TESIS - KI142502

DEKOMPOSISI LTL UNTUK OPTIMASI MODEL PROSES BISNIS TREE MINER PADA TERMINAL PETIKEMAS

AFINA LINA NURLAILI
NRP: 05111650010053

DOSEN PEMBIMBING
Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D

PROGRAM MAGISTER
BIDANG KEAHLIAN MANAJEMEN INFORMASI
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

TESIS - KI142502

DEKOMPOSISI LTL UNTUK OPTIMASI MODEL PROSES BISNIS TREE MINER PADA TERMINAL PETIKEMAS

AFINA LINA NURLAILI
NRP: 05111650010053

DOSEN PEMBIMBING
Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D
NIP: 195908031986011001

PROGRAM MAGISTER
BIDANG KEAHLIAN MANAJEMEN INFORMASI
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2018

Tesis disusun untuk memenuhi salah satu syarat memperoleh gelar
Magister Komputer (M.Kom.)
di
Institut Teknologi Sepuluh Nopember Surabaya

oleh:
AFINA LINA NURLAILI
NRP. 05111650010053

Dengan judul:
DEKOMPOSISI LTL UNTUK OPTIMASI MODEL PROSES BISNIS TREE MINER PADA TERMINAL PETIKEMAS

Tanggal ujian: 08-1-2018
Periode wisuda: 2018 Gasal

Disetujui oleh:

Prof. Drs.Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D (Pembimbing 1)
NIP. 19590803 198601 1 001

Dr. Ir. Raden Venantius Hari Ginardi,M.Sc (Penguji 1)
NIP. 19650518 199203 1 003

Dr.Eng Darlis Herumurti, S.Kom.,M.Kom (Penguji 2)
NIP. 19771217 200312 1 001

Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D (Penguji 3)
NIP. 19770824 200604 1 001

Dekan Fakultas Teknologi Informasi dan Komunikasi,

Dr.Agus Zainal Arifin, S.Kom, M.Kom
NIP. 19720809 199512 1 001

Dekomposisi LTL Untuk Optimasi Model Proses Bisnis Tree Miner Pada Terminal Petikemas

Nama mahasiswa: Afina Lina Nurlaili
NRP : 05111650010053
Pembimbing : Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D

ABSTRAK

Process discovery atau penemuan model proses adalah kegiatan mengamati perilaku dalam *event log* dan membangun model dari *event log*. Telah banyak metode *process discovery* yang diusulkan dalam beberapa tahun terakhir. Sebagian besar menghasilkan model dengan kualitas yang buruk karena event log yang besar. Maka dari itu perlu dilakukan dekomposisi model proses. Dekomposisi dilakukan setelah menentukan struktur kausalnya untuk menghasilkan bagian-bagian yang dapat dipakai ulang dalam membangun model proses. Keuntungan dari dekomposisi adalah dapat mempermudah proses analisis dan mudah dibangun kembali serta dapat mengurangi waktu komputasi.

Dekomposisi *Linear Temporal Logic* (LTL) akan digunakan sebagai dasar pembentukan model proses. LTL adalah notasi formal yang dapat menggambarkan relasi aktivitas yang berkaitan dengan waktu. LTL dipilih karena dapat menjelaskan relasi proses secara fleksibel sehingga analis dapat lebih mudah menganalisa dan memodifikasi. Sedangkan *process tree* digunakan untuk menggambarkan secara keseluruhan model proses yang dibentuk dari dekomposisi LTL. *Process tree* digunakan untuk memodelkan proses karena dapat menangani struktur blok dan memiliki kesamaan notasi dengan LTL. Hasil pembentukan model process tree menunjukkan bahwa *fitness, precision, simplicity,* dan *generalization* adalah 0.92, 0.68 , 0.87, dan 0.97

Penelitian ini juga mengusulkan optimasi waktu dan biaya untuk mengurangi *dwelling time*. *Dwelling time* adalah waktu berapa lama petikemas (barang impor) ditimbun di Tempat Penimbunan Sementara (TPS) di pelabuhan sejak dibongkar dari kapal sampai dengan barang impor keluar dari TPS. Optimasi dilakukan mengacu pada proses paralel barang dan dokumen yang telah diusulkan. Goal programming digunakan sebagai metode optimasi.

Kata kunci: *Process discovery;* dekomposisi LTL*; process tree; dwelling time;* optimasi waktu dan biaya; goal programming

[Halaman ini sengaja dikosongkan]

Decomposition LTL to Optimize Model Process Business Tree Miner at Port Container

Name : Afina Lina Nurlaili

NRP : 05111650010053

Pembimbing : Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D

ABSTRACT

Process discovery is the activity of observing the behavior in the event log and building the model of the event log. There have been many methods of process discovery proposed in recent years. Most produce models with poor quality due to large log events. Therefore it is necessary to decompose the process model. Decomposition is done after determining the causal structure to produce reusable parts in building process models. The advantage of decomposition is that it simplifies the analysis process and is easily rebuilt and can reduce computational time.

Linear Temporal Logic (LTL) decomposition will be used as the basis for forming process models. LTL is a formal notation that can describe time-related activity relationships. LTL is chosen because it can explain the process relation flexibly so that analysts can more easily analyze and modify. While the process tree is used to describe the overall process model formed from LTL decomposition. Process tree is used to model process because it can handle block structure and have similar notation with LTL. The result of forming a process tree model shows that fitness, precision, simplicity, and generalization are 0.92, 0.68 , 0.87, and 0.97.

The study also proposes time and cost optimization to reduce dwelling time. Dwelling time is how long the container (imported goods) has been stockpiled in Temporary Landfill (TPS) at the port since it was unloaded from the ship until the imported goods came out of the TPS. Optimization is done in reference to the parallel process of goods and documents that have been proposed. Goal programming is used as an optimization method.

Keywords: Process discovery; decomposition of LTL; process tree; dwelling time; optimation time and cost; goal programming

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Alhamdulillahirabbil'alamin. Puji dan syukur penulis panjatkan kehadirat Allah SWT atas berkat, rahmat dan hidayah-Nya, penyusunan Tesis ini dapat diselesaikan. Tesis ini dibuat sebagai salah satu syarat dalam menyelesaikan Program Studi Magister di Institut Teknologi Sepuluh Nopember Surabaya. Penulis menyadari bahwa Tesis ini dapat diselesaikan karena dukungan dari berbagai pihak, baik dalam bentuk dukungan moral dan material.

Melalui kesempatan ini dengan kerendahan hati penulis mengucapkan terima kasih dan penghargaan setinggi-tingginya kepada semua orang untuk semua bantuan yang telah diberikan, antara lain kepada:

1. Ibu dan bapak tercinta yang tiada henti selalu mendukung anaknya, tetap sabar mendengar keluhannya, selalu mendoakan anaknya yang terbaik dan selalu menjadi panutan yang baik.
2. Kakak dan adik yang selalu saja cerewet, sehingga dapat membangkitkan semangat penulis untuk mengerjakan thesis ini.
3. Bapak Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D selaku pembimbing yang senantiasa memberikan arahan dan bimbingan kepada penulis. Semoga Allah SWT senantiasa merahmati bapak dan keluarga.
4. Bapak Royyana M Ijtihadie, S.Kom.,M.Kom.,Ph.D, Dr. Ir. Raden Venantius Hari Ginardi,M.Sc., Dr.Eng Darlis Herumurti,S.Kom.,M.Kom. sebagai tim Penguji Tesis yang memberikan masukan dan kritik yang membangun untuk Tesis ini.
5. Seluruh dosen S2 Teknik Informatika yang telah memberikan ilmu dan pengetahuan kepada penulis selama menempuh studi.
6. Teman seperjuangan, Dewi Rahmawati, Kelly Rossa Sungkono, Yutika Amelia Effendi, Abd. Charis Fauzan dan sahabat serta teman lainnya yang tidak dapat disebutkan satu persatu, terima kasih atas bantuan dan motivasi yang telah diberikan.

Akhirnya dengan segala kerendahan hati penulis menyadari masih banyak terdapat kekurangan pada Tesis ini. Oleh karena itu, segala tegur sapa dan kritik

yang sifatnya membangun sangat penulis harapkan demi kesempurnaan Tesis ini. Penulis berharap bahwa perbuatan baik dari semua orang yang dengan tulus memberikan kontribusi terhadap penyusunan Tesis ini mendapatkan pahala dari Allah. Aamiin Alluhamma Aamiin.

Surabaya, Januari 2018

Penulis

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

BAB 1
PENDAHULUAN

1.1. Latar belakang

Process mining adalah metode dan teknik yang digunakan untuk mendukung proses bisnis dalam hal memodelkan, mengontrol, dan menganalisa proses yang melibatkan pelaku proses, organisasi, aplikasi, dokumen serta masukan informasi lainnya dengan memanfaatkan log data yang tercatat (Burattin, Maggi, & Sperduti, 2016). *Process mining* menggabungkan analisa di bidang data (*data mining* dan *machine learning*) dengan analisa pada model proses. *Process mining* terdiri dari tiga bagian yaitu penemuan model proses, pemeriksaan kualitas antara log data dengan model, dan peningkatan kualitas (Vahedian Khezerlou & Alizadeh, 2014). Bagian pertama dari *process mining* adalah penemuan model proses. Penemuan model proses menggali informasi dari log data dan membentuk model proses yang menggambarkan aktivitas sesuai informasi tersebut (Loreto & Vargas, 2012). Telah banyak metode *process discovery* yang diusulkan dalam beberapa tahun terakhir. Sebagian besar menghasilkan model dengan kualitas yang buruk karena event log yang besar (Verbeek, 2016). Maka dari itu perlu dilakukan dekomposisi model proses. Dekomposisi dilakukan setelah menentukan struktur kausalnya untuk menghasilkan bagian-bagian yang dapat dipakai ulang dalam membangun model proses. Keuntungan dari dekomposisi adalah dapat mempermudah proses analisis dan mudah dibangun kembali serta dapat mengurangi waktu komputasi.

Dekomposisi model proses dilakukan dengan menggunakan LTL. LTL digunakan karena berisi aturan-aturan yang menggambarkan relasi antar aktivitas berkaitan dengan waktu pelaksanaan aktivitas terhadap aktivitas lain sehingga dapat mengelompokkan aktivitas-aktivitas sesuai operator atau *gate* pada log data (Maggi & Westergaard, 2012). Setelah dilakukan penulisan notasi LTL berdasarkan relasi antar aktivitas, tahap selanjutnya yaitu penemuan model proses dengan menggali informasi dari data proses dan membentuk model proses yang menggambarkan aktivitas sesuai informasi tersebut. Pembentukan model proses dibentuk berdasarkan dekomposisi LTL. Bentuk model proses yang digunakan sebagai

bentuk penggabungan dekomposisi LTL adalah model *process tree*. Model ini dipilih karena model ini banyak digunakan untuk menggambarkan penguraian formula matematika, yang memiliki kemiripan dengan formula LTL, sehingga mempermudah penggabungan dekomposisi LTL (J. C. a M. Buijs, 2014). Algoritma yang diusulkan dapat secara efektif dan otomatis membedakan relasi sequence, single choice XOR, conditional OR dan parallel AND.

Pada penelitian ini tidak hanya memodelkan proses namun juga dilakukan analisis optimasi waktu dan biaya. Analisis optimasi waktu diperlukan karena *dwelling time* yang terlampau lama di Indonesia dibandingkan dengan negara-negara lain. *Dwelling time* adalah waktu yang dihitung mulai dari suatu petikemas (*container*) dibongkar dan diangkut dari kapal sampai petikemas tersebut meningggalkan terminal melalui pintu utama (Rizkikurniadi, 2014). Usulan metode paralel antar aktivitas dokumen dan barang diharapkan dapat membantu mengurangi waktu *dwelling time.* Optimasi dilakukan dengan menggunakan goal programming. Dimana Goal Programming merupakan salah satu metode yang biasa digunakan untuk mencari solusi dengan fungsi tujuan lebih dari satu (multi objective) (Levi, 2014). Satu pendekatan goal programming adalah untuk memenuhi goal dalam suatu urutan prioritas. Goal prioritas kedua ditetapkan tanpa mengurangi goal prioritas pertama. Untuk tiap tingkat prioritas, fungsi obyektifnya adalah meminimumkan jumlah dari simpangan. Goal programming pada penelitian ini digunakan untuk mengoptimasi waktu dan biaya. Dengan adanya optimasi waktu dan biaya dengan menggunakan goal programming, maka pelabuhan dapat mengurangi *dwelling time* sesuai dengan aturan pemerintah.

1.2. Perumusan Masalah

Rumusan masalah yang diangkat dalam penelitian ini adalah sebagai berikut.

1. Bagaimana membentuk dekomposisi LTL pada log data?
2. Bagaimana membentuk model *process tree* dari dekomposisi LTL?
3. Berapa optimasi waktu dan biaya agar didapatkan waktu dan biaya optimal dengan menggunakan *goal programming*?

1.3. Tujuan

Tujuan yang akan dicapai dalam pembuatan tesis ini adalah untuk mengetahui bahwa LTL dapat dimanfaatkan bukan hanya sebagai analisis kebenaran suatu kasus kelompok aktivitas namun dapat juga dimanfaatkan sebagai pembentuk potongan-potongan model sebelum ditemukan model proses secara keseluruhan. Pada penelitian ini juga bertujuan mengoptimalkan waktu dan biaya aktivitas pada setiap .

1.4. Manfaat

Manfaat dari penelitian ini adalah untuk mengetahui bahwa LTL dapat dimanfaatkan bukan hanya sebagai analisis kebenaran suatu kasus kelompok aktivitas namun dapat juga dimanfaatkan sebagai pembentuk potongan-potongan model sebelum ditemukan model proses secara keseluruhan. Pada penelitian ini juga bermanfaat mengoptimalkan waktu dan biaya aktivitas pada setiap *trace*.

1.5. Kontribusi Penelitian

Kontribusi penelitian ini adalah mengusulkan pendekatan dekomposisi model menggunakan LTL dan optimasi waktu dan biaya menggunakan *goal programming.* Dekomposisi LTL dilakukan pada kasus *process mining* dengan mengelompokkan aktivitas berdasarkan operator. Pada penelitian ini juga dilakukan analisis waktu dan biaya dengan memparalelkan aktivitas dokumen dengan barang. Hasil dari throughput time dari paralel dijadikan sebagai goal. Pada penelitian sebelumnya goal programming digunakan untuk menaikkan keuntungan dan produk suatu perusahaan, namun pada penelitian ini goal programming digunakan untuk mengurangi waktu dan biaya berseumber dari data event log.

1.6. Batasan Masalah

Untuk menghindari meluasnya permasalahan yang akan diselesaikan, maka dalam penelitian ini masalah akan dibatasi pada adalah:

1. Pengujian dilakukan pada studi kasus proses bisnis *container handling* di pelabuhan petikemas PT.TPS Surabaya dan menggunakan event log contoh yang terdapat operator AND dan OR

2. Implementasi penemuan model dan dekomposisi model proses dikerjakan dengan menggunakan bahasa Java dan Python
3. Dekomposisi model proses dilakukan hanya dengan menggunakan tiga operator yaitu AND, XOR, OR
4. Metode goal programming yang digunakan pada penelitian ini adalah *linear goal programming*.

BAB 2
KAJIAN PUSTAKA DAN DASAR TEORI

Kajian pustaka yang telah dilakukan adalah dengan melakukan ujicoba penelitian sebelumnya dan melakukan kajian terhadap literatur yang berhubungan. Penelitian yang dilakukan adalah pembentukan model proses dari model tree sebagai acuan dalam memparalelkan *trace* barang dengan dokumen dan optimasi waktu dan biaya. Pembentukan model proses dilakukan dengan menggabungkan dekomposisi LTL. Sedangkan optimasi waktu dan biaya dilakukan dengan menggunakan goal programming.

2.1 Log Data

Log data menggambarkan perilaku historis, dan informasi yang berupa keadaan dari alur kerja untuk menciptakan simulasi yang lebih akurat sehingga dapat memprediksi perilaku potensial masa depan dengan beberapa scenario (Alves de Medeiros, Weijters, & van der Aalst, 2006). Log data yaitu kumpulan urutan, dimana setiap urutan adalah urutan aktivitas yang terjadi pada model proses. Kejadian pada log data memiliki atribut umum untuk dianalisis yaitu:

1. Case ID merupakan nomor unik setiap kasus log data dan terdiri dari rangkaian aktivitas dari awal hingga akhir.
2. Originator merupakan *agent* yang bertugas menyelesaikan setiap aktivitas.
3. Aktivitas merupakan nama aktivitas– nama aktivitas dari setiap kejadian.
4. Time merupakan waktu eksekusi yang dicatat pada setiap aktivitas.
5. Cost merupakan biaya yang dikeluarkan pada setiap aktivitas.

2.2 Control Flow Pattern

Control flow pattern pertama kali diusulkan oleh Wil van der Aalst pada tahun 2003 dalam tulisan ilmiah berjudul "*Workflow Patterns*". Control flow pattern ini berfokus pada satu aspek spesifik dari pengembangan aplikasi berorientasi proses, yaitu deskripsi dependensi control-flow antara aktivitas dalam alur proses (Sarno, Wibowo, Effendi, & Sungkono, 2016). Terdapat tiga dasar control flow pattern

yaitu XOR-operator, AND –operator, dan OR-operator (Burattin et al., 2016). Seperti terlihat pada Tabel 3.17 XOR-operator memiliki satu sumber aktivitas yang sama, namun berbeda tujuan aktivitas. Sedangkan AND-operator memiliki satu sumber aktivitas yang sama dan semua tujuan aktivitas harus dilalui semua. OR-operator memiliki satu sumber aktivitas yang sama, namun dari semua aktivitas tujuan hanya dapat melalui n-1 aktivitas tujuan. Selain tiga operator penting, workflow pattern juga terdapat sequence pattern dan loop. Sequence berarti aktivitas harus dilalui sesuai urutannya sedangkan loop adalah perulangan aktivitas.

2.3 Process Tree

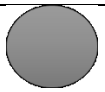
Process tree adalah notasi model proses yang bisa digunakan untuk memvisualisasikan dan mewakili model proses (M. Van Eck, 2013). Beberapa contoh notasi lain yang digunakan untuk menunjukkan perspektif aliran kontrol proses yaitu Business Process Model and Notation (BPMN), petri net, dan Event Driven Process Chains (EPC) (J. C. A. M. Buijs, Van Dongen, & Van Der Aalst, 2012). Process tree digunakan oleh algoritma ETM karena bahasa pemodelan tradisional seperti petri net, BPMN, dan diagram aktivitas UML memungkinkan terciptanya model yang tidak sempurna, yaitu model ini mengandung kebuntuan dan anomali lainnya (J. C. a M. Buijs, 2014). Process tree dijamin mewakili model proses sempurna, yang mengurangi ruang pencarian algoritma ETM (Vázquez-Barreiros, Mucientes, & Lama, 2014).

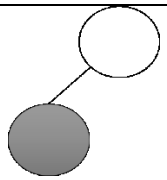
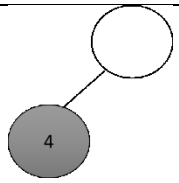
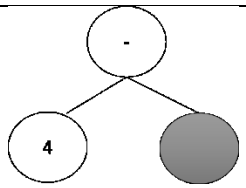
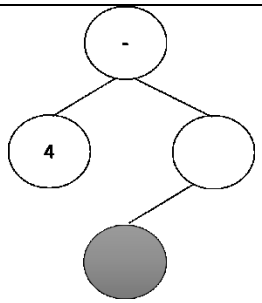
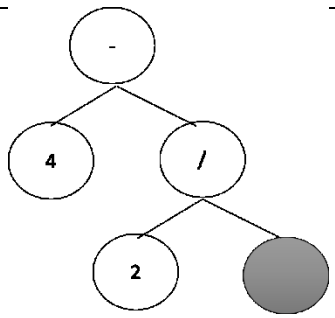
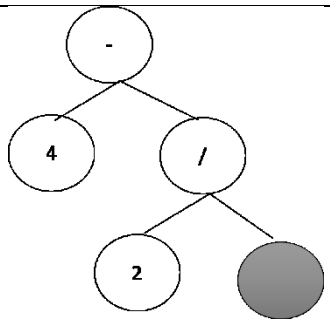
Process tree memiliki node, operator, dan garis penghubung. Sebuah node dapat berupa node cabang atau node daun. Node cabang adalah node operator dan node daun adalah aktivitas (M. L. van Eck, Buijs, & van Dongen, 2015). Lima operator yang terdapat pada process tree yaitu: operator SEQUENCE (→), operator XOR (×), operator AND (∧), operator OR (∨) dan operator LOOP (↺). Operator SEQUENCE (→) menjalankan node cabang (*sub-tree*) secara berurutan, operator XOR (×) menjalankan pemilihan salah satu node cabang, operator AND (∧) menjalankan seluruh node cabang secara bersamaan, operator OR (∨) jika terdapat tiga pilihan menjalankan dua node cabang, dan operator LOOP (↺) menjalankan seluruh node cabang secara berulang (M. L. van Eck et al., 2015).

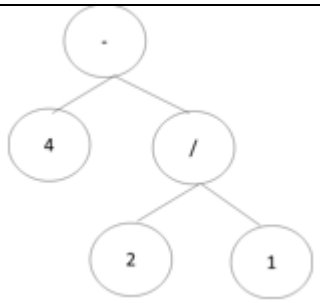
Pembuatan model *process tree* dengan contoh (3 + (4 * 5)) yang akan diubah menjadi bentuk sebagai berikut ['(', '3', '+', '(', '4', '*', '5' ,')',')'] agar dapat dibuatkan model *process tree* nya dapat dilihat pada Tabel 2.1.

a. Buat *empty tree*
b. Baca (sebagai token pertama. Dengan aturan 1, buat simpul baru sebagai *left child of the root*. Buat simpul saat ini pada *child* baru ini.
c. Baca 4 sebagai token berikutnya. Dengan aturan 3, tetapkan nilai *root* simpul saat ini ke 4 dan kembali ke *parent*.
d. Baca - sebagai token berikutnya. Dengan aturan 2, tetapkan nilai akar simpul saat ini ke - dan tambahkan simpul baru sebagai *right child*. *Right child* baru menjadi simpul saat ini.
e. Baca sebuah (sebagai token berikutnya. Dengan aturan 1, buat simpul baru sebagai *left child* simpul saat ini. *Left child* yang baru menjadi simpul saat ini.
f. Baca 2 sebagai token berikutnya. Dengan aturan 3, atur nilai simpul saat ini menjadi 2. Jadikan *parent* dari 4 simpul saat ini.
g. Baca / sebagai token berikutnya. Dengan aturan 2, tetapkan nilai *root* simpul saat ini ke / dan buat *new right child*. *The new right child* menjadi simpul saat ini.
h. Baca 1 sebagai token berikutnya. Dengan aturan 3, atur nilai *root* dari simpul saat ini menjadi 1. Buatlah *parent* dari 1 simpul saat ini.
i. Baca) sebagai token berikutnya. Dengan aturan 2 kita membuat parent dari / node saat ini.
j. Baca) sebagai token berikutnya. Dengan aturan 4 kita membuat *parent* dari - simpul saat ini. Pada titik ini tidak ada parent untuk - . Selesai.

Tabel 2.1 Cara pembuatan model *process tree*

a	

b	
c	
d	
e	
f	
g	

h	

2.4 Kualitas Model Proses

Terdapat empat aspek dalam mengukur kualitas model. Ke-empat aspek tersebut adalah *fitness*, presisi*, simplicity,* dan generalisasi. *Fitness* mengukur seberapa besar proses pada *event log* dapat tergambar di model proses. Presisi mengukur seberapa besar *trace* yang terbetuk dari model proses tergambar dalam *event log*. *Simplicity* mengukur kesederhanaan model proses tanpa menghilangkan proses dari *event log*. Sedangkan, generalisasi mengukur generalisasi model proses.

Sebuah proses model dengan *fitness* yang baik menunjukkan "kecocokan" dari model proses yang ditemukan dengan "realita" . Sebuah model proses dikatakan memiliki *fitness* sempurna jika semua *trace* di *event log* dapat diwakili oleh model proses dari awal sampai akhir. Sebaliknya, jika banyak *trace* di *event log* tidak dapat diwakili oleh proses model dari awal sampai akhir, maka proses model disebut memiliki *fitness* yang buruk. Perhitungan *fitness* menggunakan Persamaan (1).

Presisi menyatakan bahwa suatu model proses tidak seharusnya menunjukkan proses yang cenderung berbeda dengan proses yang terlihat pada *event log*. Presisi berkaitan dengan notasi *overfitting* dalam konteks *data mining*. Suatu model proses dikatakan *overfitting* apabila model proses tersebut sangat spesifik dan berpatokan penuh pada contoh proses di *event log*. Semakin tinggi nilai presisi suatu model proses, maka semakin besar kecenderungan model tersebut dalam notasi *overfitting*. Perhitungan presisi menggunakan Persamaan (2).

Perhitungan *simplicity* suatu model proses dengan cara membandingkan ukuran *tree model* dari suatu model proses dengan aktivitas pada *event log*. Apabila aktivitas yang muncul pada model proses hanya sekali dan semua aktivitas pada

event log tergambar pada *tree model*, maka model tersebut memiliki nilai *similarity* yang tinggi. Perhitungan *simplicity* menggunakan Persamaan (3).

Generalisasi menyatakan bahwa suatu model proses seharusnya menunjukkan generalisasi dari contoh proses yang terlihat pada *event log*. Generalisasi berkaitan dengan notasi *underfitting* dalam konteks *data mining*. Suatu model proses dikatakan *underfitting* apabila model proses tersebut juga menunjukkan proses yang cenderung berbeda dengan proses yang terlihat pada *event log*. Semakin tinggi nilai generalisasi suatu model proses, maka semakin besar kecenderungan model tersebut dalam notasi *underfitting*. Perhitungan generalisasi suatu model proses memperhatikan frekuensi dari kemunculan *node* pada *process tree* berdasarkan *event log*. Perhitungan generalisasi menggunakan Persamaan (4).

$$rf = \frac{CM}{CL} \tag{1}$$

$$p = \frac{TC}{TM} \tag{2}$$

$$s = 1 - \frac{DA + MA}{\#NT} \tag{3}$$

$$g = 1 - \frac{\sum_1^{NT} (\sqrt{\#EN})^{-1}}{\#NT} \tag{4}$$

Keterangan:

rf	:	nilai kualitas model proses *fitness*
p	:	nilai kualitas model proses presisi
s	:	nilai kualitas model proses *simplicity*
g	:	nilai kualitas model proses generalisasi
CM	:	jumlah case dalam *event log* yang tergambar di model proses
CL	:	jumlah case di *event log*
TC	:	jumlah *traces* (variasi proses yang dapat dibentuk dari model proses) yang terekam dalam proses di *event log*
TM	:	jumlah *traces* atau variasi proses yang dapat dibentuk dari model proses

DA : jumlah aktivitas yang digambarkan duplikasi dalam *tree model* serta jumlah duplikasinya.

MA : jumlah aktivitas pada *event log* yang tidak tergambar dalam *model tree* serta aktivitas yang tergambar tetapi tidak ada dalam *event log*.

EN : jumlah *node* dalam *model tree* dilalui sesuai dengan proses dalam *event log*.

NT : jumlah *node* yang tedapat dalam *tree model*

2.5 Linear Temporal Logic (LTL)

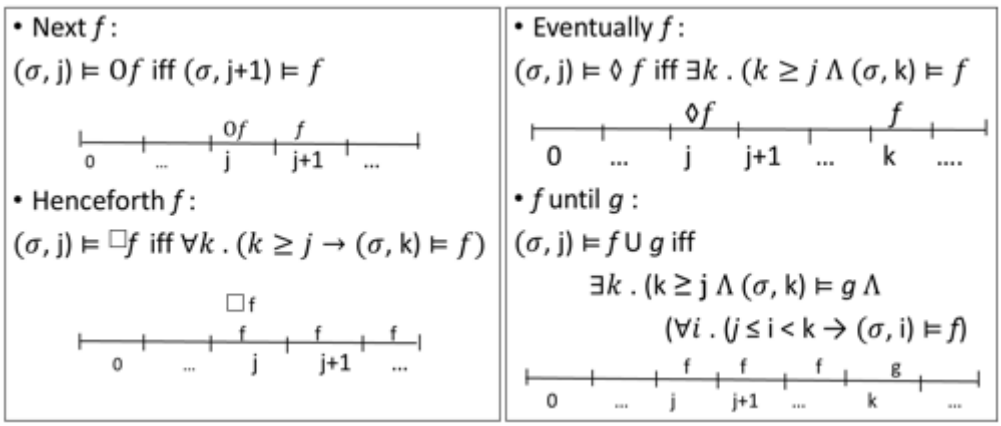
Logika temporal adalah aturan dan simbol digunakan untuk menggambarkan setiap sistem aturan. LTL sering digunakan untuk memeriksa kebenaran atau pelanggaran dalam sebuah sistem (Burattin et al., 2016). Operator LTL dapat dilihat pada sedangkan pada merupakan terjemahan notasi LTL ke dalam notasi Boolean. Pada Tabel.2.2 dijelaskan bahwa LTL memiki empat notasi dasar yaitu next yang dilambangkan dengan O, henceforth yang dilambangkan dengan □ , eventually yang dilambangkan dengan ◊, dan until dilambangkan dengan U (Maggi, Di Francescomarino, Dumas, & Ghidini, 2013). Untuk setiap notasi LTL memiliki arti tersendiri seperti terlihat pada Tabel 2.3 merupakan perbandingan notasi LTL dengan notasi Boolean. Sedangkan pada Gambar 2.1 merupakan ilustrasi penjelasan keadaan benar semantics LTL.

Tabel.2.2 Notasi LTL beserta artinya

Notasi Operator	Arti
O *f*	*f* benar pada state selanjutnya
□ *f*	*f* benar pada semua state selanjutnya
◇ *f*	*f* benar pada state selanjutnya (namun bisa dua state selanjutnya)
f U *g*	*g* benar pada state selanjutnya dan *f* benar pada setiap state sampai state saat itu

Tabel 2.3 Notasi LTL dan terjemahan ke notasi Boolean

Notasi LTL	Notasi Boolean
O f	$\neg f$
$\Box f$	$f \wedge g$
$\Diamond f$	$f \vee g$
f U g	$f \rightarrow g$



Gambar 2.1 Ilustrasi Semantiks LTL

2.6 Goal Programming

Goal Programming merupakan suatu teknik penyelesaian problema pengambilan keputusan yang melibatkan banyak tujuan. Pendekatan dasar yang digunakan dalam *goal programming* adalah meminimalkan deviasi antara tujuan yang ditetapkan dan usaha yang akan dilakukan dalam suatu himpunan kendala sistem. *Goal Programming* adalah salah satu model matematis (empiris) yang dipakai sebagai dasar dalam pengambilan keputusan dan karenanya pendekatan *Goal Programming* ini disebut dengan pendekatan kuantitatif. Model *Goal Programming* merupakan perluasan dari model pemrogaman linier, sehingga seluruh asumsi, notasi formulasi model matematis, prosedur perumusan model dan penyelesaiannya tidak berbeda. Perbedaannya hanya terletak pada kehadiran sepasang variabel deviasioanal yang akan muncul difungsi tujuan dan fungsi-fungsi

kendala. Langkah yang harus dilakukan dalam pembentukan model *Goal Programming* antara lain:

1. Penentuan variabel keputusan, yaitu parameter-parameter yang berpengaruh terhadap keputusan
2. Formulasi Fungsi Tujuan
3. Menyusun persamaan matematis untuk tujuan yang telah ditetapkan tiap fungsi tujuan harus digambarkan sebagai fungsi variabel keputusan.
   $g_i = f_i(x)$, $f_i(x)$ = fungsi variabel keputusan pada tujuan ke i.
4. Tiap fungsi harus memiliki ruas kanan dan ruas kiri. Nilai di^- menunjukkan besarnya deviasi negatif $f_i(x)$ dari b_i, sedangkan nilai di^+ menunjukkan besarnya nilai deviasi positif.
   $f_i(x) + d_i^- - d_i^+ = b_i$ dimana i = 1,2,3,...m
5. Menyederhanakan model. Langkah ini perlu dilakukan untuk mendapatkan model yang cukup besar sehingga model dapat mewakili semua tujuan.
6. Menyusun fungsi pencapaian

2.7 Ringkasan penelitian terdahulu

Bagian ini dijelaskan mengenai studi komparasi pada beberapa paper pada

Tabel 2.4 Tabel penelitian terdahulu

Peneliti	**Tahun**	**Metode**	**Kelebihan**	**Kekurangan**
Andrea Burattin, Fabrizio M. Maggi, Alessandro Sperduti	2016	LTL	Memiliki hasil yang lebih baik daripada pendekatan pemeriksaan kesesuaian berdasarkan control flow dan pengguna dapat mendeteksi pelanggaran yang mana tidak dapat diidentifikasi dengan	Dalam process mining banyak terdapat perspektif, namun pada paper ini hanya menggunakan tiga

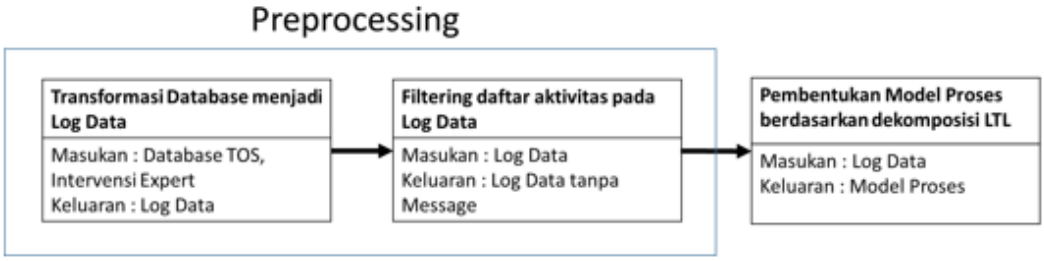
Peneliti	Tahun	Metode	Kelebihan	Kekurangan
			hanya mempertimbangkan perspektif control flow	perspektif saja.
Amin Vahedian Khezerlou, Somayeh Alizadeh	2014	Process Tree	Process tree dapat mengatasi struktur blok petri net, process tree dapat dengan mudah dikonversi menjadi petri net	Waktu eksekusi yang lama karena harus mencapai fitness terbaik atau jumlah maksimum generasi
J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst	2015	Kualitas Model	Pemilihan urutan kualitas model yang didahulukan sehingga dapat memaksimalkan kualitas model berdasarkan pilihan yang dikehendaki.	Waktu eksekusi lama saat ada operator loop
Tri Harjiyanto	2014	Goal Programming	Pemilihan goal programming tanpa prioritas lebih baik karena pendapatan yang diterima lebih besar dan perusahaan tidak memiliki batasan dana untuk proses produksi.	Menambah jumlah kendala sebagai acuan agar mendapatkan hasil sesuai goal

BAB 3
METODA PENELITIAN

3.1. Metode Usulan

3.1.1 Model Sistem Metode Usulan

Metode usulan pertama berkaitan dengan *process discovery* adalah penentuan operator menggunakan notasi LTL dan memodelkan kedalam bentuk model *process tree*. Alur metode usulan dapat dilihat pada Gambar 3.1.



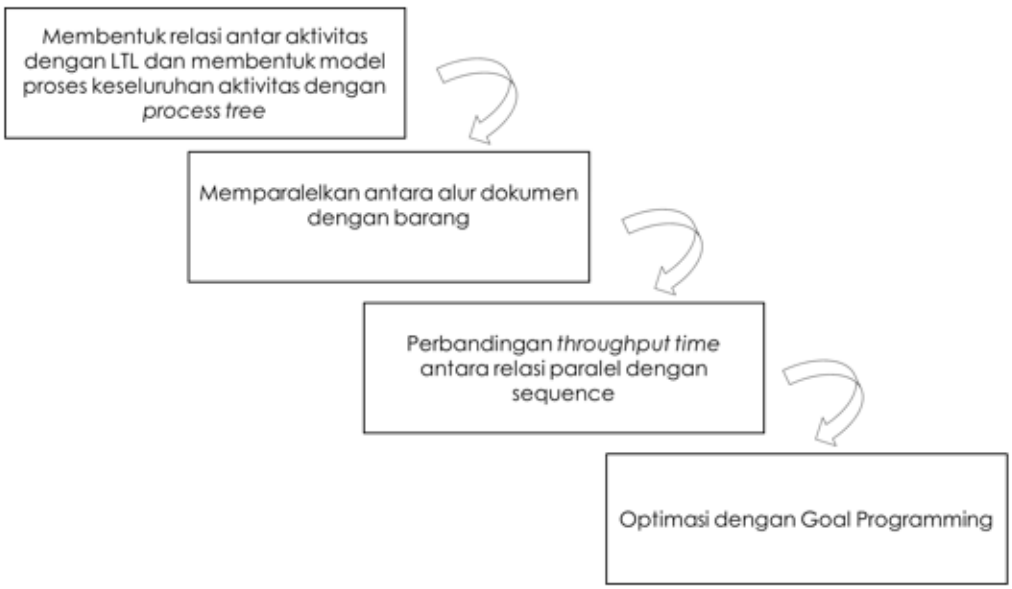
Gambar 3.1 Alur Metode Usulan *Process Discovery*

Tahapan pertama adalah melakukan *pre-processing*. Tahapan *pre-processing* dibagi menjadi tiga bagian, yaitu transformasi *database* ke log data, *filtering* aktivitas pada log data, dan pemisahan log data. Transformasi dilakukan untuk mengubah data di database TOS (Terminal Operating System) ke dalam log data yang berisi proses impor barang di Terminal Peti Kemas. Transformasi dilakukan untuk mendapatkan isi dari log data, seperti ID proses, aktivitas atau *message*, pelaku aktivitas atau *message*, waktu pelaksanaan, dan atribut pendukung aktivitas atau *message* (data lain yang mempengaruhi pelaksanaan suatu aktivitas atau *message*). Kemudian, proses *filtering* dilakukan setelah proses transformasi. Proses ini memisahkan *message* dari log data sehingga hanya aktivitas yang termasuk dalam log data. Hal ini dikarenakan algoritma proses model tidak memerlukan *message*.

Tahapan kedua adalah pembentukan model proses dari operator dekomposisi LTL. Dimana operator yang digunakan adalah tiga operator penting dalam pemodelan proses bisnis. Penentuan operator didapatkan dengan melihat relasi urutan antar aktivitas. Operator XOR memiliki konsep bahwa satu sumber aktivitas

terdiri dari banyak tujuan aktivitas namun hanya boleh memilih satu aktivitas. Sedangkan operator AND memiliki konsep bahwa satu sumber aktivitas terdiri dari banyak tujuan aktivitas dan banyak tujuan tersebut harus dilalui semuanya. Untuk operator OR memiliki konsep bahwa satu sumber aktivitas terdiri dari banyak tujuan aktivitas dan boleh memilih diantara tujuan tersebut minimal dua aktivitas tujuan karena jika hanya memilih satu aktivitas tujuan maka bisa disebut XOR sedangkan jika memilih seluruh aktivitas tujuan maka disebut AND. Langkah selanjutnya kemudian dilakukan dibentuk ke dalam notasi LTL dan dari LTL tersebut digabungkan menjadi keseluruhan model proses dengan menggunakan *process tree*.

Metode usulan kedua berkaitan dengan optimasi waktu dan biaya dengan memparalelkan *trace* barang dan dokumen dengan menggunakan goal programming. Kontribusi pada penelitian ini ada pada tahapan optimasi paralel dengan goal programming. Selama ini alur *container handling* di TPS adalah sequence atau flow shop, maka dari itu diusulkan parallel pada aktivitas dokumen dan barang untuk menghemat waktu. Alur usulan kontribusi dan metode penelitian ini dapat dilihat pada Gambar 3.2.



Gambar 3.2 Gambaran usulan kontribusi penelitian

3.1.2 Jadwal Kegiatan

Jadwal kegiatan dalam penyusunan penelitian ini dapat dilihat pada Tabel 3.1.

Tabel 3.1 Jadwal Kegiatan Penelitian

Aktivitas	Bulan																			
	8				9				10				11				12			
Studi Literatur																				
Analisis dan Desain																				
Pembuatan Perangkat Lunak																				
Uji Coba dan Analisis Hasil																				
Dokumentasi																				

3.2 Masukan

Dalam penelitian ini, log data yang digunakan adalah log data mengenai proses impor barang di Terminal Peti Kemas Surabaya. Log data merupakan hasil *pre-processing* yang diambil dari database TOS (Terminal Operating System) mengenai proses impor barang di Terminal Peti Kemas. Data yang diambil adalah data yang berjalan dari bulan Januari sampai berakhir di bulan Maret 2016. Penanda dari awal proses pada database adalah tanggal pada kolom VESEL_ATB dan penanda akhir proses adalah tanggal yang tersimpan pada kolom TRUCK_OUT.

Database TOS memiliki beberapa kolom dimana kolom- kolom tersebut berisi tanggal dan waktu eksekusi untuk menunjukkan aktivitas pada proses, serta data lainnya yang menunjang pembentukan log data. Tabel 3.2 menjelaskan keterangan dari kolom database TOS yang digunakan dalam pembentukan log data.

Log data yang dimiliki PT.TPS memiliki sebelas jalur *trace* tanpa anomali. Dimana *trace* pertama dan kedua memiliki 30 aktivitas yang digunakan yang merupakan log data jalur merah dengan proses karantina dan kontainer bertipe *refeer*. Sedangkan trace kedua memiliki 29 aktivitas yang digunakan yang merupakan jalur merah dengan proses karantina dan kontainer bertipe *dry*. Trace

ketiga memiliki 25 aktivitas yang digunakan yang merupakan log data jalur hijau dengan proses karantina dan kontainer bertipe *refeer*. Trace keempat memiliki 25 aktivitas yang digunakan yang merupakan log data jalur hijau dengan proses karantina dan kontainer bertipe *dry*. Trace kelima dan keenam memiliki 24 aktivitas yang digunakan yang merupakan log data jalur merah dan kontainer bertipe *refeer* atau *uncontainer*. Trace ketujuh memiliki 23 aktivitas yang digunakan yang merupakan log data jalur merah dengan proses karantina dan kontainer bertipe *dry*. Trace kedelapan dan kesembilan memiliki 18 aktivitas yang digunakan yang merupakan log data jalur hijau dan kontainer bertipe *refeer* ata *uncontainer*. Trace kesepuluh memiliki 17 aktivitas yang digunakan yang merupakan log data jalur merah dengan proses karantina dan kontainer bertipe *dry*. Trace kesebelas memiliki 31 aktivitas yang digunakan yang merupakan log data jalur merah dengan proses karantina dan kontainer bertipe *uncontainer*. Untuk mengetahui seluruh aktivitas setiap trace dapat dilihat pada Lampiran.

Tabel 3.2 Kode dan keterangan kolom database TOS

Kolom	**Keterangan**
CONTAINER_KEY	Kode unik untuk membedakan data proses impor barang yang satu dengan yang lain
CTR_TYPE	Tipe umum barang yang di-impor. Tipe ini dibedakan ke dalam tiga macam yaitu Reefer, Dry, dan Uncontainer yang dituliskan dalam kode khusus.
YARD_BLOCK	Blok dari tempat untuk menumpuk barang (letak penumpukkan barang dalam zona vertikal)
YARD_SLOT	Slot dari tempat untuk menumpuk barang (letak penumpukkan barang dalam zona horizontal)
HAS_QUARANTINE_FLAG	Penunjuk dilakukannya proses di karantina atau tidak, yang disimbolkan “YES” dan “NO”
CUSTOM_BEHANDLE_COUNT	Penunjuk dilakukannya proses di behandle atau tidak, yang disimbolkan “0” dan “1”

3.3 Preprocessing (Transformasi ke Log Data berdasarkan Database)

Pada tahap ini adalah transformasi dari log data berdasarkan database TOS. Log data yang digunakan berisi Case_ID, nama aktivitas atau *message*, waktu pelaksanaan, pelaku aktivitas (*originator*), pelaku *message* (sender receiver), *cost, atribut* tambahan yang mendukung aktivitas (*input, output, detail attachment*). Cara transformasi secara garis besar dapat dilihat pada Tabel 3.3.

Tabel 3.3 Cara Transformasi ke Log Data secara Garis Besar

Isi Log Data	**Cara Transformasi**
Case_ID	Mengambil data pada kolom CONTAINER_KEY
Waktu pelaksanaan	Melakukan *inverse* distribusi kumulatif normal berdasarkan database dan rentang waktu intervensi dari *expert*
Nama aktivitas dan *message*	Berdasarkan intervensi *expert*
Pelaku *message* (sender receiver)	
Cost	Melakukan *inverse* distribusi kumulatif normal berdasarkan list harga pengeluaran proses secara keseluruhan
Atribut tambahan (*input output*)	Berdasarkan list dokumen yang dibutuhkan pada proses impor barang
Atribut tambahan (*detail attachment*)	Berdasarkan kolom pada database, yaitu CTR_TYPE, HAS_QUARANTINE_FLAG, dan CUSTOM_BEHANDLE_COUNT

Database TOS menyimpan waktu dari aktivitas dan *message*, akan tetapi tidak semua kolom menyimpan waktu setiap aktivitas dan *message*. Terdapat beberapa kolom yang menunjukkan rentang waktu pelaksanaan beberapa aktivitas serta *message* sekaligus. Sedangkan, log data membutuhkan waktu setiap aktivitas atau *message*. Oleh karena itu, diperlukan perkiraan waktu untuk setiap aktivitas dan *message*.

Selain database TOS, *expert* juga memberikan perkiraan rentang waktu eksekusi setiap aktivitas. Perkiraan rentang waktu eksekusi tersebut dimanfaatkan untuk menentukan perkiraan waktu setiap aktivitas dan *message.* Penentuan waktu eksekusi setiap aktivitas menggunakan *inverse normal distribution* yang mengacu pada perkiraan rentang waktu eksekusi dan rentang waktu beberapa aktivitas dari

database. Pengunaan *normal distribution* dipilih dengan asumsi bahwa persebaran datanya adalah sebaran normal.

Persamaan untuk perhitungan waktu aktivitas dapat dilihat pada Persamaan (7) sampai Persamaan (9). Waktu aktivitas didapatkan dari penambahan waktu sebelumnya dan perkiraan waktu eksekusi aktivtias menggunakan *inverse normal distribution*. Rata-rata untuk *inverse normal distribution* adalah nilai perkiraan berdasarkan median rentang waktu eksekusi dan rentang waktu beberapa aktivitas database. Simpangan baku adalah 5% dari nilai rata-rata. Pemilihan 5% untuk memberikan persebaran yang tidak lebar dalam hasil perkiraan waktu eksekusi aktivitas. Untuk contoh intervensi waktu dijelaskan pada Tabel 3.4. Pada tabel tersebut proses blok Delivery dengan menormalisasi durasi dari aktivitas Truck In hingga aktivitas Truck Out. Normalisasi dilakukan untuk mendapatkan standar deviasi durasi tiap aktivitas.

$$time(Act\_now) = time(Act\_before) + NormInv(p, \mu, \sigma_x) \quad (5)$$

$$\mu = \frac{\widetilde{Es.Act}}{\sum_{i=0}^{n} \widetilde{Es.Act_i}} \times \overline{ActinLog} \quad (6)$$

$$\sigma_x = 0{,}05 \times \mu \quad (7)$$

dimana:

time(Act_now)	=	*waktu pelaksanaan aktivitas atau message sekarang*
time(Act_bfeore)	=	*waktu pelaksanaan aktivitas atau message sebelumnya*
NormInv	=	*inverse normal distribution untuk memperkirakan rentang waktu tiap aktivtias*
$\widetilde{Es.Act}$	=	*median dari interval rentang waktu yang didapatkan dari expert*
$\overline{ActinLog}$	=	*rata-rata dari rentang waktu kelompok aktivitas di database*
n	=	*jumlah aktivitas yang berada pada log data*

p = *nilai probabilitas secara random*

μ = *rata-rata untuk perhitungan inverse normal distribution*

σ_x = *simpangan baku untuk perhitungan inverse normal distribution*

Contoh dari pembentukan waktu pelaksanaan adalah pembentukan waktu aktivitas-aktivitas yang terjadi saat truk datang untuk mengambil barang (TRUCK IN) dan truk keluar dari Terminal Peti Kemas (TRUCK OUT). Perhitungan dapat dilihat pada Tabel 3.4. Batas atas dan batas bawah adalah intervensi yang diberikan oleh *expert* mengenai rentang waktu eksekusi setiap aktivitas. Kemudian, dari rentang waktu eksekusi tersebut, diambil nilai median yang ditunjukkan oleh nilai pada Median(Es.Act). Kemudian, nilai rata-rata didapatkan dari nilai pada Median(Es.Act) dibagi jumlah keseluruhan nilai media dan dikali dengan rentang waktu kelompok aktivitas pada database, yang ditunjukkan oleh nilai ActinLog. Kemudian, nilai standar deviasi menunjukkan nilai simpangan baku. Nilai rata-rata dan simpangan baku yang didapatkan kemudian diolah menggunakan Persamaan (7) untuk menghasilkan waktu setiap aktivitas atau *message*. Hasil waktu setiap aktivitas dapat dilihat pada Tabel 3.5.

Tabel 3.4 Perhitungan Waktu Aktivitas untuk Log Data

Aktivitas	Batas Atas	Batas Bawah	Median(Es.Act)	Rata-rata	Standar Deviasi
Dispatch WQ Delivery to CHE	0:00:30	0:00:50	0:00:40	0:00:40	0:00:02
Determine Container Type	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Determining Uncointaner	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Decide Task Before Lift Container	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Prepare Tools	0:10:00	0:15:00	0:12:30	0:12:35	0:00:38
Lift on Container Truck	0:02:00	0:03:00	0:02:30	0:02:31	0:00:08
Truck Go To Gate Out	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
Check Container before Truck out	0:01:00	0:02:00	0:01:30	0:01:31	0:00:05
		sum(median(Es.Act))	0:23:10		
		ActInLog	**0:23:20**		

Tabel 3.5 Contoh Hasil Estimasi Waktu Setiap Aktivitas

CaseID	Activity	Time	NormInv
4619882	Truck in	3/14/16 19:30:21	
4619882	Dispatch WQ Delivery to CHE	3/14/16 19:31:02	0:00:41
4619882	Determine Container Type	3/14/16 19:32:30	0:01:28
4619882	Determining Uncointaner	3/14/16 19:34:06	0:01:36
4619882	Decide Task Before Lift Container	3/14/16 19:35:36	0:01:31
4619882	Prepare Tools	3/14/16 19:48:30	0:12:54
4619882	Lift on Container Truck	3/14/16 19:51:05	0:02:35
4619882	Truck Go To Gate Out	3/14/16 19:52:34	0:01:29
4619882	Check Container before Truck out	3/14/16 19:54:13	0:01:39
4619882	Truck Out	3/14/16 22:48:25	

3.3.1 Filtering antara Aktivitas dan Message

Dari Tabel 3.6, selanjutnya akan difilter antara aktivitas dan message dimana hal ini bertujuan untuk proses discovery dan optimasi. Proses optimasi hanya melibatkan aktivitas sehingga dilakukan proses filtering antara aktivitas dan message. Proses filtering dilakukan dengan membuat rule jika terdapat sender dan receive maka dikategorikan sebagai message, dan sebaliknya jika terdapat originator maka dikategorikan sebagai aktivitas.

Tabel 3.6 Tabel hasil filtering antara aktivitas dan message

Aktivitas
Entry Document via PDE
Vessel Berthing Process
Discharge Container
Bring Container to Yard
Stack Container in Yard
Verification Document Behandle
Create Job Order Document Behandle
Bring Container from Yard to Behandle
Stack Container in Behandle Area
Check Goods Behandle
Create document LHP
Stack Container in Yard From Behandle

Aktivitas
Create document SPPB
Verification Document Quarantine
Create Job Order Document Quarantine
Bring Container from Yard to Quarantine
Stack Container in Quarantine Area
Check Goods Quarantine
Create document KH/KT
Stack Container in Yard From Quarantine
Create Job Order Document Delivery
Truck in
Dispatch WQ Delivery to CHE
Determine Container Type
Determining Refeer/ Determining Dry/ Determining Uncointaner
Decide Task Before Lift Container
Unplug Refeer Cable/Prepare Tools
Lift on Container Truck
Truck Go To Gate Out
Check Container before Truck out
Truck Out

3.3.2 Pengelompokan Aktivitas Berdasarkan Kategori Dokumen dan Barang

Setelah diperoleh keseluruhan aktivitas di PT TPS maka selanjutnya adalah menentukan aktivitas tersebut adalah kategori dokumen atau barang. Maksud kategori barang atau dokumen adalah proses yang dilakukan oleh suatu aktivitas adalah terkait dokumen atau barang (container). Dengan menggunakan rule input dan output akan dihasilkan aktivitas dan kategorinya seperti pada Tabel 3.7.

Tabel 3.7 Aktivitas kategori dokumen dan barang

Aktivitas	Kategori
Entry Document via PDE	Dokumen

Aktivitas	Kategori
Vessel Berthing Process	Barang
Discharge Container	Barang
Bring Container to Yard	Barang
Stack Container in Yard	Barang
Verification Document Behandle	Dokumen
Create Job Order Document Behandle	Dokumen
Bring Container from Yard to Behandle	Barang
Stack Container in Behandle Area	Barang
Check Goods Behandle	Barang
Create document LHP	Dokumen
Stack Container in Yard From Behandle	Barang
Create document SPPB	Dokumen
Verification Document Quarantine	Dokumen
Create Job Order Document Quarantine	Dokumen
Bring Container from Yard to Quarantine	Barang
Stack Container in Quarantine Area	Barang
Check Goods Quarantine	Barang
Create document KH/KT	Dokumen
Stack Container in Yard From Quarantine	Barang
Create Job Order Document Delivery	Dokumen
Truck in	Barang
Dispatch WQ Delivery to CHE	Barang
Determine Container Type	Barang
Determining Refeer/ Determining Dry/ Determining Uncointaner	Barang
Decide Task Before Lift Container	Barang

Aktivitas	Kategori
Unplug Refeer Cable/Prepare Tools	Barang
Lift on Container Truck	Barang
Truck Go To Gate Out	Barang
Check Container before Truck out	Barang
Truck Out	Barang

3.3.3 Membuat Single Timestamp menjadi Double Timestamp

Data log dapat dibagi menjadi dua jenis berdasarkan waktu yang terekam. Jenisnya adalah single timestamp dan double timestamp. Dalam organisasi yang menjalankan proses bisnis, data log tersedia baik dalam bentuk single timestamp atau double timestamp. Namun ada organisasi yang hanya menyediakan waktu akhir untuk semua aktivitas yang dieksekusi, maka hal pertama yang perlu dilakukan adalah mengubah single timestamp menjadi double timestamp dengan menggunakan sojourn time. Sojourn time adalah adalah total waktu (termasuk durasi eksekusi dan waktu tunggu) yang dibutuhkan setiap aktivitas untuk menyelesaikan proses. Langkah-langkah untuk mengubah single timestamp dan double timestamp adalah sebagai berikut:

1. Hitung sojourn time untuk semua aktivitas di data log
$$Sojourn\ time\ B = et_{activity_B} - et_{activity_A}$$
2. Pilih batas atas dan batas bawah setiap aktivitas dari sojourn time
3. Hitung nilai median antara batas atas dan batas bawah untuk setiap aktivitas
4. Lakukan normalisasi untuk setiap aktivitas
$$Normalization = \frac{Median\ value}{Average\ of\ Sojourn\ Time}$$
5. Hitung standar deviasi untuk setiap aktivitas
$$Stdev = \frac{0.05}{Normalization}$$

6. Dapatkan durasi eksekusi untuk setiap aktivitas dengan menggunakan bilangan acak normal

$$Norminv = Rand(); normalization; stdev$$

7. Untuk mendapatkan waktu mulai untuk setiap aktivitas, kurangkan waktu akhir dengan durasi eksekusi dan untuk mendapatkan waktu tunggu, kurangkan waktu akhir dengan waktu mulai aktivitas.

$$Start\ time\ act\ B = End\ time\ B - Execution\ duration\ B$$

$$Waiting\ time\ act\ B = End\ time\ B - Start\ time\ C$$

Misalnya, terdapat data log dengan single timestamp seperti pada Tabel 3.8. Dengan menggunakan langkah 1-7, kita dapat mengubah single timestamp menjadi double timestamp dengan menggunakan sojourn time. Kami menerapkan langkah-langkah untuk mendapatkan durasi eksekusi untuk aktivitas B dan aktivitas C serta mengurangi waktu akhir dengan durasi eksekusi untuk mendapatkan waktu mulai untuk aktivitas B dan aktivitas C. Tabel 3.9 menyajikan data log double timestamp sebagai hasilnya.

Tabel 3.8 Langkah-langkah mengubah single timestamp menjadi double timestamp

Case ID	Activity	End Time	Sojourn Time	Average Sojourn Time	Upper bound	Lower bound	Median	Normalization	Stdev	Execution Duration
ID001	A	20/06/2014 08.32	03.17.00							
ID001	B	20/06/2014 13.42	05.10.18	06.36	09.59	05.05	07.32	03.22	01.22	04.39
ID001	B	20/06/2014 23.41	09.59.22							04.18
ID001	C	21/06/2014 08.16	08.34.09	10.16	15.35	06.41	11.08	01.59	01.18	01.54
ID002	A	21/06/2014 10.46	02.30.18							
ID002	B	21/06/2014 16.57	06.11.36							03.09
ID002	D	21/06/2014 18.09	01.11.27							
ID002	E	21/06/2014 19.15	01.06.29							
ID002	C	22/06/2014 10.51	15.35.29							01.16
ID003	A	22/06/2014 16.28	05.36.48							
ID003	D	22/06/2014 23.42	07.14.33							
ID003	B	23/06/2014 04.48	05.05.50							03.10
ID003	E	23/06/2014 16.44	11.56.24							
ID003	C	23/06/2014 23.26	06.41.04							03.54

Tabel 3.9 Hasil data log dengan double timestamp

Case ID	Activity	Start Time	End Time
ID001	A		20/06/2014 08.32
ID001	B	20/06/2014 09.03	20/06/2014 13.42
ID001	B	20/06/2014 19.23	20/06/2014 23.41
ID001	C	21/06/2014 06.22	21/06/2014 08.16
ID002	A		21/06/2014 10.46
ID002	B	21/06/2014 13.48	21/06/2014 16.57
ID002	D		21/06/2014 18.09
ID002	E		21/06/2014 19.15
ID002	C	22/06/2014 09.34	22/06/2014 10.51
ID003	A		22/06/2014 16.28
ID003	D		22/06/2014 23.42
ID003	B	23/06/2014 01.38	23/06/2014 04.48
ID003	E		23/06/2014 16.44
ID003	C	23/06/2014 19.31	23/06/2014 23.26

3.4 Pembentukan Model *Process Tree* dari Dekomposisi LTL

Bagian ini menjelaskan langkah-langkah algoritma penemuan model proses dengan menggunakan dekomposisi LTL. Langkah utamanya adalah, pertama, mengelompokkan semua input dan output aktivitas untuk setiap trace; kemudian, klasifikasi relasi sequence dan paralel, dan terakhir, model proses lengkap dengan relasi sequence dan paralel, input dan output.

Langkah 1. Kelompokkan relasi sequence (>) setiap aktivitas di seluruh case di data log. Pada tahapan ini merupakan penentuan relasi aktivitas. Dasar dari semua penentuan relasi aktivitas adalah dengan pertama kali dilakukan penentuan direct succession dapat dilihat pada Tabel 3.10 dan Tabel 3.11 dimana a > b untuk semua case a diikuti secara langsung oleh b.

Tabel 3.10 Tabel Direct Succession setiap case

Input		Output
{}	>	A
A	>	B
B	>	C
C	>	D
D	>	E
E	>	F
F	>	I
I	>	{}

Input		Output
{}	>	A
A	>	C
C	>	B
B	>	D
D	>	E
E	>	G
G	>	I
I	>	{}

Input		Output
{}	>	A
A	>	D
D	>	B
B	>	C
C	>	E
E	>	H
H	>	I
I	>	{}

Tabel 3.11 Tabel Direct Succesion semua case

A	>	B
A	>	C
A	>	D
B	>	C
B	>	D
C	>	D
C	>	B
C	>	E
D	>	E
D	>	B
E	>	F
E	>	G
E	>	H
F	>	I
G	>	I
H	>	I

Langkah 2. Kelompokkan relasi paralel (||) setiap aktivitas di seluruh case di data log. Penentuan relasi causal dapat dilihat pada Tabel 3.12 ditentukan a → b jika a > b dan tidak berlaku bagi b > a. Relasi causal berguna saat penentuan relasi paralel dan choice dan juga untuk memodelkan secara keseluruhan proses model. Relasi paralel dapat dilihat pada Tabel 3.13 dengan cara a || b jika a > b dan b > a. Relasi choice dapat dilihat pada Tabel 3.14 dilakukan a # b jika tidak a > b dan tidak b > c.

Tabel 3.12 Tabel Causality

A	→	B
A	→	C
A	→	D
C	→	E
D	→	E
E	→	F
E	→	G
E	→	H
F	→	I
G	→	I
H	→	I

Tabel 3.13 Tabel Paralel

B	||	C
C	||	B
B	||	D
D	||	B
C	||	D

Tabel 3.14 Tabel Choice

F	#	G
F	#	H
G	#	H

Langkah 3. Hapus relasi sequence dan paralel yang duplikat untuk mendapatkan jumlah trace di data log

Langkah 4. Mendapatkan semua trace dengan relasi paralel dan sequence dari data log

Langkah 5. Menentukan transisi (TL) dari workflow net . Pada tahapan pembentukan kumpulan transisi dilakukan dengan cara membuat kumpulan aktivitas awal hingga aktivitas akhir pada suatu CaseID dan dikelompokkan berdasarkan *trace* yang sama. Contoh event log dapat dilihat pada Tabel 3.15 dimana semua transisi yang dibentuk adalah A B C D E F G H I. Persamaan yang digunakan untuk mencari T_L dapat dilihat pada persamaan (8).

$$\boldsymbol{T_L = \{t \in T \mid \exists_{\sigma \epsilon L} t \in \sigma\}} \quad (8)$$

Tabel 3.15 Contoh event log

Case	**Aktivitas**						
Case001	A	B	C	D	E	F	I
Case002	A	C	B	D	E	G	I
Case003	A	D	B	C	E	H	I

Langkah 6. Menentukan transisi input dari workflow net

$$\boldsymbol{T_I = \{t \in T \mid \exists_{\sigma \epsilon L} t = first(\sigma)\}} \quad (9)$$

Langkah 7. Menentukan transisi output dari workflow net

$$\boldsymbol{T_o} = \{\boldsymbol{t} \in \boldsymbol{T} \mid \exists_{\sigma \epsilon L} \boldsymbol{t} = \boldsymbol{last}(\sigma)\} \quad (10)$$

Langkah 8. Pembentukan operator dengan notasi LTL

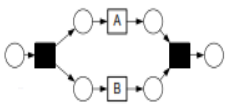
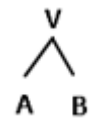
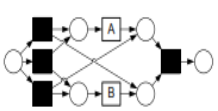
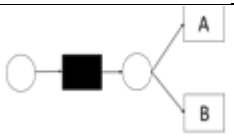
Pada tahapan ini pembentukan operator dilakukan dengan menggunakan notasi LTL. *Control-flow pattern* umum, yaitu *control-flow pattern* yang membentuk relasi sekuensial serta relasi paralel, seperti XOR, OR dan AND, dibentuk secara langsung dari *rules* LTL. Relasi sequence ditentukan dari relasi causal sebagai contoh A → B dapat dilihat pada Tabel 3.16 maka penulisan LTLnya adalah A → O (B). Operator XOR dan AND ditentukan dari tabel choice dan causal. Sebagai contoh pada Tabel 3.15 operator XOR terdapat pada aktivitas E → F, E → G, E → H, F # G, F # H, G # H maka jika dibuat dekomposisi LTL adalah E → O (F ∨ G ∨ H). Sebagai contoh pada Tabel 3.15 operator AND terdapat pada aktivitas A → B, A → C, A → D, B || C, C || B, B || D, D || B, C || D maka jika dibuat dekomposisi LTL adalah A → <> (B ∨ C ∨ D). Pembentukan operator berdasarkan LTL dapat dilihat pada Tabel 3.17.

Tabel 3.16 Tabel Aturan Dekomposisi LTL

Pattern	**Causality**	**Paralel**	**Choice**	**LTL**
Sequence	trans_bef → trans_next			trans_before → O (trans_next)
Parallel (AND)	trans_bef1 → trans_next1, trans_bef1 → trans_next2, trans_befn → trans_nextn	trans_next1 || trans_next2, trans_next2 || trans_next1, trans_befn || trans_nextn		trans_bef1 -> ◊ ((trans_next1 ∧ trans_next2…. ∧ trans_nextn))
Exclusive choice (XOR)	trans_bef1 → trans_next1, trans_bef1 → trans_next2, trans_befn		trans_next1 # trans_next2, trans_next2 # trans_next1, trans_befn # trans_nextn	trans_bef1 -> O ((trans_next1 ∨ trans_next2…. ∨ trans_nextn))

Pattern	Causality	Paralel	Choice	LTL
	→ trans_nextn			
Multi Choice Pattern (OR)	trans_bef1 → trans_next1, trans_bef1 → trans_next2, trans_bef1 → trans_next3	trans_next1 || trans_next2, trans_next2 || trans_next1		trans_bef1 -> <> ((trans_next1 ∨ trans_next2))
Invisible Task in Skip and Switch Condition	trans_bef1 → trans_next1, trans_bef1 → trans_next2, trans_next1 → trans_next2			trans_bef1 -> O ((trans_next1 ∨ Invisibe Task))

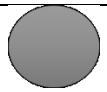
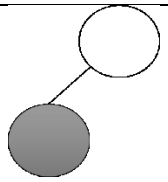
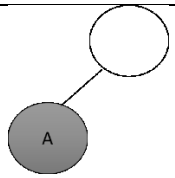
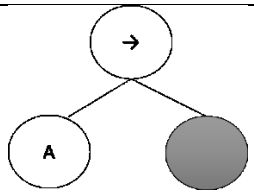
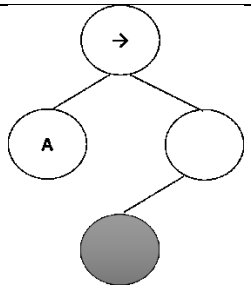
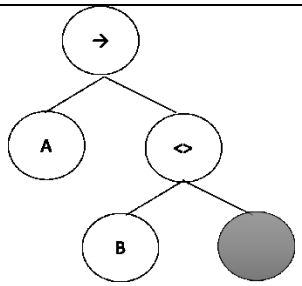
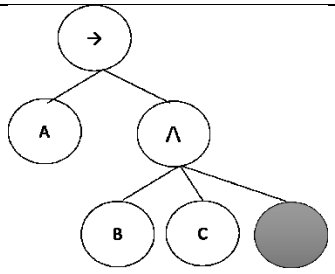
Tabel 3.17 Notasi setiap model proses berdasarkan Operator

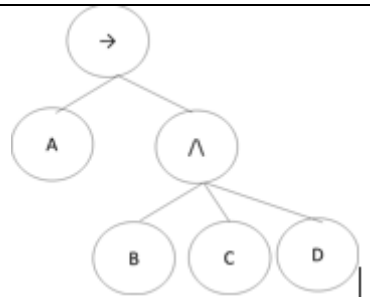
Operator	LTL	Process Tree	Petri net
SEQUENCE	A→O(B)		
AND	Initial → ◊ (A Λ B)		
OR	Initial → ◊ (A V B)		
XOR	Initial → O (A V B)		

Langkah 9. Pembacaan notasi LTL dibentuk kedalam model process tree. Pada tahapan ini merupakan pe yang selanjutnya akan digunakan oleh LTL untuk memodelkan menggunakan process tree. Pembuatan model *process tree* dengan contoh A → <> ((B \/ C \/ D)) yang akan diubah menjadi bentuk sebagai berikut ['A', '→', '<>', '(', '(', 'B' ,'\/','C', '\/' ,'D',')', ,')'] agar dapat dibuatkan model *process tree* nya. Langkah pembentukan model process tree dapat dilihat pada Tabel 3.18:

a. Buat *empty tree*
b. Dengan aturan 1, buat simpul baru sebagai *left child of the root*. Buat simpul saat ini pada *child* baru ini.
c. Baca A sebagai token berikutnya. Dengan aturan 3, tetapkan nilai *root* simpul saat ini A dan kembali ke *parent*.
d. Baca → sebagai token berikutnya. Dengan aturan 2, tetapkan nilai akar simpul saat ini → dan tambahkan simpul baru sebagai *right child*. *Right child* baru menjadi simpul saat ini.
e. Baca sebuah <> sebagai token berikutnya. Karena variable "<>" maka mengikuti aturan 1, buat simpul baru sebagai *left child* simpul saat ini. *Left child* yang baru menjadi simpul saat ini.
f. Baca B sebagai token berikutnya. Dengan aturan 3, atur nilai simpul saat ini menjadi B.
g. Baca /\ sebagai token berikutnya. Karena <> bertemu dengan \/ maka /\. Dengan aturan 2, tetapkan nilai *root* simpul saat ini ke /\ dan buat *new right child*. *The new right child* menjadi simpul saat ini.
h. Baca C sebagai token berikutnya. Dengan aturan 3, atur nilai *root* dari simpul saat ini menjadi 1. Buatlah *parent* dari 1 simpul saat ini.
i. Baca D sebagai token berikutnya. Dengan aturan 3, atur nilai *root* dari simpul saat ini menjadi 1. Buatlah *parent* dari 1 simpul saat ini.
j. Baca) sebagai token berikutnya. Dengan aturan 2 kita membuat parent dari / node saat ini.
k. Baca) sebagai token berikutnya. Dengan aturan 4 kita membuat *parent* dari - simpul saat ini. Pada titik ini tidak ada parent untuk - . Selesai.

Tabel 3.18 Pembentukan model *process tree* dari LTL

a	
b	
c	
d	
e	
f	
g	

h	

3.5 Optimasi waktu dan biaya menggunakan goal programming

3.5.1 Paralelisasi Log Data

Untuk memparalelkan aktivitas sebuah proses bisnis, aktivitas yang independen harus diidentifikasi terlebih dahulu, seperti aktivitas manakah dari proses bisnis yang dapat dilakukan bersamaan. Tingkat paralelisme tertinggi dicapai jika jumlah aktivitas yang diidentifikasi sebagai independen dapat dimaksimalkan. Secara umum, identifikasi ini didasarkan pada waktu dan tempat eksekusi aktivitas, aktivitas dapat di paralelisasi jika aktivitas pada entitas simulasi yang sama dijalankan dalam urutan timestamp. Untuk meningkatkan tingkat paralelisme, kami mengusulkan sebuah pendekatan baru yang mengidentifikasi kriteria independensi lain: Jika dua aktivitas pada entitas simulasi yang sama mengakses item data yang sama dengan cara yang berbeda, mereka dapat dieksekusi secara paralel. Pada penelitian ini, paralelisasi yang diusulkan adalah trace dokumen dengan trace barang. Kedua trace tersebut dapat dilihat pada Tabel 3.19 sedangkan penerapan pada event log dapat dilihat pada . Sedangkan ilustrasi paralelisasi dapat dilihat pada . Kriteria aktivitas dapat diparalelkan adalah sebagai berikut:

1. Aktivitas setiap departemen independen
2. Waktu dan tempat eksekusi berbeda
3. Resource yang menangani setiap aktivitas berbeda

Tabel 3.19 Trace Dokumen dan Trace Barang di Event Log TPS

Trace Dokumen	**Trace Barang**
Document_Entry_via_PDE	Vessel_Berthing_Process

Trace Dokumen	Trace Barang
Verification_Document_Behandle Create_Job_Order_Document_Behandle Create_document_LHP Create_document_SPPB Verification_Document_Quarantine Create_Job_Order_Document_Quarantine Create_document_KH/KT Create_Job_Order_Document_Delivery	Discharge_Container Bring_Container_to_Yard Stack_Container_in_Yard Stack_Container_in_Behandle_Area Check_Goods_Behandle Bring_Container_from_Yard_to_Behandle Stack_Container_in_Yard_From_Behandle Bring_Container_from_Yard_to_Quarantine Stack_Container_in_Quarantine_Area Check_Goods_Quarantine Stack_Container_in_Yard_From_Quarantine Truck_in Dispatch_WQ_Delivery_to_CHE Determine_Container_Type Determining_Dry Determining_Refeer Determining_Uncontainer Decide_Task_Before_Lift_Container Unplug_Refeer_Cable Prepare_Tools Lift_on_Container_Truck Truck_Go_To_Gate_Out Check_Container_before_Truck_out Truck_Out

Tabel 3.20 Contoh event log paralelisasi dokumen dan barang

Activity	Start	Finish
Document_Entry_via_PDE	1/31/16 10:51	1/31/16 12:07
Vessel_Berthing_Process	1/31/16 14:35	2/2/16 13:20
Discharge_Container	2/1/16 6:45	2/1/16 6:49
Bring_Container_to_Yard	2/1/16 6:49	2/1/16 7:01
Stack_Container_in_Yard	2/1/16 7:01	2/1/16 7:04
Verification_Document_Behandle	1/31/16 12:07	2/1/16 7:04
Create_Job_Order_Document_Behandle	2/1/16 7:04	2/8/16 4:05
Stack_Container_in_Behandle_Area	2/8/16 4:05	2/12/16 2:03
Check_Goods_Behandle	2/12/16 2:03	2/12/16 4:21
Create_document_LHP	2/12/16 4:21	2/12/16 8:56
Bring_Container_from_Yard_to_Behandle	2/12/16 8:56	2/12/16 20:16
Stack_Container_in_Yard_From_Behandle	2/12/16 20:16	2/12/16 20:23
Create_document_SPPB	2/12/16 20:23	2/13/16 18:10
Create_Job_Order_Document_Delivery	2/13/16 18:10	2/15/16 1:10
Truck_in	2/15/16 1:10	2/15/16 19:45
Dispatch_WQ_Delivery_to_CHE	2/15/16 19:45	2/15/16 19:46
Determine_Container_Type	2/15/16 19:46	2/15/16 19:49
Determining_Dry	2/15/16 19:49	2/15/16 19:51
Decide_Task_Before_Lift_Container	2/15/16 19:51	2/15/16 19:53
Lift_on_Container_Truck	2/15/16 19:53	2/15/16 19:55
Truck_Go_To_Gate_Out	2/15/16 19:55	2/15/16 21:32
Check_Container_before_Truck_out	2/15/16 21:32	2/15/16 21:33
Truck_Out	2/15/16 21:33	2/15/16 21:34

3.5.2 Implementasi Uji Coba Optimasi

Pada penelitian tesis ini, tujuan yang ingin dicapai adalah meminimumkan waktu. Goal waktu diperoleh dari rata-rata setiap aktivitas dari kumpulan case pada trace yang terdapat trace dokumen. Sedangkan yang ingin dioptimasi adalah waktu maksimum setiap aktivitas per trace agar mencapai waktu minimum. Optimasi dilakukan per trace proses, trace barang dan dokumen. Sementara, prinsip biaya pada optimasi ini adalah jika goal waktu berhasil dicapai, maka berapakah biaya yang bisa dihemat untuk setiap trace.

Langkah-langkah optimasi waktu dan biaya dengan Goal programming adalah sebagai berikut:

1. Memisahkan trace barang dan dokumen seperti terlihat pada Tabel 3.14
2. Menghitung waktu rata-rata serta biaya rata-rata setiap aktivitas per trace
3. Formulasi fungsi tujuan

   Untuk meminimalkan waktu setiap aktivitas per blok proses, fungsi tujuan dapat diformulasikan sebagai berikut :

   Meminimumkan $Z = \sum_{i=1}^{m}(d11 + d12)$

   Untuk i=1,2,…,m tujuan
4. Variabel keputusan

   Dari penelitian ini ada satu tujuan atau sasaran yang ingin dicapai untuk membantu pengambil keputusan dalam membuat perencanaan, sasaran yaitu meminimalkan waktu eksekusi aktivitas pada trace tertentu.
5. Model matematika

   1. Meminimalkan waktu eksekusi aktivitas per blok proses

$$d11 - d12 = Pi$$

Keterangan:

Pi : waktu rata-rata (goal)

$d11$: nilai penyimpangan di bawah Pi

$d12$: nilai penyimpangan di atas Pi

2. Memperoleh biaya aktivitas per blok proses mengikuti goal waktu

Fungsi tujuan:

$$Min\ Z = \sum_{i=1}^{m}(BiXi)$$

Keterangan:

Bi : biaya per trace

6. Penyelesaian optimal

   Hasil kombinasi variabel keputusan dari hasil optimisasi yang dilakukan dengan LINGO diperoleh waktu yang sesuai dengan sasaran atau goal serta biaya yang harus dikeluarkan jika waktu berhasil diminimumkan.

3.6 Keluaran

Tedapat tiga log data yang akan digunakan untuk pengujian, yaitu log data pada bulan TPS, AND, dan OR. Pengujian dilakukan dengan menghitung kualitas model proses yang dibentuk. Model proses yang dibentuk oleh metode usulan akan dikomparasikan dengan metode Alpha. Metode *Alpha* dipilih sebagai metode komparasi karena metode ini adalah metode penemuan model proses yang menggunakan relasi *causal* untuk membentuk model proses.

BAB 4
HASIL PENELITIAN DAN PEMBAHASAN

4.1. Hasil Penelitian

4.1.1 Lingkungan Uji Coba

Sebagai uji coba pada penelitian ini, event log diujikan dengan menggunakan komputer dengan spesifikasi processor Intel® Core™ i7 @ 1.80 GHz, memori 4 GB. Perangkat lunak pendukung adalah sistem operasi *Windows 8 64-bit*, *Java*, *Python* dan *Lingo*.

4.1.2 Preprocessing

Database TPS yang digunakan dapat dilihat pada Tabel 4.1. Cara membaca kolom-kolom satu case yang sama pada Tabel 4.1 adalah dari kiri ke kanan yaitu dari CONTAINER_KEY hingga STACK_DATE. Nomor CONTAINER_KEY yang berbeda menunjukkan case yang berbeda pula. Berikut penjelasan kolom-kolom dari Tabel 4.1:

1. CONTAINER_KEY berisi nomor setiap kontainer yang digunakan sebagai Case_ID pada Tabel 4.2. Pada contoh Tabel 4.2 nomor CONTAINER_KEY yang ditransformasi diberi tanda kotak merah.
2. CTR_SIZE yang berisi 20 dan 40 dengan satuan feet yang akan digunakan sebagai dasar perhitungan biaya aktivitas.
3. CTR_TYPE yang berisi tipe dari kontainer DRY, RFR, atau UNCONTAINER akan digunakan sebagai salah satu data detail lampiran pada log data. DRY merupakan tipe kontainer dengan barang yang tidak memerlukan alat untuk mendinginkan dan menyejukkan udara di dalam ruangan. RFR merupakan singkatan dari refeer yaitu tipe kontainer dengan barang yang memerlukan alat untuk mendinginkan dan menyejukkan udara di dalam ruangan sebagai contoh daging dan buah-buahan. UNCONTAINER merupakan tipe kontainer yang tidak memerlukan kontainer sebagai tempat penyimpanan barang, contohnya adalah alat berat yang tidak dapat ditampung kontainer.
4. GROSS berisi berat dari kontainer termasuk isinya

5. VESSEL_ATB berisi waktu kedatangan kapal yang berisi kontainer-kontainer yang diangkut.
6. DISC_DATE adalah waktu bongkar muat kontainer dari kapal ke pelabuhan.
7. YARD_BLOCK dan YARD_SLOT merupakan tempat diletakkan kontainer di Yard. Kontainer yang memiliki jalur merah atau jalur hijau dibedakan dari letak blok dan slotnya di Yard sehingga memudahkan saat pemeriksaan fisik dan.
8. STACK_DATE adalah waktu diletakkan suatu kontainer di Yard.

Tabel 4.1 Database proses impor barang terminal petikemas

CONTAINER_KEY	CTR_SIZE	CTR_TYPE	GROSS	VESSEL_ATB	DISC_DATE	YARD_BLOCK	YARD_SLOT	STACK_DATE
4454276	20	DRY	24.7	23/01/2016 10:10	23/01/2016 20:31	I	143	23/01/2016 21:17
4453421	20	DRY	19.8	23/01/2016 10:10	23/01/2016 20:32	J	61	23/01/2016 20:54
4454029	20	DRY	19.4	23/01/2016 10:10	23/01/2016 20:32	J	65	23/01/2016 21:17
4456156	40	DRY	29.5	23/01/2016 19:10	23/01/2016 20:33	T	48	23/01/2016 20:43
4454284	20	DRY	20.1	23/01/2016 10:10	23/01/2016 20:33	I	143	23/01/2016 21:18
4454041	20	TNK	24	23/01/2016 10:10	23/01/2016 20:37	I	141	23/01/2016 20:58
4456281	40	RFR	33.5	23/01/2016 19:10	23/01/2016 20:37	M	96	23/01/2016 21:13
4454283	20	DRY	27.3	23/01/2016 10:10	23/01/2016 20:38	I	143	23/01/2016 21:20
4456322	40	DRY	7.7	23/01/2016 19:10	23/01/2016 20:38	T	40	23/01/2016 20:49
4453527	20	DRY	24.9	23/01/2016 10:10	23/01/2016 20:38	J	61	23/01/2016 21:00
4456169	40	RFR	24.8	23/01/2016 19:10	23/01/2016 20:38	D	8	23/01/2016 21:05
4453987	20	DRY	22.5	23/01/2016 10:10	23/01/2016 20:39	M	91	23/01/2016 21:10
4456343	40	DRY	14.8	23/01/2016 19:10	23/01/2016 20:40	I	80	23/01/2016 20:50

Metode pengolahan log data seperti dijelaskan pada Sub Bab 3.3. Hasil dari transformasi log data dapat dilihat pada Tabel 4.2. Pada log data yang tergambar di Tabel 4.2, *message* maupun aktivitas masih terekam menjadi satu di kolom nama aktivitas. Cara membaca kolom-kolom pada Tabel 4.2 adalah dari kiri ke kanan yaitu dari Case ID hingga Detail Lampiran kemudian turun sampai nomor akhir dari Case ID yang sama. Sehingga satu case merupakan kumpulan dari Sender, Originator, Input, Activity yang membentuk satu alur kejadian dalam kasus ini satu case terjadi dari Document Entry via PDE hingga Truck Out, Output, Receiver, Time, Cost, Yard Block, Yard Slot, dan Detail Lampiran. Berikut penjelasan kolom-kolom dari Tabel 4.2:

1. Case ID berisi nomor yang didapatkan dari CONTAINER_KEY.
2. Sender adalah pengirim message berupa *agent* dan bukan perseorangan. Sender dan Receiver yang akan dijadikan sebagai dasar filtering message seperti terlihat pada Tabel 4.3.

3. Originator atau pelaksana dari aktivitas. Originator dijadikan sebagai dasar bahwa aktivitas yang dijalankan bukan sebuah message.
4. Input atau dokumen yang dijadikan sebagai masukan aktivitas dapat dijalankan.
5. Activity adalah aktivitas atau kegiatan yang dijalankan pada satu waktu kejadian.
6. Output atau keluaran dokumen dari aktivitas yang sudah dijalankan.
7. Receiver adalah penerima pesan berupa *agent* dan bukan perseorangan. Sender dan Receiver yang akan dijadikan sebagai dasar filtering message seperti terlihat pada Tabel 4.3.
8. Time atau waktu dijalankannya sebuah aktivitas.
9. Cost atau biaya yang harus dikeluarkan dari setiap aktivitas.
10. Yard Block dan Yard Slot merupakan letak kontainer di Yard. Pengisian Yard Block maupun Yard Slot hanya diisi pada aktivitas Stack_Container_in_Yard.
11. Detail Lampiran atau lampiran tipe kontainer dan penjaluran kontainer. Dry merupakan tipe kontainer yang tidak memerlukan pendingin dan Green Line merupakan jalur hijau. Penjaluran bea cukai dibedakan menjadi tiga bagian, yaitu jalur hijau (Green Line), jalur merah (Red Line), dan jalur kuning (Yellow Line). Jalur hijau merupakan jalur yang tidak memerlukan pemeriksaan fisik dan jalur merah adalah jalur yang memerlukan pemeriksaan fisik. Kriteria kontainer termasuk jalur merah karena hal-hal sebagai berikut:
    - Importir baru;
    - Importir yang termasuk dalam kategori risiko tinggi (high risk importir);
    - Barang impor sementara;
    - Barang Operasional Perminyakan (BOP) golongan II;
    - Barang re-impor;
    - Terkena pemeriksaan acak;
    - Barang impor tertentu yang ditetapkan oleh Pemerintah;

- Barang impor yang termasuk dalam komoditi berisiko tinggi dan/atau berasal dari negara yang berisiko tinggi.

Tabel 4.2 Hasil transformasi berupa log data proses impor barang terminal petikemas

Case ID	Sender	Originator	Input	Activity	Output	Receiver	Time	Cost	Yard Block	Yard Slot	Detail Lampiran
445342		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		23/01/2016 7:22	0			Dry;Green Line
445342	Customer		BC 2.0	Request_Behandle	BC 2.0	SKP	23/01/2016 7:23	0			
445342		TPS		Vessel_Berthing_Process			23/01/2016 10:10	47836,72			
445342		TPS		Discharge_Container			23/01/2016 20:32	428,1332			
445342		TPS		Bring_Container_to_Yard			23/01/2016 20:51	50			
445342		TPS		Stack_Container_in_Yard			23/01/2016 20:54	19,34	J	61	
445342	SKP		BC 2.0	Approve_Behandle	BC 2.0	Customer	24/01/2016 11:07	1,467092			

Hasil dari filtering log data tanpa *message* dapat dilihat pada Gambar 4.3 dimana kolom Sender dan Receiver kosong dan terdapat beberapa aktivitas yang hilang. Message dihilangkan karena kebutuhan untuk *process discovery* dan optimasi waktu dan biaya hanya memerlukan aktivitas. *Message* adalah pesan singkat yang dikirim dari Sender ke Receiver. Sehingga suatu aktivitas disebut *message* jika memiliki pelaksana aktivitas yang berbeda. Pada Tabel 4.3 aktivitas Request_Behandle ditandai dengan kotak merah dengan Sender oleh Customer dan Receiver oleh SKP (Sistem Komputer Pelayanan milik Bea Cukai) yang ditunjukkan pada Tabel 4.2 hilang pada Tabel 4.3. Aktivitas Approve_Behandle yang terdapat pada Tabel 4.2 ditandai dengan kotak merah juga hilang pada Tabel 4.3 karena terdapat Sender oleh SKP dan Receiver oleh Customer.

Tabel 4.3 Hasil filtering berupa log data proses impor barang terminal petikemas tanpa *message*

Case ID	Sender	Originator	Input	Activity	Output	Receiver	Time	Cost	Yard Block	Yard Slot	Detail Lampiran
445342		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		23/01/2016 7:22	0			Dry;Green Line
445342		TPS		Vessel_Berthing_Process			23/01/2016 10:10	47836,72			
445342		TPS		Discharge_Container			23/01/2016 20:32	428,1332			
445342		TPS		Bring_Container_to_Yard			23/01/2016 20:51	50			
445342		TPS		Stack_Container_in_Yard			23/01/2016 20:54	19,34	J	61	

4.1.3 Uji Coba *Process Discovery* dengan Metode Dekomposisi LTL

Log data yang digunakan pada eksperimen tesis ini untuk modifikasi algoritma yang diusulkan ada 3 jenis log data. Log data pertama berisi operator OR dan XOR yang berisi 100 case data, log data kedua berisi operator AND dan XOR

yang berisi 100 case data log data ketiga adalah log data dari PT.TPS hasil transfomasi yang telah dijelaskan pada Subbab sebelumnya.

Mengikuti langkah-langkah pengerjaan yang telah dijelaskan pada Subbab 3.4, berikut adalah hasil metode usulan dalam menemukan model proses dari log data pada Tabel 4.4 dimana hanya terdapat CASE dan ACTIVITY. Hal ini karena keperluan untuk melakukan process discovery hanya diperlukan nomor case dan activity. Berikut hasil uji coba dari langkah-langkah yang telah dijelaskan pada Subbab 3.4.

Langkah 1. Membuat direct succession yang berisi seluruh relasi sequence (>) setiap aktivitas dari seluruh case di data log.

Case PP1: A>B, B>C, C>E, E>G,G>I
Case PP2: A>C, C>D, D>E, E>F, F >I
Case PP3: A>D, D>B, B>E, E>G, G> I
Case PP4: A>D, D>C, C>E, E>G, G> I
……
Case PP100: A>B, B>C, C>E, E>G, G>I

Tabel 4.4 Daftar aktivitas berdasarkan Case pada log data 1

CASE	ACTIVITY
PP1	A
PP1	B
PP1	C
PP1	E
PP1	F
PP1	I
PP2	A
PP2	C
PP2	D
PP2	E
PP2	G
PP2	I
PP3	A
PP3	D
PP3	B
PP3	E
PP3	H
PP3	I

Langkah 2. Membuat seluruh relasi paralel setiap aktivitas dari seluruh case di log data. Pada log data uji 1 ditemukan relasi paralel OR dan XOR. Operator OR yang dilambangkan dengan $\oplus$ dapat ditemukan pada aktivitas A OR SPLIT B, C, D OR JOIN E. Sedangkan operator XOR yang dilambangkan dengan # dapat ditemukan pada aktivitas E XOR SPLIT F, G, H XOR JOIN I.

OR : $B \oplus C \oplus D$

Choice : F # G

G # H

F # H

Langkah 3. Hapus relasi sequence dan paralel yang duplikat untuk mendapatkan jumlah trace di log data.

Langkah 4. Mendapatkan semua trace dari log data seperti yang telah ditampilkan pada Tabel 4.3. Pada Tabel 4.3 menunjukkan bahwa trace yang dapat dibentuk ada 4 trace.

Tabel 4.5 Daftar trace yang dapat dibentuk dari data uji 1

Trace 1	A B C E F G I
Trace 2	A C D E G F I
Trace 3	A D B E F G I
Trace 4	A D C E G F I

Langkah 5. Menentukan transisi (TL) dari workflow net

Transisi: A, B, C, D, E, F, G, I

Langkah 6. Menentukan transisi masukan dari workflow net

Masukan: A

Langkah 7. Menentukan transisi keluaran dari workflow net

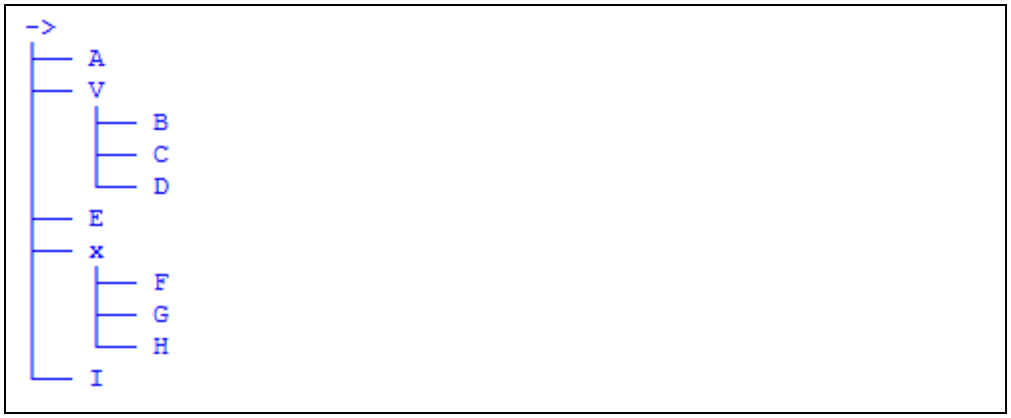
Keluaran: I

Langkah 8. Membuat dekomposis LTL dari relasi sequence dan paralel yang telah dibuat. Hasil running program untuk

LTL : A -> <> (B $\vee$ C $\vee$ D)

E -> O (F $\wedge$ G $\wedge$ H)

Langkah 9. Menemukan model proses berdasarkan relasi terlihat pada Gambar 4.4 dimana operator OR dan XOR dapat ditemukan dengan menggunakan process tree. Operator OR digambarkan dengan tanda “V” untuk aktivitas B, C, dan D. Sedangkan operator XOR digambarkan dengan “x” untuk aktivitas F, G, dan H.



Gambar 4.1 Process tree kasus OR dan XOR

Selanjutnya adalah menerapkan ke log data uji coba 2. Langkah yang sama dilakukan untuk menemukan model proses log data uji coba 2.

Langkah 1. Membuat direct succession yang berisi seluruh relasi sequence (>) setiap aktivitas dari seluruh case di data log. Log data 2 dapat dilihat pada Tabel 4.6.

Case PP1: A>B, B>C, C>D, D>E, E>F, F>I
Case PP2: A>C, C>B, B>D, D>E, E>F, G>I
Case PP3: A>D, D>B, B>C, C>E, E>G, H> I
Case PP4: A>D, D>B, B>C, C>E, E>G, H> I
……
Case PP100: A>B, B>C, C>D, D>E, E>F, F>I

Tabel 4.6 Daftar aktivitas berdasarkan Case pada log data 2

CASE	ACTIVITY
PP1	A
PP1	B
PP1	C
PP1	D
PP1	E
PP1	F
PP1	I
PP2	A
PP2	C
PP2	B
PP2	D
PP2	E
PP2	G
PP2	I

PP3	A
PP3	D
PP3	B
PP3	C
PP3	E
PP3	H
PP3	I

Langkah 2. Membuat seluruh relasi paralel (||) dan choice (#) setiap aktivitas dari seluruh case di log data. Pada log data uji 2 ditemukan relasi paralel AND dan XOR. Operator AND yang dapat ditemukan pada aktivitas A AND SPLIT B, C, D AND JOIN E. Sedangkan operator XOR yang dapat ditemukan pada aktivitas E XOR SPLIT F, G, H XOR JOIN I.

OR : B || C || D

Choice : F # G # H

Langkah 3. Hapus relasi sequence dan paralel yang duplikat untuk mendapatkan jumlah trace di log data

Langkah 4. Mendapatkan semua trace dari log data seperti yang telah ditampilkan pada Tabel 4.5 dimana terdapat 3 trace yang dapat dibentuk.

Tabel 4.7 Contoh trace yang dapat dibentuk dari log data 2

Trace 1	A B C D E F I
Trace 2	A C B D E G I
Trace 3	A D B C E H I

Langkah 5. Menentukan transisi (TL) dari workflow net

Transisi: A, B, C, D, E, F, G, I

Langkah 6. Menentukan transisi masukan dari workflow net

Masukan: A

Langkah 7. Menentukan transisi keluraran dari workflow net

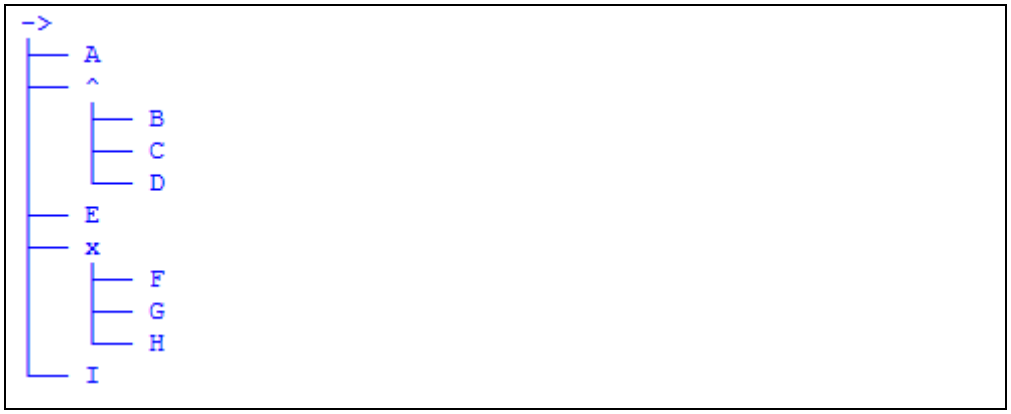
Keluaran: I

Langkah 8. Membuat dekomposisi LTL dari relasi sequence dan paralel yang telah dibuat

LTL : A -> <> (B \/ C \/ D)

E -> O (F /\ G /\ H)

Langkah 9. Menemukan model proses berdasarkan relasi dapat dilihat pada Gambar 4.5 dimana operator AND dan XOR dapat ditemukan dengan menggunakan process tree. Operator AND dilambangkan dengan ^ dan operator XOR dilambangkan dengan x.



Gambar 4.2 Process tree kasus AND dan XOR

Langkah yang sama dilakukan untuk menemukan model proses log data PT TPS. Langkah yang sama dilakukan untuk menemukan model proses log data uji coba PT TPS.

Langkah 1. Membuat direct succession yang berisi seluruh relasi sequence (>) setiap aktivitas dari seluruh case di data log. Hasil dari pembuatan direct succession pada trace jalur hijau dengan tipe kontainer Dry dapat dilihat pada Tabel 4.8.

Tabel 4.8 Direct succession trace jalur hijau

Input	**Relasi**	**Output**
Document_Entry_via_PDE	>	Vessel_Berthing_Process
Vessel_Berthing_Process	>	Discharge_Container
Discharge_Container	>	Bring_Container_to_Yard
Bring_Container_to_Yard	>	Stack_Container_in_Yard
Stack_Container_in_Yard	>	Verification_Document_Behandle
Verification_Document_Behandle	>	Create_document_SPPB
Create_document_SPPB	>	Create_Job_Order_Document_Delivery
Create_Job_Order_Document_Delivery	>	Truck_in
Truck_in	>	Dispatch_WQ_Delivery_to_CHE
Dispatch_WQ_Delivery_to_CHE	>	Determine_Container_Type
Determine_Container_Type	>	Determining_Dry
Determining_Dry	>	Decide_Task_Before_Lift_Container
Decide_Task_Before_Lift_Container	>	Lift_on_Container_Truck
Lift_on_Container_Truck	>	Truck_Go_To_Gate_Out

Input	Relasi	Output
Truck_Go_To_Gate_Out	>	Check_Container_before_Truck_out
Check_Container_before_Truck_out	>	Truck_Out

Langkah 2. Membuat seluruh relasi choice (#) setiap aktivitas dari seluruh case di log data.

Determining_Dry # Determining_Refeer

Determining_Dry # Determining_Uncontainer

Determining_Refeer # Determining_Uncontainer

Langkah 3. Hapus relasi sequence dan paralel yang duplikat untuk mendapatkan jumlah trace di log data

Langkah 4. Mendapatkan semua trace dari log data. Hasil trace yang dapat dibentuk terdapat pada LAMPIRAN dimana terdapat 12 trace. Keduabelas trace tersebut yaitu:

1. Trace jalur merah dan karantina dengan tipe kontainer Dry
2. Trace jalur merah dan karantina dengan tipe kontainer Refeer
3. Trace jalur merah dan karantina dengan tipe kontainer Uncontainer
4. Trace jalur merah tanpa karantina dengan tipe kontainer Dry
5. Trace jalur merah tanpa karantina dengan tipe kontainer Refeer
6. Trace jalur merah tanpa karantina dengan tipe kontainer Uncontainer
7. Trace jalur hijau dengan tipe kontainer Dry
8. Trace jalur hijau dengan tipe kontainer Refeer
9. Trace jalur hijau dengan tipe kontainer Uncontainer
10. Trace jalur hijau dan karantina dengan tipe kontainer Dry
11. Trace jalur hijau dan karantina dengan tipe kontainer Refeer
12. Trace jalur hijau dan karantina dengan tipe kontainer Dry

Langkah 5. Menentukan transisi (TL) dari workflow net

Transisi: Document Entry via PDE, Vessel Berthing, Bring Container to Yard, Stack Container in Yard, Verification Document Quarantine, Create Job Order Document Quarantine, Create Document KH/KT, Stack Container in Yard From Quarantine, Verification Document Behandle, Create Job Order Document Behandle, Bring Container from Yard to Behandle, Stack Container in Behandle Area, Check Goods Behandle, Create document LHP, Stack Container in Yard From Behandle, Create document SPPB, Create Job Order Document Delivery, Truck in, Dispatch WQ Delivery to CHE, Determine Container Type, Determining Dry, Decide Task Before Lift Container, Prepare Tools, Lift on Container Truck, Truck Go To Gate Out, Check Container before Truck out, Truck Out

Langkah 6. Menentukan transisi masukan dari workflow net

Masukan: Document Entry

Langkah 7. Menentukan transisi keluaran dari workflow net

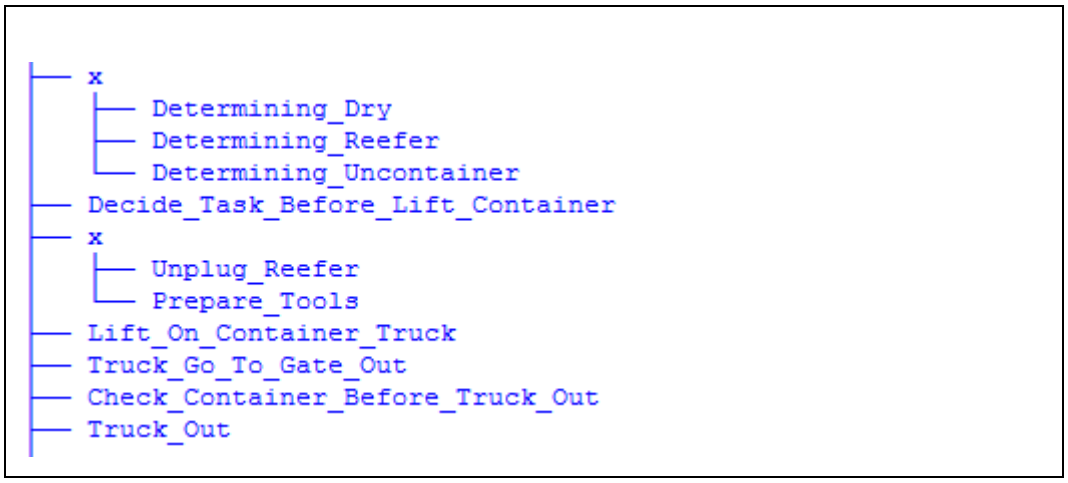
Keluaran: Truck Out

Langkah 8. Membuat dekomposis LTL dari relasi sequence dan paralel yang telah dibuat

LTL : Determine Container Type -> O (Determining Dry \/ Determining Uncontainer \/ Determining Refeer),

Decide Task Before Lift Container -> O (Prepare Tools \/ Unplug Refeer Cable)

Langkah 9. Menemukan model proses berdasarkan relasi dapat dilihat pada Gambar 4.5 dimana operator XOR dapat ditemukan dengan menggunakan process tree. Operator XOR dilambangkan dengan x. Model proses tree dibentuk dengan menggabungkan control-flow pattern yang telah diperoleh dan penambahan atribut pada model proses tree. Hasil model tree berdasarkan control-flow pattern yang didapatkan ditampilkan pada Gambar 4.10.



Gambar 4.3 Process tree kasus TPS

4.1.3. Uji coba optimasi waktu dan biaya

Dari aktivitas-aktivitas yang dijalankan pada PT. TPS, terdapat aktivitas barang dan dokumen yang dijalankan sekuential yaitu setelah melewati beberapa aktivitas dari Vessel Berthing hingga aktivitas Stack Container in Yard baru dijalankan aktivitas Verifikasi Dokumen. Kedua aktivitas ini dapat dijalankan secara paralel, karena tidak saling bergantungan satu dengan yang lainnya, dengan alasan:

- Aktivitas setiap departemen independen
- Waktu dan tempat eksekusi berbeda
- Resource yang menangani setiap aktivitas berbeda

Jika eksekusi pada kedua aktivitas ini dilakukan secara paralel, maka relasi model proses kedua aktivitas ini menjadi AND karena aktivitas Verifikasi Dokumen dan Vessel Berthing harus dijalankan secara bersamaan. Berdasarkan waktu dan biaya yang telah dijabarkan pada Subbab 3.5, yang digunakan untuk optimasi pada trace yang jalur merah dan karantina, trace jalur merah, dan trace jalur hijau adalah total dari waktu rata-rata durasi eksekusi setiap aktivitas sebagai goal dan yang akan diminimalkan adalah total waktu maksimum dari setiap aktivitas pada trace yang jalur merah dan karantina, trace jalur merah, dan trace jalur hijau. Model matematika untuk optimasi dengan goal programming trace jalur merah dijabarkan pada LAMPIRAN.

Langkah-langkah optimasi waktu dan biaya dengan Goal programming pada trace jalur merah dan karantina adalah sebagai berikut:

1. Memisahkan trace barang dan dokumen seperti terlihat pada Tabel 3.14
2. Menghitung waktu rata-rata serta biaya rata-rata setiap aktivitas per trace seperti telihat pada

Tabel 4.9 Waktu dan biaya rata-rata yang didapatkan dari event log

No.	**Aktivitas**	**Waktu (menit)**	**Biaya ($)**
1	Discharge_Container	5,095238	498,5905
2	Bring_Container_to_Yard	23,71655	61,9
3	Stack_Container_in_Yard	2,502502	23,93524
4	Verification_Document_Quarantine	2008,8	184,9767
5	Create_Job_Order_Document_Quarantine	37,61795	181,87
6	Bring_Container_from_Yard_to_Quarantine	91,0355	177,2102
7	Stack_Container_in_Quarantine_Area	2,257779	182,7841
8	Check_Goods_Quarantine	45,14105	357,8939
9	Create_document_KH/KT	90,28331	441,4857
10	Stack_Container_in_Yard_From_Quarantine	1128,54	176,0086
11	Verification_Document_Behandle	2207,359	6,441655
12	Create_Job_Order_Document_Behandle	3250,193	464,448
13	Stack_Container_in_Behandle_Area	1850,805	477,816
14	Check_Goods_Behandle	45,14105	1155,777
15	Create_document_LHP	90,28331	23360,3
16	Bring_Container_from_Yard_to_Behandle	223,4511	491,9728
17	Stack_Container_in_Yard_From_Behandle	2,256568	464,1482

No.	Aktivitas	Waktu (menit)	Biaya ($)
18	Create_document_SPPB	428,8448	2694,676
19	Create_Job_Order_Document_Delivery	610,5404	30783,18
20	Truck_in	574,3056	10602,86
21	Dispatch_WQ_Delivery_to_CHE	1,318755	941,9415
22	Determine_Container_Type	2,752009	2080,956
23	Determining_Dry	2,272167	18,87571
24	Decide_Task_Before_Lift_Container	1,526951	2097
25	Lift_on_Container_Truck	2,534427	19,78476
26	Truck_Go_To_Gate_Out	47,00956	2283,088
27	Check_Container_before_Truck_out	1,48124	2070,151
28	Truck_Out	1,516008	10562,11

3. Formulasi fungsi tujuan

   Untuk meminimalkan simpangan waktu dan biaya setiap aktivitas per blok proses, fungsi tujuan dapat diformulasikan sebagai berikut :

   Meminimumkan $Z = \mathrm{D1plus} + \mathrm{D1minus} + \mathrm{D2plus} + \mathrm{D2minus}$

   Untuk D1 merupakan simpangan waktu dan D2 merupakan simpangan biaya.

4. Variabel keputusan

   Dari penelitian ini terdapat 28 aktivitas yang menjadi variabel keputusan. Variabel keputusan merupakan variabel yang dijadikan dasar perhitungan pada model matematika yang dibuat.

Tabel 4.10 Variabel keputusan pada trace jalur merah

No.	Aktivitas	Variabel Keputusan
1	Discharge_Container	x1
2	Bring_Container_to_Yard	x2
3	Stack_Container_in_Yard	x3
4	Verification_Document_Quarantine	x4
5	Create_Job_Order_Document_Quarantine	x5
6	Bring_Container_from_Yard_to_Quarantine	x6
7	Stack_Container_in_Quarantine_Area	x7

No.	Aktivitas	Variabel Keputusan
8	Check_Goods_Quarantine	x8
9	Create_document_KH/KT	x9
10	Stack_Container_in_Yard_From_Quarantine	x10
11	Verification_Document_Behandle	x11
12	Create_Job_Order_Document_Behandle	x12
13	Stack_Container_in_Behandle_Area	x13
14	Check_Goods_Behandle	x14
15	Create_document_LHP	x15
16	Bring_Container_from_Yard_to_Behandle	x16
17	Stack_Container_in_Yard_From_Behandle	x17
18	Create_document_SPPB	x18
19	Create_Job_Order_Document_Delivery	x19
20	Truck_in	x20
21	Dispatch_WQ_Delivery_to_CHE	x21
22	Determine_Container_Type	x22
23	Determining_Dry	x23
24	Decide_Task_Before_Lift_Container	x24
25	Lift_on_Container_Truck	x25
26	Truck_Go_To_Gate_Out	x26
27	Check_Container_before_Truck_out	x27
28	Truck_Out	x28

5. Model matematika

1. Meminimalkan waktu eksekusi aktivitas per blok proses

$$D1plus - D1minus = Pi$$

Keterangan:

Pi : waktu rata-rata (goal)

$D1plus$: nilai penyimpangan di bawah Pi

$D1minus$: nilai penyimpangan di atas Pi

2. Memperoleh biaya aktivitas per blok proses mengikuti goal waktu

Fungsi tujuan:

$$D2plus - D2minus = Pi$$

Keterangan:

Pi : biaya rata-rata (goal)

$D2plus$: nilai penyimpangan di bawah Pi

$D2minus$: nilai penyimpangan di atas Pi

6. Penyelesaian optimal

Hasil kombinasi variabel keputusan dari hasil optimisasi yang dilakukan dengan LINGO diperoleh waktu yang sesuai dengan sasaran atau goal serta biaya yang harus dikeluarkan jika waktu berhasil diminimumkan.

Penyelasaian optimal untuk mendapatkan waktu minimal pada Tabel 4.11 dimana 5.09524 merupakan waktu rata-rata aktivitas dalam menit, * merupakan lambang perkalian, x1 merupakan variabel keputusan seperti yang telah ditampilkan pada Tabel 4.10, D1plus D1minus merupakan nilai simpangan dan 12688.33 merupakan tujuan yang ingin dicapai:

Tabel 4.11 Model matematika optimasi waktu

```
5.09524*x1+23.7165*x2+2.5025*x3+2008.8*x4+37.6179*x5+91.035
5*x6+2.25778*x7+45.1411*x8+90.2833*x9+1128.54*x10+2207.36*x
11+3250.19*x12+1850.8*x13+45.1411*x14+90.2833*x15+223.451*x
16+2.25657*x17+428.845*x18+610.54*x19+574.306*x20+1.31875*x
21+2.75201*x22+2.27217*x23+1.52695*x24+2.53443*x25+47.0096*
x26+1.48124*x27+1.51*x28-D1plus+D1minus=12688.33;
```

Penyelasaian optimal untuk mendapatkan biaya minimal dimana 498.59 merupakan waktu rata-rata aktivitas dalam menit, * merupakan lambang perkalian, x1 merupakan variabel keputusan seperti yang telah ditampilkan pada Tabel 4.10, D2plus D2minus merupakan nilai simpangan dan 92862.18 merupakan tujuan yang ingin dicapai:

Tabel 4.12 Model matematika optimasi biaya

498.59*x1+61.9*x2+23.9352*x3+184.977*x4+181.87*x5+177.21*x6 +182.784*x7+379.7854*x8+863.73*x9+191.9372*x10+6.44165*x11 +464.448*x12+477.816*x13+1155.78*x14+24277.5*x15+491.973*x1 6+464.148*x17+2694.68*x18+30783.2*x19+10602.9*x20+941.942*x 21+2080.96*x22+18.8757*x23+2097*x24+19.7848*x25+2283.09*x2 6+2070.15*x27+10562.11*x28-D2plus+D2minus=92862.18;

4.2. Evaluasi

Evaluasi ini dilakukan untuk proses discovery dengan metode usulan dengan membandingkan kinerja sistem yang dikembangkan dengan sistem lain. Perbandingan dilakukan dengan menggunakan hasil yang didapatkan pada hasil uji studi kasus.

4.2.1 Perbandingan hasil kualitas model proses

Pemilihan model proses yang dilakukan dengan mencari nilai kualitas tertinggi dari keseluruhan model proses tersebut. Kualitas diukur dari segi fitness, presisi, simplicity, dan generalisasi. Apabila model proses yang memenuhi kriteria tersebut lebih dari satu atau tidak memenuhi, maka akan dipilih rata-rata kualitas tertinggi dari seluruh kualitas tersebut. Hasil kualitas model proses tree dapat dilihat pada Tabel 4.13. dengan anomali.

Evaluasi kinerja pembentukan model proses imperatif berdasarkan empat sisi kualitas, yaitu fitness, presisi, simplicity, dan generalisasi. Evaluasi yang dilakukan dengan membandingkan kualitas hasil metode yang diusulkan dan kualitas hasil metode Alpha. Metode Alpha adalah metode discovery model proses yang dapat menemukan model proses dengan menggunakan hubungan causal. Hasil metode akan semakin bagus apabila nilai kualitas yang didapatkan tinggi. Rentang nilai kualitas adalah 0,0 sampai 1,0.

Dari hasil evaluasi tersebut, hasil metode yang diusulkan memiliki nilai simplicity yang lebih tinggi dari hasil metode Alpha. Kemudian, nilai fitness dari hasil metode yang diusulkan maupun hasil dari metode Alpha memiliki nilai yang tinggi. Nilai tinggi dalam simplicity pada metode usulan dipengaruhi oleh

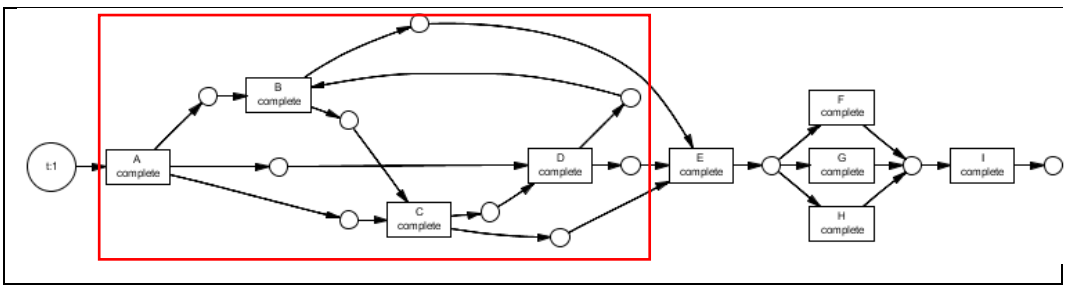
kemasmpuan metode yang diusulkan dalam mendeteksi relasi invisible task. Metode Alpha tidak mendeteksi invisible task sehingga banyak aktivitas yang didefinisikan berulang (redundan). Pendefinisian aktivitas secara redundan ini mengurangi nilai simplicity.

Tabel 4.13 Tabel perbandingan hasil perhitungan kualitas model proses

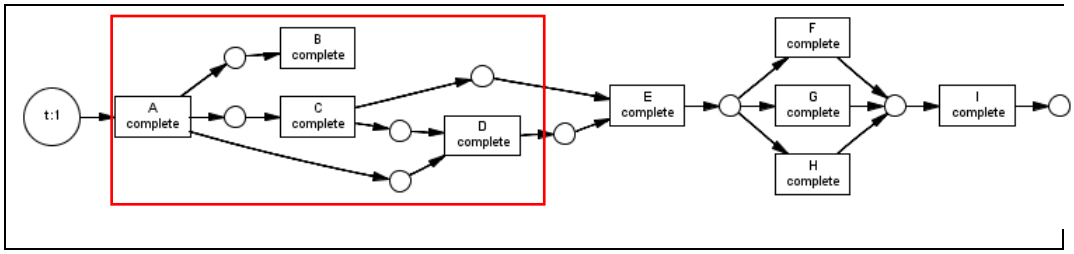
	Fitness (0,00-1,0)	**Precision (0,00-1,00)**	**Simplicity (0,00-1,00)**	**Generalization (0,00-1,00)**
Metode Usulan dengan Anomali	0.92	0.68	0.87	0.97
Alpha	0.92	0.68	0.82	0.97

4.2.2 Perbandingan model dengan Alpha

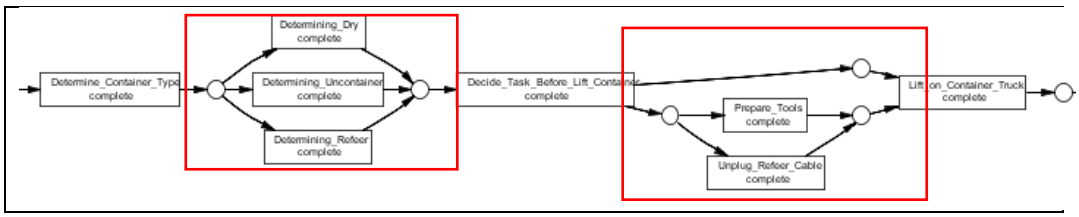
Model proses yang terbentuk dengan algoritma dekomposisi LTL dengan Data log Uji Coba 1, Data log Uji Coba 2 dan Data log PT TPS seperti pada Gambar 4.5, Gambar 4.7 dan Gambar 4.10 pada Subbab 4.1.3. Dengan menggunakan data log yang sama, model proses akan di *discover* dengan algoritma Alpha dengan kakas bantu ProM seperti ditunjukkan pada Gambar 4.15 untuk data uji coba 1, Gambar 4.16 untuk data uji coba 2 dan Gambar 4.17 untuk data log PT TPS dimana pada Gambar 4.15 tidak dapat ditemukan operator AND ditandai dengan kotak merah pada aktivitas A, B, C, dan D sedangkan pada metode usulan dapat ditemukan. Pada Gambar 4.16 tidak dapat ditemukan operator AND sedangkan metode usulan dapat ditemukan. Pada Gambar 4.17 dan metode usulan sama-sama dapat menemukan operator XOR.



Gambar 4.4 Hasil running ProM untuk data uji 1 tidak dapat menemukan operator OR



Gambar 4.5 Hasil running ProM untuk data uji 2 tidak dapat menemukan operator AND



Gambar 4.6 Hasil running ProM untuk data uji PT. TPS dapat menemukan operator XOR

4.2.3 Perbandingan hasil optimasi

Hasil perbandingan optimasi dari tiga trace dapat dilihat pada Tabel 4.13. Pada Tabel 4.13 goal merupakan tujuan yang ingin dicapai dari waktu atau biaya, awal merupakan jumlah waktu atau biaya dari penjumlahan data yang terdapat di event log, sedangkan simpangan merupakan hasil pengurangan durasi yang dapat dicapai dengan menggunakan goal programming dimana trace jalur merah merupakan trace yang jauh dari goal, oleh karena itu perlu pengurangan waktu yang cukup banyak. Durasi waktu optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah dan karantina adalah 96,31 menit, durasi waktu optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah adalah 3415,17 menit, dan durasi waktu optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur hijau adalah 96,16 menit. Biaya optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah dan karantina adalah 0,0, biaya optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah adalah $22271,05, dan biaya optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur hijau adalah $0,0. Terdapat biaya $0,0 hal ini karena biaya sudah mencapai goal sehingga tidak diperlukan pengurangan biaya.

Tabel 4.14 Hasil optimasi waktu dan biaya

	Waktu (menit)			Biaya ($)		
	Goal	Awal	Simpangan	Goal	Awal	Simpangan
Trace jalur merah dan karantina	12688,3	12784,61	96,31	92862,00	92862,18	0,00
Trace jalur merah	10853,44	14282,49	3415,17	84931,83	107202,88	22271,05
Trace jalur hijau	8964,51	9060,67	96,16	79445,55	79445,75	0,00

BAB 5
KESIMPULAN DAN SARAN

Berdasarkan uji coba dan analisis hasil pengujian pada metode, dapat diuraikan beberapa kesimpulan. Juga akan ditambahkan sedikit saran guna pengembangan kedepannya.

5.1. Kesimpulan

Dari hasil uji coba data log yang telah dilakukan terhadap sistem yang dikembangkan berdasarkan analisis seluruh metodologi, diambil kesimpulan sebagai berikut:

1. Pembentukan dekomposisi LTL dibentuk dengan memanfaatkan relasi antar aktivitas. Relasi direct succession dijadikan pembentukan dekomposisi sequence, relasi parallel (||) dijadikan sebagai dasar pembentukan dekomposisi AND, relasi choice (#) dijadikan sebagai dasar pembentukan dekomposisi XOR, sedangkan relasi OR (⊕) dijadikan sebagai dasar pembentukan dekomposisi OR.

2. Dengan menggunakan dekomposisi LTL operator paralel AND, OR dan XOR dapat ditemukan dan operator pada process tree dapat digambarkan sedangkan pada alpha tidak dapat ditemukan untuk operator AND dan OR.

3. Pembentukan process tree dari dekomposisi LTL dibentuk dengan memanfaatkan notasi pada LTL. Notasi <> jika bertemu dengan V maka akan menghasilkan operator OR. Notasi <> jika bertemu dengan /\ maka akan menghasilkan operator AND. Sedangkan notasi O jika bertemu dengan \/ maka akan menghasilkan operator XOR.

4. Nilai fitness, presisi, simplicity, dan generalisasi algoritma usulan adalah 0.92, 0.68 , 0.87, and 0.97 untuk data log PT TPS

5. Paralelisasi dapat terjadi jika aktivitas setiap departemen independen, waktu dan tempat eksekusi berbeda dan resource yang menangani setiap aktivitas berbeda. Pada studi kasus data log PT TPS, trace barang dan dokumen dapat diparalelisasikan
6. Durasi waktu optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah dan karantina adalah 96,31 menit, durasi waktu optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah adalah 3415,17 menit, dan durasi waktu optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur hijau adalah 96,16 menit. Biaya optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah dan karantina adalah 0,0, biaya optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur merah adalah $22271,05, dan biaya optimasi yang dapat dikurangi agar mencapai goal untuk trace jalur hijau adalah $0,0. Terdapat biaya $0,0 hal ini karena biaya sudah mencapai goal sehingga tidak diperlukan pengurangan biaya.
7. Optimasi waktu dan biaya dengan menggunakan goal programming dapat meminimalkan waktu rata-rata maksimum dan rata-rata menjadi waktu minimum trace tertentu.

5.2. Saran

Saran yang diberikan untuk analisis dan pengembangan sistem pada penelitian tesis ini antara lain:

1. Menemukan dekomposisi model proses dengan menggunakan LTL pada kasus pilihan tidak bebas dan loop
2. Menggunakan metode Inductive Miner untuk mendekomposisi LTL

DAFTAR PUSTAKA

Alves de Medeiros, A. K., Weijters, A. J. M. M., & van der Aalst, W. M. P. (2006). Genetic process mining: A basic approach and its challenges. *Business Process Management Workshops (BPM 2005)*, *3812*(task C), 203–215. https://doi.org/10.1007/11678564_18

Buijs, J. C. A. M., Van Dongen, B. F., & Van Der Aalst, W. M. P. (2012). On the role of fitness, precision, generalization and simplicity in process discovery. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7565 LNCS*(PART 1), 305–322. https://doi.org/10.1007/978-3-642-33606-5_19

Buijs, J. C. a M. (2014). Flexible Evolutionary Algorithms for Mining Structured Process Models. *2014*, 345. https://doi.org/10.6100/IR780920

Burattin, A., Maggi, F. M., & Sperduti, A. (2016). Conformance checking based on multi-perspective declarative process models. *Expert Systems with Applications*, *65*, 194–211. https://doi.org/10.1016/j.eswa.2016.08.040

Eck, M. Van. (2013). Alignment-based Process Model Repair and its Application to the Evolutionary Tree Miner, (December).

Levi, A. (2014). ISSN : 1963-6590 (Print) ISSN : 2442-2630 (Online), 151–167.

Loreto, D. A., & Vargas, J. C. (2012). *Sampling and Abstract Interpretation for Tackling Noise in Process Discovery*. Barcelona.

Maggi, F. M., Di Francescomarino, C., Dumas, M., & Ghidini, C. (2013). Predictive Monitoring of Business Processes. Retrieved from http://arxiv.org/abs/1312.4874

Maggi, F. M., & Westergaard, M. (2012). of LTL-Based Declarative Process Models, (257593), 131–146.

Rizkikurniadi, F. P. (2014). Studi Pengurangan Dwelling Time Petikemas (Studi Kasus : Terminal Petikemas Surabaya) Selayang Pandang.

Sarno, R., Wibowo, W. A., Effendi, Y. A., & Sungkono, K. R. (2016). Determining Process Model Using Time-Based Process Mining and Control-Flow Pattern, *14*(1). https://doi.org/10.12928/TELKOMNIKA.v14i1.3257

Vahedian Khezerlou, A., & Alizadeh, S. (2014). A new model for discovering process trees from event logs. *Applied Intelligence*, *41*(3), 725–735. https://doi.org/10.1007/s10489-014-0564-7

van Eck, M. L., Buijs, J. C. A. M., & van Dongen, B. F. (2015). Genetic process mining: Alignment-based process model mutation. *Lecture Notes in Business Information Processing*, *202*, 291–303. https://doi.org/10.1007/978-3-319-15895-2_25

Vázquez-Barreiros, B., Mucientes, M., & Lama, M. (2014). A genetic algorithm for process discovery guided by completeness, precision and simplicity. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8659 LNCS*, 118–133. https://doi.org/10.1007/978-3-319-10172-9_8

Verbeek, H. M. W. (2016). Divide and Conquer. *New Left Review*, (28), 1–3.

LAMPIRAN A

Dekomposisi LTL Log Data PT TPS

Control Flow Pattern	LTL
SEQUENCE	• Document_Entry_via_PDE -> _O (Vessel_Berthing_Process) • Vessel_Berthing_Process -> _O (Discharge_Yard_Planning) • Discharge_Yard_Planning -> _O (Bring_Container_to_Yard) • Bring_Container_to_Yard -> _O (Stack_Container_In_Yard), • Verification_Document_Quarantine -> _O (Create_Job_Order_Document_Quarantine) • Create_Job_Order_Document_Quarantine -> _O (Bring_Container_from_Yard_to_Quarantine) • Bring_Container_from_Yard_to_Quarantine -> _O (Stack_Container_in_Quarantine_Area) • Stack_Container_in_Quarantine_Area -> _O (Check_Goods_Quarantine) • Check_Goods_Quarantine -> _O (Create_document_KH/KT) • Create_document_KH/KT -> _O (Stack_Container_in_Yard_From_Quarantine), • Verification_Document_Behandle -> _O (Create_Job_Order_Document_Behandle) • Create_Job_Order_Document_Behandle -> _O (Stack_Container_in_Behandle_Area) • Stack_Container_in_Behandle_Area -> _O (Check_Goods_Behandle) • Check_Goods_ Behandle -> _O (Create_document_LHP)

	• Create_document_LHP -> _O (Bring_Container_from_Yard_to_Behandle) • Bring_Container_from_Yard_to_Behandle -> _O (Stack_Container_in_Yard_From_Behandle), • Create_document_SPPB -> _O (Create_Job_Order_Document_Delivery) • Create_Job_Order_Document_Delivery -> _O (Truck_in) • Truck_in -> _O (Dispatch_WQ_Delivery_to_CHE), • Lift_on_Container_Truck -> _O (Truck_Go_To_Gate_Out) • Truck_Go_To_Gate_Out -> _O (Check_Container_before_Truck_out), • Check_Container_before_Truck_out -> _O (Truck_out)
XOR	• Determine_Container_Type -> _O (Determining_Dry \/ Determining_Uncontainer \/ Determining_Refeer), • Decide_Task_Before_Lift_Container -> _O (Prepare_Tools \/ Unplug_Refeer_Cable) • Stack_Container_In_Yard -> _O (Verification_Document_Quarantine \/ Skip1) • Verification_Document_Behandle -> _O (Create_Job_Order_Document_Behandle \/ Skip2)
AND	[]
OR	[]
SKIP	• Verification_Document_Behandle -> _O (Create_document_SPPB) • Stack_Container_in_Yard -> _O (Verification_Document_Behandle)

LAMPIRAN B

Model Matematika Optimasi Goal Programming Trace Jalur Merah dan Karantina

```
Min=D1plus+D1minus+D2plus+D2minus;
5.09524*x1+23.7165*x2+2.5025*x3+2008.8*x4+37.6179*x5+91.0355*x6+
2.25778*x7+45.1411*x8+90.2833*x9+1128.54*x10+2207.36*x11+3250.19
*x12+1850.8*x13+45.1411*x14+90.2833*x15+223.451*x16+2.25657*x17+
428.845*x18+610.54*x19+574.306*x20+1.31875*x21+2.75201*x22+2.272
17*x23+1.52695*x24+2.53443*x25+47.0096*x26+1.48124*x27+1.51*x28-
D1plus+D1minus=12688.33;
498.59*x1+61.9*x2+23.9352*x3+184.977*x4+181.87*x5+177.21*x6+182.
784*x7+379.7854*x8+863.73*x9+191.9372*x10+6.44165*x11+464.448*x1
2+477.816*x13+1155.78*x14+24277.5*x15+491.973*x16+464.148*x17+26
94.68*x18+30783.2*x19+10602.9*x20+941.942*x21+2080.96*x22+18.875
7*x23+2097*x24+19.7848*x25+2283.09*x26+2070.15*x27+10562.11*x28-
D2plus+D2minus=92862.18;
x1>=1;
x2>=1;
x3>=1;
x4>=1;
x5>=1;
x6>=1;
x7>=1;
x8>=1;
x9>=1;
x10>=1;
x11>=1;
x12>=1;
x13>=1;
x14>=1;
x15>=1;
x16>=1;
x17>=1;
x18>=1;
x19>=1;
x20>=1;
x21>=1;
x22>=1;
x23>=1;
x24>=1;
x25>=1;
x26>=1;
x27>=1;
```

Model Matematika Optimasi Goal Programming Trace Jalur Merah

```
Min=D1plus+D1minus+D2plus+D2minus;
5.495312*x1+19.36017*x2+2.502851*x3+3409.676*x4+5020.524*x5+2858
.91*x6+69.7295*x7+
139.459*x8+345.161*x9+3.486476*x10+662.4303*x11+943.0916*x12+727
.8344*x13+1.341882*x14+
2.700437*x15+2.3036*x16+1.500249*x17+2.487358*x18+47.60374*x19+1
.521092*x20+1.49164*x21-D1plus+D1minus=10853.44;
483.6238*x1+60.93313*x2+23.56188*x3+6.69809*x4+450.0389*x5+476.0
066*x6+1131.964*x7+44542.11*x8+
436.379*x9+504.367*x10+2638.514*x11+29117.07*x12+9715.06*x13+800
.9351*x14+1723.969*x15+18.54375*x16+
```

```
1959.597*x17+19.47563*x18+1702.192*x19+1836.97*x20+9554.871*x21-
D2plus+D2minus=84931.83;
x1>=1;
x2>=1;
x3>=1;
x4>=1;
x5>=1;
x6>=1;
x7>=1;
x8>=1;
x9>=1;
x10>=1;
x11>=1;
x12>=1;
x13>=1;
x14>=1;
x15>=1;
x16>=1;
x17>=1;
x18>=1;
x19>=1;
x20>=1;
x21>=1;
```

Model Matematika Optimasi Goal Programming Trace Jalur Hijau

```
Min=D1plus+D1minus+D2plus+D2minus;
5.637634*x1+24.04248*x2+2.522576*x3+2139.478*x4+40.06516*x5+96.9
5782*x6+2.403722*x7+48.07805*x8+96.15682*x9+1201.954*x10+2350.95
5*x11+576.938*x12+650.257*x13+1746.433*x14+1.313368*x15+2.727642
*x16+2.31968*x17+1.492927*x18+2.495403*x19+65.49093*x20+1.476581
*x21+1.482398*x22-D1plus+D1minus=8964.516;
563.8099*x1+65.31645*x2+25.25452*x3+172.736*x4+191.1454*x5+184.6
901*x6+179.5299*x7+371.3689*x8+451.219*x9+174.5931*x10+7.001879*
x11+2613.377*x12+38278.9*x13+12750.86*x14+1072.528*x15+2456.123*
x16+20.04871*x17+2549.082*x18+20.8771*x19+2533.004*x20+2342.257*
x21+12421.83*x22-D2plus+D2minus=79445.55;
x1>=1;
x2>=1;
x3>=1;
x4>=1;
x5>=1;
x6>=1;
x7>=1;
x8>=1;
x9>=1;
x10>=1;
x11>=1;
x12>=1;
x13>=1;
x14>=1;
x15>=1;
x16>=1;
x17>=1;
x18>=1;
x19>=1;
x20>=1;
x21>=1;
```

```
x22>=1;
```

LAMPIRAN C
Pembuatan Transisi

```
private void createTransition() {
      Character iter = 'A';

      for (int i = 0; i < traces.size(); i++) {
         Vector<Integer> x = new Vector<Integer>();
         for (int j = 0; j < traces.get(i).size(); j++) {
            x.add(0);
            String val = traces.get(i).get(j);
            if (!val.equals(finish) && !transition.containsValue(val)) {
               transition.put(iter++, val);
               activityFlag.add(0);
            }
         }
         accessed.add(x);
      }
      transition.put(iter, finish);
      activityFlag.add(0);
   }
```

Pembuatan Tuple dan Penemuan Control Flow AND, XOR, dan OR

```
private void createTuples() {
      /* Set Map for Creating Tuples */
      map = new int[transition.size()][transition.size()];
      int size = transition.size();

      for (int i = 0; i < traces.size(); i++) {
         for (int j = 0; j < traces.get(i).size() - 1; j++) {

map[getTransitionOrder(traces.get(i).get(j))][getTransitionOrder(traces.get(i).g
et(j + 1))]++;
         }
      }

      for (int i = 0; i < traces.size() - 1; i++) {
         //System.out.println("Trace " + i + ": " + traces.get(i));
         for (int gap = traces.get(i).size() - 2; gap >= 1; gap--) {
              for (int j = 0; j < traces.get(i).size() - 2; j++) {
                String xor = new String();
                String or = new String();
                boolean flagAND = false;
                boolean flagXOR = false;
                boolean flagOR = false;
                int k = j + gap + 1;
                if (activityFlag.get(getTransitionOrder(traces.get(i).get(j))) == 1) {
                   continue;
```

```
            }
            if (k - 1 >= 2 && k < traces.get(i).size() &&
activityFlag.get(getTransitionOrder(traces.get(i).get(j))) == 0 &&
!mapTuples.contains(traces.get(i).subList(j, k + 1)) &&
!accessed.get(i).subList(j, k + 1).contains(1)) {
                for (int l = i + 1; l < traces.size(); l++) {
                   for (int m = 0; m < traces.get(l).size() - 2; m++) {
                      int same = 0;
                      int dif = 0;
                      int n = m + gap + 1;
                      if (n - m >= 2 && n < traces.get(l).size() &&
traces.get(i).get(j).equalsIgnoreCase(traces.get(l).get(m)) &&
traces.get(i).get(k).equalsIgnoreCase(traces.get(l).get(n)) &&
!traces.get(l).subList(m, n + 1).equals(traces.get(i).subList(j, k + 1))) {
                         boolean flagXorAct = true;
                         for (int zz = m + 1; zz <= n; zz++) {
                            if
(activityFlag.get(getTransitionOrder(traces.get(l).get(zz))) == 1) {
                               flagXorAct = false;
                               break;
                            }
                         }
                         if (flagXorAct) {
                            for (int loop = j + 1; loop < k; loop++) {
                               if (traces.get(l).subList(m, n +
1).contains(traces.get(i).get(loop))) {
                                  if (!traces.get(l).get(m + loop -
j).equals(traces.get(i).get(loop))) {
                                     dif++;
                                  }
                                  same++;
                               }
                            }

                            if (same == 0) {
                               if (!flagXOR) {
                                  flagXOR = true;
                                  XOR.add(traces.get(i).get(j) + " XOR SPLIT " +
traces.get(i).subList(j + 1, k) + " XOR JOIN " + traces.get(i).get(k));
                                  mapTuples.add(traces.get(i).subList(j, k + 1));
                                  setAccessed(i, j, k);
                                  setActivityFlag(traces.get(i).get(j));
                                  xor = traces.get(i).subList(j + 1, k).toString();
                               }
                               XOR.add(traces.get(i).get(j) + " XOR SPLIT " +
traces.get(l).subList(m + 1, n) + " XOR JOIN " + traces.get(i).get(k));
                               mapTuples.add(traces.get(l).subList(m, n + 1));
```

```
                        setAccessed(l, m, n);
                        setActivityFlag(traces.get(l).get(m + 1));
                        xor += "\\/" + traces.get(l).subList(m + 1,
n).toString();
                    }
                else if (same == gap && dif >= gap - 1) {
                    System.out.println("AND");
                    if (!flagAND) {
                        ltl("AND", i, j, k, "", "");
                        flagAND = true;
                        AND.add(traces.get(i).get(j) + " AND SPLIT " +
traces.get(i).subList(j + 1, k) + " AND JOIN " + traces.get(i).get(k));
                        mapTuples.add(traces.get(i).subList(j, k + 1));
                        setActivityFlag(traces.get(i).get(j));
                        setAccessed(i, j, k);
                    }
                    AND.add(traces.get(i).get(j) + " AND SPLIT " +
traces.get(l).subList(m + 1, n) + " AND JOIN " + traces.get(i).get(k));
                    mapTuples.add(traces.get(l).subList(m, n + 1));
                    setAccessed(l, m, n);
                    setActivityFlag(traces.get(l).get(m + 1));
                } else if (gap >= 2 && dif == gap - 1) {
                    if (!flagOR) {
                        flagOR = true;
                        OR.add(traces.get(i).get(j) + " OR SPLIT " +
traces.get(i).get(j + 1) + " OR JOIN " + traces.get(i).get(k));
                        mapTuples.add(traces.get(i).subList(j, k + 1));
                        setAccessed(i, j, k);
                        setActivityFlag(traces.get(i).get(j));
                        or = traces.get(i).get(j + 1).toString();
                    }
                    OR.add(traces.get(i).get(j) + " OR SPLIT " +
traces.get(l).get(m + 1) + " OR JOIN " + traces.get(i).get(k));
                    mapTuples.add(traces.get(l).subList(m, n + 1));
                    setAccessed(l, m, n);
                    setActivityFlag(traces.get(l).get(m + 1));
                    or += "\\/" + traces.get(l).get(m + 1).toString();
                    OR.add(traces.get(i).get(j) + " OR SPLIT " +
traces.get(l).get(m + 2) + " OR JOIN " + traces.get(i).get(k));
                    mapTuples.add(traces.get(l).subList(m, n + 2));
                    setAccessed(l, m, n);
                    setActivityFlag(traces.get(l).get(m + 2));
                    or += "\\/" + traces.get(l).get(m + 2).toString();
                } }
            } else if (n >= traces.get(l).size()) {
                break;
            }
```

```
                    }
                }
                if (flagXOR) {
                    ltl("XOR", i, j, k, xor, "");
                }
                if (flagOR) {
                    ltl("OR", i, j, k, "", or);
                }
            } else if (k >= traces.get(i).size()) {
                break;
            }
        }
    }
}
```

Penemuan Control Flow Skip

```
private void findSkip()
    {
        Vector<List<String>> checkSkip = new Vector<List<String>>();
        for(int i=0; i<traces.size(); i++) {
            for(int j=0, k=j+1; k<traces.get(i).size(); j++, k++) {
                boolean flagSkip = false;
                if(activityFlag.get(getTransitionOrder(traces.get(i).get(j))) == 0 &&
!checkSkip.contains(traces.get(i).subList(j, k + 1))) {
                    for(int l=i+1; l<traces.size(); l++) {
                        for(int m=0; m<traces.get(l).size()-1; m++) {
                            if(activityFlag.get(getTransitionOrder(traces.get(l).get(m)))
== 0 && traces.get(i).get(j).equalsIgnoreCase(traces.get(l).get(m)) &&
!traces.get(i).get(k).equalsIgnoreCase(traces.get(l).get(m+1))) {
                                for(int n=m+2; n<traces.get(l).size(); n++){
                                    if(!checkSkip.contains(traces.get(l).subList(m, n + 1))
&& traces.get(i).get(k).equalsIgnoreCase(traces.get(l).get(n)) &&
activityFlag.get(getTransitionOrder(traces.get(l).get(n))) == 0) {
                                        checkSkip.add(traces.get(l).subList(m, n + 1));
                                        flagSkip = true;
                                        break;
                                    }
                                    else continue;
                                }
                            } else continue;

                            if(flagSkip){
                                SKIP.add(traces.get(i).subList(j, k + 1).toString() + " SKIP "
+ checkSkip.toString());
                                activityFlag.set(getTransitionOrder(traces.get(i).get(j)), 3);
                                flagSkip = false;
```

```
                    break;
                }
            }
        }
    }
}
```

Penemuan Control Flow Sequence

```
private void findSequence() {
    for (int i = 0; i < traces.size(); i++) {
        String x = "";
        boolean flagX = false;
        for (int j = 0 ; j < traces.get(i).size(); j++) {
            if(activityFlag.get(getTransitionOrder(traces.get(i).get(j))) == 0) {
                if(!flagX && j+1<traces.get(i).size() &&
activityFlag.get(getTransitionOrder(traces.get(i).get(j+1)))==0) {
                    x=x+traces.get(i).get(j);
                    activityFlag.set(getTransitionOrder(traces.get(i).get(j)), 2);
                    flagX = true;
                }
                else if(flagX) {
                    x=x+" seq "+traces.get(i).get(j);
                    activityFlag.set(getTransitionOrder(traces.get(i).get(j)), 2);
                    if(j+1<traces.get(i).size() &&
activityFlag.get(getTransitionOrder(traces.get(i).get(j+1))) > 0)
                        flagX = false;
                }
            }
            else{
                if(!x.equalsIgnoreCase("")){
                    SEQUENCE.add(x);
                    x="";
                }
                flagX = false;
                continue;
            }
        }
        if(!x.equalsIgnoreCase("")) {
            SEQUENCE.add(x);
        }
    }
}
```

Pembuatan LTL

```
private void ltl(String notation, int x, int y, int z, String xor, String or) {
```

```
        String transformed = new String();
        if (notation.equalsIgnoreCase("AND")) {
            for (int i = y + 1; i < z; i++) {
                if (i == y + 1) {
                    transformed = traces.get(x).get(i);
                } else {
                    transformed += "/\\" + traces.get(x).get(i);
                }
            }
            ltl.add(traces.get(x).get(y) + "-> <> (" + transformed + ")");
        } else if (notation.equalsIgnoreCase("XOR")) {
            transformed = xor;
            ltl.add(traces.get(x).get(y) + "-> _o (" + transformed + ")");
        } else if (notation.equalsIgnoreCase("OR")) {
            transformed = or;
            ltl.add(traces.get(x).get(y) + "-> <> (" + transformed + ")");
        }
    }
```

Pembuatan Model Process Tree

```
def buildtree(F):
    LTL = []
    for line in F:
        line_outspace = line.replace("/\\","^").replace("\\/","V").replace("\n","")
        LTL.append(line_outspace)

    nama_node = "node"
    node = Node("->")
    count = 1
    brack_front = 0
    now = node
    list_node = []

    #build next string
    for index_LTL in xrange(0, len(LTL)):
        #print "\nLTL No: " + str(index_LTL)

        componen_LTL = LTL[index_LTL].split(" ")
        for var in componen_LTL:
            if var != "(" and var != "_O" and var != "<>":
                start_index = componen_LTL.index(var)
                break

        for index_now in xrange(start_index, len(componen_LTL)):
            already_node = 0
            #if index_LTL == 0:
            #    print "var: " + str(componen_LTL[index_now])
```

```
        #    print "node_first: " + str(now.name)

        if componen_LTL[index_now] == "(":
            for node_search in list_node:
                if str("/")+str(componen_LTL[index_now+1]) in str(node_search):
                    already_node += 1
                    break

            if already_node == 0:
                if brack_front == 0:
                    child_node = str(nama_node)+str(count)
                    child_node = Node(str(nama_node) + str(count), parent=now)
                    now = child_node
                    count += 1
                else:
                    brack_front -= 1

        elif componen_LTL[index_now] == ")":
            if now.parent != None:
                now = now.parent

        elif componen_LTL[index_now] == "_O" or componen_LTL[index_now] == "<>":
            if now.name == "V":
                now.name = "loop"
            else:
                if now.name != "->":
                    now.name = componen_LTL[index_now]
                brack_front += 1

        elif componen_LTL[index_now] == "^" or componen_LTL[index_now] == "V" or componen_LTL[index_now] == "->":
            if now.name == "V" or now.name == "x":
                if componen_LTL[index_now] == "->" or componen_LTL[index_now] == "^":
                    for node_search in list_node:
                        if componen_LTL[index_now-1] in str(node_search):
                            temp_now = node_search
                            break

                    now = temp_now
                    now.name = "->"
                    child_node = str(nama_node)+str(count)
                    child_node = Node(componen_LTL[index_now-1], parent=now)
                    count += 1
                    for no in xrange(len(list_node)):
                        if str(temp_now) in str(list_node[no]):
```

```
                    del list_node[no]
                    list_node.append(child_node)
                    break

            elif now.name == "_O":
                #print "x"
                now.name = "x"

            elif now.name == "<>":
                print "^"
                now.name = "V"

            for node_search in list_node:
                if str("/")+str(componen_LTL[index_now+1]) in str(node_search)
and LTL[index_LTL].count(componen_LTL[index_now+1]) == 1 :
                    already_node += 1
                elif componen_LTL[index_now+1] == "(":
                    already_node += 1
                elif componen_LTL[index_now+1] == "_O" and
componen_LTL[index_now] == "V":
                    already_node += 1
                    break


            if already_node == 0:
                child_node = str(nama_node)+str(count)
                child_node = Node(str(nama_node) + str(count), parent=now)
                count += 1
                now = child_node

        else:
            for node_search in list_node:
                if componen_LTL[index_now] in str(node_search) and
LTL[index_LTL].count(componen_LTL[index_now]) == 1:
                    already_node += 1
                    now = node_search
                    break

            if already_node == 0:
                parent_node = now.parent
                if parent_node == None:
                    child_node = str(nama_node)+str(count)
                    child_node = Node(componen_LTL[index_now], parent=now)
                    count += 1
                    list_node.append(child_node)

                else:
```

```
                now.name = componen_LTL[index_now]
                list_node.append(now)
                now = parent_node

            else:
                now = now.parent

    return node
```

BIOGRAFI PENULIS

Afina Lina Nurlaili, lahir di Karanganyar, 13 Desember 1993. Penulis menempuh pendidikan dasar mulai kelas 1 sampai 6 di SDN Kleco II. Untuk pendidikan menengah, penulis tempuh di SMP Negeri 1 Surakarta dan selanjutnya di SMA Negeri 1 Surakarta. Penulis melanjutkan pendidikan sarjana di Jurusan Teknik Informatika, Institut Teknologi Sepuluh Nopember Surabaya. Penulis dalam menyelesaikan pendidikan S1 mengambil bidang minat Manajemen Informasi (*Information Management*) dan memiliki ketertarikan di bidang *Enterprise Resource Planning* khususnya modul Finance. Penulis dalam menyelesaikan pendidikan S2 mengambil bidang minat Manajemen Informasi (*Information Management*) dan memiliki ketertarikan di bidang *Process Mining*. Penulis dapat dihubungi melalui email: afina.lina12@gmail.com