessary transfers => 0.0 Violations on the gender policy => 110.0 Violations on room not having needed and preferred room properties => 522.0 Violations on the room preference of the patients => 996.0 Violations on patient not being assigned to a department with the right specialism => 0.0 Total cost is 1685.0

LAMPIRAN B:
HASIL UJI COBA AWAL MENGGUNAKAN SATU NEIGHBORHOOD

Lampiran ini merupakan hasil lengkap uji coba optimasi menggunakan satu *neighborhood*. Merujuk pada Tabel B.1.

Tabel B.1 Hasil Uji Coba Awal 30S – Satu Neighborhood 10 kali

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
0	*Complete Move* (CM)	1501.6	1520.0	1501.0	1515.2	1511.6	1534.8	1568.2	1494.8	1574.0	1494.6
1		1214.6	1203.4	1194.6	1216.0	1208.2	1209.6	1210.4	1200.0	1208.8	1208.8
2		2029.6	2019.4	1967.6	1978.0	2027.4	1981.2	2062.6	2035.6	2034.6	2031.0
3		1406.6	1385.2	1463.2	1458.2	1455.0	1500.4	1442.6	1470.6	1443.4	1428.2
4		2282.8	2241.6	2202.8	2249.0	2233.8	2195.8	2282.0	2289.2	2247.0	2179.2
5		823.6	827.6	830.0	838.8	830.0	823.6	834.0	829.2	828.4	830.8
6		1471.4	1483.8	1461.4	1414.8	1440.0	1482.6	1477.2	1487.4	1485.0	1450.6
0	*Complete Swap* (CS)	1501.6	1520.0	1501.0	1515.2	1511.6	1534.8	1568.2	1494.8	1574.0	1494.6
1		1214.6	1203.4	1194.6	1216.0	1208.2	1209.6	1210.4	1200.0	1208.8	1208.8
2		2029.6	2019.4	1967.6	1978.0	2027.4	1981.2	2062.6	2035.6	2034.6	2031.0

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
3	*Complete Swap* (CS)	1406.6	1385.2	1463.2	1458.2	1455.0	1500.4	1442.6	1470.6	1443.4	1428.2
4		2282.8	2241.6	2202.8	2249.0	2233.8	2195.8	2282.0	2289.2	2247.0	2179.2
5		823.6	827.6	830.0	838.8	830.0	823.6	834.0	829.2	828.4	830.8
6		1471.4	1483.8	1461.4	1414.8	1440.0	1482.6	1477.2	1487.4	1485.0	1450.6
0	*Partial Move* (PM)	1501.6	1520.0	1501.0	1515.2	1511.6	1534.8	1568.2	1494.8	1574.0	1494.6
1		1214.6	1203.4	1194.6	1216.0	1208.2	1209.6	1210.4	1200.0	1208.8	1208.8
2		2029.6	2019.4	1967.6	1978.0	2027.4	1981.2	2062.6	2035.6	2034.6	2031.0
3		1406.6	1385.2	1463.2	1458.2	1455.0	1500.4	1442.6	1470.6	1443.4	1428.2
4		2282.8	2241.6	2202.8	2249.0	2233.8	2195.8	2282.0	2289.2	2247.0	2179.2
5		823.6	827.6	830.0	838.8	830.0	823.6	834.0	829.2	828.4	830.8
6		1471.4	1483.8	1461.4	1414.8	1440.0	1482.6	1477.2	1487.4	1485.0	1450.6
0	*Partial Swap* (PS)	1501.6	1520.0	1501.0	1515.2	1511.6	1534.8	1568.2	1494.8	1574.0	1494.6
1		1214.6	1203.4	1194.6	1216.0	1208.2	1209.6	1210.4	1200.0	1208.8	1208.8
2		2029.6	2019.4	1967.6	1978.0	2027.4	1981.2	2062.6	2035.6	2034.6	2031.0
3		1406.6	1385.2	1463.2	1458.2	1455.0	1500.4	1442.6	1470.6	1443.4	1428.2
4		2282.8	2241.6	2202.8	2249.0	2233.8	2195.8	2282.0	2289.2	2247.0	2179.2

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
5	*Partial Swap* (PS)	823.6	827.6	830.0	838.8	830.0	823.6	834.0	829.2	828.4	830.8
6		1471.4	1483.8	1461.4	1414.8	1440.0	1482.6	1477.2	1487.4	1485.0	1450.6

“Halaman ini sengaja dikosongkan”

LAMPIRAN C:
HASIL UJI COBA 30 DETIK MENGGUNAKAN GABUNGAN NEIGHBORHOOD

Lampiran ini berisikan hasil uji coba 400 detik menggunakan kombinasi *neighborhood*. Hirarki pada VNS menggunakan urutan CM>CS>PM>PS. Merujuk pada Tabel C.1 untuk *random selection*, dan Tabel C.2 untuk VNS.

Tabel C.1 Hasil Uji Coba 30 Detik – Random Selection 10 kali

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
0	CM, CS	1432.6	1346.2	1388	1316.4	1349.4	1401.4	1388.4	1327.6	1404.6	1382.2
1		1027.6	974.8	1007.6	982	1025.2	999.2	932.4	990.8	1023.8	1004
2		1783.4	1723.8	1755.6	1787.6	1698.8	1729.8	1742.8	1763.2	1714.2	1720.2
3		1300.8	1270.8	1260.8	1280.2	1201.4	1205.2	1160.8	1283.8	1250.8	1265.2
4		1894.4	1881.8	1941.2	1966.8	1912.8	1960.8	2027	1912.6	1918.8	2015.6
5		720.4	729.6	720.8	724.8	732	722	704.8	727.6	735.2	721.2
6		1118.4	1126.8	1111	1112.6	1073.8	1103.2	1118.8	1113.4	1083.2	1111.2
0	CM, PM	1529	1510.2	1432	1478.2	1456.2	1452.2	1507.6	1455.6	1451.4	1441.4
1		1087.2	1037.2	1102.6	1107.2	1135.4	1059.2	1066.4	1050	1076	1032.6

C-2

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
2	CM, PM	1826.8	1887	1893.2	1879	1859	1899.6	1855.6	1832.2	1904.6	1844.4
3		1277.2	1325.8	1366.8	1356	1316.8	1299	1327.6	1294.4	1342.4	1304.2
4		2133	2208.4	2083.2	2045.4	2063.4	2051.2	2051.6	2096.8	1967.2	2066
5		773.6	748.2	757.8	752.8	761	747.8	758.4	737.2	782.4	764.6
6		1214.8	1179.2	1206	1229	1187.6	1177.8	1193.2	1217.6	1181.8	1218.4
0	CM, PS	1463.4	1412	1504.8	1457	1433.2	1417.4	1366.8	1455.8	1369.4	1509.8
1		1050.4	1050.6	1046.6	1029.8	1030.4	1036.8	1019.8	1057.8	980	1011.4
2		1818.4	1866.2	1845.4	1841.6	1833	1885.2	1813.8	1931.2	1944	1823.2
3		1407.2	1365.6	1340.6	1324.2	1297.2	1317.6	1267.4	1363.4	1329.4	1356.8
4		2125.8	2112.8	2152.6	2061.8	2226.2	2095.4	2051.6	2141.4	2125.4	2077.2
5		750.4	731	748.8	780.4	753.6	743.6	767.8	772	763	779.2
6		1253.6	1187.8	1230.6	1229.4	1258.2	1161.8	1177.2	1181.6	1182.4	1265.8
0	CS, PM	1574	1587.8	1598.8	1543.2	1621	1640.2	1612	1655.2	1632.8	1608.8
1		1072	1084.2	1048.4	1084.6	1115.8	1052.6	1037.6	1080.8	1119.6	1065.2
2		2026.2	2017.6	1917.8	1964.4	1942.4	2019	1932.8	1980.8	1905.4	1907.8
3		1321.2	1309.4	1406.8	1343.8	1390.8	1351.4	1441.8	1380.6	1355.2	1389.8

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
4	CS, PM	2231.2	2159.2	2176.8	2166.6	2231	2163.8	2198.2	2120.2	2236.8	2271.8
5		761.6	756	757.8	742.8	754.4	751.2	757.6	744	764	762.2
6		1239.4	1274	1232.8	1232.8	1230.4	1217.2	1263.6	1214	1225.4	1218.2
0	CS, PS	1794	1758.6	1799.2	1792.8	1797	1820	1805	1738.6	1741.6	1777.8
1		1193.6	1166.2	1177	1217.6	1190.8	1163.4	1160.2	1152	1161	1157.2
2		2305.4	2167.8	2177.4	2213.8	2139.6	2194.4	2250	2120.8	2117.6	2190.4
3		1606.6	1521	1559.4	1589.8	1607.2	1566.4	1555.8	1506.8	1569.4	1619.4
4		2483.8	2429.8	2465.6	2407.4	2518.4	2493.4	2414.4	2523.8	2399.4	2417.8
5		785.2	829.2	796.4	804.8	797.2	796.4	827.6	826.8	805.6	822.2
6		1367.4	1278	1344	1381	1358.8	1354.2	1301	1370	1396	1372.6
0	PM, PS	1714.4	1689.2	1679.4	1723	1684.2	1705.8	1701.4	1544.4	1718.6	1676.4
1		1133	1205	1174.4	1151.6	1137.8	1106	1146.4	1174.4	1156.8	1149.2
2		2166.2	2202	2197	2090.2	2169.2	2211.6	2101	2127.8	2122	2167.8
3		1508.4	1548.8	1574.8	1513	1609	1520	1498.4	1528.2	1571.2	1512.6
4		2423	2271	2375.6	2466.6	2306.6	2359.6	2326.2	2333	2323.2	2456.8
5		828.6	829.2	826.4	835.6	839.4	817.4	801.2	824.4	809.8	831

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
6	PM, PS	1384.6	1382	1343.2	1413.2	1381.2	1429.6	1352.6	1444.4	1375	1328.8
0	CM, CS, PM	1496.8	1370.4	1404.8	1434.2	1385.4	1440.6	1381.2	1460.4	1536.4	1472.2
1		1050.6	989.4	995.6	1063.2	1040	1073	1014	1048.4	1024	1020.2
2		1883.2	1884.4	1830.4	1767.2	1772	1803.4	1758.4	1774	1734.6	1777.8
3		1346.8	1314.8	1212.8	1228.6	1188	1278.4	1268.4	1215.2	1283	1253
4		1966	2087.8	2071.4	2103.6	2008.4	2122.8	2015	1967.8	1967	2017.8
5		749.2	764	745.4	745	756.2	709.6	746	772	749.4	749.2
6		1143.4	1184.8	1156.8	1179.4	1199.6	1156.8	1161	1169	1179.4	1134.8
0	CM, CS, PS	1467.6	1467.6	1379.8	1472	1454.6	1452.8	1570.2	1507.2	1466.4	1463
1		1020.6	1020.6	1018.2	1015.4	1031.8	1018.6	1025.6	1013.4	1019.4	1039.4
2		1797	1797	1733.4	1811.2	1811	1856.8	1788	1850.6	1855.6	1929.8
3		1269.6	1269.6	1219.6	1316	1304	1259.6	1274	1280.2	1254.4	1276.4
4		2115.8	2115.8	2095.6	2065.8	2084.8	2072.2	2029.2	2147.6	1991.6	2126.8
5		726	726	745.4	748.6	727.6	729.6	741.4	720.8	752	769.6
6		1249.2	1249.2	1152.8	1207.2	1219.8	1219.6	1239.4	1206	1173.6	1163.8
0	CM, PM, PS	1549.6	1447.4	1628.2	1482.2	1577.4	1541.8	1548.4	1471	1435.8	1414.8

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
1	CM, PM, PS	1080.4	1089.8	1084	1087.6	1120.8	1070.4	1035.6	1048	1078.2	1038.4
2		2009.8	1947.2	1848.2	1984.4	1930.2	1905.2	1878.2	1960.4	1866.2	1948.6
3		1349.4	1467.2	1368.4	1357.4	1352.6	1309.8	1339.2	1425.4	1331.8	1360.8
4		2145.2	2139	2105.8	2114.4	2152.6	2256.8	2341.8	2071.6	2203.4	2158.2
5		772.8	742.6	739.2	804.8	766.8	768	800.4	756	769.4	786.2
6		1241.6	1206.4	1275	1234.6	1201.6	1267	1245.6	1194.2	1260.2	1233
0	CS, PM, PS	1620	1756.2	1641	1702.8	1644.8	1614	1690.6	1687.6	1671	1677.6
1		1106.6	1106.6	1049.6	1101.4	1083.6	1092.6	1074	1106.2	1086	1131
2		2111.6	2115.2	2061.6	2010.4	2042.2	2061.2	2080.4	2005.6	2148.4	1982.4
3		1422.4	1499.2	1489.8	1445.6	1490.8	1430.8	1427.2	1498.6	1408.4	1484.6
4		2438	2269.8	2249.4	2328.6	2285.4	2229.4	2341.6	2380.6	2307.2	2301.6
5		790.4	754.8	765.2	779.4	783.6	780.4	759.6	787	783	783.8
6		1326.4	1325.8	1348	1285.8	1253.2	1329.8	1302.4	1339	1292.4	1351.8
0	CM, CS, PM, PS	1640	1415.4	1543.6	1440.6	1455	1556.4	1442.8	1548.2	1526.6	1547
1		1069.4	1023	1020	1052.2	999	1036	1031.8	1056.2	970.8	1016.4
2		1870.4	1913.4	1912.4	1858.8	1910.4	1867.4	1890.6	1858.4	1910.6	1813.4

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
3	CM, CS, PM, PS	1373.4	1313.4	1325	1317	1296.8	1337	1327	1368.4	1291.6	1391.8
4		2125.6	2106.6	2051.8	2156.2	2089	2189.8	2044.4	2109	2024.8	2156.4
5		741.6	752	729.2	772.8	745.8	767.8	766.8	749.6	726.6	747.2
6		1263.6	1175.8	1217.2	1214.8	1188.6	1199	1118.4	1186.6	1202.8	1164.6

Tabel C.2 Hasil Uji Coba 30 Detik – VNS 10 kali

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
0	CM, CS	1407.6	1392.8	1322.8	1327.6	1364	1390.8	1374	1432.2	1381.6	1374
1		1014.4	1009.6	1012.8	1000.4	997.6	960.8	1009.2	1008.8	989.2	1009.2
2		1703.6	1736.4	1748	1730.8	1715.2	1819	1753	1688.8	1724	1753
3		1292.4	1259	1289.2	1242.6	1187.6	1222	1241.2	1234.6	1226.2	1241.2
4		1929	1978.4	2025.4	1973.2	1952.6	2002.4	2045	2027.8	1990.2	2045
5		728.6	712.2	734.4	741	750.4	719.6	737.8	729.2	717.6	737.8
6		1118.6	1165.8	1119.2	1151.8	1163.4	1138	1112.6	1149.8	1106.4	1112.6
0	CM, PM	1438.2	1381.2	1368.8	1363.8	1308	1454.6	1469.6	1443	1333	1407
1		1067.8	1040.6	1059.6	998.8	1014	1000.8	1020.8	991.4	1027.2	1024
2		1772	1741.2	1694.8	1796.6	1743	1788.6	1771.6	1780.2	1837.2	1790.4
3		1300.8	1279.4	1262	1240	1291.2	1360.2	1260.8	1302.8	1267	1269.2
4		2074.4	2041.2	1984.2	2011.8	1986.6	1926.8	2028.8	2068	2001.8	2057.4
5		745	752.2	730	732	743.8	740.6	723.6	737.2	747.2	723.6
6		1153.8	1142.8	1119	1108	1147	1119.4	1150	1136.2	1185.6	1160.2
0	CM, PS	1427.2	1372.2	1379.8	1396.6	1448	1351.6	1380.2	1344.2	1348	1339.8

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
1	CM, PS	1019.8	1027.2	1039	995.6	1025.4	1028.6	1013.6	999.8	1032	997.6
2		1760	1808.4	1760.6	1755.4	1773.2	1764.8	1783	1745.4	1721.4	1706.6
3		1232.2	1222	1277.2	1255.2	1250.6	1318	1268	1273.6	1269	1235.8
4		2020	1991.4	2030.4	2042.8	1963.2	2009.4	2026.8	1965.6	2013.6	2019
5		769	723.6	738.4	745.6	776.8	743.2	721.6	723.2	751.2	745.2
6		1158.4	1139.2	1190.6	1147	1157	1086.6	1141.2	1087.4	1130.8	1148.4
0	CS, PM	1606.8	1692	1670	1571.8	1670.2	1583.6	1629.6	1655	1626.2	1609.8
1		1046	1039.6	1053.2	1050.8	1050.8	1074.6	1071.2	1064.6	1085	1061.8
2		1947.4	2079.8	1850.4	2007.2	2071.4	1995	2009	2024.6	2022	2052
3		1369.4	1451	1385.6	1359.4	1391	1356.8	1406.2	1412.8	1416.8	1435.2
4		2250.4	2223.8	2190.4	2179.2	2343.2	2200.8	2130.6	2155.4	2142.2	2170.4
5		737.6	753.8	747.6	770.2	778	727.8	747.6	747.6	760.8	756.4
6		1250.6	1252.8	1254.4	1297.4	1220.4	1272.4	1260.4	1283.6	1277	1244.6
0	CS, PS	1717.2	1667.6	1772.8	1778.2	1791.4	1791.4	1714.4	1707.8	1702.2	1778.6
1		1162.8	1136.2	1157.4	1139.4	1126.6	1126.6	1104.8	1162.2	1125.4	1162.4
2		2045.6	2119.6	2069.2	2153	2107.2	2107.2	2151.2	2082	1981	2056

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
3	CS, PS	1459.4	1497.2	1454.8	1457.8	1421.2	1421.2	1444.8	1517	1478.2	1528.2
4		2255.8	2289.4	2296.6	2265.4	2358	2358	2237.6	2273.4	2315.2	2246
5		792	773.6	758.4	793.6	818.4	818.4	779.2	782	770.2	818.2
6		1256.4	1248	1290.6	1331.2	1270.2	1270.2	1320.4	1196.6	1231.6	1289.8
0	PM, PS	1741	1685.2	1714	1650.8	1696	1736.2	1687.8	1753.2	1683.2	1739
1		1170.6	1123.8	1106.8	1156.6	1167.6	1173.8	1132.4	1129.8	1175	1177.2
2		2106.6	2096	2098.8	2062	2131.4	2154.8	2091.4	2106	2154.2	2048.8
3		1490.8	1472.2	1479.8	1483.2	1450.6	1477.6	1525.8	1476	1511.6	1455.6
4		2321.4	2431.8	2316.8	2337.8	2331.4	2376.4	2322.6	2315	2348.6	2247.2
5		813.6	816	817.6	822.6	819.4	839.2	812.4	836	829	830
6		1344.2	1359.4	1359.2	1376.4	1351.8	1364.4	1355	1370.4	1389.2	1401.2
0	CM, CS, PM	1441.8	1358.2	1452.2	1389.6	1431.4	1418.2	1467	1436.2	1489.2	1453.8
1		1087.6	1036.4	1041	1010	1048	1019	1037	1023.4	996.6	1001.2
2		1790	1792.8	1760.2	1819.6	1879.4	1734	1869.2	1798	1853.4	1808
3		1307.2	1302.8	1296.2	1232	1245.8	1259	1349.8	1269.8	1327	1267.4
4		1961.4	1994.8	2004.8	2012.4	2066.6	1973.4	2070.4	2036.4	2016.6	2025.2

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
5	CM, CS, PM	755.8	749.4	742.8	740.6	746.2	718	723.4	753.2	735.2	751.8
6		1145.4	1163.8	1180.2	1172.6	1138.6	1177	1169	1170.4	1147	1129.4
0	CM, CS, PS	1420.4	1411	1340.6	1338	1331.6	1404	1403.4	1342.2	1376.8	1396.2
1		989.6	962.4	996.2	1039.2	1024.4	984.2	1015	1010.6	972.2	1007.2
2		1786.6	1733.4	1707.2	1770.2	1792.8	1725.8	1701.4	1709.2	1701	1752.6
3		1190.2	1220.4	1207.4	1279.8	1269.2	1201.2	1206.2	1251.8	1280.8	1271.8
4		1899	1887.2	1966.2	1997.8	2000.4	1963.2	1893.4	1946.4	2000	1956
5		730	720	734.8	715.4	739.2	725.6	726	750.8	728.4	733.2
6		1120.2	1168.8	1137.6	1089	1130.6	1135.2	1129.8	1130.6	1134.6	1175.2
0	CM, PM, PS	1434	1378.2	1347.2	1384.6	1441.8	1393.8	1381	1360.2	1329	1391.4
1		1032.2	1048.8	1067	1044.2	988.6	1047.8	1055.8	1026.6	1050.4	1036.2
2		1733.4	1745.2	1734.6	1793.8	1787.6	1805.2	1827.6	1763.2	1820	1823.8
3		1244.6	1255.6	1296.4	1265.6	1236.8	1264	1258.8	1286	1265	1267.8
4		1959.8	1987.6	2029.6	1978.6	2025.8	1987.6	1952	2016	1965.6	2023.4
5		724.6	720.4	736	756.2	731.6	747.2	751	733.2	730	723.6
6		1206.6	1132.4	1179.4	1099.6	1167	1143.2	1145.2	1136.4	1128.8	1166

Instance	*Neighborhood*	1	2	3	4	5	6	7	8	9	10
0	CS, PM, PS	1656.6	1658.8	1692.6	1723.6	1666	1604.8	1661.6	1671.6	1603	1594.6
1		1107.2	1075.6	1103.2	1103.8	1027.2	1080.6	1083.2	1057.8	1120.6	1091
2		2017.2	1974.6	1990.8	1983.8	1952.2	2041	1990.4	2057.4	2079.8	2017.8
3		1458.8	1416	1420.2	1367.4	1436.8	1467.8	1397.4	1439.2	1407.2	1441.2
4		2196.4	2311.8	2218.2	2272	2179.2	2144	2281.6	2199	2264.8	2266.8
5		770.2	778.4	762.8	752.6	765.4	749.8	776.4	752.2	741.2	769
6		1261.8	1288.4	1261	1259	1225	1231.6	1251	1218.4	1228.8	1237.2
0	CM, CS, PM, PS	1438	1375	1360.8	1355.8	1431.6	1394	1403.2	1411.6	1368.4	1322
1		994	1006.6	1031.6	1000	1009.2	1025	1017	984.4	969.2	1026.6
2		1736.2	1788	1680.6	1780.4	1761	1775	1805.8	1730.8	1807.6	1748
3		1247.8	1234.2	1245.6	1214.6	1276	1203.4	1247.4	1267	1257	1228.6
4		2000.4	2099.8	1950.2	2049.2	1991.8	1987.6	2143.2	2032	1960	1931
5		745.6	723.6	732.8	739.6	739	752.4	732.8	714.4	747.4	719.2
6		1159.6	1133.8	1143.6	1109	1124	1120.6	1105.6	1153.2	1154.4	1148.4

“Halaman ini sengaja dikosongkan”

LAMPIRAN D:
HASIL UJI COBA 400 DETIK MENGGUNAKAN GABUNGAN NEIGHBORHOOD

Lampiran ini berisikan hasil uji coba 400 detik menggunakan kombinasi *neighborhood*. Hirarki pada VNS menggunakan urutan CM>CS>PM>PS. Merujuk pada Tabel D.1.

Tabel D.1 Hasil Uji Coba 400 Detik 3 kali

Instance	Perpindahan	*Neighborhood*	1	2	3
0	*Random*	CM	1086	1036.8	1026.4
1			861.6	851.2	847.6
2			1470.2	1477.8	1444.2
3			1024.6	1049.6	1000.2
4			1567.6	1670.6	1668.2
5			697.6	696.8	694.4
6			958.8	947.6	983.2
0		CM, CS	986.6	978.4	1002.4
1			790.8	831.6	786.8
2			1305.6	1311.8	1315.2
3			975.8	949.8	939.6
4			1407.2	1396.8	1414
5			679.2	679.2	666.4
6			913	917.2	905.2
0		CM, CS, PM	995.6	979.2	962
1			772.8	777.6	758.4
2			1317.6	1294.6	1309.8
3			951	960.6	934.2
4			1486	1418.2	1398.2

D-2

Instance	Perpindahan	*Neighborhood*	1	2	3
5	*Random*	CM, CS, PM	674.4	671.2	680
6			893.6	931	915
0		CM, CS, PM, PS	1026.2	974	974.6
1			811.2	784.8	769.6
2			1305.8	1319	1323.2
3			968.8	957	946.8
4			1418.2	1369.2	1443.2
5			668	675.2	681.6
6			920.8	929.6	929.8
0	*VNS*	CM, CS	988.2	989.4	963.6
1			780	802	784
2			1344.4	1303.2	1302.6
3			970	967.6	945.8
4			1436.4	1450.6	1447
5			672	676.8	668.8
6			918.2	914	919.4
0		CM, CS, PM	995.2	1026.2	1012.2
1			784	805.6	811
2			1365.2	1308.2	1354.4
3			989.4	975.4	936.6
4			1484.8	1499	1481.8
5			682.4	672	666.4
6			924.2	939	930.2
0		CM, CS, PM, PS	1050.8	1015.6	1028
1			806.2	804.2	783.4
2			1353.2	1359.2	1331.2
3			964.2	971.8	1019

Instance	Perpindahan	*Neighborhood*	1	2	3
4	*VNS*	CM, CS, PM, PS	1484.2	1544.4	1475.8
5			672.8	678.4	675.2
6			922.6	938.8	917.2

“Halaman ini sengaja dikosongkan”

LAMPIRAN E:
HASIL UJI COBA 3600 DETIK MENGGUNAKAN GABUNGAN NEIGHBORHOOD

Lampiran ini berisikan hasil uji coba 3600 detik menggunakan kombinasi *neighborhood*. Hirarki pada VNS ditentukan dengan melihat pada performa *neighborhood* (lihat BAB 6.6). Merujuk pada Tabel E.1.

Tabel E.1 Hasil Uji Coba 3600 Detik

Perpindahan	*Neighborhood*	*Instance*						
		0	1	2	3	4	5	6
Random Selection	CM,CS	921.6	739.6	1236.4	882.4	1314.6	661.6	876.8
	DM,DS	919.2	724	1241.8	919.4	1303.2	655.2	871.2
	CM,CS,PM	918.2	747.2	1233.8	913.4	1335.4	664.8	880.2
	CM,CS,PS2	898	710	1239.6	913	1336.8	660	878.6
	DM,DS,PM	908.4	713.6	1225.2	882.6	1380.8	661.6	874.2
	DM,DS,PS2	937.2	723.2	1242.4	917	1321	668	872.6
	CM,CS,PM,PS	906.4	745.6	1258.6	914.8	1311.2	669.6	892.8
	CM,CS,PM,PS2	914	724	1246.8	931.8	1366.8	672.8	868.8
	CM,CS,PS,PS2	900.2	716.4	1216.4	886.2	1317	670.4	881.8
	DM,DS,PM,PS2	938.4	720.8	1246.2	914.2	1388.2	663.2	873.6
	DM,DS,PS,PS2	902.4	737.6	1239.2	900.6	1344.2	665.6	880.4
	CM,CS,PM,PS,PS2	938.6	722.4	1260.8	919.2	1357.4	663.2	879.6

E-2

Perpindahan	*Neighborhood*	*Instance*						
		0	1	2	3	4	5	6
Random Selection	DM,DS,PM,PS,PS2	920.2	714	1240.2	918	1359.8	654.4	875
VNS	CM,CS	894.8	727.2	1264	852.2	1318	672	874.6
	DM,DS	935.2	721.2	1237	902	1295.8	664	862
	CM,CS,PM	908.2	728.8	1223.6	916.8	1343	660.8	884.2
	CM,CS,PS2	918.6	733.2	1265	866.4	1354.4	663.2	881.4
	DM,DS,PM	894.4	733.2	1236.2	901	1313.8	662.4	866.6
	DM,DS,PS2	907.4	714	1231	903	1336.4	666.4	876.6
	CM,CS,PM,PS	941.6	733.2	1255	897.6	1378.4	668.8	902.2
	CM,CS,PM,PS2	898.8	752.2	1264.4	923.2	1331.4	666.4	877.6
	CM,CS,PS,PS2	919	740.6	1224.6	915.4	1331.8	666.4	884.6
	DM,DS,PM,PS2	915.8	736.8	1250.2	917.4	1357.8	662.4	875.6
	DM,DS,PS,PS2	906.6	703.2	1243.6	908.6	1342.2	668.8	871.2
	CM,CS,PM,PS,PS2	937.4	751.4	1266.8	923	1369.6	659.2	876.2
	DM,DS,PM,PS,PS2	938.6	723.8	1239.6	944.6	1346.2	666.4	872.8