TUGAS AKHIR – KI141502

OPTIMASI KINERJA PADA CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL

FITRIANING HARYADITA
NRP 5111 100 106

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

Dosen Pembimbing II
Adhatus Solichah Ahmadiyah, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI141502

PERFORMANCE OPTIMIZATION IN CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL

FITRIANING HARYADITA
NRP 5111 100 106

Supervisor I
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.

Supervisor II
Adhatus Solichah Ahmadiyah, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

OPTIMASI KINERJA PADA CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Manajemen Informasi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
FITRIANING HARYADITA
NRP. 5111 100 106

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Prof. Drs. Ec. Ir. RIYANARTO SARNO, M.Sc., Ph.D.
NIP: 19590803 198601 1 001 ..... (pembimbing 1)

ADHATUS SOLICHAH AHMADIYAH, S.Kom., M.Sc.
NIPH: 5100201405002 ..... (pembimbing 2)

SURABAYA
JUNI, 2015

LEMBAR PENGESAHAN

OPTIMASI KINERJA PADA CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Manajemen Informasi
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh
FITRIANING HARYADITA
NRP. 5111 100 106

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Prof. Drs. Ec. Ir. RIYANARTO SARNO, M.Sc., Ph.D.
NIP: 19590803 198601 1 001 ..... (pembimbing 1)

ADHATUS SOLICHAH AHMADIYAH, S.Kom., M.Sc.
NIPH: 5100201405002 ..... (pembimbing 2)

SURABAYA
JUNI, 2015

[Halaman ini sengaja dikosongkan]

OPTIMASI KINERJA PADA CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL INFORMATIKA INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA

Nama	**: Fitrianing Haryadita**
NRP	**: 5111100106**
Jurusan	**: Teknik Informatika – FTIf ITS**
Dosen Pembimbing I	**: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Dosen Pembimbing II	**: Adhatus Solichah Ahmadiyah, S.Kom., M.Sc.**

Abstrak

Optimasi cross-organizational business process adalah salah satu masalah yang harus dipecahkan. Untuk mengoptimasi kinerja dalam cross-organizational business process yang pertama dilakukan adalah melakukan process discovery terhadap model proses bisnis dari event log. Banyak algoritma process discovery yang telah diterapkan seperti Alpha, Alpha ++ dan Heuristic Miner, tetapi tidak dapat men-discover parallel OR. Oleh kerena itu Tugas Akhir ini memodifikasi Heuristic Miner dengan menggunakan interval threshold untuk men-discover parallel XOR, AND, dan OR. Interval threshold ditentukan berdasarkan rata-rata dari positive dependency measure dalam dependency matriks.

Setelah mendapatkan model dari cross-organizational business process event log, kemudian dilakukan optimasi kinerja dengan mendapatkan durasi minimum proses bisnis dan biaya tambahan yang minimum. CPM crashing project adalah salah astu metode yang digunakan untuk time–cost optimization. Tetapi CPM crashing project memerlukan beberapa data untuk melakukan percepatan durasi proses bisnis tetapi pada realitanya banyak data yang berbentuk single timestamp event log yang tidak memiliki

data yang dibutuhkan untuk CPM crashing project. Oleh karena itu dalam Tugas akhir ini menggunakan perhitungan rata-rata durasi eksekusi dan biaya setiap aktivitas dari setiap case untuk menentukan data optimasi yang digunakan untuk CPM crashing project. Kemudian linear programming digunakan untuk mendapatkan durasi minimum dan biaya tambahan minimum dari proses bisnis.

Hasil uji coba menunjukkan bahwa modified Heuristic Miner dapat discover OR split dan join, dan linear programming dengan data optimasi yang telah dihitung sebelumnya dapat melakukan optimasi terhadap cross-organizational business process dengan mendapatkan minimum durasi dan minimum biaya tambahan yang digunakan.

Kata kunci: Process Discovery, Discovery Parallel Activity OR, Modified Heuristic Miner, Time–cost Optimization, Single Timestamp Event Log, Cross-Organizaional Business Process Model, Linear Programming

PERFORMANCE OPTIMIZATION IN CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL INFORMATIKA INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA

Nama	**: Fitrianing Haryadita**
NRP	**: 5111100106**
Jurusan	**: Teknik Informatika – FTIf ITS**
Dosen Pembimbing I	**: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Dosen Pembimbing II	**: Adhatus Solichah Ahmadiyah, S.Kom., M.Sc.**

Abstract

In cross-organizational business process model performance optimization is one of the problem which should be solve. To optimize the cross-organizational business process model the first thing to do is discover the business process from event log. Many algorithms have been employed for process discovery, such as Alpha, Alpha ++ and Heuristic Miner but they cannot discover processes containing parallel OR. Therefor in this undergraduate thesis represents the modified Heuristic Miner which utilizes the threshold intervals to discover parallel XOR, AND, and OR. The threshold intervals are determined based on average positive dependency measure in dependency matrix.

After getting the model of cross-organizational business process event log then optimize the model to get the minimum duration of process and minimum additional cost which is needed. CPM crashing project is one of method to solve the time–cost optimization. But CPM crashing project need some data to speeding up the process business. But in reality there are a lot of data which is represent using single timestamp event log which does not provide data to do CPM crashing project. Therefor this undergraduate thesis represents a method to get data which is use to crashing project. The data is got from averaging time execution

of each activity in case in event log. Then to crashing the project this undergraduate thesis use linear programming to get minimum duration and minimum additional cost.

The results show that the modified Heuristic Miner can discover OR split and join, and using linear programming use the data which is calculate, this undergraduate thesis can optimize the performance of cross-organizational business process by getting minimum makspan and minimum additional cost.

Keywords: Process Discovery, Discovery Parallel Activity OR, Modified Heuristic Miner, Time–cost Optimization, Single Timestamp Event Log, Cross-Organizaional Business Process Model, Linear Programming

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Segala puji bagi Allah SWT, Tuhan semesta alam yang telah melimpahkan rahmat dan hidayah-Nya kepada penulis, sehingga tugas akhir berjudul "Optimasi Kinerja pada Cross-Organizational Business Process Model" ini dapat selesai sesuai dengan waktu yang telah ditentukan.

Pengerjaan tugas akhir ini menjadi sebuah sarana untuk penulis memperdalam ilmu yang telah didapatkan selama menempuh pendidikan di kampus perjuangan Institut Teknologi Sepuluh Nopember Surabaya, khususnya dalam disiplin ilmu Teknik Informatika. Terselesaikannya buku tugas akhir ini tidak terlepas dari bantuan dan dukungan semua pihak. Pada kesempatan kali ini penulis ingin mengucapkan terima kasih kepada:

1. Bapak, Ibu, kakak dan keluarga yang selalu memberikan dukungan penuh untuk menyelesaikan Tugas Akhir ini.
2. Bapak Riyanarto Sarno dan Ibu Adhatus Solichah selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberikan petunjuk selama proses pengerjaan Tugas Akhir ini.
3. Bapak, Ibu dosen Jurusan Teknik Informatika ITS yang telah banyak memberikan ilmu dan bimbingan yang tak ternilai harganya bagi penulis.
4. Seluruh staf dan karyawan FTIf ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
5. Segenap dosen rumpun mata kuliah Manajemen Informasi.
6. Rekan-rekan laboratorium Manajemen Informasi.
7. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak kekurangan. Dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2015

DAFTAR ISI

[Halaman ini segaja dikosongkan]

DAFTAR TABEL

[Halam ini sengaja dikosongkan]

NOMENKLATUR

$A =>_w B$	: nilai *dependency* dari aktivitas A ke B.
$\|A >_w B\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$\|B >_w A\|$	: frekuensi aktivitas B yang diikuti secara langsung oleh A.
$A =>_w A$	: nilai *dependency Length One Loop*
$\|A >_w A\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh A.
$max\{\|A >_w x\|\| x \in e\}$	: frekuensi aktivitas A yang diikuti oleh aktivitas x dimana aktivitas x merupakan bagian dari aktivitas yang ada dari event log.
$A =>_{2w} B$	: nilai *dependency Length Two Loop*
$\|A \gg_w B\|$	: frekuensi dari trace dalam bentuk ABA muncul dalam log
$\|B \gg_w A\|$	: frekuensi dari trace dalam bentuk BAB muncul dalam log
$A =>_w B \wedge C$	: *parallel measure* antara B dan C dimana *split* terjadi di A.
$\|B >_w C\|$	: frekuensi aktivitas B yang diikuti secara langsung oleh C.

$\|C >_w B\|$	: frekuensi aktivitas C yang diikuti secara langsung oleh B.
$\|A >_w B\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$\|A >_w C\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh C.
Z	: nilai fungsi tujuan.
C_i	: sumbangan per unit kegiatan, untuk masalah maksimisasi C_i menunjukkan keuntungan atau penerimaan per unit, sementara dalam kasus minimisasi menunjukkan biaya per unit.
X_i	: banyaknya kegiatan i, dimana i = 1, 2, 3, … w. berarti disini terdapat w variabel keputusan.
a_{ij}	: banyaknya sumberdaya i yang dikonsumsi sumberdaya j.
b_i	: jumlah sumber daya i (i = 1, 2, …, w)
g	: macam batasan sumber atau fasilitas yang tersedia.
h	: macam kegiatan yang menggunakan sumber atau fasilitas tersebut.
RBT	: *Relative-to-best threshold*
$Avg\ PDM$	: rata-rata positive dependency measure yang lebih dari 0 pada *matrix dependency*

DM	: *dependency measure*
$DM_{a=>b}$	: *dependency measure* antar aktivitas dalam dependency matriks.
$SD\ PDM$	: standar deviasi *positive dependency measure* pada *matrix dependency*
POT	: *Positive observations threshold*
$f_{a=>b}$	: frekuensi antar aktivitas pada matriks frekuensi.
DT	: *dependency threshold*
$A =>_w B \wedge C$	: *parallel measure* antara aktivitas pada percabangan dari A yaitu B dan C.
$\|B \gg>_w C\|$	: *undirect and direct followed frequency* aktivitas B dan C
$\|C \gg>_w B\|$	: *undirect and direct followed frequency* aktivitas C dan B
$\|A >_w B\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$\|A >_w C\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh C.
$\|B \gg> not_w C\|$	: frekuensi eksekusi aktivitas B yang tidak diikuti aktivitas C.
$\|C \gg> not_w B\|$	: frekuensi eksekusi aktivitas C yang tidak diikuti aktivitas B.
PM	: *Parallel Measure*
Limit PDM	: nilai minimum dari *positive dependency measure* pada *matrix dependency*

pt	: aktivitas pada *parallel split* dan *join*
n_s	*: noise*
m	: jumlah *case* dalam *event log*
j_t	: frekuensi *trace* dalam *event log*
t	*: trace* pada *event log*
t_{ns}	: bagian *trace* tang dipotong
m	: jumlah *case*
n	: jumlah *complete trace* dari model
X_i	: *time slope* yang terjadi untuk penyelesaian aktivit ke – i dimana i ∈ {Aktivitas} (variable keputusan)
T_{ni}	: durasi normal aktivitas ke – i dimana i ∈ {Aktivitas}
T_{ci}	: *crash* duration aktivitas i dimana i ∈ {Aktivitas}
S_i	: *Cost slope* aktivitas ke – i dimana i ∈ {Aktivitas}
C_{ni}	: *cost* normal aktivitas ke – i dimana i ∈ {Aktivitas}
C_{ci}	: *cost crash* aktivitas ke – i dimana i ∈ {Aktivitas}
ed	: durasi eksekusi
$et_{activity_{output}}$	: waktu eksekusi *output* aktivitas
$et_{activity}$	: waktu eksekusi aktivitas
ed_k	: waktu eksekusi aktivitas pada *case* ke-k
$cost_k$	: biaya aktivitas pada *case* ke-k
h	: banyaknya aktivitas dieksekusi dalam *event log*

	(banyaknya *case* yang mengandung aktivitas)
j	: banyaknya aktivitas pada proses bisnis
j	: banyaknya aktivitas pada proses bisnis
Y_i	: variabel *start time* aktivitas ke i
Y_{pi}	: variabel *start time predecessor* aktivitas ke i
X_{pi}	: variabel *time slope predecessor* aktivitas ke i
K	: *normal time predecessor* aktivitas ke i
D	: maksimum durasi pada *critical path* pada proses bisnis
Y'_{Finish}	: durasi proses bisnis paling minimum setelah dimampatkan
Y_{Finish}	: durasi hasil akhir durasi *crashing* proses bisnis
P	: *place* pada *Petri Net*
T	: transisi (aktivitas) pada *Petri Net*
M	: *message* pada *Petri Net*

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1. Latar Belakang

Cross-organizational business process merupakan suatu proses bisnis yang melibatkan lebih dari satu organisasi untuk menjalankannya. Dalam menjalankan c*ross-organizational business process*, mengoptimalkan proses bisnis penting untuk mempercepat durasi yang dibutuhkan tetapi hal ini mengakibatkan penambahan biaya pengeluaran yang dibutuhkan. Sehingga diperlukan upaya untuk meminimumkan durasi dari proses bisnis tapi dengan menggunakan biaya tambahan yang minimum. Dalam mengoptimasi diperlukan untuk mengetahui model dari proses bisnis untuk proses bisnis dari perusahaan berkembang suatu proses yaitu *process mining. Process mining* memiliki salah satu tujuannya yaitu membentuk model dari suatu proses bisnis. *Process mining* sendiri memiliki tiga proses utama yaitu *process discovery, conformance checking,* dan *enchanment*.

Process discovery sendiri merupakan salah satu dari rangkaian *process mining* dimana tujuan dari proses ini adalah untuk membentuk model dengan upaya menggali informasi dari data yang tercatat dalam suatu *event log*. Terdapat banyak algoritma yang digunakan dalam *process discovery*. Alpha ++ merupakan salah satu dari algoritma yang digunakan dalm process discovery. Algoritma ini menggunakan prinsip pemodelan *Petri net*, dengan pemodelan ini algoritma ini dapat memodelkan beberapa jenis proses yang ada pada *event log* seperti *short loop, non free choice*, dan proses paralel (Wen, Aalst, Jianmin, & Sun, 2012). Namun algoritma ini tidak dapat men-*discover event log* yang mengandung *noise* (Weber, 2013) dan karena menggunakan prinsip *Petri net* maka hanya dapat men-*discover* proses paralel

dengan menggunakan percabangan AND dan XOR. Oleh karena itu munculah algoritma baru yaitu *heuristic miner*.

Heuristic miner merpakan jenis pemodelan dengan menggunakan prinsip *Causal-net* (C-net) dalam modelnya. *Heuristic miner* merupakan salah satu algoritma yang dapat menangani event log yang mengandung *noise* (Weber, 2013). *Noise* dalam *event log* ditangani dengan penggunakan *threshold* yang ditentukan untuk memodelkan proses bisnis. Namun dengan cara ini masih terdapat kekurangan yaitu model dapat menjadi *overfitting* dan *underfitting* ketika salah dalam menentukan *threshold*-nya, dan juga pada algoritma ini belum dapat menentukan percabangan yang menggunakan OR, padahal banyak model yang menggunakan jenis percabangan ini. Selain tidak dapat men-*discover* percabangan OR *heuristic miner* juga termasuk algoritma yang tidak mudah untuk digunakan kerena penggunaan *threshold* yang harus ditentukan secara manual oleh pengguna. Sehingga hanya pengguna yang ahli dalam *process discovery* yang dapat menggunakannya. *Discover* percabangan OR dan penentuan *threshold* yang digunakan pada *heuristic miner* merupakan permasalahan pertama yang dipecahkan.

Selain pemodelan proses bisnis dari *event log,* terdapat permasalahan yang kedua yaitu dalam menentukan optimasi kinerja dalam proses bisnis, yang dimaksud kinerja disisni adalah terjadi percepatan waktu durasi dari proses bisnis. Karena hal tersebut dibutuhkan suatu mekanisme untuk optimasi biaya tambahan yang dibutuhkan untuk mencapai percepatan maksimal dari sebuah proses bisnis, dengan kata lain mencari biaya tambahan yang paling minimum untuk kemungkinan percepatan yang paling maksimal dalam istilahnya hal ini disebut dengan *Time-Cost Optimization.*

Berdasarkan permasalahan pertama dan kedua di atas Tugas Akhir ini akan memberikan solusi tentang dua cara yaitu memodifikasi dalam algoritma *heuristic miner* sehingga dapat menentukan *threshold* yang tepat dan dapat memodelkan percabangan OR dalam model yang di-*discover* dan menentukan

threshold yang digunakan secra otomatis sehinggal model dapat di-*discovery* oleh orang awam sehingga ter-*discover* model yang ideal dan untuk optimasi karena *event log* hanya memiliki *single timestamp* maka akan dilakukan perhitungan untuk durasi dan biaya yang diperlukan untuk optimasi dan menggunakan pembentukan model matematika *linear programming* akan dicari durasi terpendek dari proses bisnis dengan penambahan biaya yang paling minimum.

Event log yang diolah dalam tugas akhir ini adalah *event log* dari kerjasama antar departemen produksi dari PT. Toray Industries Indonesia untuk proses produksi benang, dan juga *event log* dari kerjasama antara PT. Toray Industries Indonesia dengan *supplier* dan Bea dan Cukai dalam melakukan *purchase order* bahan baku dari pembuatan benang.

1.2. Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana memodifikasi *heuristic miner* agar dapat memodelkan proses bisnis dengan *threshold* otomatis?
2. Bagaimana memodifikasi *heuristic miner* agar dapat memodelkan paralel OR?
3. Bagaimana menghitung data durasi dan biaya dari *single timestamp event log*?
4. Bagaimana cara mendapatkan optimasi durasi terendah dengan minimum biaya tambahan?

1.3. Batasan Permasalahan

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Bahasa pemrograman yang digunakan adalah C#.
2. Aplikasi yang dikembangkan aplikasi *dekstop*.
3. Data masukan berupa *event log* dalam bentuk Excel.
4. Data keluaran proses *discovery* dalam bentuk *graph*.

5. Data keluaran proses *discovery* hanya dapat disimpan dalam bentuk tabel Excel.
6. Data keluaran data optimasi dalam bentuk tabel dan dapat disimpan dalam Excel.
7. Data keluaran optimasi berupa biaya tambahan paling optimal, durasi proyek paling minimal, dan *crashed duration* untuk tiap aktivitas.

1.4. Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Menentukan *threshold heuristic miner* sehingga dapat membentuk model proses bisnis yang ideal tanpa harus memasukkan secara manual.
2. Memodifikasi *heuristic miner* sehingga dapat men-*discover* bentuk percabangan OR.
3. Melakukan perhitungan untuk data optimasi (durasi normal, durasi *crash,* biaya normal, biaya *crash*) yang diperlukan ketika melakukan percepatan dengan metode CPM *project crashing.*
4. Mendapatkan nilai durasi dan tambahan biaya yang paling optimal, yaitu durasi yang minimum dengan biaya yang minimum.

1.5. Manfaat

Manfaat Keilmuan

Manfaat yang dihasilkan dari pengerjaan Tugas Akhir ini adalah terbentuknya suatu metode yang dapat men-*discover* model yang mengandung paralel OR dan terbebas dari *noise,* selain mendapatkan model yang mengandung OR dalam Tugas Akhir ini juga didapatkan suatu upaya optimasi yang dilakukan pada *cross-organizational business process*. Dengan mendapatkan model yang mengandung OR maka model dapat mengembalikan *event log* sesuai dengan *event log* yang digali. Pemodelan OR dilakukan dengan memodifikasi algoritma yang sudah ada yaitu *heuristic miner,* hal ini dilakukan karena selain untuk mendapatkan model

paralel OR diharapkan model hasil *discovery* juga terbebas dari *noise* yang terdapat pada *event log*. Pemodelan paralel OR merupakan salah satu kontribusi utama yang ada dalam tugas akhir ini karena yang penulis ketahui pemodelan OR dengan memodifikasi *heuristic miner* belum pernah dilakukan. Selain pemodelan OR terdapat manfaat lainnya yaitu optimasi dari *single timestamp event log* dan hubungan optimasi dengan pola *cross-organizational business process*. Untuk optimasi dari *single timestamp event log* diperlukan perhitungan data durasi. Dalam Tugas Akhir ini data durasi dihitung dengan memodifikasi metode *time prediction* dari Van Der Aalst karena durasi yang dihitung dari *time prediction* merupakan durasi per *trace* sedangkan yang diperlukan untuk optimasi adalah durasi per aktivitas. Hubungan optimasi dengan pola *cross-organizational business process* menghubungkan bentuk dari pola dengan keterkaitan antar organisasi jika terdapat salah satu organisasi yang tidak melakukan optimasi.

Manfaat Praktis

Manfaat dari proses *discovery* sendiri adalah agar dapat menemukan proses yang sebenarnya terjadi, untuk penemuan paralel OR dimaksudkan utntuk mendapatkan model yang benar-benar dapat mengembalikan data yang ada dalam *event log*. Dari model proses yang didapatkan dapat dilihat bagaimana model proses bisnis yang sesuai untuk SOP ke depannya, hal ini dapat dilihat dengan medapatkan model secara periodik dari *event log*. Sedangkan untuk optimasinya sendiri lebih membantu memprediksi *crash time* dan biaya yang dibutuhkan walaupun data berupa *single tumestamp event log* yang tidak memiliki data durasi maupun *crash time*. Pola hubungan optimasi dengan *cross-organizational business process* dapat dimanfaatkan untuk menentukan organisasi mana saja yang akan diptimasi jika terdapat keterbatasan optimasi dari organisasi yang berhubungan dengan proses bisnis tersebut.

1.6. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas

Akhir ini yaitu:

1. **Studi literatur**

   Dalam pembuatan Tugas Akhir ini telah dipelajari tentang hal-hal yang dibutuhkan sebagai ilmu penunjang dalam penyelesaiannya. Pertama adalah tentang struktur utama dari *Heuristic Miner*, CPM *Crashing Project*, dan Optimasi dengan *Linear Programming*. Kemudian adalah menentukan batasan-batasan pemetaan. Selain itu, juga dibantu beberapa literatur lain yang dapat menunjang proses penyelesaian Tugas Akhir ini.

2. **Pemodifikasian Metode**

   Pada tahap ini penulis menjabarkan cara pemecahan masalah yang terdapat dalam rumusan masalah. Selain itu penulis juga menjabarkan tentang modifikasi yang telah dilakuakan pada algoritma sebelumnya sebagai salah satu kontribusi dalam Tugas Akhir ini.

3. **Analisis dan Perancangan Sistem**

   Aktor yang menjadi pelaku adalah pengguna perangkat lunak yang dibangun oleh penulis. Kemudian beberapa kebutuhan fungsional dari sistem ini adalah sebagai berikut :

   a. Melakukan proses *discovery* dari *file event log* menjadi model proses bisnis;
   b. Melakukan perhitungan untuk penentuan data optimasi berupa durasi dan biaya.
   c. Melakukan optimasi minimum durasi proses bisnis dan minimum biaya tambahan yang dibutuhkan.

4. **Implementasi**

   Pada tahap ini dilakukan pembuatan elemen perangkat lunak yang merupakan implementasi dari rancangan yang telah dibuat sebelumnya.

5. Pengujian dan evaluasi

Pada tahap ini dilakukan pengujian terhadap elemen perangkat lunak dengan menggunakan data uji yang telah dipersiapkan pada jurnal. Pengujian dan evaluasi perangkat dilakukan untuk mengevaluasi jalannya perangkat lunak, mengevaluasi fitur utama, mengevaluasi fitur-fitur tambahan, mencari kesalahan yang timbul pada saat perangkat lunak aktif, mengadakan perbaikan jika ada kekurangan. Tahapan-tahapan dari pengujian adalah sebagai berikut:

a. pencocokan hasil *discovery* perangkat lunak dengan model *cross-organizational* dan *single-organizational* yang sudah ada;
b. pengujian ketahanan perangkat lunak dalam menangani *noise* pada proses *discovery*; dan
c. pencocokan hasil uji sistem dengan hasil uji pada eksperimen yang telah dilakukan.

6. Penyusunan buku Tugas Akhir

Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan Tugas Akhir.

1.7. Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan Tugas Akhir ini.

Bab III Metode Pemecahan Masalah

Bab ini membahas cara penulis memecahkan masalah yang ada. Penjelasan tentang algoritma yang dikembangkan penulis dan langkah-langkahnya sehingga dapat memecahkan masalah yang ada.

Bab IV Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan alur, proses dan perancangan antarmuka pada perangkat lunak.

Bab V Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak perangkat lunak dan implementasi fitur-fitur penunjang perangkat lunak.

Bab VI Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan serta evaluasi terhadap fitur-fitur perangkat lunak.

Bab VII Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada aplikasi ini.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir. Teori-teori tersebut meliputi *Process Mining, Process Discovery,* model proses bisnis, *Heuristic Miner Algorithm*, *Causal Net* (C-net)*, Noise*, proses paralel, *Critical Path Method* (CPM), *Linear Programming*.

2.1 Model Proses Bisnis

Proses bisnis merupakan sekumpulan aktivitas yang dibuat untuk menghasilkan keluaran spesifik dengan tujuan tertentu (Wang, et al., 2010). Dari model tersebut juga dapat diketahui informasi dimana dan kapan suatu aktivitas dilakukan, kondisi awal sebelum aktivitas dilakukan, kondisi akhir setelah aktivitas dilakukan, serta masukan dan keluaran yang jelas. Adapun ciri-ciri dari proses bisnis itu sendiri adalah sebagai berikut :

1. Mempunyai tujuan tertentu
2. Mempunyai masukan yang spesifik.
3. Mempunyai keluaran yang spesifik.
4. Memanfaatkan *resource*.
5. Memiliki aktivitas yang dapat dieksekusi dengan urutan tertentu.
6. Dapat melibatkan lebih dari satu organisasi.

Model proses bisnis merupakan representasi dari proses bisnis. Sehingga sebuah model proses bisnis harus secara jelas mendefinisikan setiap ciri-ciri yang harus dimiliki oleh suatu proses bisnis. *Unified Modeling Language* (UML) merupakan salah satu representasi dasar dari proses bisnis. Saat ini representasi dari model proses bisnis itu sendiri sudah banyak berkembang dan banyak jenisnya. Mulai dari *Causal Net,* UML, *Business* BPEL, *Business Process Model and Notation* (BPMN), EPC, PNML, dan masih banyak lagi. Tetapi, masing-masing jenis tersebut juga memiliki kegunaan dan fungsi sendiri-sendiri.

2.2 *Event Log*

Dalam *process mining* untuk menganalisis suatu *business process* digunakan *event log* dari *business process* tersebut sebagai acuannya. *Event log* didefinisikan sebagai suatu set proses eksekusi yang mengambil dari data aktivitas proses bisnis yang dilakukan dalam konteks tertentu (Wen, Aalst, Jianmin, & Sun, 2012). Atau dengan kata lain merupakan catatan dari eksekusi aktivitas dalam suatu proses bisnis, catatan eksekusi ini dapat menyimpan data berupa waktu dilaksanakannya suatu aktivitas, *resource* yang melaksanakan aktivitas, dan lain-lain sesuai dengan kebutuhan dari perusahaan yang menghidupkan *event log*-nya. Dalam *event log* dapat terdiri dari berbagai macam *case, trace*, dan *activity* (van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, 2011).

➨ *Case* dan *Trace*

Case merupakan suatu kasus tertentu yang ada pada *event log*. Kasus tertentu tersebut dapat berupa suatu kasus dalam memproduksi suatu barang tertentu, karena *event log* dapat terdiri dari catatan dari proses eksekusi pembuatan banyak barang atau proses eksekusi dari banyak kasus proses. Sedangkan *trace* merupakan alur dari aktivitas yang dijalankan dalam suatu proses.

Sebagai gambarannya misal dalam suatu *event log*:

$$L = [\langle a, c, d\rangle^{45}, \langle b, c, d\rangle^{42}, \langle a, c, e\rangle^{38}, \langle b, c, e\rangle^{22}]$$

Dalam *event log* tersebut:

1. Terdapat 4 *trace* yaitu (a,c,d), (b,c,d), (a,c,d), (b,c,e)
2. Terdapat 147 *case* karena (a,c,d) dilakukan sebanyak 45 kali, (b,c,d) sebanyak 42 kali, (a,c,e) sebanyak 38 kali, dan (b,c,e) sebanyak 22 kali.

- Activity

Merupakan bagian dari case yang merupakan sub proses dalam pembuatan suatu barang atau dalam suatu proses tertentu.

Misal dalam event log

$$L = [\langle a, c, d\rangle^{45}, \langle b, c, d\rangle^{42}, \langle a, c, e\rangle^{38}, \langle b, c, e\rangle^{22}]$$

Memiliki lima aktivitas yaitu {a, b, c, d, e}.

2.3 Noise

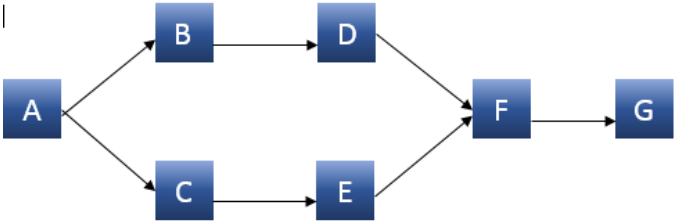
Noise dedifinisikan sebagai '*outlier'* atau event yang jarang terjadi atau exeptional event. Maka diasumsikan bahwa model yang di-*mining* tidak harus mengandung *noise* yang menyebabkan model berantakan. *Noise* sendiri dapat berupa *traceted trace* yang memiliki frekuensi yang jarang terjadi dalam *event log* (Weber, 2013; van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, 2011).

Menurut cara pembentukannya terdapat tiga jenis *noise* yaitu (Loreto, 2012).

- Menghapus *head* dari *trace* aktivitas

Pada cara pembuatan *noise* ini dilakukan dengan menghapus *head* dari *trace* yang ada pada *event log*. Aktivitas yang dihapus dapat berupa aktivitas pertama atau serangkaian aktivitas awal dari *trace* yang ada pada *event log*.

Contohnya adalah sebagai berikut:



Gambar 2.1 Proses Bisnis

Gambar 2.1 akan menghasikan complete trace event log sebagai berikut:

Tabel 2.1 *Event Log* Tanpa *Noise* Gambar 2.1

No. Trace	Trace Utuh
1	ABDCEFG
2	ABCEDFG
3	ABCDEFG
4	ACEBDFG
5	ACBEDFG
6	ACBDEFG

Pada *event log* pada Tabel 2.1 terdapat penghapusan pada aktivitas pertama pada *trace* 1 dan terdapat penghapusan serangkaian aktivitas awal pada *trace* 4, sehingga *trace* 1 dan 4 menjadi noise dengan jenis pembentukan dengan penghapusan *head* dari *trace. Event log* yang mengandung *noise* adalah sebagai berikut:

Tabel 2.2 *Event Log* dengan *Noise* Hasil Perpotongan *Head*

No. Trace	Trace
1	BDCEFG
2	ABCEDFG
3	ABCDEFG
4	EBDFG
5	ACBEDFG
6	ACBDEFG

➨ Menghapus tail dari trace aktivitas

Pada cara pembuatan *noise* ini dilakukan dengan menghapus *head* dari *trace* yang ada pada *event log*. Aktivitas yang dihapus dapat berupa aktivitas pertama atau serangkaian aktivitas awal dari *trace* yang ada pada *event log.*

Contoh yang digunakan adalah Gambar 2.1 Proses Bisnis dengan hasil *event log* pada Tabel 2.1.

Pada *event log* pada Tabel 2.1 terdapat penghapusan pada aktivitas terakhir pada *trace* 1 dan terdapat penghapusan serangkaian aktivitas akhir pada *trace* 4, sehingga *trace* 1 dan 4 menjadi noise dengan jenis pembentukan dengan penghapusan *tail* dari *trace. Event log* yang mengandung *noise* adalah sebagai berikut:

Tabel 2.3 *Event Log* dengan *Noise* Hasil Perpotongan *Tail*

No. Trace	Trace
1	ABDCEF
2	ABCEDFG
3	ABCDEFG
4	ACEBD
5	ACBEDFG
6	ACBDEFG

➨ Menghapus bagian acak dari tubuh trace aktivitas

Pada cara pembuatan *noise* ini dilakukan dengan menghapus *body* dari *trace* yang ada pada *event log*. Aktivitas yang dihapus dapat berupa seluruh *body* dari *trace* atau salah satu aktivitas *body* dari *trace* yang ada pada *event log.*

Cara pembentukan *noise* ini akan menghasilkan *noise* yang melanggar aturan hubungan antar aktivitas yang ada pada model misal sebenarnya antar aktivitas tidak terhubung, tiba-tiba terdapat hubungan antar aktivitas.

Contoh yang digunakan adalah Gambar 2.1 Proses Bisnis dengan hasil *event log* pada Tabel 2.1.

Pada *event log* pada Tabel 2.1 terdapat penghapusan seluruh *body* pada *trace* 1 dan terdapat penghapusan satu aktivitas *body* pada *trace* 4, sehingga *trace* 1 dan 4 menjadi *noise* dengan jenis pembentukan dengan penghapusan *body* dari *trace. Event log* yang mengandung *noise* adalah sebagai berikut:

Tabel 2.4 *Event Log* dengan *Noise* Hasil Perpotongan *Body*

No. Trace	Trace
1	AG
2	ABCEDFG

No. Trace	Trace
3	ABCDEFG
4	ACEBDG
5	ACBEDFG
6	ACBDEFG

Banyak algoritma yang tidak dapat menangangani masalah *noise* seperti *alpha*, *alpha plus* maupun *alpha plus plus* tidak dapat mengananinya (Wen, Aalst, Jianmin, & Sun, 2012). *Heuristic miner* merupakan algoritma yang dapat mengani *noise* dengan menggunakan *thershold* yang telah ditentukan (Weber, 2013).

Dalam Tugas Akhir ini akan dicari *limitation* dari *heuristic miner* dalam menangani *noise* yang ada pada *event log.*

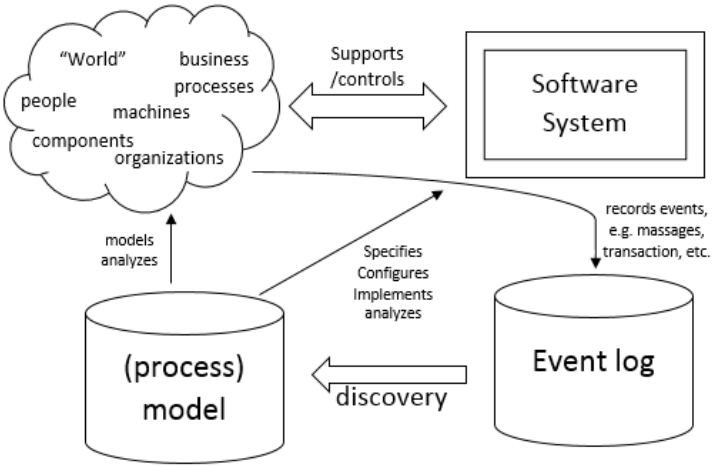
2.4 *Process Mining*

Process Mining merupakan suatu teknologi yang relatif masih baru dalam kaitannya dengan BPM – *Business Process Management* (van der Aalst, Process Mining: Beyond Business Intelligence, 2013). BPM sendiri bertujuan untuk mendapatkan model dengan cara mengamati perilaku proses bisnis di suatu organisasi. Pada *process mining*, pengamatan dilakukan terhadap proses bisnis yang telah tercatat dalam suatu *event log*. Dengan cara ini diharapkan akan ditemukan struktur proses baru yang sebelumnya tidak disadari sedang terjadi. Berdasarkan siklus yang konsisten serta frekuensi aliran informasi yang terjadi maka dapat diketahui apakah selama ini proses bisnis yang diterapkan oleh sistem informasi telah sesuai dengan pedoman yang dimiliki oleh organisasi ataukah sebaliknya. Berbagai manfaat bisa didapat dengan adanya *Process Mining*, seperti untuk mengetahui bagaimanakah proses yang sebenarnya terjadi. Mengetahui apakah proses yang berjalan sudah sesuai dengan model yang dirancang sebelumnya. Mengetahui di tahapan manakah terjadi perlambatan proses. Selain itu yang cukup menarik bahwasannya *Process Mining* juga mampu melakukan prediksi atas jumlah keterlambatan yang mungkin timbul serta membuat rancangan model seperti apa yang lebih tepat guna menyelesaikan permasalahan. *Event log* sebagai sumber data dari teknik *Process Mining* dirasa tepat karena

umumnya *log* sebuah sistem informasi berisi data dari berbagai kasus yang dieksekusi organisasi. Data yang dicatat umumnya berupa waktu mulai dan selesainya pekerjaan di suatu bagian, siapa saja pelakunya, dan lain sebagainya (Wicaksono, Atastina, & Kurniati, 2014).

2.5 *Process Discovery*

Process discovery merupakan salah satu proses yang paling menantang dari rangkaian *process mining*. Tujuan dari proses ini adalah untuk membentuk model dengan cara menggali informasi dari data yang tercatat dalam suatu *event log* (Goedertier, De Weerdt, Martens, Vanthienen, & Baesens, 2011). Dalam struktur proses bisnis, model dapat dianggap sebagai *graph* untuk mengandung satu set *node* dihubungkan dengan *edge* (Sarno, Ginardi, Pamungkas, & Sunaryono, 2013).



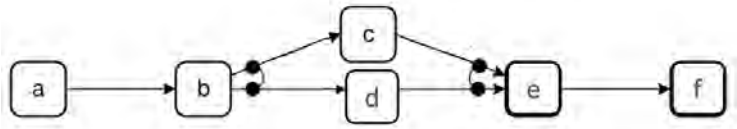
Gambar 2.2 Skema *Process Discovery*

Adapun algoritma yang digunakan dalam *Process Discovery* adalah algoritma *alpha miner, alpha plus miner, alpha plus plus miner, heuristics miner* (van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, 2011). Tiap algoritma memiliki pendekatan yang berbeda-beda dalam menganalisis proses yang terjadi. Dalam Tugas Akhir ini

akan mengembangkan algoritma *heuristics miner* untuk melakukan *process discovery* karena algoritma ini merupakan algoritma yang dapat menangani adanya *noise* dalam *event log*.

2.6 *Causal Net* (C-Net)

Causal Net (C-Net) adalah suatu *graph* yang menggambarkan suatu proses bisnis, dimana *nodes* dari *graph* menggambarkan aktivitas yang terjadi pada proses bisnis dan *arcs* atau *edges* pada *gaph* menggambarkan hubungan antar aktivitas yang ada pada proses bisnis. Dalam C-Net setiap aktivitas memiliki serangkaian *input bindings* dan *output bindings* (van der Aalst, Adriansyah, & van Dongen, Causal Nets: A Modeling Language Tailored Towards Process Discovery, 2011). Salah satu contoh dari causal net adalah sebagai berikut:



Gambar 2.3 Contoh *Causal Net*

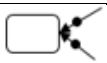
Pada contoh C-Net pada Gambar 2.3 aktivitas a memiliki *input bindings* kosong sehingga a merupakan *start activity* dan a memiliki *output bindings:* {b}, ini berarti setelah aktivitas a diikuti oleh aktivitas b. Aktivitas b memiliki *input bindings:*{a} dan *output bindings:* {c,d}, karena *output bindings* yang dimiliki b lebih dari satu maka menggunakan *split model* yang dimiliki C-Net, *input bindings* dan *output bindings* untuk C-Net begitu seterusnya sehingga akan menghasilkan *Causal Matrix*. Untuk Gambar 2.3 *Causal Matrix*-nya adalah sebagai berikut:

Tabel 2.5 *Causal Matrix* Gambar 2.3

Input	Aktivitas	Output
{}	a	b
a	b	c,d
b	c	e
b	d	e
c, d	e	f
e	f	{}

Bentuk model *split* dan *join* yang dimiliki C-Net adalah sebagai berikut (van der Aalst, Adriansyah, & van Dongen, Causal Nets: A Modeling Language Tailored Towards Process Discovery, 2011):

Tabel 2.6 C-Net *Split* dan *Join*

C-Net Split Model	C-Net Join Model
 XOR-Split	 XOR-Join
 AND-Split	 AND-Join
 OR-Split	 OR-Join

2.7 Jenis *Split* dan *Join*

Dalam proses bisnis dalam alurnya dimungkinkan untuk adanya percabangan sehingga membentuk suatu pilihan ataupun suatu proses paralel. Proses paralel sendiri terdapat dua jenis *split join* AND dan *split join* OR, sedangkan untuk pilihan menggunakan *split* dan *join* XOR (Weske, 2011). Sifat-sifat untuk masing-masing *split* dan *join* adalah sebagai berikut:

➨ *Single Choice* XOR

Single Choice XOR terjadi jika titik dalam proses alur kerja di mana satu cabang dibagi menjadi dua atau lebih tetapi trace hanya dapat memilih salah satu cabang saja. Semua jenis pemodelan dapat memodelkan XOR oleh karena itu hampir semua algoritma dapat memodelkan XOR (Weske, 2011).

➨ *Parallel* AND

Parallel AND terjadi jika *parallel split pattern* muncul. *Parallel split pattern* didefinisikan sebagai mekanisme yang memungkinkan dua kegiatan yang berbeda dilakukan secara

bersamaan. Sifat dasar dari pola ini sendiri adaah semua aktivitas yang ada di percabangan harus dijalankan, baik itu dijalankan secara bersamaan atau secara bergantian (Weske, 2011).

- *Conditional* OR

Conditional OR digunakan ketika *multiple choice pattern* muncul. *Multiple choice pattern* pemilihan satu atau lebih aktivitas dalam percabangan untuk dijalankan. Dalam *multiple choice pattern* satu aktivitas dapat dijalankan sendiri tanpa harus menjalankan aktivitas lain yang ada dipercabangan, atau juga dapat menjalankan beberapa aktivitas baik secara bersamaan maupun tidak (Weske, 2011).

Dalam *process mining* banyak algoritma *process discovery* yang tidak dpat mngidentifikasi atau memodelkan secara benar suatu *event log* yang mengandung *multiple choice pattern* (Sarno, Indita, Ginadi, Sunaryono, & Mukhlash, 2013). Hal ini dikarenakan kebanyakan algoritma terutama algoritma yang menggunakan pemodelan dengan *Petri Net* tidak memiliki aturan dalam memodelkan OR, hal ini dikarenakan bentuk pemodelan dari *Petri Net* sendiri memamng tidak dapat memodelkan bentuk *split* dan *join* OR (Weske, 2011).

2.8 *Heuristic Miner Algorithm*

Heuristic miner meupakan salah satu algoritma yang digunakan untuk melakukan *process discovery* dalam *process mining*. *Heuristic miner* menggunakan model C-Net dalam merepresentasikan model proses bisnis yang dihasilkan. Algoritma ini menggunakan frekuensi dari suatu hubungan aktivitas dan urutannya untuk membangun sebuah model proses bisnis. Ide dasar dari algoritma ini adalah suatu urutan atau hubungan dari aktivitas yang jarang terjadi dalam *event log* tidak harus dimasukkan dalam model. Penggunaan frekuensi dalam pembentukan model membuat algoritma ini lebih akurat dibandingkan dengan algoritma yang lainnya selain itu juga membuat algoritma ini dapat menangani *noise,* walaupun belum ada algoritma yang data sepenuhnya menangani masalah *noise* (van der Aalst, Process Mining -

Discovery, Conformance and Enhancement of Business Processes, 2011).

Dalam *heuristic miner* tedapat tiga langkah utama dalam pembentukan model dari proses bisnis (Weijters, van der Aalst, & de Medeiros, 2006), yaitu:

➨ *Mining Dependency Graph*

Langkah awal dari *heutistic miner* adalah membuat suatu *dependency graph* yang merupakan hasil dari perhitungan dari frekuensi-frekuensi dari hungungan antar aktivitas yang ada. Pertama yang dilakukan adalah menghitung frekuensi dari hubungan masing masing aktivitas yang terhubung. Kemudian dari frekuensi tersebut akan menentukan hubungan ketergantungan dari masing-masing aktivitas atau dengan kata lain dapat dikatakan aktivitas yang dapat saling dihubungkan.

Perhitungan dari *dependency measure* yang digunakan untuk menentukan hubungan antar aktivitas dalam model dapat dilihat pada Persamaan 2.1.

$$A =>_w B = \left(\frac{|A>_w B| - |B>_w A|}{|A>_w B| + |B>_w A| + 1}\right) \quad (2.1)$$

Keterangan:

$A =>_w B$: nilai *dependency* dari aktivitas A ke B.

$|A >_w B|$: frekuensi aktivitas A yang diikuti secara langsung oleh B.

$|B >_w A|$: frekuensi aktivitas B yang diikuti secara langsung oleh A.

Hasil dari Persamaan 2.1 menghasilkan suatu matriks yaitu matriks *dependency measure* yang akan digunakan untuk membentuk *dependency graph*. A. Weijters dan Sofie Cnude (Weijters, van der Aalst, & de Medeiros, 2006; Cnude, 2014) menjelaskan bahwa dalam pemilihan *dependency measure* untuk pembentukan *dependency graph, heuristic miner* menggunakan beberapa jenis *threshold* yaitu:

a. *Relative-to-best threshold* (RBT)

   Threshold ini menunjukkan bahwa suatu *edge* atau hubungan akan digunakakan (yaitu untuk memasukkan

edge/hubungan antar aktivitas ke dalam *control-flow network*) jika perbedaan antara nilai *dependency measure* yang dihitung untuk *edge* dan nilai terbesar dari *dependency measure* yang ada pada matriks *dependency graph* lebih rendah dari nilai parameter ini.

b. *Positive observations threshold* (POT)
   Threshold ini mengontrol jumlah minimum berapa kali aktivitas memiliki ketergantungan hubungan dengan aktivitas lain kegiatan, dua: relasi dianggap ketika banyak frekuensi hubungan ini berada di atas nilai atau sama dengan parameter ini.

c. *Dependency threshold* (DT)
   Threshold ini berguna untuk mengabaikan semua hubungan yang nilai *dependency measure* berada dibawah nilai parameter. Dengan kata lain nilai *dependency measure* yang digunakan harus lebih besar atau sama dengan nilai *dependency threshold.*

Salah satu kekurangan dari *heuristic miner* adalah *threshold* yang tidak ditentukan sehingga pengguna harus menentukannya sendiri. Dengan kata lain pengguna harus mengerti betul dengan cara kerja algoritma ini supaya dapat menentukan *threshold* yang dapat mengahsilkan proses bisnis yang ideal. Hal ini mengakibatkan pengguna awam kesulitan dalam menggunakan algoritma ini, oleh karena itu dalam Tugas Akhir ini akan memodifikasi algorima ini sehingga pengguna awam juga dapat menggunakan algoritma ini.

➨ *Mining Short Loop (Length One Loop dan Length Two Loop)*

Dalam proses, dimungkinkan untuk menjalankan kegiatan yang sama beberapa kali. Jika ini terjadi, ini biasanya mengacu pada *Loop* yang terjadi pada model. Untuk *Long Loop* sudah dapat diatasi dengan menggunakan *dependency measure* tetapi untuk *Short Loop* diperlukan pengecekan tersendiri.

Rumus untuk *Length One Loop:*

$$A =>_w A = \left(\frac{|A>_w A|}{\max\{|A>_w x|\,|\, x \in e\}}\right) \qquad (2.2)$$

Keterangan:
$A =>_w A$: nilai *dependency Length One Loop*
$|A >_w A|$: frekuensi aktivitas A yang diikuti secara langsung oleh A.
$\max\{|A >_w x|| \, x \in e\}$: frekuensi aktivitas A yang diikuti oleh aktivitas x dimana aktivitas x merupakan bagian dari aktivitas yang ada dari event log.

Hasil dari Persamaan 2.2 menghasilkan suatu matriks yaitu matriks *length one loop* yang digunakan untuk membentuk *one loop* pada *dependency graph.*

Rumus untuk *Length Two Loop:*

$$A =>_{2w} B = \left(\frac{|A \gg_w B| + |B \gg_w A|}{|A \gg_w B| + |B \gg_w A| + 1}\right) \quad (2.3)$$

Keterangan:
$A =>_{2w} B$: nilai *dependency Length Two Loop*
$|A \gg_w B|$: frekuensi dari *trace* dalam bentuk ABA muncul dalam log
$|B \gg_w A|$: frekuensi dari *trace* dalam bentuk BAB muncul dalam log

Hasil dari Persamaan 2.3 menghasilkan suatu matriks yaitu matriks *length two loop* yang digunakan untuk membentuk *two loop* pada *dependency graph.*

➨ *Mining Process Parallel*

Untuk mendapatkan aktivitas paralel pada algoritma ini menggunakan *parallel measure* untuk menentukan suatu aktivitas tersebut paralel atau tidak.

Sebelum menghitung nilai dari *parallel measure* ditentukan terlebig dahulu proses mana saja yang paralel dengan menggunakan *causal* matriks seperti pada Tabel 2.5.

Jika *input output*-nya lebih dari satu aktivitas maka ada kemungkinan paralel maka dihitung nilai *parallel measure* dari aktivitas tersebut.

Persamaan 2. 4 merupakan rumus untuk penentuan dari *parallel measure.*

$$A =>_w B \wedge C = \left(\frac{|B >_w C| + |C >_w B|}{|A >_w B| + |A >_w C| + 1}\right) \quad (2.4)$$

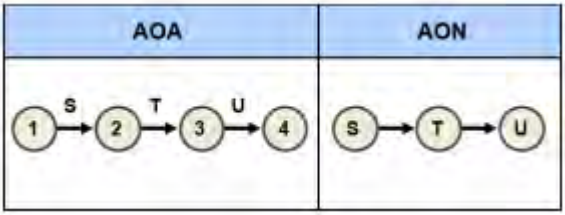
Keterangan:
$A =>_w B \wedge C$: *parallel measure* antara B dan C dimana *split* terjadi di A.
$|B >_w C|$: frekuensi aktivitas B yang diikuti secara langsung oleh C.
$|C >_w B|$: frekuensi aktivitas C yang diikuti secara langsung oleh B.
$|A >_w B|$: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$|A >_w C|$: frekuensi aktivitas A yang diikuti secara langsung oleh C.

Kekurangan dari *heuristic miner* yang lainnya adalah hanya dapat men-*discover* jenis *split* dan *join* XOR dan AND, sementara untuk OR tidak bisa walaupun sebenarnya C-Net dapat memodelkan bentuk *split* dan *join* OR. Oleh karena itu dalam Tugas Akhir ini akan memodifikasi algoritma ini sehingga dapat men-*discovery split* dan *join OR*.

2.9 *Critical Path Method*

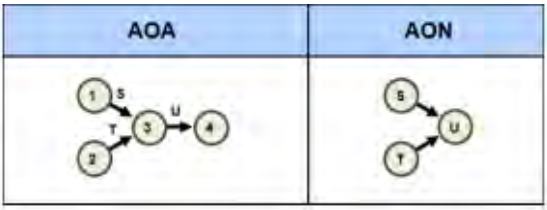
Critical Path Method (CPM) adalah teknik untuk menganalisa jaringan kegiatan/aktivitas-aktivitas ketika menjalankan proyek dalam rangka memprediksi durasi total. *Critical path* sendiri dalam sebuah proyek adalah jalur terpanjang dalam *network diagram* (Prawira, 2014).

Critical path didapatkan dari sebuah diagram jaringan (*network diagram*) yang memperlihatakan hubungan dan urutan aktitivas-aktivitas dalam suatu proyek. Secara umum network diagram digambarkan menggunakan *activity on node* (AON) dan *activity on arrow* (AOA) (Larson & Gray, 2006). Pada AON, aktivitas proyek direpresentasikan dengan titik (*node*), sementara pada AOA, aktivitas kegiatan direpresentasikan dengan panah (*arrow*). Aktivitas proyek yang mendahului/menjadi syarat dillakukan aktivitas lainnya disebut *predesesor*. Gambar 2.4 sampai dengan gambar Gambar 2.6, menunjukkan hubungan aktivitas dalam proyek menggunakan AOA dan AON.

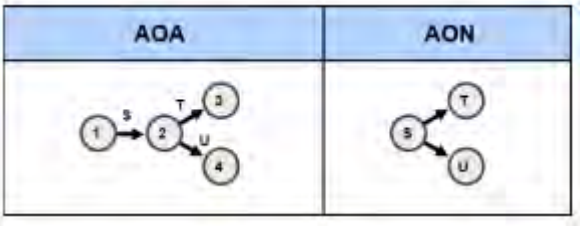


Gambar 2.4 Bentuk Serial

Pada Gambar 2.4 (bentuk serial), diperlihatkan tiga aktivitas proyek yaitu S, T, dan U. Pada gambar tersebut ditunjukkan bahwa aktivitas S merupakan *predesesor* bagi aktivitas T, sementara aktivitas T menjadi *predesesor* bagi aktivitas U. Pada Gambar 2.5 Bentuk (bentuk *join*), diperlihatkan bahwa aktivitas S dan T menjadi *predesesor* bagi aktivitas U, atau dengan kata lain dapat dikatakan bahwa aktivitas U bisa dilaksanakan jika aktivitas S dan T sudah dilaksanakan terlebih dahulu. Gambar 2.6 (bentuk *split*) memperlihatkan aktivitas S menjadi *predesesor* bagi aktivitas T dan U. Hal ini menggambarkan bahwa aktivitas T dan U bisa dilaksanakan jika aktivitas S telah dilaksanakan terlebih dahulu.



Gambar 2.5 Bentuk *Join*



Gambar 2.6 Bentuk *Split*

Pada Tugas Akhir ini digunakan jenis AON untuk menentukan *critical path* pada jaringan atau proses bisnis yang digunakan yang berguna untuk menentukan durasi dari proses bisnis.

Selain digunakan untuk menentukan durasi dari proses bisnis salah satu metode CPM yaitu *crashing project* digunakan sebagai dasar untuk optimasi durasi. Dasar yang digunakan yaitu perhitungan dari perhitungan *cost slope* dan *time slope* yang akan digunakan untuk menjadi konstanta dalan model matematika dari *linear programming*.

2.10 *Linear Programming*

Linear programming merupakan suatu teknik optimasi dimana variabel pembentuknya merupakan variabel-variabel *linear. Linear programming* digunakan untuk melakukan suatu optimasi dalam bentuk hasil keputusan yang maksimum atau minimum dengan menggunakan batasan-batasan tertentu. Pemrograman linier berkaitan dengan pemodelan suatu kasus yang ada pada dunia nyata ke dalam suatu model matematika yang terdiri dari sebuah fungsi tujuan (*objective function*) linier dengan beberapa batasan (*constrain function*) linier. Salah satu bentuk permasalahan yang dipecahkan dengan *linear programming* adalah perencanaan aktivitas untuk mendapatkan hasil optimal (menurut model matematika) diantara semua kemungkinan alternatif yang ada.

Sekilas tentang sejarah *linear programming*, Seorang Matematikawan Rusia L.V. Kantorovich pada 1939 berhasil menemukan pemecaham masalah yang berkaitan dengan program linear. Pada waktu itu Kantorovich bekerja untuk Kantor Pemerintah Uni Soviet. L.V. Kantorovich diberi tugas untuk mengoptimalkan produksi pada industri *plywood*. L.V. Kantorovich kemudian muncul dengan teknik matematis yang diakui sebagai pemrograman linear. Matematikawan Amerika George B. Dantzig mengembangkan pemecahan masalah tersebut, dimana hasil karyanya pada masalah tersebut pertama kali dipublikasikan pada tahun 1947.

Pada setiap masalah, ditentukan variabel keputusan, fungsi tujuan, dan sistem kendala, yang bersama-sama membentuk suatu model matematika dari dunia nyata. Bentuk umum dari program linier adalah (Taha):

- Fungsi Tujuan:
  $Max\ atau\ Min\ Z = \sum_{i=1}^{h} C_i X_i$ (2.5)
- Fungsi Batasan:
  $\sum_{j=1}^{h} a_{ij} X_j \leq atau = atau \geq b_i\ untuk\ i = 1, 2, 3, \ldots, g$ (2.6)

Keterangan:
Z : nilai fungsi tujuan.
C_i : sumbangan per unit kegiatan, untuk masalah maksimisasi C_i menunjukkan keuntungan atau penerimaan per unit, sementara dalam kasus minimisasi menunjukkan biaya per unit.
X_i : banyaknya kegiatan j, dimana i = 1, 2, 3, … n. berarti disini terdapat n variabel keputusan.
a_{ij} : banyaknya sumberdaya i yang dikonsumsi sumberdaya j.
b_i : jumlah sumberdaya i (i = 1, 2, …, m)
g : macam batasan sumber atau fasilitas yang tersedia.
h : macam kegiatan yang menggunakan sumber atau fasilitas tersebut.

Untuk menyelesaikan model matematika dari *linear programming* dalam tugas akhir ini menggunakan metode *simplex*. Metode *simplex* sendiri merupakan salah satu teknik penyelesaian dalam program linear yang digunakan sebagai teknik pengambilan keputusan dalam permasalahan yang berhubungan dengan pengalokasian sumberdaya secara optimal. Metode *simplex* digunakan umtuk mencari nilai optimal dari program linier yang melibatkan banyak *constraint* (pembatas) dan banyak variabel.

2.11 *Cross Organizational Bussiness Process Model*

Ada berbagai macam pola koordinasi pada Antar-Organisasi *workflow* yaitu,

2.11.1 Pola 1 : Koordinasi dengan Aktivitas yang Sinkron

Aktivitas yang dapat dilaksanakan oleh dua organisasi yang berbeda dapat didefinisikan sebagai pola koordinasi antar organisasi. Misalnya, dalam proses bisnis multi-moda transportasi, penandatanganan kontrak transportasi harus diselesaikan oleh *Sender* dan *Consignor* secara bersama-sama. Dengan demikian, penandatanganan kontrak merupakan kegiatan sinkron untuk mengkoordinasikan proses antara *Sender* dan *Consignor*. Informasi yang sama tentang awal dan akhir penandatanganan kontrak dapat direkam. Syarat pola ini adalah sebagai berikut (Zeng, Sun, Duan, Liu, & Wang, 2013):

Definisi 2.(Koordinasi dengan aktivitas yang sinkron) Misal $\Sigma_1 = (P_1,T_1;F_1,M_{01})$ and $\Sigma_2=(P_2,T_2;F_2,M_{02})$ menjadi model *workflow* dari dua organisasi jika :

(1) $T1 \cap T2 \neq \emptyset$, dan

(2) $P1 \cap P2 = \emptyset$,

Σ_1 dan Σ_2 adalah koordinasi dengan aktivitas yang sinkron. Jika $\Sigma=(P,T;F,M0)$, maka

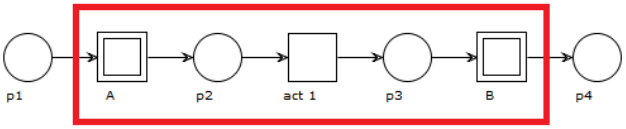
(1) $P=P_1 \cup P_2$;

(2) $T=T_1 \cup T_2$;

(3) $F=F_1 \cup F_2$;

(4) $M_0=M_{01} \cup M_{02}$.

Pada pola koordinasi ini, paling tidak ada satu aktivitas *workflow* dikerjakan oleh dua organisasi (Zeng, Sun, Duan, Liu, & Wang, 2013).



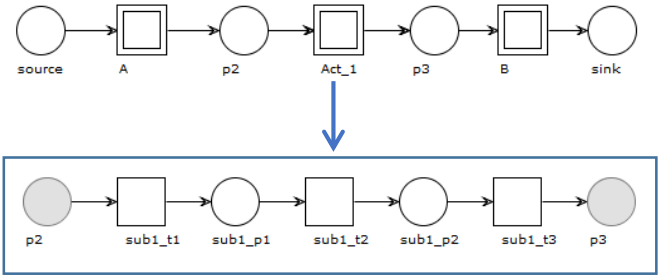
Gambar 2.7 Pola aktivitas sinkron

Aktivitas sinkron digambarkan oleh transisi “act 1” diantara organisasi A dan organisasi B. Aktivitas yang tidak sinkron berada di dalam subproses masing-masing organisasi sehingga aktivitas *private* tidak dapat diketahui oleh organisasi yang lain. Aktivitas yang dapat diketahui oleh organisasi yang lain hanyalah aktivitas yang bersifat sinkron atau dilaksanaan bersama.

Ada beberapa pola aktivitas sinkron yang mungkin terjadi dalam pelaksanaan proses bisnis, antara lain:

- Pola hubungan *Capasity sharing*

  Bentuk kerjasama *capasity sharing* adalah bentuk yang diasumsikan mempunyai pengendalian terpusat (*controlized control*), dimana sebuah urutan kasus dikendalikan oleh sebuah organisasi yang melintasi beberapa organisasi. Pelaksanaan pekerjaan didistribusikan kepada orgainsasi lain yang menjalankannya.

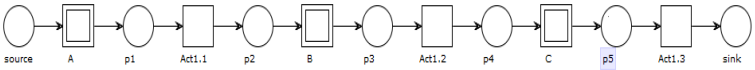


Gambar 2.8 Hubungan *Capasity sharing*

Act_1 adalah aktivitas yang akan di-*share* dari organisasi A sebagai pusat kontrol aktivitas Act_1 ke organisasi B sebagai penerima aktivitas. Act_1 adalah aktivitas subproses yang didalamnya terdiri dari aktivitas-aktivitas yang berurutan.

- Pola hubungan *chained execution*

  Bentuk kerjasama *chained execution* adalah suatu proses dipilah berdasarkan subproses (*disjoint* subproses) dimana pelaksanaannya dilakukan oleh beberapa organisasi secara berurutan. Kerjasamanya ini membutuhkan *partner* yang mengawali atau mentransfer urutan sebuah kasus sampai pada akhirnya semua pekerjaan dapat diselesaikan. Berlawanan dengan *capacity sharing*, pengendalian urutan pekerjaan (*workflow*) didistribusikan melawati berbagai organisasi.

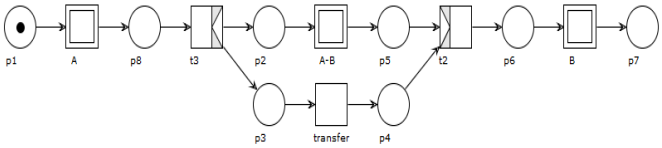


Gambar 2.9 Hubungan *chained execution*

  Act1.1, Act1.2, dan Act1.3 adalah aktivitas yang dilaksanakan secara berurutan namun memiliki karakteristik yang berbeda pada masing-masing organisasi. Perbedaan dengan aktivitas sinkron pada umumnya adalah untuk *chained execution,* setiap organisasi melakukan aktivitas yang mirip namun tidak dapat dilakukan secara bersama-sama.

- Pola hubungan *case transfer*

  Bentuk kerjasama organisasi berikutnya adalah *case transfer*. Pada bentuk kerjasama ini setiap organisasi mempunyai deskripsi proses pekerjaan yang sama. Sebuah *case process* instan dapat dilakukan oleh organisasi yang lain untuk menjaga keseimbangan pekerjaan agar tidak terjadi *workload* (kelebihan pekerjaan).

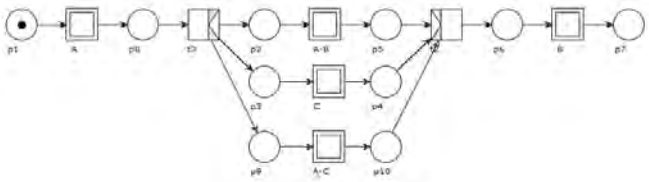


Gambar 2.10 Pola *case transfer*

Organisasi A dan B memiliki hubungan aktivitas 'transfer' yang berisi aktivitas organisasi A yang dilakukan oleh organisasi B.

- Pola hubungan *loosely coupled*

  Bentuk kerjasama organisasi berikutnya adalah *loosely coupled* dimana bentuk kerjasama ini sebuah proses dipotong-potong dalam bagian yang terjadi secara bersamaan. Ada semacam protocol yang mengkomunikasikan dan menghubungkan dengan bagian lain yang terlibat.



Gambar 2.11 Pola *loosely coupled*

Organisasi A adalah *protocol* untuk oragnisasi B dan C yang mengatur aktivitas pada hubungan A-B dan A-C.

2.11.2 Pola 2 : Koordinasi dengan pertukaran pesan

Selama pelaksanaan *workflow*, ada banyak kasus pesan-pesan yang dipertukarkan antara partner yang berbeda. Contoh, setelah *Buyer* melakukan pembayaran, verifikasi pembayaran pesan dikirim ke *Sender*, dan *workflow* dikoordinasikan dengan bertukar pesan verifikasi pembayaran antara *Buyer* dan *Sender* (Zeng, Sun, Duan, Liu, & Wang, 2013).

Definisi 3. (Koordinasi dengan pertukaran pesan) $\Sigma_1=(P_1,T_1;F_1,M_{01})$ dan $\Sigma_2=(P_2,T_2;F_2,M_{02})$ adalah *workflow* model dari dua organisasi, jika :

(1) $P_{M1}\cap P_{M2}\neq\emptyset$,

(2) $P_{L1}\cap P_{L2}=\emptyset$,

(3) $P_{R1}\cap P_{R2}=\emptyset$,

(4) $T_1\cap T_2=\emptyset$,

Σ_1 dan Σ_2 adalah koordinasi dengan pertukaran pesan. $\Sigma=(P,T;F,M_0)$ adalah model yang dibuat oleh Σ_1 dan Σ_2 dengan pertukaran pesan, sehingga

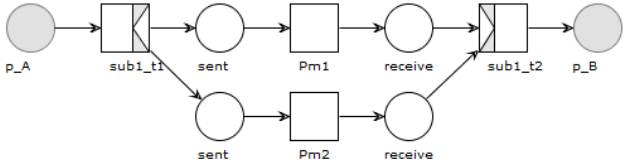
(1) $P=P_1\cup P_2$;

(2) $T=T_1\cup T_2$;

(3) $F=F_1\cup F_2$;

(4) $M_0=M_{01}\cup M_{02}$. (Zeng, Sun, Duan, Liu, & Wang, 2013)

Pada pola koordinasi ini, paling tidak ada satu pertukaran pesan pada *workflow* dikerjakan oleh dua organisasi. Pesan dapat berupa aliran data, form, dan pertukaran pesan dengan *Electronic Data Interchange (EDI)* (Zeng, Sun, Duan, Liu, & Wang, 2013).



Gambar 2.12 Pola pertukaran pesan

Pertukaran pesan berada didalam subproses yang menggambarkan hubungan antar organisasi. "p_A" adalah *place* awal dari organisasi A dan "p_B" adalah place akhir dari organisasi B. Pm1 dan Pm2 adalah pesan yang dikirimkan dari organisasi A ke organisasi B dengan keterangan *place* sebelumnya yaitu *sent* dan *receive*.

2.11.3 Pola 3 : Koordinasi dengan pembagian sumber daya

Karena sumber daya yang sering diakses secara eksklusif oleh *private workflow*, alokasi sumber daya yang penting dalam menghindari konflik sumber daya antar organisasi yang berbeda ketika lebih dari satu *private workflow* memiliki kebutuhan untuk mengakses sumber daya yang sama. Misalnya, dalam proses bisnis multi moda transportasi, sumber daya seperti mobil box, stasiun barang, dan *broker* pabean diakses oleh *private workflow*. Namun, jika sumber daya seperti mobil box dan stasiun barang ditempati oleh suatu kegiatan dalam *private workflow*, kegiatan yang membutuhkan sumber daya yang sama di tempat lain harus menunggu sampai sumber daya dilepaskan. Sehingga sumber daya alokasi dan kontrol konflik sumber daya yang penting untuk dua *workflow* dikoordinasikan dengan sumber daya bersama (Zeng, Sun, Duan, Liu, & Wang, 2013).

Definisi 4. (Koordinasi dengan pembagian sumber daya) $\Sigma_1=(P_1,T_1;F_1,M_{01})$ dan $\Sigma_2=(P_2,T_2;F_2,M_{02})$ adalah *workflow* model dari dua organisasi, jika :

(1) $P_{M1}\cap P_{M2}\neq\emptyset$,

(2) $P_{L1}\cap P_{L2}=\emptyset$,

(3) $P_{R1}\cap P_{R2}=\emptyset$,

(4) $T_1\cap T_2=\emptyset$,

Σ_1 dan Σ_2 adalah koordinasi dengan pembagian sumber daya $\Sigma=(P,T;F,M_0)$ adalah model yang dibuat oleh Σ_1 dan Σ_2 dengan pembagian sumber daya, sehingga

(1) $P=P_1\cup P_2$;

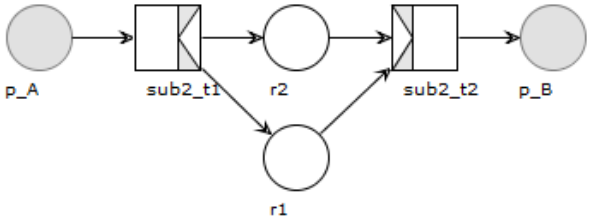
(2) $T=T_1\cup T_2$;

(3) $F=F_1\cup F_2$;

(4) $M_0=M_{01}\cup M_{02}$. (Zeng, Sun, Duan, Liu, & Wang, 2013)

Pada pola ini, paling tidak terdapat satu sumber daya yang dibagikan antara dua organisasi. Pembagian sumber daya ini dapat menggunkaan sistem *cloud-based* sehingga pada masing-masing organisasi, dilakukan perekaman *resource* yang

dibutuhkan untuk memudian di koordinasikan dengan *cloud* untuk organisasi lain (Zeng, Sun, Duan, Liu, & Wang, 2013).



Gambar 2.13 Pola *resource sharing*

Resource tidak dapat digambarkan sebagai transisi karena tidak termasuk aktivitas yang menghubungkan antar organisasi sehingga *resource* digambarkan sebagai place yang menghubungkan antar organisasi didalam subproses hubungan. Pada gambar 3, r1 dan r2 adalah *resource* yang saling digunakan oleh organisasi A dan organisasi B. Pada subproses hubungan antara organisasi B dengan A, maka *place resource* juga harus disertakan.

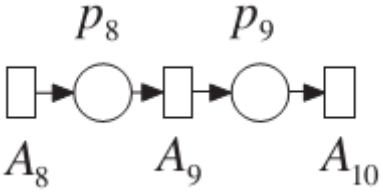
2.11.4 Pola 4 : Koordinasi dengan prosedur yang abstrak

Saat melakukan koordinasi dengan organisasi lain, persyaratan dan output dari *private workflow* dapat dipublikasikan dengan rincian yang dilakukan oleh organisasi lain. Rincian aktivitas tersebut merupakan satu aktifitas oleh organisasi lain. Sehingga antar-organisasi sebetulnya melakukan aktifitas yang sama.

Σ_1 dan Σ_2 adalah koordinasi dengan prosedur yang abstrak $\Sigma=(P,T;F,M_0)$ adalah model yang dibuat oleh Σ_1 dan Σ_2 dengan prosedur yang abstrak, sehingga

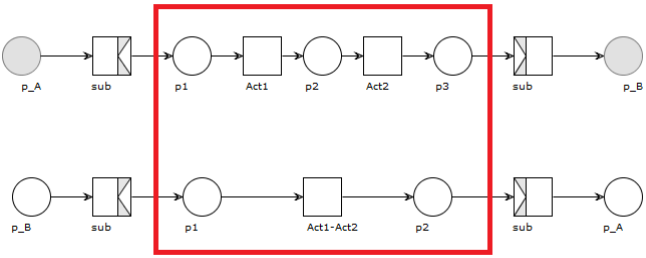
(1) $P=P_1 \cup P_2$;
(2) $T=T_1 \cup T_2$;
(3) $F=F_1 \cup F_2$;
(4) $M_0=M_{01} \cup M_{02}$. (Zeng, Sun, Duan, Liu, & Wang, 2013)

Jika *Consignor* tidak ingin membuat dokumen prosedur bea cukai, tiga kegiatan yang terlibat, yaitu *Generate Declaration Form, Declaration Application*, dan *Declaration Acceptance* dapat dikemas sebagai prosedur abstrak yang diterbitkan untuk kepentingan partner. Contoh prosedur abstrak dijelaskan pada Gambar 2.14 yang merepresentasikan kegiatan membuat dokumen prosedur Bea dan Cukai.



Gambar 2.14 Contoh prosedur abstrak

Pada pola prosedur abstrak, terdapat jenis koordinasi lain yaitu *subcontracting*. Pada koordinasi ini sebuah organisasi membagi aktivitas-aktivitas tertentu menjadi subproses kepada organisasi yang lain. Hubungan subcontracting dapat dilakukan untuk mengatasi aktivitas berurutan yang dilakukan oleh dua organisasi yang berbeda secara bersama-sama.



Gambar 2.15 Pola prosedur *subcontracting*

Prosedur abstrak hampir sama dengan aktivitas sinkron, namun pada satu organisasi tidak melakukan semua aktivitas secara

terperinci dan organisasi lain melakukan seluruh rangkaian aktivitas. Pada Gambar 2.15 dijelaskan untuk hubungan organisasi A ke B, organisasi A harus melakukan aktivitas Act1 dan Act2 secara berurutan, sedangkan pada hubungan organisasi B ke organisasi A, organisasi B hanya mekukan satu aktivitas yaitu Act1-Act2 yang dilakukan secara bersama dan tidak terperinci.

BAB III
METODE PEMECAHAN MASALAH

Pada bab ini akan dibahas mengenai metodologi pemecahan masalah yang digunakan sebagai dasar solusi dari pembuatan Tugas Akhir. Metodologi tersebut menerangkan langkah demi langkah proses hingga dapat menghasilkan model dari suatu *event log* permasalahan ini dibahas pada subbab 3.2 sampai dengan 3.4 dan dapat mengasilkan suatu persamaan linear guna untuk mengoptimasi model dari *event log* permasalahan ini akan dibahas pada subbab 3.5.

3.1 Cakupan Permasalahan

Permasalahan utama yang diangkat dalam pembuatan Tugas Akhir ini adalah memodifikasi *heuristic miner algorithm,* menentukan limitasi dari *heuristic miner*, dan bagaimana mengoptimasi waktu *makespan* dari suatu proses bisnis dan biaya optimasi waktu yang diperlukan (total *crash cost*).

Heuristic miner algorithm merupakan salah satu algoritma *process discovery* yang sulit untuk digunakan, hal ini dikarenakan pada *heuristic miner algorithm* tedapat *threshold-trheshold* yang harus ditentukan pengguna. Sementara pengertian dari *threshold* itu sendiri tidak semua pengguna mengerti. Oleh karena itu dalam Tugas Akhir ini yang dimaksud dengan memodifikasi *heuristic miner algorithm* adalah menentukan nilai dari *threshold* yang ada. Selain penentuan *threshold* yang dimaksud dalam memodifikasi *heuristic miner algorithm* disini adalah merubah rumus untuk *mining parallel activity* dan penambahan beberapa interval sehingga dapat men-*discover split* dan *join* OR. Sehingga dalam modifikasi *heuristic miner algorithm* terdapat dua tahap utaha yaitu penentuan nilai *threshold* dan perubahan pada rumus *mining parallel activity.* Selain memodifikasi *heuristic miner algorithm* Tugas Akhir ini juga menentukan limitasi dari *heuristic miner algorithm* baik yang sudah dimodikasi maupun limitasi sebelum dan sesudah modifikasi.

Permasalahan selanjutnya adalah pengoptimasian waktu atau *makespan* dari proses bisnis dan biaya yang dibutuhkan (total *crash cost*). Karena bentuk dari *event log* yang ada hanya memiliki *single timestamp* mengakibatkan aktivitas dalam proses bisnis memiliki durasi yang tidak tentu oleh karena itu langkah awal dalam memecahkan masalah ini adalah menentukan durasi untuk setiap aktivitas yang ada dalam proses bisnis. Selanjutnya untuk pengotimalan waktu atau *makespan* dan biaya yang dibutuhkan diperlukan *crash time* (waktu yang dikurangkan dari normal waktu aktivitas) dan *crash cost* (biaya yang diperlukan untuk pemampatan per aktivitas), maka langkah selanjutnya adalah penentuan durasi (*normal duration*), *crash duration,* dan *crash cost* per aktivitas, yang terakhir adalah pemodelan bentuk linear untuk mendapatkan waktu dan biaya paling optimal.

3.2 Modifikasi *Heuristic Miner Algorithm*

Pada bagian ini dijelaskan tentang bagaimana *heuristic miner algorithm* digunakan dalam Tugas Akhir ini dan bagaimana *heuristic miner algorithm* dimodifikasi dalam Tugas Akhir ini. Tahapan *heuristic miner algorithm* yang telah dikembangkan memiliki kesamaan dengan *heuristic miner algorithm* yang telah dikembangkan sebelumnya hanya saja terdapat tambahan dan perubahan pada tahapan dan rumus yang ada pada tiga tahap utama.

3.2.1 *Mining Dependency Graph*

Sama seperti tahapan yang ada pada *heuristic miner algorithm* yang telah dikembangkan sebelumnya, pada *heuristic miner algorithm* yang telah dimodifikasi memerlukan matriks frekuensi dari hubungan antar aktivitas dalam proses bisnis. Kemudian frekuensi dari matriks frekuensi akan dihitung menggunakan Persamaan 2.1. Modifikasi yang dilakukan pada bagian ini adalah pada penentuan *threshold* yang akan digunakan untuk memilih *dependency measure* pada dependency matriks yang akan digunakan dalam model. Penentuan dari nilai-nilai *threshold* ini dimaksudkan untuk mempermudah pengguna agar

dapat men-*discover* model yang benar. Penentuan dari *threshold-threshold* itu adalah sebagai berikut:

a. Relative-to-best threshold (RBT)
   Threshold ini digunakan untuk memilih *dependency measure* yang memiliki selisih dengan maximum *dependency measure* pada *matrix dependency* yang lebih kecil dari nilai *threshold* ini.
   - Langkah pertama dalam menghitung *Relative-to-best threshold* (RBT) ini adalah menghitung rata-rata (Avg) dari *positive dependency measure* pada *matrix dependency* (PDM).
   - Setelah itu menentukan nilai dari RBT menggunakan Persamaan 3.1.

     $$RBT = Avg\ PDM - \left(\frac{SD\ PDM}{2}\right) \quad (3.1)$$

     Keterangan:
     RBT : *Relative-to-best threshold*
     $Avg\ PDM$: rata-rata *positive dependency measure* pada *matrix dependency*
     $SD\ PDM$: standar deviasi *positive dependency measure* pada *matrix dependency*
     Hubungan antar aktivitas diambil jika memenuhi Persamaan 3.2.

     $$Max(DM) - DM_{a=>b} \leq RBT \quad (3.2)$$

     Keterangan:
     DM : *dependency measure*
     $DM_{\boldsymbol{a=>b}}$: *dependency measure* antar aktivitas dalam dependency matriks.

   Diambil menggunakan rata-rata karena untuk mengambil nilai yang paling general dari *dependency measure* yang ada.

b. Positive observations threshold (POT)
   Threshold ini mengontrol jumlah minimum berapa kali aktivitas memiliki ketergantungan hubungan dengan aktivitas lain. Persamaan 3.3 merupakan rumus untuk penentuan *threshold* ini.

$POT = Min\ (f_{a=>b})$ (3.3)

Hubungan antar aktivitas diambil jika memenuhi Persamaan 3.4.

$f_{a=>b} \geq POT$ (3.4)

Keterangan:

POT : *positive observations threshold.*

$f_{a=>b}$: frekuensi antar aktivitas pada matriks frekuensi.

c. Dependency Threshold (DT)

Threshold ini berguna untuk mengabaikan semua hubungan yang nilai *dependency measure* berada dibawah nilai parameter. Persamaan 3.5 merupakan rumus untuk penentuan *threshold* ini.

$DT = Avg\ PDM - SD\ PDM$ (3.5)

Hubungan antar aktivitas diambil jika memenuhi Persamaan 3.6.

$DM_{a=>b} \geq DT$ (3.6)

Keterangan:

DT : *dependency threshold*

3.2.2 *Mining Short Loop (Length One Loop dan Length Two Loop)*

Pada tahap ini tidak terdapat perubahan. Untuk menghitung *short loop heuristic miner* yang dimodifikasi tetap menggunakan Persamaan 2.2 dan Persamaan 2.3.

3.2.3 *Mining Process Parallel*

Pada *heuristic miner algorithm* yang telah dikembangkan sebelumnya hanya dapat men-*discover split* dan *join* AND dan XOR. Oleh karena itu dalam *heuristic miner algorithm* yang telah dimodifikasi ini akan membuat *heuristic miner algorithm* agar dapat men-*discover* semua jenis *split* dan *join* baik itu AND, XOR, maupun OR. Untuk dapat men-*discover* OR yang pertama dilakukan sama dengan *heuristic miner algorithm* yaitu

membentuk *causal matrix* dari hasil *dependency graph,* tapi dalam memodifikasi ini dilakukan perubahan pada rumus yang digunakan untuk *parallel measure*. Pada persamaan sebelumya pada Persamaan 2.4 hanya mementingkan *direct followed frequency* dari aktivitas pada cabang, padahal sebenarnya untuk *split* dan *join* AND sendiri memeperbolehkan adanya *undirect followed frequency*.

Perubahan dilakukan dengan mengganti frekuensi yang hanya menghitung *direct followed frequency* pada aktivitas percabangan dengan juga menghitung semua *undirect followed frequency* karena sifat dari AND sendiri yang tidak hanya dapat dikakukan secara bersamaan atau *sequence* tapi aktivitas yang ada dalam percabangan dapat dilakukan secara tidak *sequence* yang penting semua aktivitas yang ada dalam percabangan dilakukan.

Untuk perhitungan OR dalam Tugas Akhir ini juga diperhitungkan frekuensi dari aktivitas yang tidak dijalankan secara bersamaan. Rumus untuk *parallel measure* adalah sebagai berikut:

$$A =>_w B \wedge C = \left(\frac{|B \gg>_w C| + |C \gg>_w B|}{|A >_w B| + |A >_w C| + |B \gg> not_w C| + |C \gg> not_w B| + 1} \right) \quad (3.7)$$

Keterangan:

$A =>_w B \wedge C$: *parallel measure* antara aktivitas pada percabangan dari A yaitu B dan C.

$|B \gg>_w C|$: *undirect and direct followed frequency* aktivitas B dan C

$|C \gg>_w B|$: *undirect and direct followed frequency* aktivitas C dan B

$|A >_w B|$: frekuensi aktivitas A yang diikuti secara langsung oleh B.

$|A >_w C|$: frekuensi aktivitas A yang diikuti secara langsung oleh C.

$|B \gg> not_w C|$: frekuensi eksekusi aktivitas B yang tidak diikuti aktivitas C.

$|C \gg> not_w B|$: frekuensi eksekusi aktivitas C yang tidak diikuti aktivitas B.

3.2.4 Pemodelan OR, XOR, dan AND

Untuk pemodelan XOR, OR dan AND penentuannya menggunakan interval yang digunakan untuk menentukan letak dari rata-rata *parallel measure* dari aktivitas dengan *input split* yang sama ataupun *output join* yang sama.Yang pertama dilakukan adalah menentukan rata-rata dari *dependency measure* yang masuk dalam *dependency graph* dan rata-rata dari *parallel measure.*

$$Avg\ PDM = \frac{\sum_{i=1}^{n_e} e_i}{n_e} \quad (3.8)$$

$$Avg\ PM = \frac{\sum_{i=1}^{n_{PM}} PM_i}{n_{PM}} \quad (3.9)$$

Interval untuk XOR, OR, dan AND adalah sebagai berikut:

- XOR
  $If\ Avg\ PM\ \leq Limit\ PDM\ then\ XOR$ (3.10)
- OR
  $If\ Limit\ PDM \leq Avg\ PM\ \leq AVG\ PDM\ then\ OR$ (3.11)
- AND
  $If\ Avg\ PDM\ \leq Avg\ PM\ then\ AND$ (3.12)

Keterangan:

$Avg\ PDM$: rata-rata *positive dependency measure* yang lebih dari 0 dari *positive dependency measure*

e_i : *dependency measure* pada *dependency matrix*

n_e : jumlah *dependency measure*

PM : *Parallel measure* dari Persamaan 3.7

$Avg\ PM$: rata-rata dari *parallel measure* yang memiliki induk aktivitas yang sama.

n_{PM} : jumlah *parallel measure* yang memiliki *input* atau *output* yang sama.

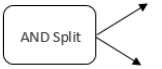
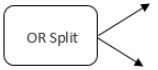
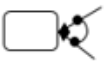
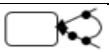
PDM *: positive dependency measure*

$Limit\ PDM$: nilai minimum dari positive *dependency measure* pada *matrix dependency measure.*

3.2.5 Penggambaran OR, XOR, dan AND pada Causal Net

Karena bentuk dari pemodelan untuk OR, XOR, dan AND yang telah dilimiki oleh C-Net kurang informatif dan susah dbaca oleh pengguna awam, maka dalam Tugas Akhir ini modelnya dirubah menjadi sebagai berikut:

Tabel 3.1 C-Net dan *Proposed Parallel Model*

C-Net Model	Proposed Model
	
	
	
	
	
	

3.3 Contoh *Discovery* Menggunakan Modifikasi *Heuristic Miner*

Diberikan *event log* sebagai berikut*:*

$$L = [(ABC), (ABDEC), (ADBEC), (ADEBC), (ABDEDEC)]$$

Sesuai dengan tahapan dari *heuristic miner* yang telah diterangkan pada bagian 3.2 maka yang pertama dilakukan adalah *mining*

dependency graph. Dalam *mining dependency graph* diperlukan matriks frekuensi. Matriks frekuensi dari *event log* L menghasilkan Tabel 3.2.

Tabel 3.2 Matriks Frekuensi *Event Log* L

Aktivitas	A	B	C	D	E
A	0	3	0	2	0
B	0	0	2	2	1
C	0	0	0	0	0
D	0	1	0	0	4
E	0	1	3	1	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.3.

Tabel 3.3 Matriks *Dependency Measure*

Aktivitas	A	B	C	D	E
A	0	0.75	0	0.667	0
B	0	0	0.667	0.25	0
C	0	0	0	0	0
D	0	-0.25	0	0	0.5
E	0	0	0.75	-0.5	0

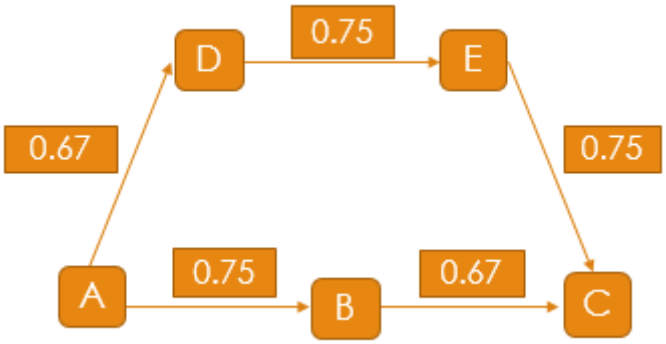
Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RBT = 0.5

PO = 1

DT = 0.42

Dengan menggunakan nilai dari *threshold* tersebut maka didapatkan *dependency graph* sebagi berikut:



Gambar 3.1 *Dependency Graph* dengan *Dependency Measure*

Setelah mendapatkan *dependency graph* dilakukan *mining* terhadap *short loop.* Dari matriks frekuensi pada Tabel 3.2 dan menggunakan Persamaan 2.2 didapatkan matriks *Length One Loop* sebagai berikut:

Tabel 3.4 Matriks *Length One Loop*

Aktivitas	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

Dari matriks *Length One Loop* pada Tabel 3.4 dapat diketahui bahwa model dari *event log* L ini tidak memiliki *Length One Loop.* Sedangkan untuk *Length Two Loop,* yang pertama harus dilakukan adalah menghitung frekuensi *Length Two Loop* sehingga mendapatkan matriks frekuensi *Length Two Loop.*

Tabel 3.5 Matriks Frekuensi *Length Two Loop*

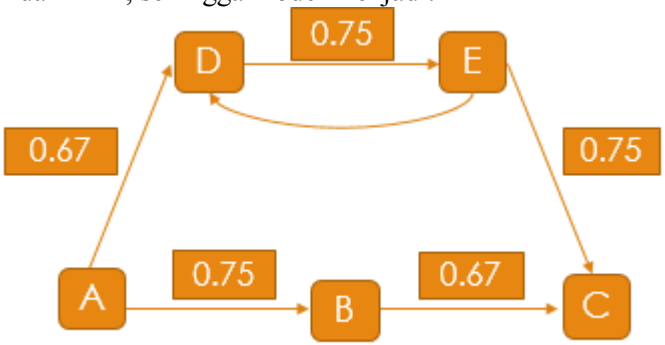
Aktivitas	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

Kemudia menggunakan Persamaan 2.3 mengitung *Length Two Loop* dan membentuk matriks mengitung *Length Two Loop.*

Tabel 3.6 Matriks *Length Two Loop*

Aktivitas	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0.667
E	0	0	0	0.667	0

Dari Tabel 3.6 diketahui bahwa model memiliki *Length Two Loop* DED dan EDE, sehingga model menjadi:



Gambar 3.2 *Dependency Graph* dengan *Dependency Measure* dan *Length Two Loop*

Selanjutnya adalah *mining parallel activity,* yang pertama dilakukan dalam *mining parallel activity* adalah menentukan *causal* matriks. *Causal* matriks untuk hasil *dependency graph* untuk contoh terdapat pada Tabel 3.11.

Tabel 3.7 *Causal* Matriks Gambar 3.1

Input	Activity	Output
{}	A	B , D
A	B	C
E, B	C	{}
A	D	E
D	E	C

Dari *causal* matriks pada Tabel 3.7 didapatkan terdapat satu *split* dan *join* yaitu *split* dengan *input* A dan aktivitas pada percabangan adalah B dan D, dan *join* dengan *output C* dengan aktivitas pada percabangan E dan B. Dengan Persamaan 3.7 maka didapatkan *parallel measure:*

$$a \rightarrow w^{b\hat{}d} = 0.57$$

$$c \rightarrow w^{b\hat{}e} = 0.57$$

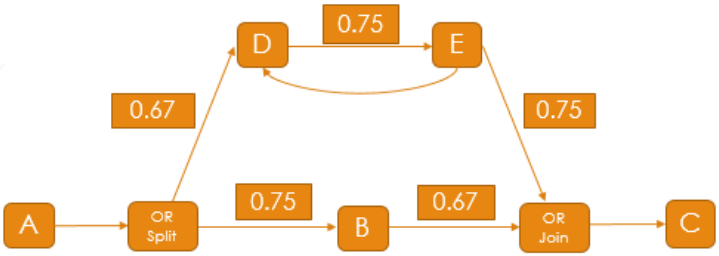
Lalu untuk nilai Avg PM *input* A adalah 0.57 dan Avg PM *output* C adalah 0.57.

Selain Avg PM yang dibutuhkan selanjutnya adalah:

Avg PDM dengan persamaan Persamaan 3.8 adalah sebesar = 0.59

Limit PDM = 0.25

Dengan menggunakan interval dari Persamaan 3.10, Persamaan 3.11, Persamaan 3.12 maka *split* dan *join* yang digunakan model yang di-*discovery* keduanya adalah OR. Sehingga modelnya menjadi sebagai berikut:



Gambar 3.3 Model Akhir dengan *Dependency Measure*

3.4 Limitasi *Heuristic Miner* dalam Menangani *Noise*

Dalam menangani *noise heuristic miner algorithm* memiliki keterbatasan-keterbatasan. Dalam Tugas Akhir ini juga mencari limitasi dari *heuristic miner algorithm.* Limitasi dari *heuristic miner algorithm* dalam menangani *noise* adalah sebagai berikut:

a. Jika *noise* digabungkan masih membentuk *trace* utuh maka *discovery* masih bisa berhasil. Dengan catatan *noise* membentuk *trace* yang hilang. *Noise* terbentuk dari pemotongan jumlah *head* dan *tail* yang sama.

$$If\ n_{s1} + n_{s2} = pt\ then\ discover\ success \quad (3.13)$$

Keterangan:

pt : aktivitas pada paralel *split* dan *join*

n_s: *noise*

Contoh:

Pada model dengan bentuk sesuai dengan Gambar 2.1 yang memiliki *event log* sesuai dengan Tabel **2.1**. Kemudian terbentuk noise dengan pemotongan kepala dan ekor dari dua buah *trace* sehingga *event log* menjadi sebagai berikut:

Tabel 3.8 *Event Log* dengan *Noise* Perpotongan Kepala dan Ekor

No. Trace	Trace Utuh
1	ABDCEFG
2	ABCEDFG
3	ABCDEFG

No. Trace	Trace Utuh
4	ACEBDFG
5	EDFG
6	ACBE

Jika *noise* yang terletak pada *trace* kelima dan keenam digabungkan maka akan membentuk salah satu *trace* utuh yang telah hilang yaitu *trace* ACBEDFG.

Jika *event log* tersebut di *discovery* menggunakan *heuristic miner* maka langkah-langkahnya adalah sebagai berikut:

- *Mining Dependency Graph*

  Matriks frekuensi dari *event log* pada Tabel 3.8 menghasilkan Tabel 3.9.

Tabel 3.9 Matriks Frekuensi Tabel 3.8

Aktivitas	A	B	C	D	E	F	G
A	0	3	2	0	0	0	0
B	0	0	2	2	1	0	0
C	0	1	0	1	3	0	0
D	0	0	1	0	1	3	0
E	0	1	0	2	0	2	0
F	0	0	0	0	0	0	5
G	0	0	0	0	0	0	0

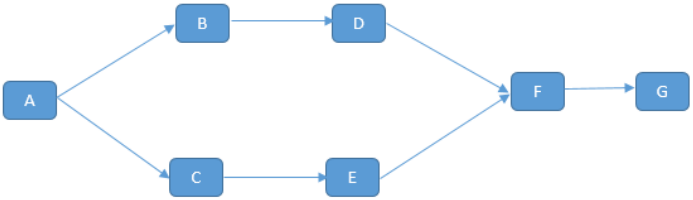
Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.10.

Tabel 3.10 Matriks *Dependency Measure* Tabel 3.8

Aktivitas	A	B	C	D	E	F	G
A	0	0.75	0.67	0	0	0	0
B	0	0	0.25	0.67	0	0	0
C	0	-0.25	0	0	0.75	0	0
D	0	0	0	-0.67	-0.25	0.75	0
E	0	0	0	0.25	0	0.67	0
F	0	0	0	0	0	0	0.83
G	0	0	0	0	0	0	0

Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RBT = 0.72
PO = 1
DT = 0.41



Gambar 3.4 *Dependency Graph* Tabel 3.8

- Mining Short Loop
  Model untuk Tabel 3.8 tidak memiliki *short loop.*
- Mining Parallel Activity

Tabel 3.11 Causal Matriks

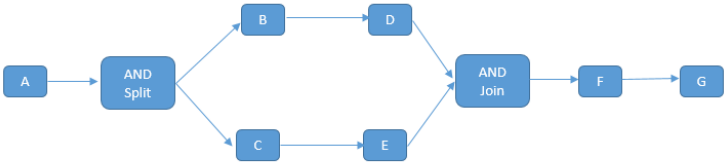
Input	Activity	Output
{}	A	B, C
A	B	D
A	C	E
B	D	F
C	E	F
D, E	F	G
F	G	{}

Dari *causal* matriks pada Tabel 3.11 didapatkan terdapat satu *split* dan *join* yaitu *split* dengan *input* A dan aktivitas pada percabangan adalah B dan C, dan join dengan *output* F dengan aktivitas pada percabangan D dan E. Dengan Persamaan 3.7 maka didapatkan *parallel measure:*

$A => w^{B\hat{}C} = 0.83$
$F => w^{D\hat{}E} = 0.83$

Nilai Avg PM *input* A adalah 0.83 dan Avg PM *output* F adalah 0.83.
Avg PDM = 0.62
Limit PDM = 0.25

Menurut interval pada Persamaan 3.9, Persamaan 3.10, Persamaan 3.11 maka split dan join yang digunakan model yang di-*discovery* keduanya adalah AND.



Gambar 3.5 Model Akhir dari Tabel 3.8

b. Untuk *noise* tidak membentuk suatu *trace* yang utuh jika *noise* saling digabungkan, minimum *trace* utuh yang dibutuhkan agar proses *discovery* berhasil adalah sebagai berikut:

Untuk jumlah *case* genap:

$$j_t = \frac{m}{2} + 1 \quad (3.14)$$

Untuk jumlah *case* genap:

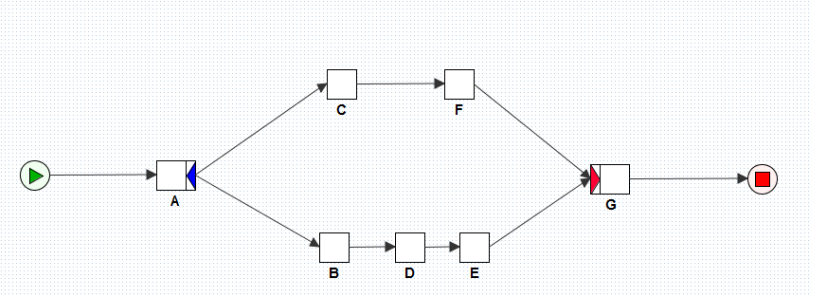
$$j_t = \frac{m+1}{2} \quad (3.15)$$

Keterangan:

m : jumlah *case* dalam *event log*

j_t : frekuensi *trace* dalam *event log*

Contoh pembuktian:
Terdapat suatu model:



Gambar 3.6 Model AND YAWL

Dari model YAWL tersebut dibangkitkan suatu *event log* sebagai berikut:

L1 = {(ACFBDEG), (ACBFDEG), (ACBDFEG), (ACBDEFG), (ABDECFG), (ABCDEFG), (ABCFDEG), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Dari *event log* L1 yang memiliki jumlah *case* 10, maka sesuai dengan yang ada pada Persamaan 3.14 dan Persamaan 3.15, maka jumlah *trace* utuh minimal yang diperlukan adalah 6 *trace* untuk dapat men-*discover event log* diatas walaupun terdapat *noise*. Maka untuk membuktikannya *event log* pada L1 akan dimasukkan *noise* yang pertama sesuai dengan Persamaan 3.14 dan Persamaan 3.15, dan yang satu melanggar jumlah minimum *trace* utuh yang dianjurkan pada Persamaan 3.14 dan Persamaan 3.15.

Contoh pertama:

Event log pada L1 dirubah menjadi *event log*

L1' = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Event log L1' memiliki jumlah *trace* utuh sesuai dengan persamaan Persamaan 3.14 dan Persamaan 3.15.

Dari *event log* L1' didapatkan perhitungan *heuristic miner algorithm* secagai berikut:

- *Mining Dependency Graph*
  Matriks frekuensi dari *event log* L1' menghasilkan Tabel 3.12.

Tabel 3.12 Matriks Frekuensi L1'

Aktivitas	A	B	C	D	E	F	G
A	0	5	2	0	0	0	1
B	0	0	2	4	0	1	0
C	0	2	0	2	1	3	0
D	0	0	2	0	4	2	0
E	0	0	0	0	0	3	4
F	0	0	0	2	3	0	4
G	0	0	0	0	0	0	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.13.

Tabel 3.13 Matriks *Dependency* Measure L1'

Aktivitas	A	B	C	D	E	F	G
A	0	0.83	0.67	0	0	0	0.5
B	0	0	0	0.8	0	0.5	0
C	0	0	0	0	0	0.75	0
D	0	0	0	0	0.8	0	0
E	0	0	0	0	0	0	0.8
F	0	0	0	0	0	0	0.8
G	0	0	0	0	0	0	0

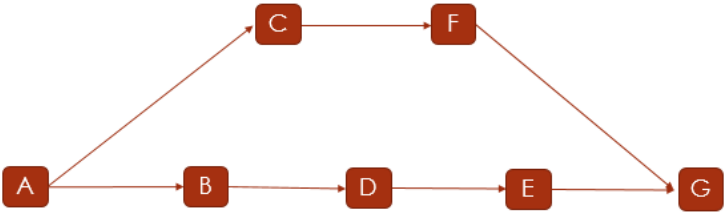
Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RTB = 0.65

PO = 1

DT = 0.59

Maka *dependency graph*-nya adalah sebagai berikut:



Gambar 3.7 *Dependency Graph* L1'

Untuk *short loop* model event log ini tidak memilikinya karena hasil matriks *length one loop* dan *two loop* semuanya 0.

Setelah itu melakukan *mining parallel activity* untuk itu memerkukan *causal* matriks.

Tabel 3.14 *Causal* Matriks L1'

Input	Activity	Output
{}	A	B, C
A	B	D
A	C	F
B	D	E
D	E	G
C	F	G
F, E	G	{}

Dari *causal* matriks pada Tabel 3.14 didapatkan terdapat satu *split* dan *join* yaitu *split* dengan *input* A dan aktivitas pada percabangan adalah B dan C, dan *join* dengan *output* G dengan aktivitas pada percabangan F dan E. Dengan Persamaan 3.7 maka didapatkan *parallel measure:*

$A => w^{B\hat{}C} = 0.77$

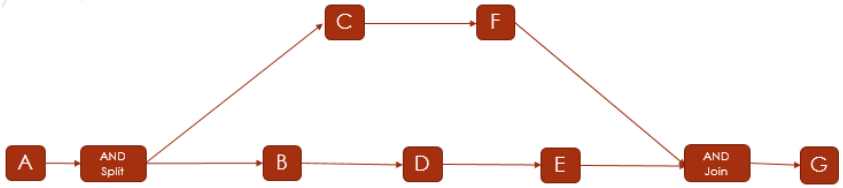
$G => w^{F\hat{}E} = 0.8$

Nilai Avg PM *input* A adalah 0.77 dan Avg PM *output* G adalah 0.8.

Avg PDM = 0.62

Limit PDM = 0

Menurut interval pada Persamaan 3.9, Persamaan 3.10, Persamaan 3.11 maka split dan join yang digunakan model yang di-*discovery* keduanya adalah AND.



Gambar 3.8 Model Akhir Hasil *Discover* L1'

Untuk *event log* yang jumlah trace utuhnya tidak sesuai dengan jumlah minimal pada Persamaan 3.14 dan Persamaan 3.15, sebagai contohnya adalah L1".

L1" = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCD)}

Dari *event log* L1" didapatkan perhitungan *heuristic miner algorithm* secagai berikut:

Matriks frekuensi dari *event log* L1" menghasilkan Tabel 3.15.

Tabel 3.15 Matriks Frekuensi L1"

Aktivitas	A	B	C	D	E	F	G
A	0	5	2	0	0	0	1
B	0	0	2	4	0	1	0
C	0	2	0	2	1	3	0
D	0	0	2	0	4	1	0
E	0	0	0	0	0	3	3
F	0	0	0	2	2	0	4
G	0	0	0	0	0	0	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.16.

Tabel 3.16 Matriks *Dependency Measure* L1"

Aktivitas	A	B	C	D	E	F	G
A	0	0.83	0.67	0	0	0	0.5

Aktivitas	A	B	C	D	E	F	G
B	0	0	0	0.8	0	0.5	0
C	0	0	0	0	0	0.75	0
D	0	0	0	0	0.8	-0.25	0
E	0	0	0	0	0	0.167	0.75
F	0	0	0	0.25	-0.167	0	0.8
G	0	0	0	0	0	0	0

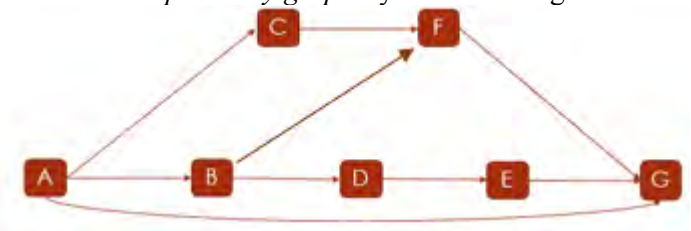
Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RTB = 0.5

PO = 1

DT = 0.39

Maka *dependency graph*-nya adalah sebagai berikut:



Gambar 3.9 *Dependency Graph* L1"

Dari *dependency graph*-nya sudah terjadi kesalahan *discovery* yaitu adanya hubungan antar aktivitas A dan G, B dan F yang sebenarnya pada model aslinya tidak terdapat hubungan antara A dan G, B dan F. Dengan ini maka proses *discovery* L1" gagal.

c. Jika jumlah *case* yang ada dari suatu *event log* sama dengan jumlah *complete trace*-nya (tanpa duplikasi), jumlah *complete trace* dari model tersebut senilai P_1^n, dan *noise* yang terdapat pada *event log* berupa *noise* yang terbentuk karena penghilangan *body* dari *trace* maka akan terjadi kegagalan *discovery*.

$$if\ m = n\ \&\&\ n = P_1^p\ \&\&\ t_{ns} = body\ t$$
$$then\ discover\ not\ success \quad (3.16)$$

Keterangan:

t : $trace$ pada $event\ log$

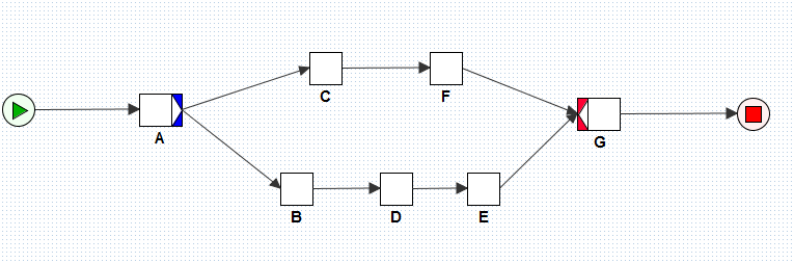
t_{ns}: bagian dari $trace$ yang dipotong

m : jumlah $case$ dalam $event\ log$

n : jumlah $complete\ trace$ dari model

Contohnya adalah sebagai berikut:

Suatu model XOR YAWL sebagai berikut:



Gambar 3.10 Model XOR YAWL

Dari model tersebut dibangkitkan *complete event log* sebagai berikut:

L2 = {(ACFG), (ABDEG)}

Kemudian dari *event log* ditambahkan *noise* menjadi: L2' = {(ACFG), (ABDEG), (AG)}

Dari *event log* L1" didapatkan perhitungan *heuristic miner algorithm* secagai berikut:

Matriks frekuensi dari *event log* L1" menghasilkan Tabel 3.17.

Tabel 3.17 Matriks Frekuensi L2'

Aktivitas	A	B	C	D	E	F	G
A	0	1	1	0	0	0	1
B	0	0	0	1	0	0	0
C	0	0	0	0	0	1	0
D	0	0	0	0	1	0	0
E	0	0	0	0	0	0	1
F	0	0	0	0	0	0	1
G	0	0	0	0	0	0	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.18.

Tabel 3.18 Matriks *Dependency Measure* L2'

Aktivitas	A	B	C	D	E	F	G
A	0	0.5	0.5	0	0	0	0.5
B	0	0	0	0.5	0	0	0
C	0	0	0	0	0	0.5	0
D	0	0	0	0	0.5	0	0
E	0	0	0	0	0	0	0.5
F	0	0	0	0	0	0	0.5
G	0	0	0	0	0	0	0

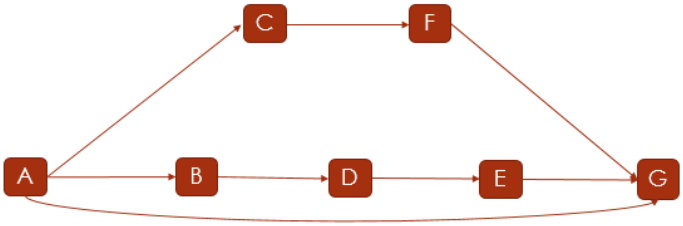
Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RTB = 0.5

PO = 1

DT = 0.5

Maka *dependency graph*-nya adalah sebagai berikut:



Gambar 3.11 *Dependency Graph* L2'

Dari *dependency graph*-nya sudah terjadi kesalahan *discovery* yaitu adanya hubungan antar aktivitas A dan G yang sebenarnya pada model aslinya tidak terdapat hubungan antara A dan G. Dengan ini maka proses *discovery* L2' gagal.

Hal ini mengakibatkan bagaimanapun nilai dari *Relative to the best threshold* maupun *dependency threshold* dan *positive observation threshold* AG akan tetap memiliki hubungan sehingga proses *discovery* salah.

Untuk mengatasi masalah ini maka paling tidak harus terdapat duplikasi untuk masing-masing *trace* pada *event log*. Dan jumlah frekuensi dari *noise* harus lebih kecil dari jumlah frekuensi dari *trace* utuh yang ada dalam *event log*.

3.5 Optimasi Waktu dan Biaya Proses Bisnis

Optimasi yang dilakukan dalam Tugas Akhir ini adalah optimasi antara tambahan biaya yang harus dibayarkan untuk memampatkan waktu eksekusi (*makespan*) dari proses bisnis. Kasus ini juga dapat disebut dengan *Time-cost Tradeoff Prolem*. Tugas akhir ini menggunakan metode *Crasing Project* yang ada pada CPM. *Crashing Project* sendiri merupakan suatu metode untuk mempersingkat lamanya waktu proyek dengan mengurangi waktu dari satu atau lebih aktivitas proyek yang penting menjadi kurang dari waktu normal aktivitas (Elmabrouk, 2011).

Terdapat dua nilai waktu yang ditunjukkan tiap aktifitas dalam suatu jaringan kerja saat terjadi c*rashing project* yaitu:

1. *Normal Duration*
   Waktu yang dibutuhkan untuk menyelesaikan suatu aktivitas atau kegiatan dengan sumber daya normal yang ada tanpa adanya biaya tambahan lain dalam sebuah proyek.
2. *Crash Duration*
   Waktu yang dibutuhkan suatu proyek dalam usahanya mempersingkat waktu yang durasinya lebih pendek dari normal duration.

Untuk membuat model matematika yang digunakan untuk optimasi diperlukan *time*. *Time slope* sendiri merupakan percepatan maksimum yang dapat dilakukan untuk *crashing project.*

$$X_i = T_{ni} - T_{ci} \quad (3.17)$$

Keterangan:

X_i : *Time slope* aktivitas ke – i dimana i ∈ {Aktivitas}

T_{ni} : durasi normal aktivitas ke – i dimana i ∈ {Aktivitas}

T_{ci} : *crash* duration aktivitas ke – i dimana i ∈ {Aktivitas}

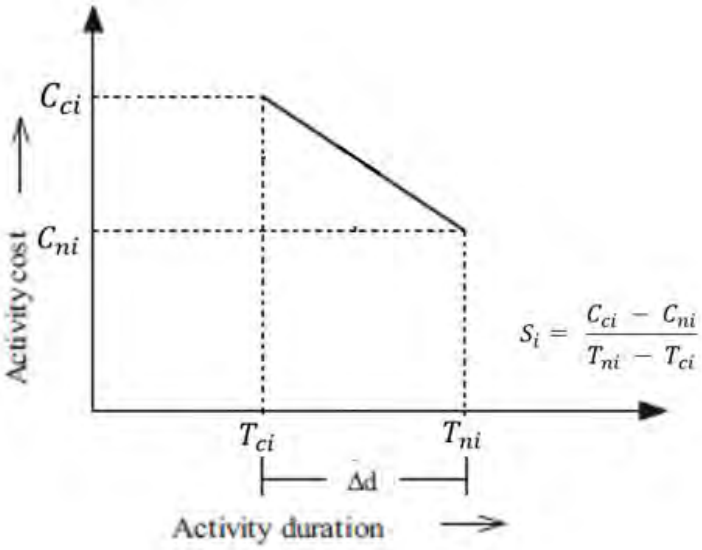
Crashing Project juga menyebabkan perubahan pada elemen biaya yaitu:

1. *Normal Cost*
   Biaya yang dikeluarkan dengan penyelesaian proyek dalam waktu normal. Perkiraan biaya ini adalah pada saat perencanaan dan penjadwalan bersamaan dengan penentuan waktu normal.
2. *Crash Cost*
   Biaya yang dikeluarkan dengan penyelesaian proyek dalam jangka waktu sebesar durasi *crash*-nya. Biaya setelah di *crashing* akan menjadi lebih besar dari biaya normal.

Perhitungan *Time-Cost TradeOff* diutamakan pada kegiatan-kegiatan yang memiliki nilai *cost slope* terendah. *Cost slope* merupakan perbandingan antara pertambahan biaya dengan percepatan waktu penyelesaian proyek. Persamaan 3.18 adalah sebagai berikut:

$$S_i = \frac{C_{ci} - C_{ni}}{T_{ni} - T_{ci}} \quad (3.18)$$

Keterangan:
S_i : *cost slope* aktivitas ke – i dimana i ∈ {Aktivitas}
C_{ni} : *cost* normal aktivitas ke – i dimana i ∈ {Aktivitas}
C_{ci} : *cost crash* aktivitas ke – i dimana i ∈ {Aktivitas}
T_{ni} : durasi normal aktivitas ke – i dimana i ∈ {Aktivitas}
T_{ci} : *crash duration* aktivitas ke- i dimana i ∈ {Aktivitas}



Gambar 3.12 ***Cost Slope***

3.5.1 Penentuan Durasi Normal dan *Cost* Normal

Penentuan durasi dengan rata-rata sudah dikembangkan oleh Van Der Aalst, tetapi yang yang dicari merupakan durasi dari eksekusi satu *trace* dalam *event log* (van der Aalst, Schonenberg, & Song, Time Prediction Based on Process Mining, 2011). Oleh karena itu dalam Tugas Akhir ini dikembangkan metode untuk menentukan durasi normal dan biaya normal dari setiap aktivitas dengan merata-rata setiap durasi dan biaya dari aktivitas pada setiap *case* yang ada pada *event log*. Durasi eksekusi diperoleh dengan mengurangkan *output* dari aktivitas dengan aktivitas (aktivitas yaitu A memiliki aktivitas B sebagai *output* maka eksekusi durasi A adalah waktu eksekusi B mengurangkan dengan waktu pelaksanaan A).

$$if\ activity_{output} = true\ then$$
$$ed = et_{activity_{output}} - et_{activity} \quad (3.19)$$

Keterangan:
ed : durasi eksekusi per *case*
$et_{activity_{output}}$: waktu eksekusi *output* dari aktivitas
$et_{activity}$: waktu eksekusi aktivitas

Setelah mendapatkan semua eksekusi durasi setiap aktivitas pada semua kasus di *event log* maka rata-rata pelaksanaan durasi per aktivitas dihitung untuk menentukan durasi normal.

$$T_{ni} = \frac{\sum_{i=1}^{h} ed_i}{h} \quad (3.20)$$

Setelah menghitung durasi dari menghitung biaya normal setiap aktivitas. *Event log* berisi biaya pelaksanaan setiap aktivitas dalam setiap *case,* oleh karena itu untuk mendapatkan biaya normal setiap aktivitas digunakan perhitungan biaya rata-rata aktivitas dalam setiap kasus.

$$C_{ni} = \frac{\sum_{i=1}^{h} cost_i}{h} \quad (3.21)$$

Keterangan:
T_{ni} : durasi normal setiap aktivitas ke-i dimana i ∈ {Aktivitas}
ed_q : waktu eksekusi aktivitas pada *case* ke-q dimana q = 1, 2, 3, … w
C_{ni} : *cost* normal setiap aktivitas ke-i dimana i ∈ {Aktivitas}
$cost_q$: biaya aktivitas pada *case* ke-i ke-q dimana q = 1, 2, … w
h : banyaknya aktivitas dieksekusi dalam *event log* (banyaknya *case* yang mengandung aktivitas)

3.5.2 Penentuan *Crash Time* dan *Crash Cost*

Dalam menentukan *crash time* dan *crash cost* dari setiap aktivitas diambil dengan mengurangi normal durasi dengan standar deviasi waktu eksekusi masing-masing aktivitas untuk *crash time* dan menambahkan biaya normal dengan standar deviasi biaya dari masih-masing aktivitas untuk *crash cost*.

3.5.3 Hubungan Cross-Organizational dengan Optimasi

Pola pada *cross-organizational business process* mempengaruhi optimasi jika terdapat salah satu organisasi yang ada pada pola tidak melakukan optimasi. Salah satu organisasi tidak dapat melakukan optimasi dapat dikarenakan banyak hal bisa jadi dikarenakan organisasi tersebuat memiliki *workload* yang sudah tidak dapat diganggu ataupun karena organisasi utama tidak mampu mengoptimasi seluruh organisasi yang diajak kerjasama sehingga mengharuskan untuk memilih organisasi yang tidak dioptimasi dan yang dioptimasi. Dari pola hubungan *cross-organizational business process* dapat dibedakan lagi dari bentuk polanya yaitu kelompok pola yang sekuensial dan paralel. Yang termasuk dalam pola sekuensial adalah pola *cross-organizational business process:*

- *Capacity sharing*
- *Chain execution*
- *Message exchange*

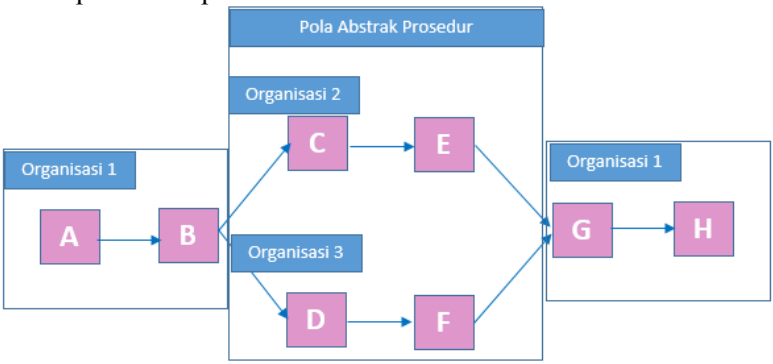
Yang termasuk dalam pola paralel adalah pola *cross-organizational business process:*

- *Case transfer*
- *Loosely couple*
- *Astrack procedure*

Untuk *cross-organizational business process* yang sekuensial, jika salah satu organisasi tidak dioptimasi maka hanya berpengaruh terhadap organisasi itu sendiri dan tidak berpengaruh terhadap *crash time* yang dapat dilakukan oleh organisasi lain yang terdapat pada pola tersebut.

Sedangkan untuk pola paralel jika salah satu organisasi yang terdapat pada pola tersebut tidak dioptimasi maka berpengaruh terhadap organisasi lain yang ada pada pola tersebut. Contohnya pada *cross-organizational business process* dengan pola *abstract procedure* memiliki 3 organisasi yang ada dalam proses bisnis

dengan satu organisasi utama dan 2 organisasi yang masuk dalam pola *abstract procedure* model dari *cross-organizational business process* dapat dilihat pada Gambar 3.13.



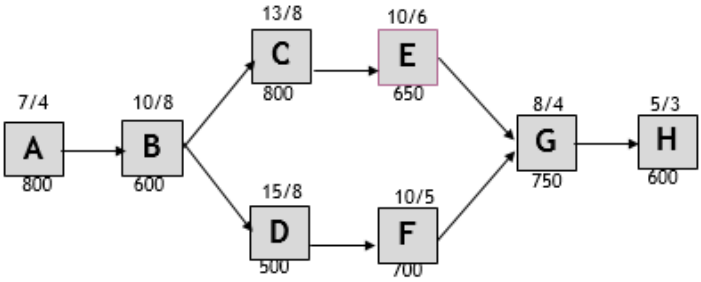
Gambar 3.13 Model *Pola Abstract Procedure*

Model pada Gambar 3.13 memiliki data untuk optimasi seperti pada Tabel 3.19.

Tabel 3.19 Data Optimasi Gambar 3.13

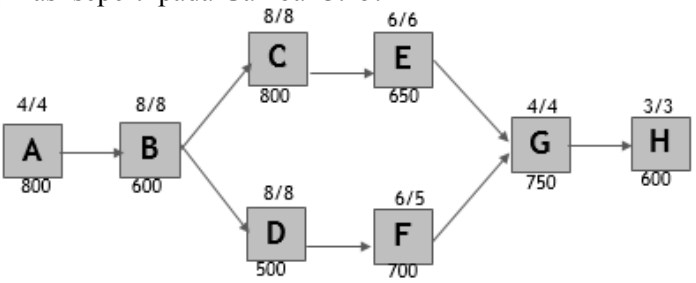
Aktivitas	Normal Dur	Crashed dur	Time Slope	Cost Slope
A	7	4	3	800
B	10	8	2	600
C	13	8	5	800
D	15	8	7	500
E	10	6	4	650
F	10	5	5	700
G	8	4	4	750
H	5	3	2	600

Jika model pada Gambar 3.13 digambarkan dengan nilai data optimasinya maka menjadi seperti pada Gambar 3.14.



Gambar 3.14 Model Dengan Nilai Data Optimasi

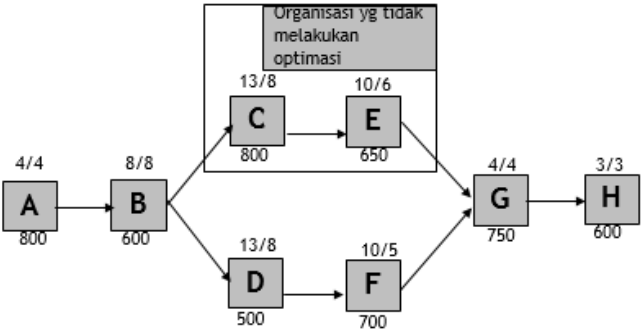
Dengan CPM maka didapatkan *critical path* A-B-D-F-G-H dengan *makespan* 55. Jika semua organisasi dapat dioptimasi maka hasil optimasi seperti pada Gambar 3.15.



Gambar 3.15 Model Optimasi dengan Semua Organisasi Dioptimasi

Maka didapatkan hasil durasi setelah dioptimasi adalah 33. Jika organisasi 2 dalam model Gambar 3.13 merupakan organisasi yang tidak dapat dioptimasi. Maka hasil optimasi model Gambar 3.13 akam menjadi seperti pada Gambar 3.16 dengan hasil optimasi 42. Hal ini dikarenakan aktivitas pada organisasi 3 yaitu D dan F tidak dapat dioptimasi secara optimal, karena organisasi 2 tidak dioptimasi maka batas untuk optimasi organisasi 3 adalah durasi yang dimiliki oleh organisasi 2. Dapat dilihat pada aktivitas D yang seharusnya dapat dioptimasi menjadi 8 jika semua organisasi dioptimasi tapi karena organisasi 2 tidak dioptimasi maka hanya bisa dioptimasi menjadi 13, begitu pula organisasi F yang seharusnya dapat dioptimasi menjadi 6 karena organisasi 2 tidak

dioptimasi maka F tetap memiliki nilai durasi 10. Hal ini menunjukkan bahwa untuk pola paralel *cross-organizational business process* jika salah satu organisasi tidak dioptimasi maka mempengaruhi organisasi lain yang ada pada pola tersebut.



Gambar 3.16 Model Optimasi dengan Organisasi 2 Tidak Dioptimasi

3.5.4 Pemodelan Fungsi Linear untuk Optimasi

Dalam pemodelan fungsi matematika untuk model optimasi ini menggunakan fungsi objektif dan beberapa fungsi batasan. Fungsi objektf untuk pemodelan matematika disini menggunakan fungsi objektif maksimum dua fungsi objektif yang pertama adalah fungsi untuk mencari waktu durasi *crash* proyek yang paling minimum. Untuk mencari durasi *crash* proses bisnis yang paling minimum maka fungsi objektif yang digunakanan adalah maksimal dari biaya *crashing* proses bisnis.

- Fungsi Objektif 1

$$MAXIMIZE \sum_{i=1}^{j} S_i X_i \quad (3.22)$$

Keterangan:

j : banyaknya aktivitas pada proses bisnis

S_i : *cost slope* aktivitas ke – i dimana i ∈ {Aktivitas}

X_i : *time slope* yang terjadi untuk penyelesaian aktivitas ke – i dimana i ∈ {Aktivitas} (variable keputusan)

Sedangkan untuk fungsi batasannya adalah sebagai berikut:

- Fungsi batasan
- *Maximum reduction constrain*

$$X_i \leq T_{ni} - T_{ci} \; for \; all \; i \tag{3.23}$$

2. *Non-negative constraint*

$$X_i \geq 0 \; for \; all \; i \tag{3.24}$$

$$Y_i \geq 0 \; for \; all \; i \tag{3.25}$$

Keterangan:

Y_i : variabel *start time* aktivitas ke – i dimana i ∈ {Aktivitas}

3. *Start time constraint*

$$Y_i - Y_{pi} + X_{pi} \geq K \; for \; all \; i \tag{3.26}$$

Keterangan:

Y_{pi} : variabel *start time predecessor* aktivitas ke – i dimana i ∈ {Aktivitas}

X_{pi} : variabel *time slope predecessor* aktivitas ke – i dimana i ∈ {Aktivitas}

K : nilai *normal time predecessor* aktivitas ke – i dimana i ∈ {Aktivitas}

4. *Project duration constraint*

$$Y'_{Finish} - D \leq 0 \tag{3.27}$$

Keterangan:

D : maksimum durasi pada critical path pada proses bisnis

Y'_{Finish}: durasi proses bisnis paling minimum setelah dimampatkan

Kemudian untuk mencari biaya tambahan yang paling minimum untuk durasi proses bisnis yang paling minimum dilakukan dengan merubah fungsi objektif dan *project duration constraint,* sedangkan untuk *maximum reduction constrain, non-negative constraint,* dan *start time constraint* tetap sama.

- Fungsi Objektif 2

$$MINIMIZE \sum_{i=1}^{j} S_i X_i \tag{3.28}$$

Keterangan:

j : banyaknya aktivitas pada proses bisnis

S_i : *cost slope* aktivitas ke – i dimana i ∈ {Aktivitas}

X_i : *time slope* yang terjadi untuk penyelesaian aktivit ke – i dimana i ∈ {Aktivitas} (variable keputusan)

- *Project duration constraint*

$$Y_{Finish} - Y'_{Finish} \leq 0 \quad (3.29)$$

Keterangan:

D : maksimum durasi pada critical path pada proses bisnis

Y_{Finish} : durasi hasil akhir durasi *crashing* proses bisnis

Y'_{Finish} : durasi proses bisnis paling minimum setelah dimampatkan

3.7 Contoh Optimasi Waktu dan Biaya

Tabel 3.20 menunjukkan *single timestamps event log* yang akan digunakan untuk menghitung data optimasi. *Single timestamps* adalah adanya satu waktu untuk setiap aktivitas pada *event log*.

Tabel 3.20 Event Log

Case ID	Activity	TimeStamp	Originator	Cost
300.PO.IMP.201406.0024	A	6/20/2014 8:32	AN	400
300.PO.IMP.201406.0024	D	6/20/2014 13:42	AN	1500
300.PO.IMP.201406.0024	B	6/20/2014 23:41	AN	1000
300.PO.IMP.201406.0024	E	6/21/2014 8:16	AN	2000
300.PO.IMP.201406.0024	C	6/21/2014 10:46	AN	1200
300.PO.IMP.201406.0015	A	6/21/2014 16:57	AN	1000
300.PO.IMP.201406.0015	B	6/21/2014 18:09	AN	400
300.PO.IMP.201406.0015	C	6/22/2014 5:15	AN	500
300.PO.IMP.201408.0021	A	6/22/2014 10:51	AN	50
300.PO.IMP.201408.0021	B	6/22/2014 23:42	AN	1000
300.PO.IMP.201408.0021	D	6/22/2014 23:42	AN	1000
300.PO.IMP.201408.0021	E	6/23/2014 4:48	AN	2000
300.PO.IMP.201408.0021	C	6/23/2014 16:44	AN	1000
300.PO.IMP.201406.0025	A	6/23/2014 23:26	AN	400
300.PO.IMP.201406.0025	D	6/24/2014 4:19	AN	1000
300.PO.IMP.201406.0025	E	6/24/2014 7:48	AN	500
300.PO.IMP.201406.0025	B	6/25/2014 1:08	AN	500
300.PO.IMP.201406.0025	C	6/25/2014 3:02	AN	500
300.PO.IMP.201408.0026	A	6/25/2014 5:11	AN	500
300.PO.IMP.201408.0026	B	6/25/2014 12:45	AN	1000
300.PO.IMP.201408.0026	D	6/25/2014 12:45	AN	1000
300.PO.IMP.201408.0026	E	6/26/2014 0:12	AN	2000
300.PO.IMP.201408.0026	D	6/26/2014 4:48	AN	1000
300.PO.IMP.201408.0026	E	6/26/2014 15:16	AN	100
300.PO.IMP.201408.0026	C	6/26/2014 22:49	AN	1000

Dari Tabel 3.20 didapatkan dengan cara yang telah disebutkan pada 3.5.1 dan 3.5.2 maka didapatkan data seperti pada Tabel 3.21.

Tabel 3.21 Tabel data Optimasi

Activity	Normal Duration	Crassh Duration	Normal Cost	Cash Cost	Time Slope	Cost Slope
A	1.66	0.42	470	812.05	1.24	275.85
D	1.97	0.73	1100	1323.61	1.24	180.33
B	2.9	0.65	680	974.96	2.25	131.09
E	2.13	1.01	1320	2261.81	1.12	840.9
C	1.89	0.95	840	1160.94	0.94	341.43

Model matematika untuk mencari *crashing* durasi proses bisnis paling minimum menggunakan Persamaan 3.22 sampai Persamaan 3.29:

- Fungsi objektif 1

  $MAXIMUM = (275.85*X_A + 180.33*X_D + 131.09*X_B + 804.9*X_E + 341.43*X_C);$

- Fungsi batasan

1. Maximum reduction constrain

   $X_A<=1.24;\ X_D<=1.24;\ X_B<=2.25;\ X_E<=1.12;\ X_C<=0.94;$

2. Non-negative constraint

   $X_A>=0;\ X_D>=0;\ X_B>=0;\ X_E>=0;\ X_C>=0;$
   $Y_A>=0;\ Y_D>=0;\ Y_B>=0;\ Y_E>=0;\ Y_C>=0;$

3. Start time constraint

   $Y_D - Y_A + X_A >= 1.66;\ Y_B - Y_A + X_A >= 1.66;\ Y_E - Y_D + X_D >= 1.97;\ Y_C - Y_E + X_E >= 2.13;\ Y_C - Y_B + X_B >= 2.9;\ Y_{FINISH} - Y_C + X_C >= 1.89;$

4. Project duration constraint

   $Y'_{FINISH} - 7.65 <= 0$

Diketahui *crashing* durasi proses bisnis paling minimum adalah 3.11.

Model matematika untuk mencari biaya tambahan minimum:

- Fungsi objektif 2

  $MINIMUM = (275.85*X_A + 180.33*X_D + 131.09*X_B + 804.9*X_E + 341.43*X_C);$

- Fungsi batasan

2. Maximum reduction constrain
   $XA<=1.24;\ XD<=1.24;\ XB<=2.25;\ XE<=1.12;\ XC<=0.94;$
5. Non-negative constraint
   $X_A>=0;\ X_D>=0;\ X_B>=0;\ X_E>=0;\ X_C>=0;$
   $Y_A>=0;\ Y_D>=0;\ Y_B>=0;\ Y_E>=0;\ Y_C>=0;$
6. Start time constraint
   $Y_D - Y_A + X_A >= 1.66;\ Y_B - Y_A + X_A >= 1.66;\ Y_E - Y_D + X_D >= 1.97;\ Y_C - Y_E + X_E >= 2.13;\ Y_C - Y_B + X_B >= 2.9;\ Y_{FINISH} - Y_C + X_C >= 1.89;$
7. Project duration constraint
   $Y_{FINISH} - 3.11 <= 0$

Dari perhitungan menggunakan metode *simplex linear programming* didapatkan biaya minimum tambahan sebesar 1980.48 dan durasi project menjadi 3.1 jam dari yang sebelumnya 5.5 jam.

BAB IV
ANALISIS DAN PERANCANGAN SISTEM

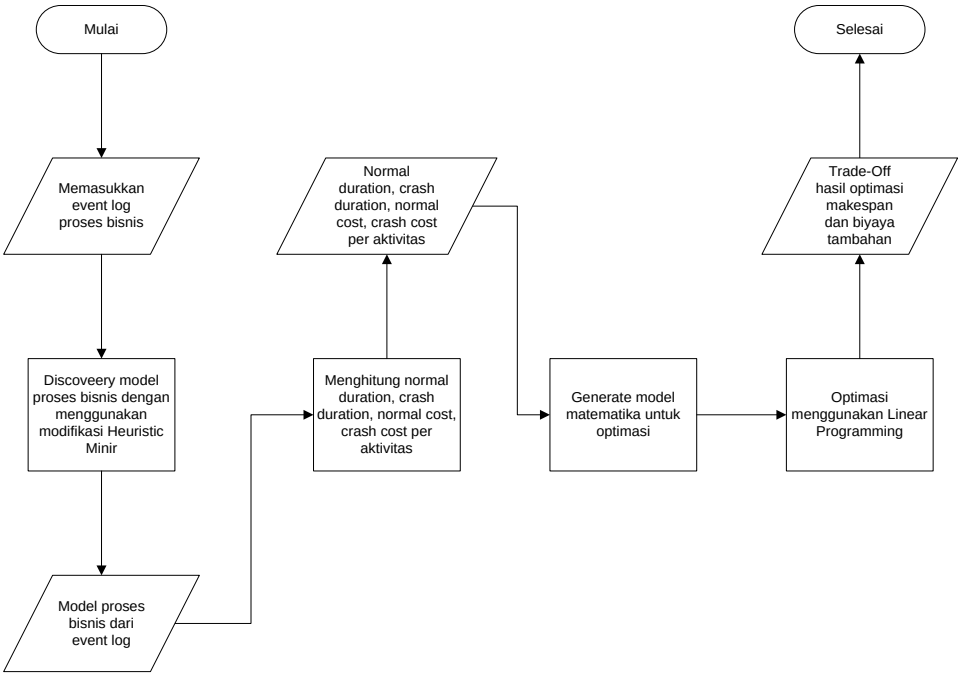
Bab ini membahas tahap analisis permasalahan dan perancangan Tugas Akhir. Analisis permasalahan membahas permasalahan yang yang diangkat dalam pengerjaan Tugas Akhir. Solusi yang ditawarkan oleh penulis juga dicantumkan pada tahap permasalahan analisis ini. Analisis kebutuhan mencantumkan kebutuhan-kebutuhan yang diperlukan perangkat lunak. Selanjutnya dibahas mengenai perancangan sistem yang dibuat. Perancangan direpresentasikan dengan diagram UML (*Unified Modelling Language*).

4.1 Analisis

Tahap analisis dibagi menjadi beberapa bagian antara lain cakupan permasalahan, deskripsi umum sistem, kasus penggunaan sistem, dan kebutuhan perangkat lunak.

4.2 Deskripsi Umum Sistem

Perangkat lunak yang akan dibangun dapat menghasilkan model dari proses bisnis, durasi normal, durasi *crash*, biaya normal dan biaya *crash* per aktivitas yang ada dalam proses bisnis, dan biaya tambahan dan *makespan* yang paling optimum untuk melakukan percepatan durasi proses bisnis. Proses pemodelan proses bisnis tersebut membutuhkan data atau *event log* proses bisnis yang sudah ada dan berjalan. Dari *event log* yang digunakan sebagai masukan dilakukan serangkaian proses hingga menghasilkan keluaran yang diharapkan. Gambar 4.1 merupakan alur pemrosesan dan bentuk arsitektur perangkat secara sederhana.



Gambar 4.1. Alur Sistem

Pada fase *discovery*, pengguna akan memilih *file event log* yang akan di-*discovery* bentuk proses modelnya. Setelah pengguna memilih *file event log* yang di-*discovery* bentuk proses modelnya hal yang paling awal dilakukan adalah melakukan proses pada *file event log* yang dipilih dengan menggunakan teknik *reading from* Excel *file* menggunakan Bytescout Spreadsheet. Setiap elemen-elemen *case id* pada *file* Excel *event log* diidentifikasi *trace-trace* yang terdapat pada *event log* dan juga frekuensi *trace* yang ada pada event log. Setelah mendapatkan *trace* dan masing-masing frekuensinya, kemudian menggunakan modifikasi dari *heuristic miner* dicari hubungan dari masing-masing aktivitas dari *event log,* sehingga dapat mengasilkan model dari *event log* yang dipilih oleh pengguna.

Setelah mendapatkan model dari *event log* yang dimasukkan, selanjutnya memasuki tahap perhitungan data optimasi berupa durasi normal, durasi *crash,* biaya normal, dan biaya *crash.* Seperti yang telah dijelaskan pada 3.5.1 dan 3.5.2 penentuan data optimasi dilakukan dengan rata-rata dan standar deviasi. Sesuai dengan *time prediction* yang dikembangkan oleh Van Der Aalst rata-rata yang diperoleh harus didapatkan dari eksekusi antar waktu aktivitas yang saling berhubungan oleh karena itu sebelum mencari data optimasi diharuskan melalui proses *discovery* terlebih dahulu.

Selanjutnya adalah fase optimasi yang akan mendapatkan nilai optimum dari percepatan *makespan* dengan biaya tambahan yang paling minimum dengan menggunakan *linear programming* sesuai yang dijelaskan pada 3.5.3.

4.3 Spesifikasi Kebutuhan Perangkat Lunak

Bagian ini berisi semua kebutuhan perangkat lunak yang diuraikan secara rinci dalam bentuk diagram kasus, diagram urutan, dan diagram aktivitas. Masing-masing diagram menjelaskan perilaku atau sifat dari sistem ini. Kebutuhan perangkat lunak dalam sistem ini mencakup kebutuhan fungsional saja. Pada bab ini juga dijelaskan tentang spesifikasi terperinci pada masing-masing kebutuhan fungsional. Rincian spesifikasi dari kasus penggunaan disajikan dalam bentuk tabel.

4.4 Kebutuhan Fungsional

Kebutuhan fungsional berisi kebutuhan utama yang harus dipenuhi oleh sistem agar dapat bekerja dengan baik. Kebutuhan fungsional mendefinisikan layanan yang harus disediakan oleh sistem, bagaimana reaksi terhadap masukan, dan apa yang harus dilakukan sistem pada situasi khusus. Daftar kebutuhan fungsional dapat dilihat pada Tabel 4.1.

Tabel 4.1 Daftar Kebutuhan Fungsional Perangkat Lunak

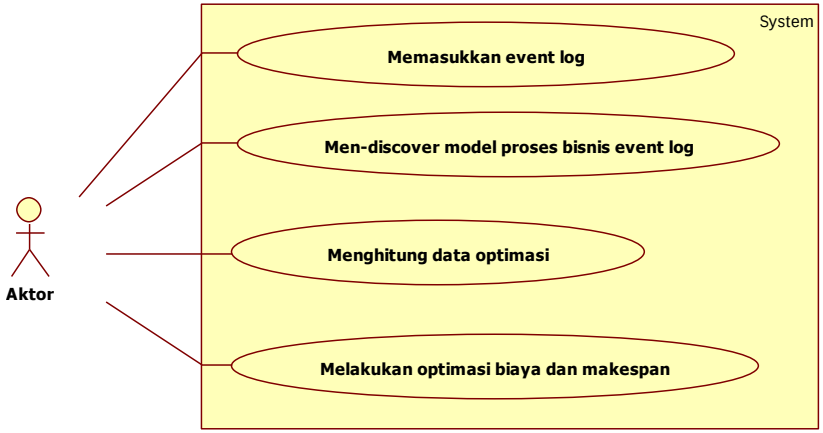
Kode Kebutuhan	Kebutuhan Fungsional	Deskripsi
TA-F0001	Memasukkan event log	Pengguna dapat memasukkan *event log* dalam bentuk Excel
TA-F0002	Men-*disover* bisnis proses model	Pengguna dapat men-*disover* bisnis proses model sehingga mengetahui bentuk *graph* bisnis proses dan hubungan antar aktivitas.
TA-F0003	Menghitung data optimasi	Pengguna dapat menghitung data optimasi berupa durasi normal, durasi *crash*, biaya normal dan biaya *crash* per-aktivitas yang ada dalam proses bisnis
TA-F0004	Optimasi biaya tambahan dan percpatan *makespan*	Pengguna dapat mengentahui hasil optimasi yaitu berupa biaya tambahan dan *makespan* percepatan yang minimum dari proses bisnis.

4.5 Aktor

Aktor mendefinisikan entitas-entitas yang terlibat dan berinteraksi langsung dengan sistem. Entitas ini bisa berupa manusia maupun sistem atau perangkat lunak yang lain. Penulis mendefinisikan aktor untuk sistem ini yaitu pengguna dari aplikasi yang dibangung oleh penulis.

4.6 Kasus Penggunaan

Kasus-kasus penggunaan dalam sistem ini akan dijelaskan secara rinci pada subbab ini. Kasus penggunaan secara umum akan digambarkan oleh salah satu model UML, yaitu diagram kasus penggunaan. Rincian kasus penggunaan berisi spesifikasi kasus penggunaan, diagram aktivitas, dan diagram urutan untuk masing-masing kasus penggunaan. Diagram kasus penggunaan dapat dilihat pada Gambar 4.2. Daftar kode diagram kasus penggunaan sistem dapat dilihat pada Tabel 4.2.



Gambar 4.2 Diagram Kasus Penggunaan Sistem

Tabel 4.2 Daftar Kode Diagram Kasus Penggunaan

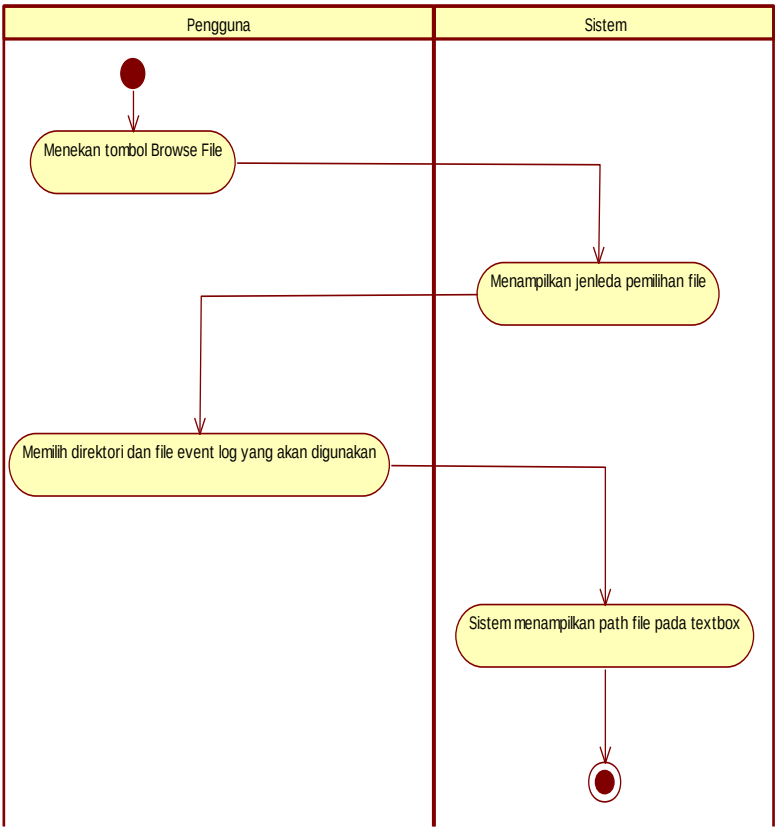
Kode Kasus Penggunaan	Nama
TA-UC0001	Memasukkan data event log
TA-UC0002	Men-*discover* model proses bisnis
TA-UC0003	Menghitung data optimasi
TA-UC0004	Melakukan optimasi biaya dan *makespan*

4.6.1 Memasukkan dan Membaca Data Event Log

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk memasukkan data yang berupa *event log* proses bisnis dalam notasi Excel. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.3 dan diagram aktivitas dari kasus penggunaan ini bisa dilihat pada Gambar 4.3.

Tabel 4.3 Spesifikasi Kasus Penggunaan Memasukkan dan Membaca *Data Event Log*

Nama	Menampilkan similarity metric
Kode	TA-UC0001
Deskripsi	Memasukkan data event log.
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Browse File* dan memilih *file* Excel *event log* yang akan digunakan
Aktor	Pengguna
Kondisi Awal	*Event log* yang akan diproses belum ada
Aliran:	
- **Kejadian Normal**	1. Pengguna menekan tombol *browse* 2. Pengguna memilih *file* Excel *event log* yang terdapat dalam direktori 3. Sistem mengambil *path* direktori dan mengambil *file* Excel *event log* ke dalam sistem
- **Kejadian Alternatif**	Tidak ada
Kondisi Akhir	Sistem menampilkan *path file* Excel *event log* dalam *textbox* pada sistem
Kebutuhan Khusus	Tidak ada



Gambar 4.3 Diagram Aktivitas Memasukkan dan Membaca Data *Event Log*

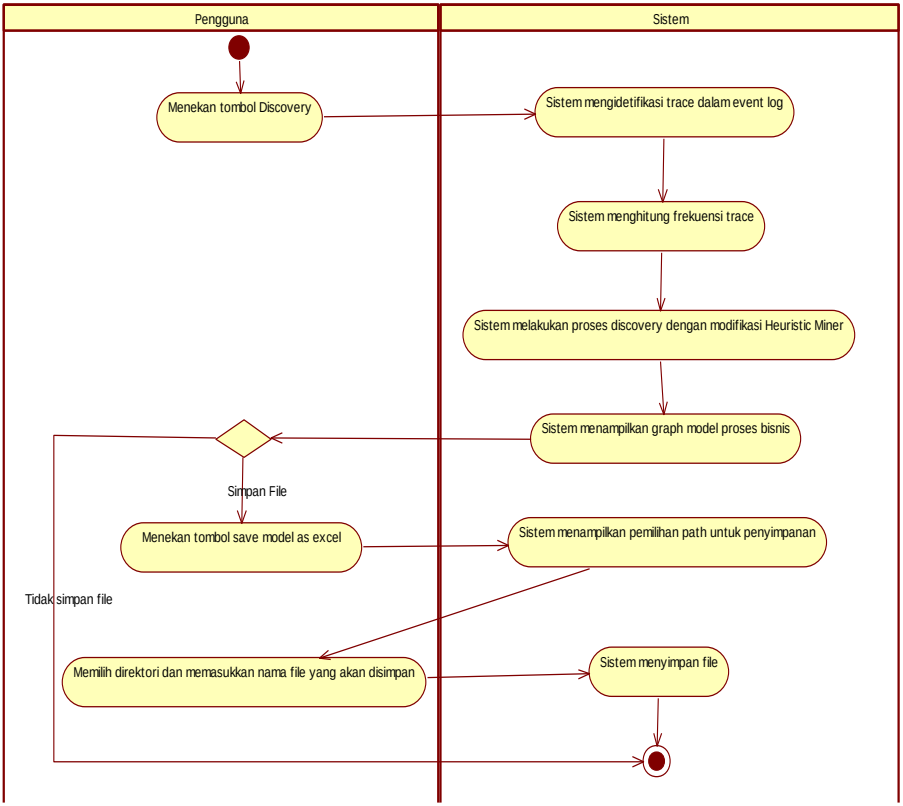
4.6.2 Men-*discover* Model Proses Bisnis

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk melakukan proses *discovery* pada *event log* yang telah dimasukkan oleh pengguna. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.4 dan diagram aktivitas dan diagram urutan dari kasus penggunaan ini bisa dilihat pada Gambar 4.4.

Tabel 4.4 Spesifikasi Kasus Penggunaan Men-*discover* Model Proses Bisnis

Nama	Men-*discover* model proses bisnis
Kode	TA-UC0002
Deskripsi	Pengguna dapat men-*disover* bisnis proses model sehingga mengetahui bentuk *graph* bisnis proses dan hubungan antar aktivitas.
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Discover* dan kemudian untuk menyimpan hasil dalam bentuk Excel pengguna menekan tombol *Save Model As* Excel
Aktor	Pengguna
Kondisi Awal	*Path file* Excel *event log* dalam *textbox* sudah terisi
Aliran:	
- **Kejadian Normal**	1. Pengguna menekan tombol *Discover.* 2. Sistem mengolah *file event log* dan menampilkan bentuk model proses bisnis dari *event log.* 3. Pengguna menekan tombol *Save Model As* Excel. 4. Pengguna memilih direktori penyimpanan dan memasukkan nama *file.* 5. Sistem mengolah data sehingga berubah dalam bentuk tabel Excel dan menyimpannya sesuai dengan nama dan direktori yang telah dipilih pengguna.
- **Kejadian Alternatif**	Pengguna tidak menyimpan hasil proses *discovery.* Maka sistem tidak akan menyimpan hasil proses *discovery.*

Kondisi Akhir	Sistem menampilkan *graph* model proses bisnis dan menyimpannya dalam bentuk tabel di Excel jika pengguna ingin menyimpannya.
Kebutuhan Khusus	Tidak ada



Gambar 4.4 Diagram Aktivitas Men-*discover* Model Proses Bisnis

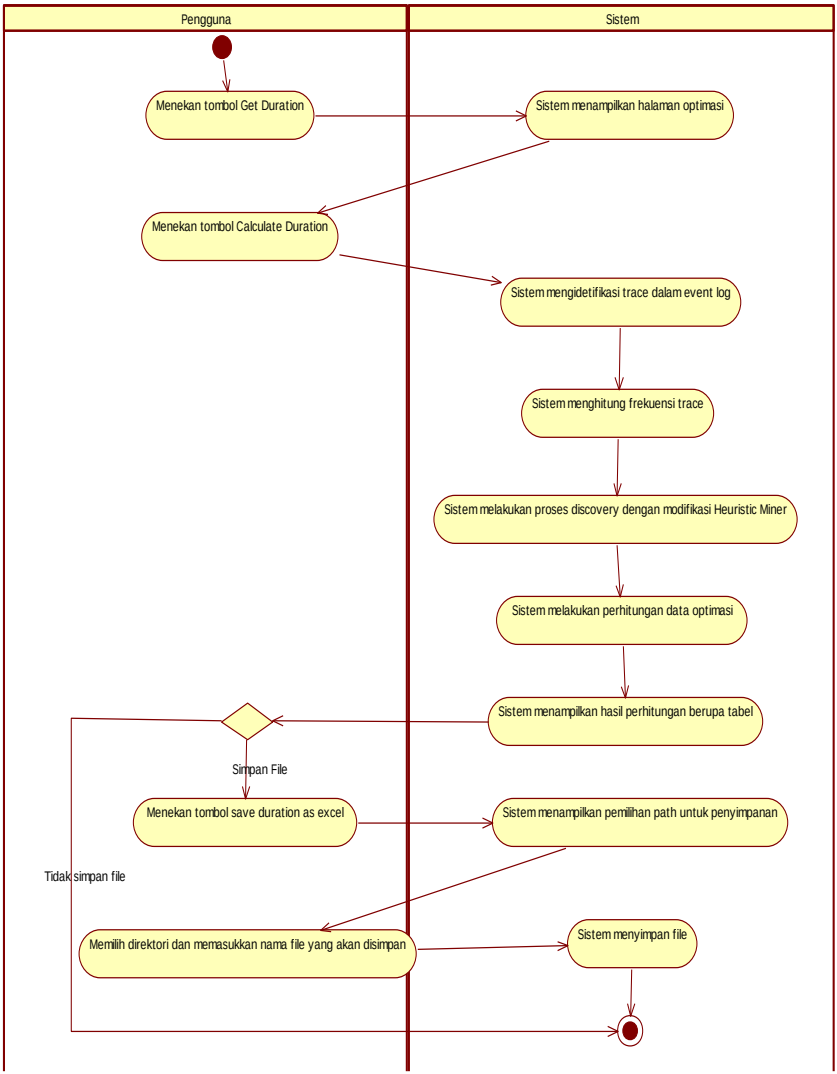
4.6.3 Menghitung Data Optimasi

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk mulai melakukan perhitungan data optimasi dari *event log* yang telah dipilih. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.5. Diagram aktivitas dari kasus penggunaan ini bisa dilihat pada Gambar 4.5.

Tabel 4.5. Spesifikasi Kasus Penggunaan Menghitung Data Optimasi

Nama	Menghitung data optimasi
Kode	TA-UC0003
Deskripsi	Pengguna dapat menghitung data optimasi berupa durasi normal, durasi *crash*, biaya normal dan biaya *crash* per aktivitas yang ada dalam proses bisnis
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Get Duration* untuk pindah ke halaman optimasi dan kemudian untuk menghitung data optimasi pengguna menekan tombol *Calculate Duration*, kemudian untuk menyimoan hasil dalam bentuk Excel pengguna menekan tombol *Save Duration As* Excel
Aktor	Pengguna
Kondisi Awal	*Path file* Excel *event log* dalam *textbox* sudah terisi
Aliran:	
- Kejadian Normal	1. Pengguna menekan tombol *Get Duration.* 2. Sistem menampilkan halaman optimasi. 3. Pengguna menekan tombol *Calculate Duration.* 4. Sistem mengolah *file event log* dengan fungsi *discovery* dan fungsi perhitungan data optimasi. 5. Sistem menampilkan hasil perhitungan data optimasi dalam bentuk tabel. 6. Pengguna menekan tombol *Save Duration As* Excel. 7. Pengguna memilih direktori penyimpanan dan memasukkan nama *file.*

	8. Sistem menyimpan hasil perhitungan sesuai dengan nama dan direktori yang telah dipilih pengguna.
- **Kejadian Alternatif**	Pengguna tidak menyimpan hasil perhitungan. Maka sistem tidak akan menyimpan hasil perhitungan.
Kondisi Akhir	Sistem menampilkan data optimasi dalam bentuk table
Kebutuhan Khusus	Tidak ada



Gambar 4.5. Diagram Aktivitas Menghitung Data Optimasi

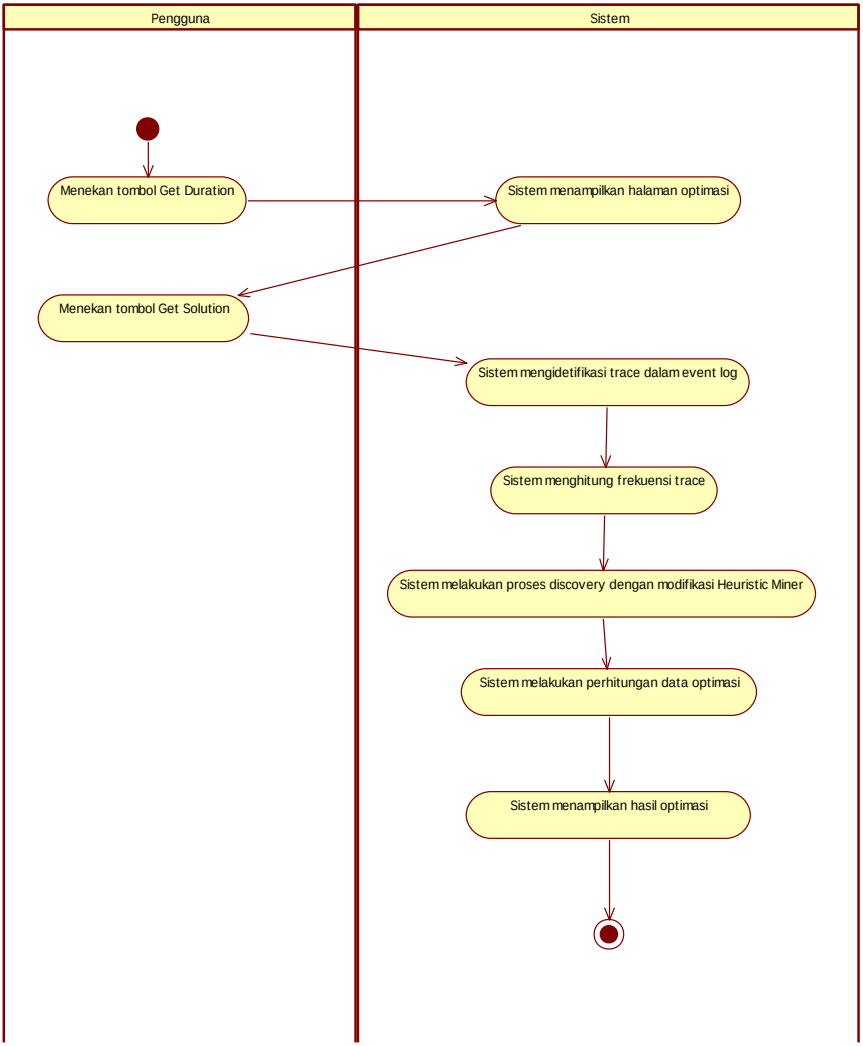
4.6.4 Melakukan Optimasi Biaya dan *Makespan*

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk mulai melakukan optimasi biaya dan *makespan* dari event log yang telah dipilih. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.6. Diagram aktivitas urutan dari kasus penggunaan ini bisa dilihat pada Gambar 4.6.

Tabel 4.6. Spesifikasi Kasus Penggunaan Melakukan Optimasi Biaya dan *Makespan*

Nama	Melakukan optimasi biaya dan *makespan*
Kode	TA-UC0004
Deskripsi	Pengguna dapat mengentahui hasil optimasi yaitu berupa biaya tambahan dan *makespan* percepatan yang minimum dari proses bisnis.
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Get Duration* untuk pindah ke halaman optimasi dan kemudian untuk menghitung data optimasi pengguna menekan tombol *Get Solution*.
Aktor	Pengguna
Kondisi Awal	*Path file* Excel *event log* dalam *textbox* sudah terisi
Aliran:	
- **Kejadian Normal**	1. Pengguna menekan tombol *Get Duration.* 2. Sistem menampilkan halaman optimasi. 3. Pengguna menekan tombol *Get Solution.* 4. Sistem mengolah *file event log* dengan fungsi fungsi perhitungan data optimasi, *generate* model matematika, dan fungsi *solve* model matematika. 5. Sistem menampilkan hasil optimasi berupa biaya tambahan minimum, aktivitas yang dimampatkan, dan durasi proyek yang telah dimampatkan.
- **Kejadian Alternatif**	1. Pengguna dapat memilih organisasi yang tidak dioptimasi pada *dropdown* dengan label *Organization which is not optimized.*

	2. Pengguna dapat menyimpan hasil optimasi pada Excel.
Kondisi Akhir	Sistem menampilkan hasil optimasi berupa biaya tambahan minimum dan durasi yang telah dimampatkan. Hasil optimasi dapat tersimpan dalam Excel.
Kebutuhan Khusus	Tidak ada



Gambar 4.6. Diagram Aktivitas Melakukan Optimasi Biaya dan *Makespan*

4.7 Perancangan Sistem

Penjelasan tahap perancangan perangkat lunak dibagi menjadi beberapa bagian yaitu perancangan proses analisis dan perancangan antarmuka.

4.8 Perancangan Antarmuka Pengguna

Bagian ini membahas mengenai perancangan antarmuka pada sistem. Terdapat dua kelas *view* yang masing-masing kelas memiliki fungsionalnya masing-masing. Kedua kelas tersebut merupakan form dan membutuhkan interaksi dari pengguna.

4.8.1 Halaman *Process Discovery*

Gambar 4.7 Rancangan Halaman *Process Discovery*

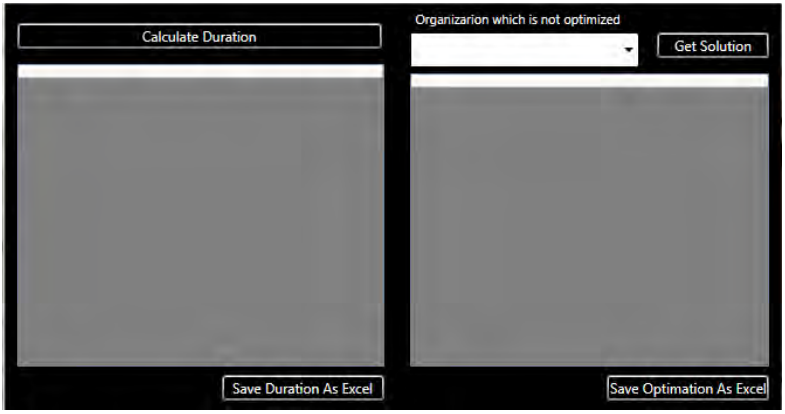
Halaman ini merupakan halam pertama yang muncul ketika aplikasi dijalankan. Pada halaman ini terdapat beberapa parameter yang harus diisi dan dibutuhkan untuk pemrosesan selanjutnya. Parameter tersebut antara lain *path* direktori tempat *event log* berada. Setelah mendapat path sistem akan membuka *file event log* yang telah dipilih. Setelah *file* terbuka maka sistem akan melakukan proses *discovery* atau bisa langsung melakukan optimasi tergantung kebutuhan dari pengguna. Rancangan halaman

utama ini dapat dilihat pada Gambar 4.7. Penjelasan mengenai atribut-atribut yang terdapat pada halaman ini bisa dilihat pada Tabel 4.7.

Tabel 4.7 Spesifikasi Atribut Antarmuka *Process Discovery*

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Jenis Masukan / Keluaran
1	Tombol *Browse File*	ActionButton	Mencari *file* Excel yang akan dikonversi.	*Action*
2	Teks Box	TextBox	Menampilkan *path* dari *file event log* yang telah dipilih	*String*
3	Tombol *Discovery*	ActionButton	Memproses *file event log* untuk mendapatkan model dari proses bisnis.	*Action*
4	Tombol *Get Duration*	ActionButton	Menuju halaman optimasi	*Action*
5	Tombol *Save Model As* Excel	ActionButton	Menyimpan hasil proses *discovery* dalam bentuk Excel	*Action*
6	*Scroll bar*	Action	Mengatur ukuran graph hasil *discovery*	*Action*

4.8.2 Halaman Optimasi



Gambar 4.8 Rancangan Antarmuka Optimasi

Halaman ini akan muncul ketika pengguna ingin melakukan optimasi tambahan biaya dan *makespan* percepatan yang dilakukan pada proses bisnis. Halaman ini akan menampilkan tombol-tombol yang digunakan untuk melakukan optimasi. Rancangan halaman optimasi ini dapat dilihat pada Gambar 4.8. Penjelasan mengenai atribut-atribut yang terdapat pada halaman ini bisa dilihat pada Tabel 4.8.

Tabel 4.8 Spesifikasi Atribut Antarmuka Optimasi

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Jenis Masukan / Keluaran
1	Tombol *Calculate Duration*	ActionButton	Memproses *file event log* dan melakukan perhitungan data optimasi	*Action*
2	Tombol *Save Duration As* Excel	ActionButton	Menyimpan hasil proses perhitungan data optimasi dalam bentuk Excel	*Action*
3	*Dropdown Organization*	Combox	Memilih organisasi yang tidak dioptimasi	*String*

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Jenis Masukan / Keluaran
	which is optimized			
4	Tombol *Get Solution*	ActionButton	Memproses *file event log* dan menghitung hasil optimasi minimum dutasi proyek dan minimum biaya tambahan yang dibutuhkan.	*Action*

[Halaman ini sengaja dikosongkan]

BAB V
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bab ini berisi proses implementasi dari setiap kelas pada semua modul. Bahasa pemrograman yang digunakan adalah bahasa pemrograman C#.

5.1 Lingkungan Implementasi

Lingkungan implementasi merupakan lingkungan dimana sistem ini dibangun, dimana akan dijelaskan mengenai kebutuhan perangkat yang diperlukan untuk membangun sistem ini. Lingkungan implementasi dibagi menjadi dua, yaitu lingkungan implementasi terhadap perangkat keras dan lingkungan implementasi terhadap perangkat lunak.

5.1.1 Perangkat Keras

Implementasi dilakukan pada sebuah Laptop dengan spesifikasi sebagai berikut:

- Merk : Lenovo
- Seri : Idea Pad Z470
- Processor : Processor Intel Core i3-2310M CPU @ 2.10GHz
- RAM : 4 GB

5.1.2 Perangkat Lunak

Suatu sistem atau perangkat lunak belum tentu bisa berdiri sendiri, dan sistem ini membutuhkan perangkat lunak lain yang dapat mendukung fungsionalitasnya, yang antara lain:

1. Sistem Operasi Windows

   Sistem operasi yang digunakan adalah Microsoft Windows 7 Professional 32-bit.

2. Microsoft Visual Studio

   Kakas bantu yang digunakan dalam mengembangkan sistem ini adalah Microsoft Visual Studio 2012 dengan bahasa pemrograman C#.

3. Framework .NET
   Pembuatan GUI menggunakan WPF, dibutuhkan Framework .NET 4.5.
4. Microsoft Solver Foundation
   Microsoft Solver Foundation mereupakan salah satu *reference* dari Microsoft yang digunakan untuk melakukan perhitungan aplikasi matematika. Dalam penggunaaan Microsoft Solver Foundation membutuhkan Framework .NET minimum .NET 4.0.
5. Bytescout Spreadsheet
   Bytescout Spreadsheet mereupakan salah satu *reference* yang dapat digunakan untuk membaca file Excel.

5.2 Penjelasan Implementasi

Pada subbab ini dijelaskan implementasi setiap metode yang dijelaskan pada bab metode pemecahan masalah, sehingga terbentuk suatu perangkat lunak yang mengimplementasi metode-metode pada bab metode pemecahan masalah.

5.2.1 Implementasi Heuristic Miner

Pada bagian ini mengimplementasi *heuristic miner* agar dapat melakukan proses *discovery* terhadap *event log* agar mendaoatkan model proses busnis yang terjadi. Proses dalam implementasi ini terdapat dua jenis yaitu:

- Pertama proses pembacaan *file event log* yang sekaligus melakukan pengidentifikasin *trace* yang terkandung dalam *event log* dan frekuensinya. Hal ini dapat dilihat pada Kode Sumber 5.1 dan Kode Sumber 5.2.

```
public void HeuristicMiner(string name)
{
document.LoadFromFile(name);
Worksheet worksheetNode = document.Workbook.Worksheets[0];
int i = 1;
int j = 0;
while (Convert.ToString(worksheetNode.Cell(i, 0)) != "")
{
```

```
  Cell currentCellCaseID = worksheetNode.Cell(i, 0);
  Cell currentCellCaseID1 = worksheetNode.Cell(i + 1, 0);
  Cell currentCellCaseID2 = worksheetNode.Cell(i - 1, 0);
  Cell currentCellActivity = worksheetNode.Cell(i, 1);
  Cell currentCellTime1 = worksheetNode.Cell(i, 2);
  Cell currentCellTime2 = worksheetNode.Cell(i + 1, 2);
  Cell oroginator = worksheetNode.Cell(i, 3);
  Cell currentCost = worksheetNode.Cell(i, 4);
  Cell Time = worksheetNode.Cell(i, 5);
  double ab = Convert.ToDouble(currentCellTime1.Value);
  double ac = Convert.ToDouble(currentCellTime2.Value);
  DateTime time1 = DateTime.FromOADate(ab);
  DateTime time2 = DateTime.FromOADate(ac);
  if (currentCellCaseID1.Value == currentCellCaseID.Value)
  {
    timeGap = time2 - time1;
    Time.Value = timeGap;
  }
  if (currentCellCaseID1.Value == currentCellCaseID.Value &&
((timeGap >= TimeSpan.Zero) || currentCellTime2.Value ==
currentCellTime1.Value))
{if(!(durationCase.ContainsKey(currentCellCaseID.Value.ToString
()())))
   {
   durationCase.Add(currentCellCaseID1.Value.ToString(), new
   List<double>());
    }
   activity.Add(currentCellActivity.Value.ToString());
}
  if (currentCellCaseID2.Value == currentCellCaseID.Value)
  {
    if
(!activity.Contains(currentCellActivity.Value.ToString()))
    {
      activity.Add(currentCellActivity.Value.ToString());
    }}
  if (!listAct.Contains(currentCellActivity.Value.ToString()))
  {
     listAct.Add(currentCellActivity.Value.ToString());
     input.Add(currentCellActivity.Value.ToString(), new
     List<string>());
     output.Add(currentCellActivity.Value.ToString(), new
     List<string>());
  }
  if (currentCellCaseID1.Value != currentCellCaseID.Value)
  {
    if (j == 0)
    {
      int w = 0;
      int t = activity.Count();
```

```
        myTrace.Add(j, new List<string>());
        myTrace[j] = activity.GetRange(w, t);
        freq.Add(0);
         j++;
            }
           if (j > 0)
           {
          int count = 0;
          for (int m = 0; m < myTrace.Count(); m++)
          {
         bool equal = myTrace[m].SequenceEqual(activity);
         if (equal == true)
         {
            count++;
         }
         }
         if (count == 0)
         {
             int w = 0;
             int t = activity.Count();
             myTrace[j] = activity.GetRange(w, t);
             freq.Add(0);
             j++;
         }
         }
         for (int m = 0; m < myTrace.Count(); m++)
         {
         bool equal = myTrace[m].SequenceEqual(activity);
         if (equal == true)
            {
              freq[m]++;
         }
         }
         activity.Clear();
   }
  i++;
}
document.Close();
```

Kode Sumber 5.1 Fungsi `HeuristicMiner()` bagian Membaca Data dan Melihat *Trace*

```
for (int m = 0; m < myTrace.Count(); m++)
{
   Console.WriteLine("trace: " + (m + 1));
   for (int z = 0; z < myTrace[m].Count(); z++)
   {
     Console.WriteLine(myTrace[m][z]);
     if (z < myTrace[m].Count() - 1)
```

```
      {
      matriksFrekuensi[myTrace[m][z], myTrace[m][z + 1]] +=
freq[m];
   }
   if ((z < myTrace[m].Count() - 2) &&
myTrace[m][z].Equals(myTrace[m][z + 2]))
   {
      matriksFreTwoLoop[myTrace[m][z], myTrace[m][z + 1]] +=
freq[m];
   }
}
```

Kode Sumber 5.2 Fungsi `HeuristicMiner()` bagian Menghitung Frekuensi *Trace*

- Yang kedua adalah bagian perhitungan matriks yang digunakan untuk melakukan proses *discovery* sehingga menghasilkan model proses bisnis. Pengimplementasian dapat dilihat pada Kode Sumber 5.3 sampai dengan Kode Sumber 5.6.

```
for (int m = 0; m < myTrace.Count(); m++)
{
  for (int z = 0; z < myTrace[m].Count(); z++)
  {
    if (z < myTrace[m].Count() - 1)
    {
       matriksDependency[myTrace[m][z], myTrace[m][z + 1]] =
       (double)((matriksFrekuensi[myTrace[m][z], myTrace[m][z +
       1]]) - (matriksFrekuensi[myTrace[m][z + 1],
       myTrace[m][z]])) /
       (double)((matriksFrekuensi[myTrace[m][z], myTrace[m][z +
       1]]) + (matriksFrekuensi[myTrace[m][z + 1],
       myTrace[m][z]]) + 1);
    }
    if ((z < myTrace[m].Count() - 1) &&
    myTrace[m][z].Equals(myTrace[m][z + 1]))
    {
       matriksOneLoop[myTrace[m][z], myTrace[m][z + 1]] =
       (double)(matriksFrekuensi[myTrace[m][z], myTrace[m][z +
       1]]) /
       (double)((matriksFrekuensi[myTrace[m][z], myTrace[m][z +
       1]]) + 1);
    }
    if ((z < myTrace[m].Count() - 2) &&
    myTrace[m][z].Equals(myTrace[m][z + 2]))
   {
    matriksTwoLoop[myTrace[m][z], myTrace[m][z + 1]] =
    (double)((matriksFreTwoLoop[myTrace[m][z], myTrace[m][z +
    1]]) + (matriksFreTwoLoop[myTrace[m][z + 1],
```

```
    myTrace[m][z]])) /
    (double)((matriksFreTwoLoop[myTrace[m][z], myTrace[m][z +
    1]]) + (matriksFreTwoLoop[myTrace[m][z + 1], myTrace[m][z]])
    + 1);
    }}}
```

Kode Sumber 5.3 Fungsi `HeuristicMiner()` bagian Perhitungan Matriks *Dependency* dan *ShortLoop*

```
RTB = matriksDependency.avg() -
(matriksDependency.standradDeviation() / 2);
DT = matriksDependency.avg() -
matriksDependency.standradDeviation();
for (int m = 0; m < myTrace.Count(); m++)
{
  for (int z = 0; z < myTrace[m].Count(); z++)
  {
    if (z < myTrace[m].Count() - 1 && (matriksDependency.maxi()
    - matriksDependency[myTrace[m][z], myTrace[m][z + 1]]) <=
    RTB && (matriksDependency[myTrace[m][z], myTrace[m][z + 1]]
    >= DT)
    || (z < myTrace[m].Count() - 1 &&
    (Math.Round((matriksDependency.maxi() -
    matriksDependency[myTrace[m][z], myTrace[m][z + 1]]), 2) <=
    Math.Round(RTB, 2) &&
    Math.Round(matriksDependency[myTrace[m][z], myTrace[m][z +
    1]], 2) >= Math.Round(DT, 2))))
    {
      if (!(output[myTrace[m][z]].Contains(myTrace[m][z +
      1])))
      {
      output[myTrace[m][z]].Add(myTrace[m][z + 1]);
      }
      if (!(input[myTrace[m][z + 1]].Contains(myTrace[m][z])))
      {
      input[myTrace[m][z + 1]].Add(myTrace[m][z]);
      }
    }
     if (z < myTrace[m].Count() - 1 &&
     matriksFrekuensi[myTrace[m][z], myTrace[m][z + 1]] > 0 &&
     matriksDependency[myTrace[m][z], myTrace[m][z + 1]] >= 0)
     {
        if (minPDM > matriksDependency[myTrace[m][z],
        myTrace[m][z + 1]])
        {
        minPDM = matriksDependency[myTrace[m][z], myTrace[m][z +
        1]];
        }
```

```
    }}}
```

Kode Sumber 5.4 Fungsi `HeuristicMiner()` bagian Penentuan *Causal* Matriks dan *Dependency Graph*

```
for (int m = 0; m < myTrace.Count(); m++)
{
  for (int z = 0; z < myTrace[m].Count(); z++)
  {
    if ((z < myTrace[m].Count() - 1) &&
    output[myTrace[m][z]].Count() > 1)
    {
      if (!output[myTrace[m][z]].Except(myTrace[m]).Any())
      {
        matriksParallel[myTrace[m][z], myTrace[m][z + 1]] +=
      freq[m];
      }
      if (!(!output[myTrace[m][z]].Except(myTrace[m]).Any()))
      {
        nonParallelSplit[myTrace[m][z]]++;
      }
    }
    if ((z < myTrace[m].Count() - 1) && (input[myTrace[m][z +
    1]].Count() > 1))
    {
       if (!input[myTrace[m][z + 1]].Except(myTrace[m]).Any())
       {
         matriksParallel[myTrace[m][z + 1], myTrace[m][z]] +=
       freq[m];
       }
       if (!(!input[myTrace[m][z +
       1]].Except(myTrace[m]).Any()))
       {
         nonParallelJoin[myTrace[m][z + 1]]++;
       }
    }
  }
}
foreach (string key in input.Keys)
{
    if (input[key].Count() == 0)
    {
      start = key;
    }
    if (output[key].Count() == 0)
    {
      end = key;
    }
    for (int m = 0; m < input[key].Count; m++)
```

```
    {
      if (m < (input[key].Count) - 1 && matriksParallel[key,
      input[key][m]] >= 0 && matriksParallel[key, input[key][m +
      1]] >= 0 && input[key].Count > 1)
      {
      matriksParallel[input[key][m], input[key][m + 1]] =
      matriksParallel[key, input[key][m]];
      matriksParallel[input[key][m + 1], input[key][m]] =
      matriksParallel[key, input[key][m + 1]];
      PMinput[key].Add((double)(matriksParallel[input[key][m],
      input[key][m + 1]] + matriksParallel[input[key][m + 1],
      input[key][m]]) /
       (double)(nonParallelJoin[key] +
      matriksFrekuensi[input[key][m], key] +
      matriksFrekuensi[input[key][m + 1], key] + 1));
      matriksParallel[key, input[key][m]] = 0;
      matriksParallel[key, input[key][m + 1]] = 0;
    }
  }
}
foreach (string key in output.Keys)
{
  for (int m = 0; m < output[key].Count; m++)
  {
    if (m < (output[key].Count) - 1 && matriksParallel[key,
    output[key][m]] >= 0 && matriksParallel[key, output[key][m +
    1]] >= 0 && output[key].Count > 1)
    {
      matriksParallel[output[key][m], output[key][m + 1]] =
      matriksParallel[key, output[key][m]];
      matriksParallel[output[key][m + 1], output[key][m]] =
      matriksParallel[key, output[key][m + 1]];
      PMoutput[key].Add((double)(matriksParallel[output[key][m],
      output[key][m + 1]] + matriksParallel[output[key][m + 1],
      output[key][m]]) /
      (double)(nonParallelSplit[key] + matriksFrekuensi[key,
      output[key][m]] + matriksFrekuensi[key, output[key][m +
      1]] + 1));
      matriksParallel[key, output[key][m]] = 0;
      matriksParallel[key, output[key][m + 1]] = 0;
    }
  }
}
```

Kode Sumber 5.5 Fungsi `HeuristicMiner()` bagian Perhitungan Matriks Frekuensi *Parallel* dan *Parellel Measure*

```
foreach (string key in PMinput.Keys)
```

```
{
  if (PMinput[key].Count > 0)
  {
    avgPMJoin[key] = (double)PMinput[key].Sum() /
    (double)PMinput[key].Count();
  }
}
foreach (string key in PMoutput.Keys)
{
  if (PMoutput[key].Count > 0)
  {
    avgPMSplit[key] = (double)PMoutput[key].Sum() /
    (double)PMoutput[key].Count();
  }
}
foreach (string key in avgPMSplit.Keys)
{
  if (avgPMSplit[key] <= minPDM)
  {
     split.Add(key, "XOR Split");
  }
  if (avgPMSplit[key] > minPDM && avgPMSplit[key] <=
  matriksDependency.avg())
  {
     split.Add(key, "OR Split");
  }
  if (avgPMSplit[key] > matriksDependency.avg())
  {
     split.Add(key, "AND Split");
  }
}
foreach (string key in avgPMJoin.Keys)
{
  if (avgPMJoin[key] <= minPDM)
  {
    join.Add(key, "XOR Join");
  }
  if (avgPMJoin[key] > minPDM && avgPMJoin[key] <=
  matriksDependency.avg())
  {
    join.Add(key, "OR Join");
  }
  if (avgPMJoin[key] > matriksDependency.avg())
  {
    join.Add(key, "AND Join");
  }
}}
```

Kode Sumber 5.6 Fungsi `HeuristicMiner()` bagian Penentuan *Split* dan *Join* untuk Paralel

5.2.2 Implementasi Perhitungan Data untuk Optimasi

Pada bagian ini mengimplementasi perhitungan dari optimasi data. Pengimplementasian ini bertujuan untuk mendapatkan nilai dari durasi normal, biaya normal, durasi *crash,* biaya *crash,* dan juga *cost slope* dan *time slope* yang akan digunakan dalam pembentukan model matematika *linear programming.* Pengimplementasiannya dapat dilihat pada Kode Sumber 5.7.

```
public void GetDuration(string name)
{
HeuristicMiner(name);
List<string> temp = new List<string>();
Worksheet worksheetNode = document.Workbook.Worksheets[0];
int o = 1;
while (Convert.ToString(worksheetNode.Cell(o, 0)) != "")
{
  Cell currentCellCaseID = worksheetNode.Cell(o, 0);
  Cell currentCellCaseID1 = worksheetNode.Cell(o + 1, 0);
  Cell currentCellCaseID2 = worksheetNode.Cell(o - 1, 0);
  Cell currentCellActivity = worksheetNode.Cell(o, 1);
  Cell currentCellActivity1 = worksheetNode.Cell(o + 1, 1);
  Cell Time = worksheetNode.Cell(o, 7);
  Cell currentCost = worksheetNode.Cell(o, 4);
  Cell currentCellTime1 = worksheetNode.Cell(o, 2);
  if (currentCellCaseID1.Value == currentCellCaseID.Value)
  {
    int m = o;
    double minim = 10000000;
    Cell currentCellTime2 = worksheetNode.Cell(m + 1, 2);
    double ab = Convert.ToDouble(currentCellTime1.Value);
    double ac = Convert.ToDouble(currentCellTime2.Value);
    double lalala = 0;
    DateTime time1 = DateTime.FromOADate(ab);
    DateTime time2 = DateTime.FromOADate(ac);
    Cell currentCellCaseID3 = worksheetNode.Cell(m + 1, 0);
    Cell currentCellActivity3 = worksheetNode.Cell(m + 1, 1);
    string hihihi = currentCellCaseID.Value.ToString();
    string hahaha = currentCellActivity.Value.ToString();
    while (hihihi == currentCellCaseID3.Value.ToString())
    {
       if
       (output[hahaha].Contains(currentCellActivity3.Value.ToString
       ()) && output[hahaha].Count() > 1)
        {
            timeGap = time2 - time1;
```

```
        lalala = (ac - ab) / (double)0.0416666666666667;
        if (lalala < minim)
        {
          duration[currentCellActivity.Value.ToString()].Add(lalala
          );
          durationCase[currentCellCaseID.Value.ToString()].Add(lala
          la);
          Time.Value = lalala;
          minim = lalala;
        }
        temp.Add(hahaha);
       }
       if
       (output[hahaha].Contains(currentCellActivity3.Value.ToString
       ()) && output[hahaha].Count() < 2)
        {
           timeGap = time2 - time1;
           lalala = (ac - ab) / (double)0.0416666666666667;
           Time.Value = lalala;

           duration[currentCellActivity.Value.ToString()].Add(lalal
           a);

           durationCase[currentCellCaseID.Value.ToString()].Add(lal
           ala);
           temp.Add(hahaha);
        }
        m++;
        currentCellCaseID3 = worksheetNode.Cell(m + 1, 0);
        currentCellActivity3 = worksheetNode.Cell(m + 1, 1);
        currentCellTime2 = worksheetNode.Cell(m + 1, 2);
        ac = Convert.ToDouble(currentCellTime2.Value);
        time2 = DateTime.FromOADate(ac);
        if (currentCellCaseID3.Value == null)
        {
            break;
        }
}
cost[currentCellActivity.Value.ToString()].Add(Convert.ToDouble(c
urrentCost.Value));
}
if (currentCellCaseID2.Value == currentCellCaseID.Value)
{
   if (!(temp.Contains(currentCellActivity.Value.ToString())))
   {

      cost[currentCellActivity.Value.ToString()].Add(Convert.ToD
      ouble(currentCost.Value));
```

```
            duration[currentCellActivity.Value.ToString()].Add(durat
            ionCase[currentCellCaseID.Value.ToString()].Average());
            Time.Value =
            duration[currentCellActivity.Value.ToString()];
            temp.Add(currentCellActivity.Value.ToString());
           }
         }
     if (currentCellCaseID1.Value != currentCellCaseID.Value)
     {
        temp.Clear();
     }
     o++;
}

document.Close();
foreach (string key in duration.Keys)
{
   normalDuration.Add(key, Math.Round((duration[key].Sum() /
   duration[key].Count()), 2));
   double sumOfSquaresOfDiffDur =
   Math.Round(duration[key].Select(val => (val -
   duration[key].Average()) * (val -
   duration[key].Average())).Sum(), 2);
   double timeSlope = Math.Sqrt(sumOfSquaresOfDiffDur /
   (duration[key].Count() - 1));
   durationSlope.Add(key, Math.Round(timeSlope, 2));
}
foreach (string key in normalDuration.Keys)
{
   crashDuration.Add(key, Math.Round((normalDuration[key] -
   durationSlope[key]), 2));
}

foreach (string key in cost.Keys)
{
   normalCost.Add(key, Math.Round(cost[key].Average(), 2));
   double sumOfSquaresOfDiffCost = Math.Round(cost[key].Select(val
   => (val - cost[key].Average()) * (val -
   cost[key].Average())).Sum(), 2);
   double slopeCost = Math.Sqrt(sumOfSquaresOfDiffCost /
   (cost[key].Count() - 1));
   sdCost.Add(key, Math.Round(slopeCost, 2));
}

foreach (string key in normalCost.Keys)
{
   Crashcost.Add(key, Math.Round((normalCost[key] + sdCost[key]),
   2));
```

```
    costSlope.Add(key, Math.Round((Crashcost[key] - normalCost[key])
    / (durationSlope[key]), 2));
}
        }
```

Kode Sumber 5.7 Fungsi `GetDuration()` untuk Menghitung Data Optimasi

5.2.3 Implementasi Pemecahan Optimasi Durasi dan Biaya Tambahan Minimum

Pada bagian ini mengimplementasi cara untuk pengoptimasian durasi dan biaya tambahan percepatan proses bisnis. Pengimplementasian ini bertujuan untuk mendapatkan nilai durasi proyek paling minimal dan biaya tambahan yang minimal pula untuk peminimalan durasi tersebut. Dalam pengimplementasiannya terdapat dua fungsi yang digunakan yaitu:

- Yang pertama adalah fungsi `GenerateMathModel()` dalam fungsi ini yang pertama dilakukan adalah memanggil fungsi `GetDuration()` untuk mendapatkan data optimasi dan kemudian menginisialisasi kelas CPM dan memanggil fungsi-fungsi yang ada di dalalamnya yaitu `CPMethod()` untuk mengetahui *critical path* dan fungsi `GetDistance()` untuk mendapatkan durasi proses bisnis. Fungsi `GenerateMathModel()` implemetasinya dapat dilihat pada Kode Sumber 5.8 dan pengimplementasian kelas CPM dapat dilihat pada Kode Sumber 5.9 sampai Kode Sumber 5.11.

```
public void GenerateMathModel(string m)
{
GetDuration(m);
CPM cpm1 = new CPM();
foreach (string act in listAct)
{
  if (!cpm1.GetDistance().ContainsKey(act))
  {
    cpm1.GetDistance().Add(act, new KeyValuePair<List<string>,
    double>(new List<String> { "" }, -1.0));
  }
}
cpm1.CPMethod(start, end, output, normalDuration);
List<string> criticalPath = cpm1.getCriticalAct();
```

```
obj = "(" + Math.Round(costSlope[start], 2) + "*X" +
start.Replace(" ", string.Empty);
foreach (string act in listAct)
{
   if (criticalPath.Contains(act))
   {
     obj = obj + " + " + Math.Round(costSlope[act], 2) + "*X" +
     act.Replace(" ", string.Empty);
   }
   if (!criticalPath.Contains(act) && act != start)
   {
     obj = obj + " + " + Math.Round(costSlope[act], 2) + "*X" +
     act.Replace(" ", string.Empty);
   }
   maxRed = maxRed + "X" + act.Replace(" ", string.Empty) + "<="
   + Math.Round(durationSlope[act], 2) + "; ";
   nonNegX = nonNegX + "X" + act.Replace(" ", string.Empty) +
   ">=" + 0 + "; ";
   nonNegY = nonNegY + "Y" + act.Replace(" ", string.Empty) +
   ">=" + 0 + "; ";
   if (act != end)
   {
     durCons = durCons + "X" + act.Replace(" ", string.Empty) +
     "+";
   }
   if (input[act].Count != 0 && input[act].Count < 2)
   {
     startCons = startCons + "Y" + act.Replace(" ",
     string.Empty) + " - " + "Y" + input[act][0].Replace(" ",
     string.Empty) + " + X" + input[act][0].Replace(" ",
     string.Empty) + " >= " +
     Math.Round(normalDuration[input[act][0]], 2) + "; ";
   }
   if (input[act].Count != 0 && input[act].Count > 1)
   {
     for (int g = 0; g < input[act].Count(); g++)
     {
       startCons = startCons + "Y" + act.Replace(" ",
       string.Empty) + " - " + "Y" + input[act][g].Replace(" ",
       string.Empty) + " + X" + input[act][g].Replace(" ",
       string.Empty) + " >= " +
       Math.Round(normalDuration[input[act][g]], 2) + "; ";
     }
   }
}
obj = obj + "); ";
startCons = startCons + "YFINISH - Y" + end.Replace(" ",
string.Empty) + " + X" + end.Replace(" ", string.Empty) + " >= "
+ Math.Round(normalDuration[end], 2) + "; ";
```

```
durCons = "YFINISH - " +
Math.Round(cpm1.GetDistance()[end].Value, 2).ToString() + " <= "
+ 0;
makespan=Math.Round(cpm1.GetDistance()[end].Value,
2).ToString();
}
```

Kode Sumber 5.8 Fungsi `GenerateMathModel()` untuk Menghitung Data Optimasi

```
public void CPMethod(string start, string finish,
Dictionary<string, List<string>> output, Dictionary<string,
double> normalDuration)
{
start1 = start;
end1 = finish;
distances[start] = new KeyValuePair<List<string>, double>(new
List<String> { "" }, normalDuration[start]);
List<String> successors = new List<string>();
successors.Add(start);
while (successors.Count() > 0)
{
  current = successors[0];
  if (current == finish)
  {
    successors.RemoveAt(0);
    if (successors.Count() > 0)
        current = successors[0];
    else
        break;
  }

  successors.RemoveAt(0);
  for (int g = 0; g < output[current].Count(); g++)
  {
   if (distances[current].Value +
   normalDuration[output[current][g]] >
   distances[output[current][g]].Value)
   {
     distances[output[current][g]] = new
     KeyValuePair<List<string>, Double>(new List<string> {
     current }, distances[current].Value +
     normalDuration[output[current][g]]);
     if (!successors.Contains(output[current][g]))
       successors.Add(output[current][g]);
   }

   else if (distances[current].Value +
   normalDuration[output[current][g]] ==
   distances[output[current][g]].Value)
```

```
    {
       if (!distances[output[current][g]].Key.Contains(current))
          distances[output[current][g]].Key.Add(current);
       if (!successors.Contains(output[current][g]))
         successors.Add(output[current][g]);
     }
  }
}
}
```

Kode Sumber 5.9 Fungsi `CPMethod()` pada Kelas CPM untuk Mendapatkan *critical path*

```
public void GetPath(string current, String path)
{
String pathTemp = start1;
while (current != start1)
{
  if (distances[current].Key.Count() > 1)
  {
    for (int i = 1; i < distances[current].Key.Count(); i++)
    {
      GetPath(distances[current].Key[i], path + current);
      path += current + "->";
      current = distances[current].Key[0];
    }
  }
}
paths.Add(path);
}
```

Kode Sumber 5.10 Fungsi `GetPath()` pada Kelas CPM untuk Mendapatkan *critical path*

```
public List<string> getCriticalAct()
{
string[] result = null;
GetPath(end1, "");
foreach (var path in paths)
{
  result = path.Split(new string[] { "->" },
  StringSplitOptions.None);
  foreach (string a in result)
  {
    if (!(criticalAct.Contains(a)))
    {
      criticalAct.Add(a);
    }
  }
```

```
}


return criticalAct;
}
```

Kode Sumber 5.11 Fungsi `getCtiticalAct()` pada Kelas CPM untuk Mendapatkan Aktivitas yang masuk *critical path*

- Yang kedua merupakan fungsi yang digunakan untuk menyelesaikan model matematika dari *linear programming*. Penyelesaian ini digunakan untuk mendapatkan durasi proses bisnis yang paling minimum dengan biaya yang digunakan untuk meminimumkan durasi juga minimum. Penyelesaian ini dilakukan dengan menggunakan metode *simplex* yang ada pada Microsoft Solver Foundation. Pengimplementasiannya dapat dilihat Kode Sumber 5.12.

```
public string SolveMathModel()
{
string [] constrain1, constrain2;
string goal, constrain3;
Dictionary<string,Decision> dec= new
Dictionary<string,Decision>();
SolverContext context = SolverContext.GetContext();
context.ClearModel();
Model model = context.CreateModel();
foreach (string act in listAct)
{
  dec.Add("X" + act.Replace(" ", string.Empty), new
  Decision(Domain.RealNonnegative, "X" + act.Replace(" ",
  string.Empty)));
  dec.Add("Y" + act.Replace(" ", string.Empty), new
  Decision(Domain.RealNonnegative, "Y" + act.Replace(" ",
  string.Empty)));
}
Decision YFINISH = new Decision(Domain.RealNonnegative,
"YFINISH");
foreach (string key in dec.Keys)
{
  model.AddDecision(dec[key]);
}
model.AddDecision(YFINISH);
goal = GetMathModel()[0].Substring(0, GetMathModel()[0].Length -
2);
```

```
constrain1 = GetMathModel()[1].Substring(0,
GetMathModel()[1].Length - 2).Split(new string[] { "; " },
StringSplitOptions.None);
constrain2 = GetMathModel()[2].Substring(0,
GetMathModel()[2].Length - 2).Split(new string[] { "; " },
StringSplitOptions.None);
constrain3 = GetMathModel()[3];
int con1 = constrain1.Length - 1;
foreach (string a in constrain1)
{
  if (con1 >= 0)
  {
    model.AddConstraint("MaxRed" + con1, a);
  }
  con1--;
}
int con2 = constrain2.Length - 1;
foreach (string a in constrain2)
{
  if (con2 >= 0)
  {
    model.AddConstraint("StartConstraint" + con2, a);
  }
  con2--;
}
model.AddConstraint("DurationConstraint", constrain3);
model.AddGoal("AdditionalCost", GoalKind.Maximize, goal);
Solution solution = context.Solve(new SimplexDirective());
Report report = solution.GetReport();
string minimumMakespan1 = report.ToString().After("YFINISH:");
double minimumMakespan2 = Convert.ToDouble(minimumMakespan1);
foreach (Goal a in model.Goals)
{
  model.RemoveGoal(a);
  break;
}
foreach (Constraint a in model.Constraints)
{
  if (a.Expression.ToString().Equals(constrain3))
  {
    model.RemoveConstraint(a);
    break;
  }
}
model.AddGoal("AdditionalCost", GoalKind.Minimize, goal);
model.AddConstraints("DurationConstraint", YFINISH -
minimumMakespan2 <= 0);
Solution solution2 = context.Solve(new SimplexDirective());
Report report2 = solution2.GetReport();
string solutionMath = report2.ToString();
```

```
return solutionMath;
}
```

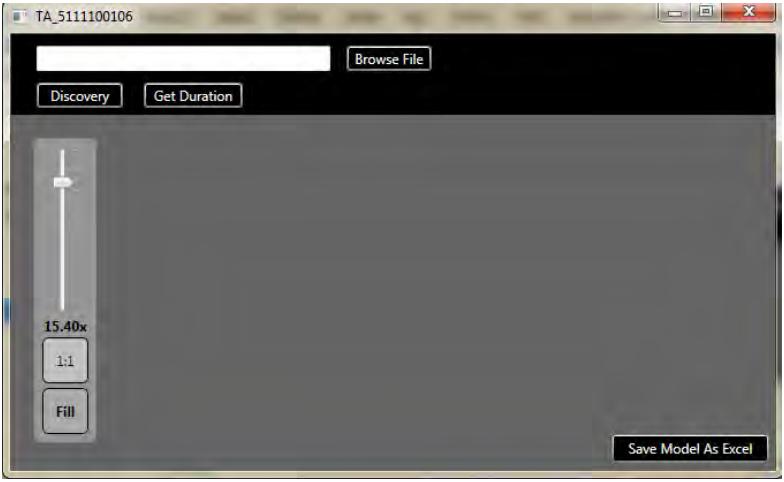
Kode Sumber 5.12 Fungsi `SolveMathModel()` Digunakan unruk Menyelesaikan Optimasi

5.3 Implementasi Antar Muka

Implementasi tampilan antarmuka pengguna pada Visual Studio 2012 dengan menggunakan jenis tampilan WPF. Berikut ini akan dijelaskan mengenai implementasi tampilan antarmuka pengguna yang terdapat pada perangkat lunak.

5.3.1 Halaman Proses *Discovery*

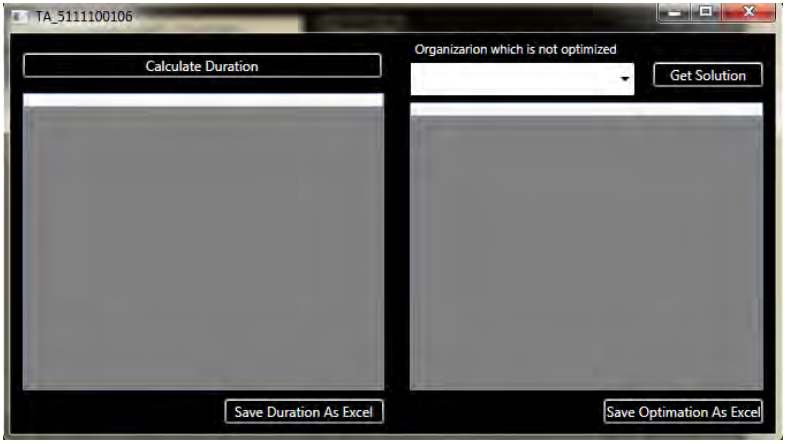
Halaman ini merupakan halam pertama yang muncul ketika aplikasi dijalankan. Pada halaman ini terdapat beberapa parameter yang harus diisi dan dibutuhkan untuk pemrosesan selanjutnya. Parameter tersebut antara lain *path* direktori tempat *event log* berada. Setelah mendapat *path* sistem akan membuka *file event log* yang telah dipilih. Setelah *file* terbuka maka sistem akan melakukan proses *discovery* atau bisa langsung melakukan optimasi tergantung kebutuhan dari pengguna. Selain itu pada halaman ini juga terdapat fungsi *Get Duration* yang digunakan untuk menuju pada halaman optimasi. Hasil implementasi pada gambar ini dapat dilihat pada Gambar 5.1. Sementara untuk implementasi kode sumber pada halaman ini dapat dilihat pada lampiran A.



Gambar 5.1 Hasil Implementasi Halaman *Process Discovery*

5.3.2 Halaman Proses Optimasi

Halaman ini akan muncul ketika pengguna menekan tombol *Get Duration* pada halaman proses *discovery,* hal ini dilakukan ketika pengguna ingin melakukan optimasi tambahan biaya dan *makespan* percepatan yang dilakukan pada proses bisnis. Halaman ini akan menampilkan tombol-tombol yang digunakan untuk melakukan optimasi. Hasil implementasi pada gambar ini dapat dilihat pada Gambar 5.2. Sementara untuk implementasi kode sumber pada halaman ini dapat dilihat pada lampiran A.



Gambar 5.2 Hasil Implementasi Halaman Optimasi

[Halaman ini sengaja dikosongkan]

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini membahas hasil dan pembahasan pada aplikasi yang dikembangkan. Pada bab ini akan dijelaskan tentang data yang digunakan, hasil yang didapatkan dari penggunaan perangkat lunak dan uji coba yang dilakukan pada perangkat lunak yang telah dikerjakan untuk menguji apakah fungsionalitas aplikasi telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya.

6.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi pembuatan sistem pada tugas akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
   a. Prosesor: Intel® Core™ i3 CPU @ 2.10GHz
   b. Memori(RAM): 4 GB
   c. Tipe sistem: 32-bit sistem operasi
2. Perangkat lunak
   a. Sistem operasi: Windows 7 Professional
   b. Perangkat pengembang: Microsoft Visual Studio 2012

6.2 Tahapan Uji Coba

Dalam uji coba yang dilakukan dalam tugas akhir ini memiliki beberapa tahapan yang dijelaskan pada subbab ini.

6.2.1 Memasukkan Data *Event Log*

Yang pertama kali dilakukan untuk tahapan uji coba adalah memasukkan *event log* pada program. *Event log* yang digunakan harus mengandung informasi sebagai berikut:

- *Case* Id
  Case merupakan suatu kasus tertentu yang ada pada *event log*. Kasus tertentu tersebut dapat berupa suatu kasus

dalam memproduksi suatu barang tertentu, karena *event log* dapat terdiri dari catatan dari proses eksekusi pembuatan banyak barang atau proses eksekusi dari banyak kasus proses.

- Aktivitas yang dijalankan pada proses bisnis.
- Waktu dijalankannya aktiviast dalam suatu *case* dalam proses bisnis (*timestamp*)
- Originator pelaksana aktivitas dalam setiap *case* pada proses bisnis.
- Biaya setiap aktitivas pada setiap *case* dalam proses bisnis.

Data masukkan yang digunakan dalam program ini memiliki ekstensi Excel. Contoh bentuk data masukkan yang digunakan terdapat pada. Dalam perangkat lunak yang dikembangkan bagian ini dilakukan dengan menekan timbol *Browse.*

Tabel 6.1 Contoh Format Data Masukkan

Case ID	Activity	Time Stamp	Organisator	Cost
PP1	Sending good receive	6/20/2014 8:32	Purchase Department	243303
PP1	Getting good receive	6/20/2014 13:42	Spinning Department	138941
PP1	Bale opening	6/20/2014 23:41	Spinning Department	508599
PP1	Conditioning of MMP Fiber	6/21/2014 8:16	Spinning Department	63706
PP1	Blending	6/21/2014 10:46	Spinning Department	496301
PP1	Opposing spike	6/21/2014 16:57	Blowing Department	352814
PP1	Striking cotton	6/21/2014 18:09	Blowing Department	125149
PP1	Carding	6/21/2014 19:15	Spinning Department	957335
PP1	Roving frame	6/22/2014 10:51	Framing Department	516512
PP1	Drawing frame	6/22/2014 16:28	Framing Department	455652
PP1	Combing	6/22/2014 23:42	Spinning Department	394780
PP1	Ring framing	6/23/2014 4:48	Spinning Department	359985
PP1	Cone winding	6/23/2014 16:44	Spinning Department	319064
PP2	Sending good receive	6/23/2014 23:26	Purchase Department	212893
PP2	Getting good receive	6/24/2014 4:19	Spinning Department	157720
PP2	Bale opening	6/24/2014 7:48	Spinning Department	405832
PP2	Conditioning of MMP Fiber	6/25/2014 1:08	Spinning Department	137116
PP2	Blending	6/25/2014 3:02	Spinning Department	450040
PP2	Air current blowing	6/25/2014 5:11	Blowing Department	269791
PP2	Striking cotton	6/25/2014 8:25	Blowing Department	186637
PP2	Carding	6/25/2014 12:45	Spinning Department	1079023
PP2	Drawing frame	6/26/2014 0:12	Framing Department	460565
PP2	Combing	6/26/2014 4:48	Spinning Department	417705
PP2	Ring framing	6/26/2014 15:16	Spinning Department	360005
PP2	Cone winding	6/26/2014 22:49	Spinning Department	313302
PP3	Sending good receive	6/27/2014 5:19	Purchase Department	180588
PP3	Getting good receive	6/27/2014 11:40	Spinning Department	159531
PP3	Bale opening	6/27/2014 20:00	Spinning Department	413071

6.2.2 Melakukan Proses *Discovery*

Setelah memasukkan *event log,* proses selanjutnya yang dilakukan pada *event log* adalah melakukan proses *discovery*. Proses *discovery* dilakukan dengan algoritma modifikasi *heuristic miner* yang telah dijelaskan pada subbab 3.2. Proses *discovery* dimaksudkan untuk mengetahui hubungan antar aktivitas. Hubungan antar aktivitas digunakan untuk proses selanjutnya yaitu untuk melakuan perhitungan durasi yang digunakan untuk proses optimasi. Para perangkat lunak yang dikembangkan proses ini dilakukan dengan menekan tombol *Discovery.*

6.2.3 Melakukan Perhitungan Data Optimasi

Perhitungan data optimasi ditujukan untuk mendapatkan data optimasi berupa durasi normal, durasi *crash*, biaya normal, biaya *crash*, dan *cost slope.* Hubungan antar aktivitas digunakan untuk mencari durasi per *case* setiap aktivitas yang ada, oleh karena itu sebelum melakukan proses ini diperlukan proses *discovery* untuk mengetahui hubungan antar aktivitas. Perhitungan data untuk optimasi telah dijelaskan pada subbab 3.5. Pada perangkat lunak yang dikembangkan proses ini dilakukan dengan yang pertama menekan tombol *Get Duration* pada halaman awal, dan kemudian menekan tombol *Calculate Duration.*

6.2.4 Melakukan Optimasi dengan Linear Programming

Setelah mendapatkan data optimasi yang kemudian dilakukan adalah memetakan data optimasi menjadi model matematika. Model matematika pertama menggunakan fungsi objektif *maximize* hal ini dikarenakan fungsi *maximize* pada biaya tambahan menghasilkan durasi proses bisnis yang paling minimal. Untuk model matematika yang kedua menggunakan fungsi objektif *minimize* karena fungsi *minimize* digunakan untuk menghasilkan biaya tambahan yang paling minimum dengan durasi batasan menggnakan durasi yang dihasilkan pada fungsi objektif model matematika yang pertama. Pada perangkat lunak yang dikembangkan proses ini dilakukan dengan yang pertama menekan tombol *Get Duration* pada halaman awal, dan kemudian menekan

tombol *Get Solution*. Pada proses optimasi ini pengguna juga dapat memilih organisasi yang tidak dioptimasi pada *dropdown* dengan label *Organization which is not optimized.*

6.3 Data Studi Kasus

Data uji yang digunakan dalam tugas akhir terdapat tiga jenis data uji yaitu untuk data *cross-organizational* didapatkan dari data *event log purchace order* bahan pembuatan benang PT Toray Industries Indonesia dan data *event log* produksi benang dari PT Toray Industries Indonesia, sedangkan untuk data *single-organization* yang digunakan untuk pembuktian *noise* diambil dari model yang dibuat menggunakan YAWL yang kemudian dibangkitkan *event log*-nya yang kemudian dirubah-rubah sehingga mengandung *noise.*

6.3.1 Data *Event Log Purchace Order* Bahan Pembuatan Benang PT Toray Industries Indonesia

Data *event log purchase order* bahan pembuatan benang dari PT Toray Industries Indonesia merupakan jenis *event* log yang melibatkan lebih dari satu departemen atau lebih dari satu organisasi. Dalam data ini melibatkan empat departmen yaitu *Purchase Order, Supplier*, dan Bea dan Cukai. Dalam data ini *Supplier* terdapat dua jenis yaitu *Supplier* kawasan berikat dan *Supplier* dari kawasan non berikat. Hal ini terjadi karena perbedaan alur proses bisnis antara *Supplier* kawasan berikat dan *Supplier* dari kawasan non berikat, kerana PT Toray Industries Indonesia merupakan perusahaan kawasan berikat maka ketika PT Toray Industries Indonesia memembeli bahan baku *Supplier* dari kawasan non berikat maka *Supplier* tersebut harus mengurus perijinan dan pembayaran pajak terhadap Bea dan Cukai, tetapi jika PT Toray Industries Indonesia memembeli bahan baku *Supplier* dari kawasan berikat maka dapat langsung diproses tanpa harus melakukan perijinan terhadap Bea dan Cukai. Adapun aktivitas dari masing-masing departemen adalah sebagai berikut:

1. Aktivitas pada departemen *Purchase*
   - *Sending PO Number*
   - *Receiving Good Receive*
2. Aktivitas pada *Supplier*
   - *Supplier* Berikat
     - *Producing Good Orders Sup1*
     - *Pakcaging Good Orders*
   - *Supplier* Non Berikat
     - *Sending Permitting Doc*
     - *Paying PPh*
     - *Producing Good Orders Sup2*
     - *Packaging Good Orders and Getting PPh Confirm*
   - Aktivitas yang dilakukan kedua jenis *Supplier*
     - *Sending Good Orders*
3. Aktivitas pada Bea dan Cukai
   - *Determining PPh and Giving Permission*

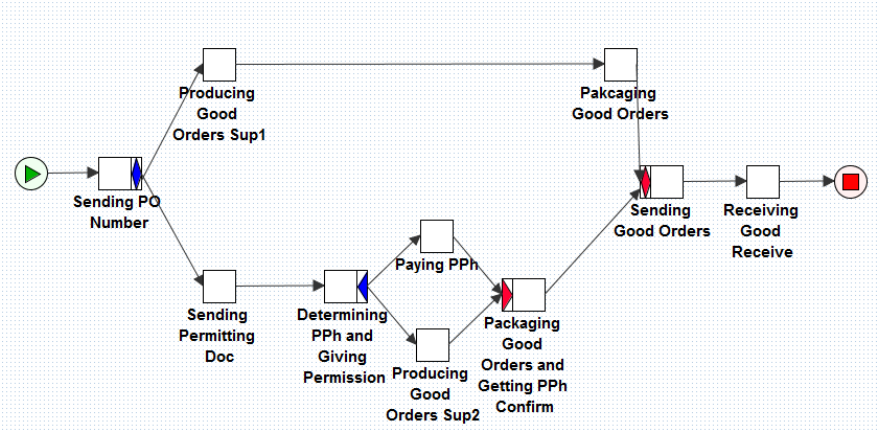
Bentuk *event log* dari data ini diperlihatkan pada Tabel 6.2.

Tabel 6.2 *Event Log* Data *Purchase Order*

Case Id	Activity	Time Stamp	Organisator	Cost
PO1	Sending PO Number	1/20/2014 8:32	Purchasing Department	14117
PO1	Sending Permitting Doc	1/20/2014 12:03	Supplier	34664
PO1	Producing Good Orders Sup1	1/20/2014 20:43	Supplier	738443
PO1	Pakcaging Good Orders	1/22/2014 21:57	Supplier	12063
PO1	Determining PPh and Giving Permission	1/23/2014 10:01	Beacukai	8291
PO1	Paying PPh	1/23/2014 14:10	Supplier	40600
PO1	Producing Good Orders Sup2	1/23/2014 15:11	Supplier	610468
PO1	Packaging Good Orders and Getting PPh Confirm	1/25/2014 7:53	Supplier	29462
PO1	Sending Good Orders	1/25/2014 8:51	Supplier	128884
PO1	Receiving Good Receive	1/26/2014 17:05	Purchasing Department	16950

Pada Tabel 6.2 menunjukan salah satu *case* dari *event log* data produksi. Sebenarnya dalam event log data produksi terkandung 100 *case* produksi, dengan 45 jenis *trace.* Pada kolom aktivitas terlihat tedapat perbedaan warna, perbedaan ini menunjukkan bahwa aktivitas itu dikerjakan oleh organisasi yang berbeda.

Model dari *event log* data prodroduksi dapat dilihat pada Gambar 6.1. Hubungan antar aktivitas dari model data purchase dapat dilihat pada Tabel 6.3.



Gambar 6.1 Model Data *Purchase Order*

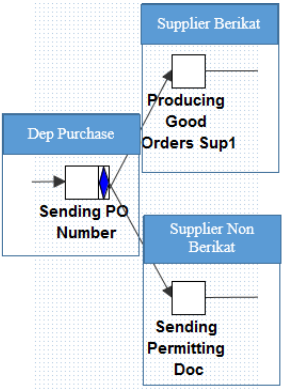
Tabel 6.3 Hubungan Antar Aktivitas dari Gambar 6.1

Input	Split / Join	Output
{Start}		Sending PO Number
Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,
Producing Good Orders Sup1		Pakcaging Good Orders
Sending Permitting Doc		Determining PPh and Giving Permission
Determining PPh and Giving Permission	AND Split	Paying PPh, Producing Good Orders Sup2,
Producing Good Orders Sup2, Paying PPh	AND Join	Packaging Good Orders and Getting PPh Confirm
Packaging Good Orders and Getting	OR Join	Sending Good Orders

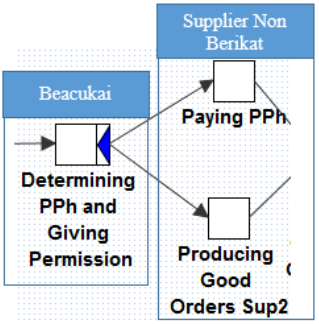
Input	Split / Join	Output
PPh Confirm, Pakcaging Good Orders		
Sending Good Orders		Receiving Good Receive
Receiving Good Receive		{end}

Jenis pola koordinasi *cross-organizational* yang ada pada data *event log* produksi adalah sebagau berikut:

1. Pola Koordinasi Pertukaran Pesan
   Pola ini terjadi antara departemen *Purchase Supplier* dan *Supplier* Bea dan Cukai. *Purchase Supplier* pada saat aktivitas *sending po number* dan *getting good receive* saat itu terjadi pertukaran pesan berupa pemesanan barang oleh *Purchase* terhadap *Supplier* jika *Supplier* Berikat maka *Supplier* akan langsung memproduksi pesanan tapi jika *Supplier* Non Berikat maka *Supplier* akan mengurus perijinan pada Bea dan Cukai, mengurus perijinan ini juga masuk dalam alur pertukaran pesan antara *Supplier* dan Bea dan Cukai. Alur pertukaran pesan *Purchase Supplier* dapat dilihat pada Gambar 6.2 dan alur pertukaran pesan *Supplier* Bea dan Cukai dapat dilihat pada Gambar 6.3.



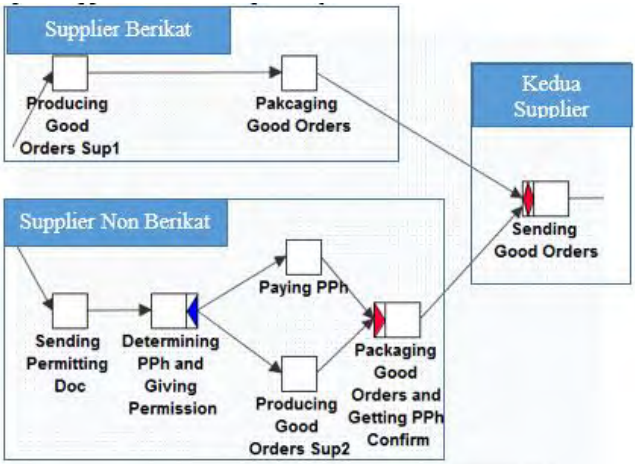
Gambar 6.2 Alur Pertukaran Pesan *Purchase Supplier* Data *Purchase*



Gambar 6.3 Alur Pertukaran Pesan *Supplier* Bea dan Cukai Data *Purchase*

2. Pola Koordinasi Prosedur Abstrak

   Pola ini terjadi antara departemen antara *Supplier* Berikat dan Non Berikat, hal ini dikarenakan sebenarnya keduanya melakukan proses yang sama yaitu memproduksi dan mengirimkan barang ke PT Toray Industries Indonesia tapi pada *Supplier* Non Berikat prosesnya lebih rinci karena belum memiliki perijinan dari Bea dan Cukai. Alur prosedur abstrak dapat dilihat pada Gambar 6.4.



Gambar 6.4 Alur Prosedur Abstrak *Supplier* Berikat dan Von Berikat Data *Purchase*

6.3.1 Data *Event Log* Produksi Benang dari PT Toray Industries Indonesia

Data *event log* produksi benang dari PT Toray Industries Indonesia merupakan jenis *event* log yang melibatkan lebih dari satu departemen atau lebih dari satu organisasi. Dalam data ini melibatkan empat departmen yaitu *Purchase, Spinning*, *Blowing*, dan *Framming.* Keempat departemen ini memiliki kewenangan masing-masing seperti untuk departemen *Purchase* memiliki kewenangan dalam mengirimkan bahan baku pada departemen *Spinning,* sedangkan *Spinning* memiliki kewenangan untuk pemintalan benang sehingga menjadi bentuk *cone* atau gulungan besar yang siap dipasarkan, dalam pemintalan benang sehingga sampai siap untuk dipasarkan. Departemen *Spinning* melakukan kerjasama dengan dua departemen lain yaitu *Blowing* dan *Framming* dalam memproduksi benang. Departemen *Blowing* berperan dalam pembersihan bahan benang yang akan diproses sedangkan *Framming* berperan dalam pemilihan kualitas serat yang baik. Adapun aktivitas dari masing-masing departemen adalah sebagai berikut:

1. Aktivitas pada departemen *Purchase*
   - *Sending good receive*
2. Aktivitas pada departemen *Spinning*
   - *Getting good receive*
   - *Bale Opening*
   - *Conditioning of MMP Fiber*
   - *Blending*
   - *Carding*
   - *Combing*
   - *Ring framing*
   - *Cone winding*
3. Aktivitas pada departemen *Blowing*
   - *Opposing spike*
   - *Air current blowing*
   - *Striking cotton*
4. Aktivitas pada departemen *Framming*
   - *Drawing frame*
   - *Roving frame*

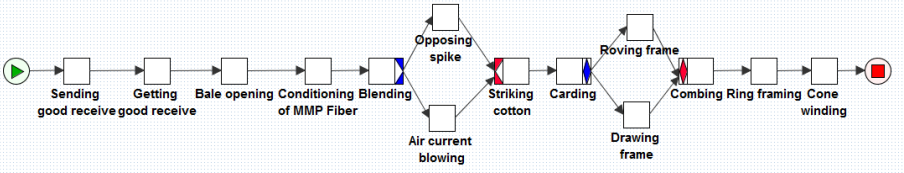
Bentuk *event log* dari data ini diperlihatkan pada Tabel 6.4.

Tabel 6.4 *Event Log* Data Produksi

Case ID	Activity	Time Stamp	Organisator	Cost
PP1	Sending good receive	6/20/2014 8:32	Purchase Department	243303
PP1	Getting good receive	6/20/2014 13:42	Spinning Department	138941
PP1	Bale opening	6/20/2014 23:41	Spinning Department	508599
PP1	Conditioning of MMP Fiber	6/21/2014 8:16	Spinning Department	63706
PP1	Blending	6/21/2014 10:46	Spinning Department	496301
PP1	Opposing spike	6/21/2014 16:57	Blowing Department	352814
PP1	Striking cotton	6/21/2014 18:09	Blowing Department	125149
PP1	Carding	6/21/2014 19:15	Spinning Department	957335
PP1	Roving frame	6/22/2014 10:51	Framing Department	516512
PP1	Drawing frame	6/22/2014 16:28	Framing Department	455652
PP1	Combing	6/22/2014 23:42	Spinning Department	394780
PP1	Ring framing	6/23/2014 4:48	Spinning Department	359985
PP1	Cone winding	6/23/2014 16:44	Spinning Department	319064

Pada Tabel 6.4 menunjukan salah satu *case* dari *event log* data produksi. Sebenarnya dalam event log data produksi terkandung 50

case produksi, dengan 8 jenis *trace.* Pada kolom aktivitas terlihat tedapat perbedaan warna, perbedaan ini menunjukkan bahwa aktivitas itu dikerjakan oleh organisasi yang berbeda.
Model dari *event log* data prodroduksi dapat dilihat pada Gambar 6.5. Hubungan antar aktivitas dari model data purchase dapat dilihat pada Tabel 6.5.



Gambar 6.5 Model Data Produksi

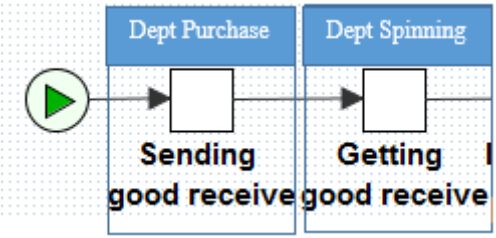
Tabel 6.5 Hubungan Antar Aktivitas dari Gambar 6.5

Input	Split / Join	Output
{Start}		Sending good receive
Sending good receive		Getting good receive
Getting good receive		Bale opening
Bale opening		Conditioning of MMP Fiber
Conditioning of MMP Fiber		Blending
Blending	XOR Split	Opposing spike, Air current blowing,
Opposing spike, Air current blowing,	XOR Join	Striking cotton
Striking cotton		Carding
Carding	OR Split	Roving frame, Drawing frame,
Drawing frame, Roving frame,	OR Join	Combing

Input	Split / Join	Output
Combing		Ring framing
Ring framing		Cone winding
Cone winding		{end}

Jenis pola koordinasi *cross-organizational* yang ada pada data *event log* produksi adalah sebagau berikut:

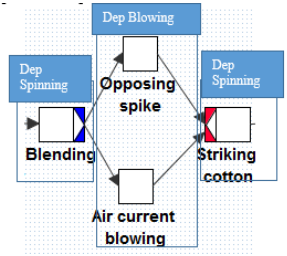
3. Pola Koordinasi Pertukaran Pesan
   Pola ini terjadi antara departemen *Purchase* dan *Spinning* pada saat aktivitas *sending* dan *getting good receive* saat itu terjadi pertukaran pesan berupa *good receive* apa saja yang diterima dan dikirimkan, pertukaran peran ini terjadi untuk melakukan pengecekan barang keluar dan masuk. Alur pertukaran pesan dapat dilihat pada Gambar 6.6.



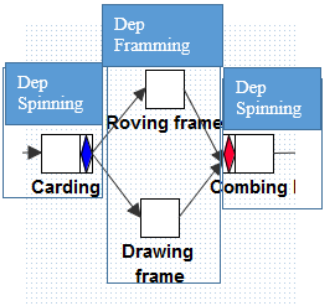
Gambar 6.6 Alur Pertukaran Pesan Data Produksi

4. Pola Koordinasi Aktivitas Sinkron
   Dalam data ini terdapat dua jenis pola hubungan aktivitas sinkron yaitu:
   - Pola hubungan *Capacity Sharing*
     Pola hubungan ini terjadi antara departemen *Spinning Blowing* dan *Spinning Framming*. Pada *Spinning Blowing* walaupun departemen *Blowing* yang melakukan pembersihan namun departemen *Spinning* yang mengatur bahan yang akan dibersihkan, begitu pula dengan

departemen *Framming,* walaupaun yang melakukan pemilihan bahan adalah departemen *Framming* namun keputusan untuk memilih bentuk pemilihan yang akan dilakukan tetap oleh departemen *Spinning.* Jadi departemen *Spinning* yang memegang control walaupun pengerjaannya dilakukan oleh departemen *Blowing* dan *Framming.* Alur hubungan *capacity sharing Spinning Blowing* akan diperlihatkan pada Gambar 6.7 dan *Spinning Framming* akan diperlihatkan pada Gambar 6.8.



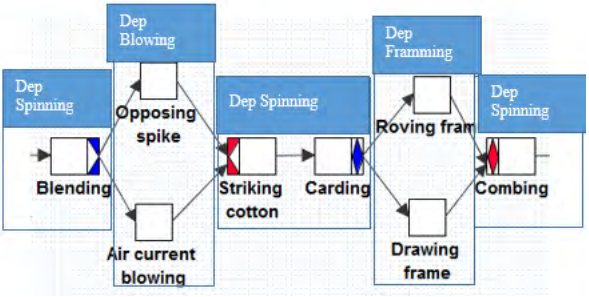
Gambar 6.7 Alur *Capacity Sharing Spinning Blowing* Data Produksi



Gambar 6.8 Alur *Capacity Sharing Spinning Framming* Data Produksi

- Pola hubungan *Chain Execution*
  Pola hubungan ini terjadi antara departemen *Spinning, Blowing,* dan *Framming.* Karena departemen ini harus

berjalan berurutan prosesnya. Alur proses tersebut dapat dilihat pada Gambar 6.9.



Gambar 6.9 Alur *Chain Execution*

6.4 Uji Kebenaran dan Hasil Uji Coba

Uji coba pada sistem ini mengacu pada pengujian *Blackbox* untuk menguji apakah fungsionalitas sistem telah berjalan sebagaimana mestinya. Pengujian mengacu pada setiap fitur yang telah diimplementasikan.

6.4.1. Pengujian Fungsionalitas

Pengujian fungsionalitas sistem dilakukan secara mandiri dengan menyiapkan sejumlah skenario sebagai tolak ukur keberhasilan pengujian. Pengujian fungsionalitas dilakukan dengan mengacu pada kasus penggunaan yang telah dijelaskan pada subbab 4.6. Pengujian pada kebutuhan fungsionalitas dapat dijabarkan pada subbab berikut.

6.4.1.1 Pengujian Fitur Memasukkan *Event Log*

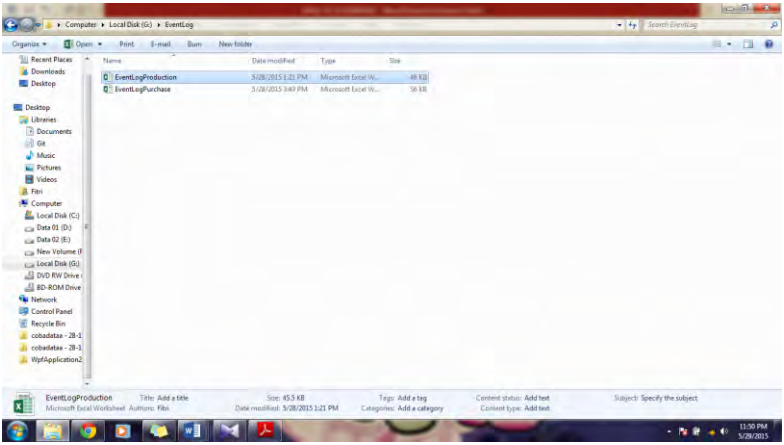
Pengujian fitur memasukkan *file event log* dilakukan dengan melakukan impor pada direktori yang di dalamnya ada *file event log*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.6.

Tabel 6.6 Pengujian Fitur Memasukkan Data *Event Log*

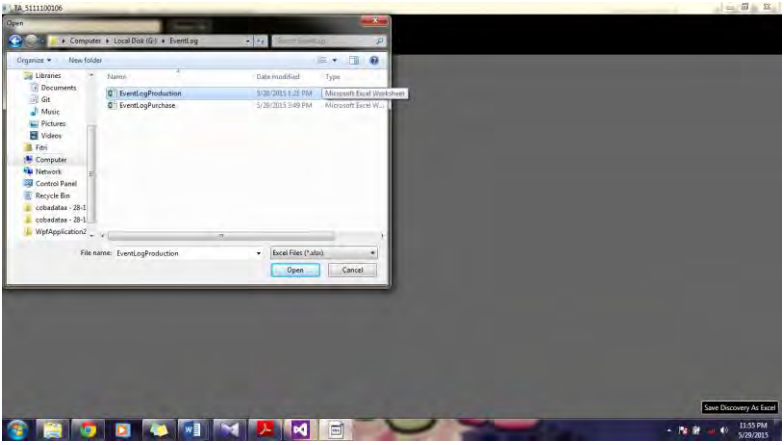
ID	**TA-UJ.UC0001**
Referensi Kasus Penggunaan	TA-UC0001

Nama	Pengujian fitur memasukkan *file event log*
Tujuan Pengujian	Menguji fitur untuk memasukkan *file event log* dengan memilih direktori tempat *file Petri Net* berada
Skenario 1	Pengguna memilih direktori tempat *file event log* berada dan sistem akan mengambil *file* tersebut
Kondisi Awal	• Sistem sudah dijalankan pada dan sistem menampilkan halaman Proses *Discovery* • Sistem menampilkan jendela pencari direktori
Data Uji	Data uji menggunakan direktori yang dibuat sendiri dan *file event log* yang dibuat di dalamnya
Langkah Pengujian	Pengguna memilih *file* yang sesuai pada jendela pencarian
Hasil Yang Diharapkan	Sistem mampu memasukkan *file event log* dengan menyimpan *path* dari *file* tersebut
Hasil Yang Didapat	*Path* dari *file event log* dapat digunakan untuk proses selanjutnya
Hasil Pengujian	100% Berhasil
Kondisi Akhir	*Path* dari *file event log* telah didapatkan oleh sistem

Dalam uji ini digunakan *file* EventLogProduction.xlsx seperti yang terlihat pada Gambar 6.10. Kemudian pengguna akan memilih *file event log* tersebut saat sistem menampilkan halaman Proses *Discovery*. Proses tersebut dapat dilihat pada Gambar 6.11. Kemudian untuk *file* yang telah berhasil diimpor dapat dilihat pada Gambar 6.12. Terisinya *textbox* dengan *path file* tersebut dapat disimpulkan bahwa sistem telah mampu mengambil *file event log* untuk proses selanjutnya.



Gambar 6.10 *File* EventLogProduction.xlsx pada Direktori



Gambar 6.11. Pengguna Memilih *File Event Log*

Gambar 6.12 *File Event Log* yang Telah Berhasil Diimpor

6.4.1.2 Pengujian Fitur Men-*discover* Proses Bisnis

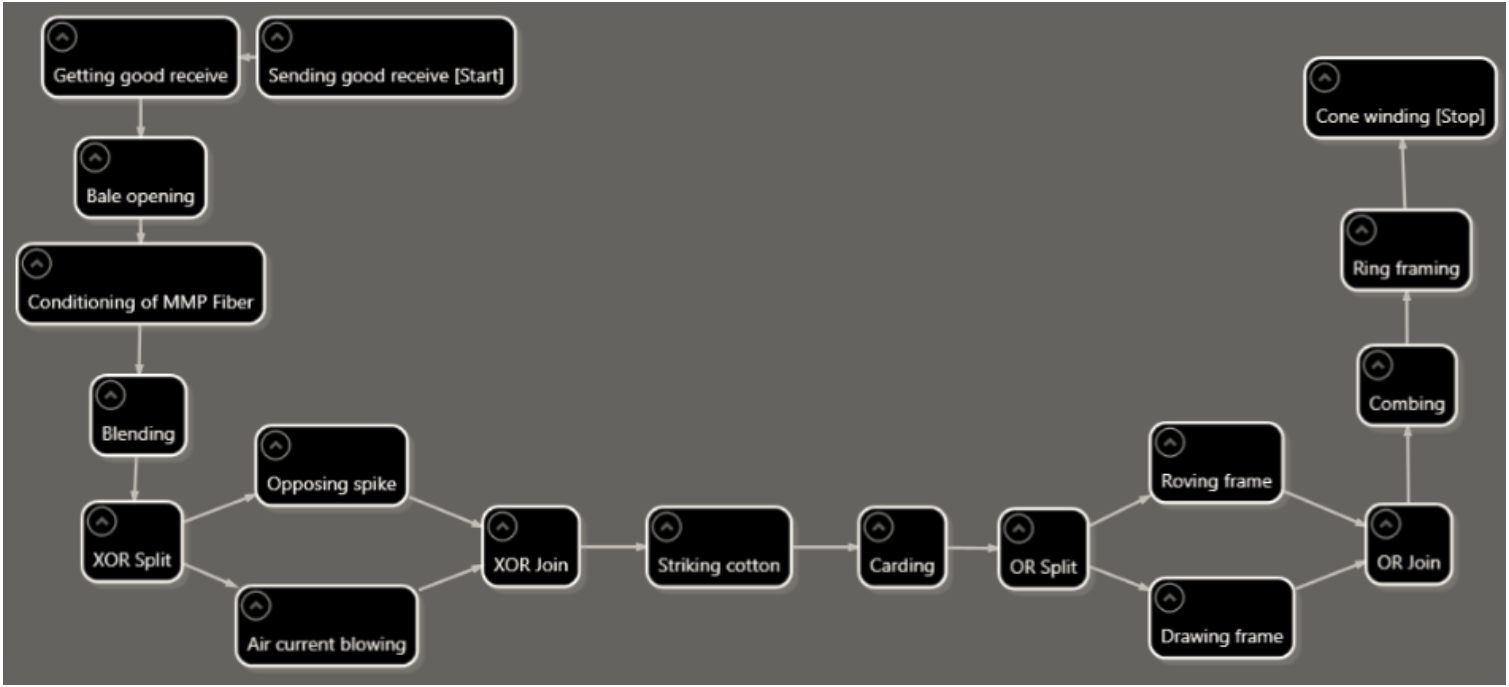
Pengujian fitur *discover* proses bisnis dilakukan dengan melakukan pengisian atribut-atribut yang tersedia pada halaman konfigurasi. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.7.

Tabel 6.7 Pengujian Fitur Konfigurasi BPMN

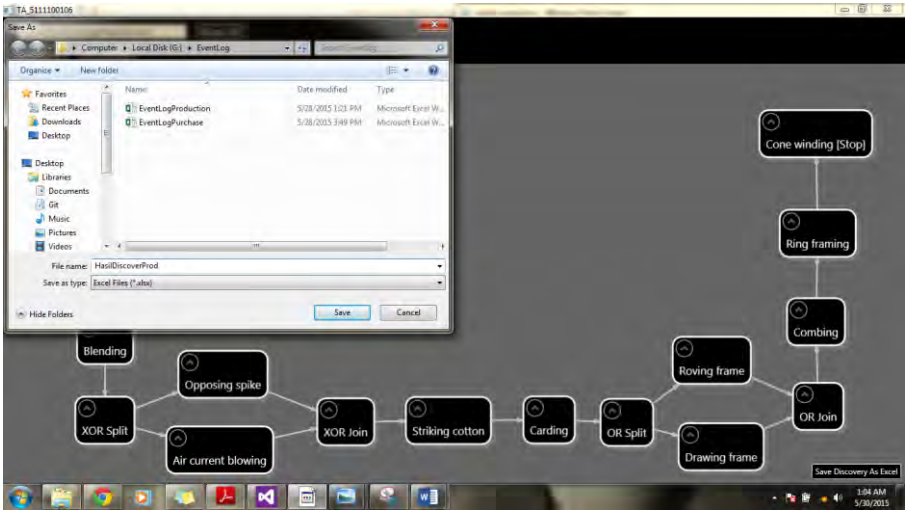
ID	**TA-UJ.UC0002**
Referensi Kasus Penggunaan	TA-UC0002
Nama	Pengujian fitur men-*discover* proses bisnis
Tujuan Pengujian	Menguji fitur untuk melakukan *discover* proses bisnis dari *event log* yang dimasukkan
Skenario 1	Pengguna menekan tombol *Discovery* setelah *textbox* terisi *path file*

Skenario 2	Pengguna menekan tombol *Save Discovery as* Excel setelah menekan tombol *Discovery*
Kondisi Awal	• Sistem sudah dijalankan dan sistem menampilkan halaman proses *discovery*
Data Uji	Data uji file *event log* yang telah dimasukkan
Langkah Pengujian	Setelah memasukkan *file* pengguna menekan tombol *Discovery*, setelah itu pengguna menekan tombol *Save Discovery as* Excel
Hasil Yang Diharapkan	Sistem mampu men-*discovery event log*
Hasil Yang Didapat	*Graph* model proses bisnis dari *event log* dan *file* Excel yang berisi tabel *input output* dan jenis paralel yang berhubungan dengan aktivitas.
Hasil Pengujian	100% Berhasil
Kondisi Akhir	*Graph* model proses bisnis *File* Excel tabel dari *graph*

Dalam uji ini digunakan *file* hasil dari masukan *event log* pada proses sebelumnya. Kemudian pengguna akan menekan tombol D*iscovery* dan kemudian menekan *Save Discovery as* Excel untuk menyimpan hasil *discovery* dalam bentuk Excel. Kemudian untuk *file* yang telah hasil *discovery* dalam bentuk *graph* dapat dilihat pada Gambar 6.13, proses penyimpanan dapat dilihat pada Gambar 6.14 dan Gambar 6.15 menunjukkan isi dari *file* Excel yang disimpan dari hasil proses *discovery*.



Gambar 6.13. Hasil *Discovery* EventLogProduction.xlsx



Gambar 6.14. Penyimpanan *File* Hasil *Discovery*

Input	Split/Join	Output		OneLoop	TwoLoop
{start}		Sending good receive			
Sending good receive		Getting good receive			
Getting good receive		Bale opening			
Bale opening		Conditioning of MMP Fiber			
Conditioning of MMP Fiber		Blending			
Blending	XOR Split	Opposing spike, Air current blowing,			
Opposing spike, Air current blowing,	XOR Join	Striking cotton			
Striking cotton		Carding			
Carding	OR Split	Roving frame, Drawing frame,			
Drawing frame, Roving frame,	OR Join	Combing			
Combing		Ring framing			
Ring framing		Cone winding			
Cone winding		{end}			

Gambar 6.15. Isi *File* Hasil *Discovery*

6.4.1.3 Pengujian Fitur Menghitung Data Optimasi

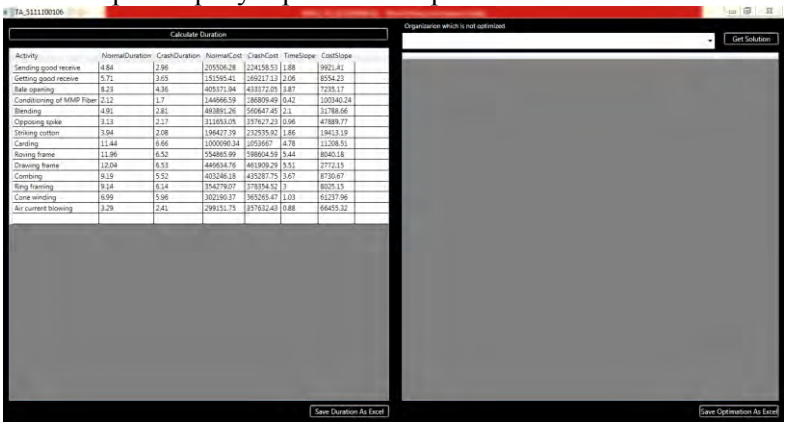
Pengujian menghitng data optimasi dilakukan dengan menekan tombol *Get Duration* sehingga menampilkan halaman optimasi kemudian pada halaman optimasi pengguna menekan tombol *Calculate Duration*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.8.

Tabel 6.8. Pengujian Fitur Menghitung Data Optimasi

ID	**TA-UJ.UC0003**
Referensi Kasus Penggunaan	TA-UC0003
Nama	Pengujian fitur menghitung data optimasi
Tujuan Pengujian	Menguji fitur untuk melakukan perhitungan data optimasi
Skenario 1	Pengguna menekan tombol *Calculate Duration*
Skenario 2	Pengguna menekan tombol *Save Duration as* Excel setelah menekan tombol *Calculate Duration*
Kondisi Awal	• Sistem sudah dijalankan pada dan sistem menampilkan halaman optimasi
Data Uji	Data uji file *event log* yang telah dimasukkan
Langkah Pengujian	Pengguna menekan tombol *Calculate Duration* pada halaman optimasi
Hasil Yang Diharapkan	Sistem mampu menghasilkan perhitungan data optimasi berupa *normal duration, crash duration, normal cost, crash cost, time slope,* dan *cost slope*
Hasil Yang Didapat	Data optimasi berupa *normal duration, crash duration, normal cost, crash cost, time slope, dan cost slope*
Hasil Pengujian	100% Berhasil
Kondisi Akhir	Tabel data optimasi berupa *normal duration, crash duration, normal cost, crash cost, time slope, dan cost slope* baik tampilan pada sistem maupun hasil penyimpanan dalam bentuk Excel

Dalam uji ini digunakan *file* hasil dari masukan *event log* pada proses sebelumnya. Kemudian pengguna akan menekan tombol *Calculate Duration* dan kemudian menekan *Save Duration as* Excel untuk menyimpan hasil perhitungan data optimasi dalam

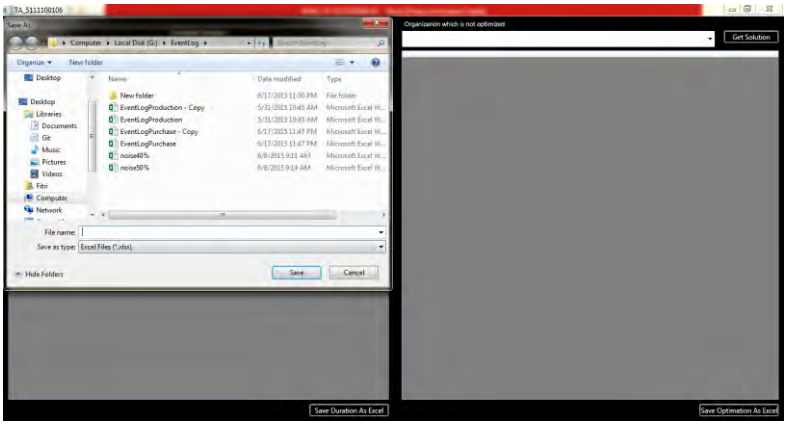
bentuk Excel. Hasil perhitungan data optimasi beserta halaman optimasi secara penuh dapat dilihat pada Gambar 6.16, hasil perhitungan data optimasinya sendiri dapat dilihat pada Gambar 6.17 dan proses penyimpanan data optimasi Gambar 6.18.



Gambar 6.16. Halaman Optimasi dan Hasil Perhitungan Data Optimasi

Activity	NormalDuration	CrashDuration	NormalCost	CrashCost	TimeSlope	CostSlope
Sending good receive	4.84	2.96	205506.28	224158.53	1.88	9921.41
Getting good receive	5.71	3.65	151595.41	169217.13	2.06	8554.23
Bale opening	8.23	4.36	405371.94	433372.05	3.87	7235.17
Conditioning of MMP Fiber	2.12	1.7	144666.59	186809.49	0.42	100340.24
Blending	4.91	2.81	493891.26	560647.45	2.1	31788.66
Opposing spike	3.13	2.17	311653.05	357627.23	0.96	47889.77
Striking cotton	3.94	2.08	196427.39	232535.92	1.86	19413.19
Carding	11.44	6.66	1000090.34	1053667	4.78	11208.51
Roving frame	11.96	6.52	554865.99	598604.59	5.44	8040.18
Drawing frame	12.04	6.53	446634.76	461909.29	5.51	2772.15
Combing	9.19	5.52	403246.18	435287.75	3.67	8730.67
Ring framing	9.14	6.14	354279.07	378354.52	3	8025.15
Cone winding	6.99	5.96	302190.37	365265.47	1.03	61237.96
Air current blowing	3.29	2.41	299151.75	357632.43	0.88	66455.32

Gambar 6.17. Hasil Perhitungan Data Optimasi



Gambar 6.18. Penyimpanan *File* Data Optimasi

6.4.1.4 Pengujian Fitur Melakukan Optimasi Biaya dan *Makespan*

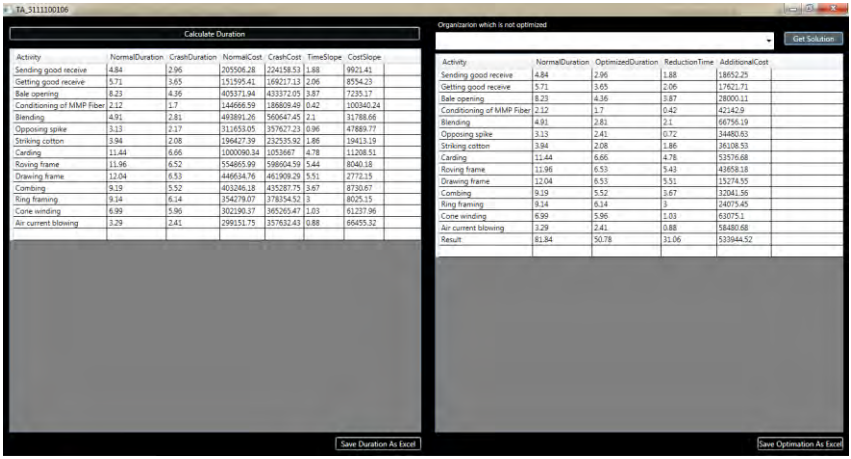
Pengujian menghitng data optimasi dilakukan dengan menekan tombol *Get Duration* sehingga menampilkan halaman optimasi kemudian pada halaman optimasi pengguna menekan tombol *Get Solution*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.9.

Tabel 6.9. Pengujian Fitur Melakukan Optimasi Biaya dan *Makespan*

ID	**TA-UJ.UC0004**
Referensi Kasus Penggunaan	TA-UC0004
Nama	Pengujian fitur melakukan optimasi biaya dan makespan
Tujuan Pengujian	Menguji fitur untuk melakukan optimasi biaya dan makespan
Skenario 1	Pengguna menekan tombol *Get Solution*
Kondisi Awal	• Sistem sudah dijalankan pada dan sistem menampilkan halaman optimasi

Data Uji	Data uji file *event log* yang telah dimasukkan
Langkah Pengujian	Pengguna menekan tombol *Get Solution* pada halaman optimasi
Hasil Yang Diharapkan	Sistem mampu menghasilkan perhitungan durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit
Hasil Yang Didapat	Durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit
Hasil Pengujian	100% Berhasil
Kondisi Akhir	Durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit

Dalam uji ini digunakan *file* hasil dari masukan *event log* pada proses sebelumnya. Kemudian pengguna akan menekan tombol *Get Solution*. Hasil perhitungan durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit beserta halaman optimasi secara penuh dapat dilihat pada Gambar 6.19, hasil perhitungan durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit sendiri dapat dilihat pada Gambar 6.20.



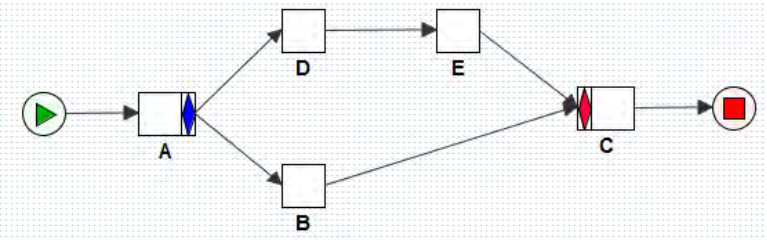
Gambar 6.19. Halaman Optimasi dan Durasi Proses Bisnis yang Paling Kecil dan Biaya Tambahan yang Paling Sedikit

Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
Sending good receive	4.84	2.96	1.88	18652.25
Getting good receive	5.71	3.65	2.06	17621.71
Bale opening	8.23	4.36	3.87	28000.11
Conditioning of MMP Fiber	2.12	1.7	0.42	42142.9
Blending	4.91	2.81	2.1	66756.19
Opposing spike	3.13	2.41	0.72	34480.63
Striking cotton	3.94	2.08	1.86	36108.53
Carding	11.44	6.66	4.78	53576.68
Roving frame	11.96	6.53	5.43	43658.18
Drawing frame	12.04	6.53	5.51	15274.55
Combing	9.19	5.52	3.67	32041.56
Ring framing	9.14	6.14	3	24075.45
Cone winding	6.99	5.96	1.03	63075.1
Air current blowing	3.29	2.41	0.88	58480.68
Result	81.84	50.78	31.06	533944.52

Gambar 6.20. Hasil Optimasi Berupa Durasi Proses Bisnis yang Paling Kecil dan Biaya Tambahan yang Paling Sedikit

6.4.2. Pengujian Validitas Hasil

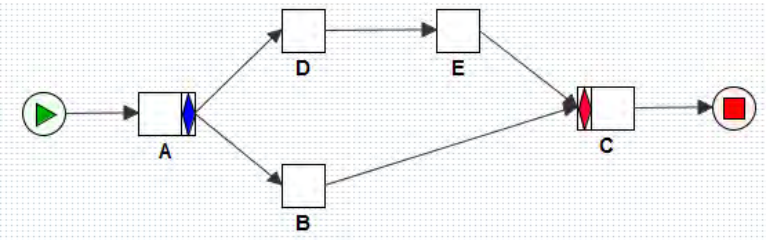
Pada bagian ini akan dijelaskan pengujian validitas hasil sistem. Uji coba dilakukan dengan bantuan kakas YAWL, Excel, dan LINGO. Dalam pengujian ini yang pertama dilakukan adalah membentuk model dengan menggunakan YAWL, model dapat dilihat pada Gambar 6.21. Setelah itu dibangkitkan *event log* untuk model tersebut, *event log* yang dibangkitkan dari model YAWL kemudian dirubah dalam bentuk Excel dan ditambahkan atribut pendukung yang tidak terdapat dalam *event log* hasil dari YAWL seperti *cost* teteapi banyak *trace*, aktivitas, dan *timestamp* tetap sama sesuai dengan *event log* yang dibangkitkan dari YAWL. *Event log* dari YAWL dapat dilihat pada Gambar 6.22 dan *event log* yang sidah dirubah dalam bentuk Excel dapat dilihat pada Tabel 6.10.



Gambar 6.21 Model YAWL

6.4.2. Pengujian Validitas Hasil

Pada bagian ini akan dijelaskan pengujian validitas hasil sistem. Uji coba dilakukan dengan bantuan kakas YAWL, Excel, dan LINGO. Dalam pengujian ini yang pertama dilakukan adalah membentuk model dengan menggunakan YAWL, model dapat dilihat pada Gambar 6.21. Setelah itu dibangkitkan *event log* untuk model tersebut, *event log* yang dibangkitkan dari model YAWL kemudian dirubah dalam bentuk Excel dan ditambahkan atribut pendukung yang tidak terdapat dalam *event log* hasil dari YAWL seperti *cost* teteapi banyak *trace*, aktivitas, dan *timestamp* tetap sama sesuai dengan *event log* yang dibangkitkan dari YAWL. *Event log* dari YAWL dapat dilihat pada Gambar 6.22 dan *event log* yang sidah dirubah dalam bentuk Excel dapat dilihat pada Tabel 6.10.



Gambar 6.21 Model YAWL

```
<trace>
    <string key="concept:name" value="300.PO.IMP.201408.0021"/>
    <event>
        <string key="Operation" value="A"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="A"/>
        <date key="time:timestamp" value="2014-08-27T10:30:42+07:00"/>
    </event>
    <event>
        <string key="Operation" value="B"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="B"/>
        <date key="time:timestamp" value="2014-08-27T10:30:53+07:00"/>
    </event>
    <event>
        <string key="Operation" value="D"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="D"/>
        <date key="time:timestamp" value="2014-08-27T11:31:34+07:00"/>
    </event>
    <event>
        <string key="Operation" value="E"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="E"/>
        <date key="time:timestamp" value="2014-08-27T12:31:34+07:00"/>
    </event>
    <event>
        <string key="Operation" value="C"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="C"/>
        <date key="time:timestamp" value="2014-08-27T14:31:34+07:00"/>
    </event>
```

Gambar 6.22 Bentuk *Event Log* YAWL

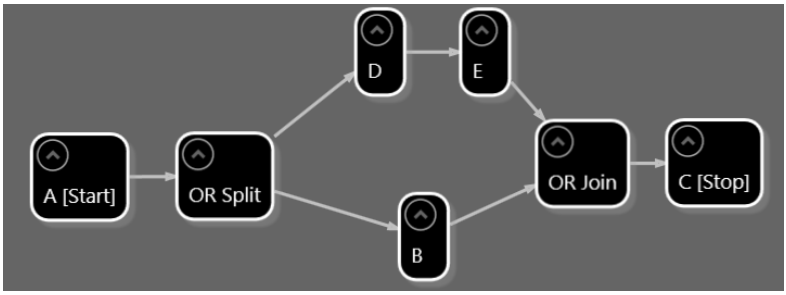
Tabel 6.10 Bentuk *Event Log* YAWL yang Dirubah Excel

Case ID	Activity	TimeStamp	Originator	Cost
300.PO.IMP.201406.0024	A	6/20/14 12:32	AN	400
300.PO.IMP.201406.0024	D	6/20/14 13:10	AN	1500
300.PO.IMP.201406.0024	B	6/20/14 15:00	AN	1000
300.PO.IMP.201406.0024	E	6/20/14 16:00	AN	2000
300.PO.IMP.201406.0024	C	6/20/14 18:00	AN	1200
300.PO.IMP.201406.0015	A	6/20/14 11:32	AN	1000
300.PO.IMP.201406.0015	B	6/20/14 12:32	AN	400
300.PO.IMP.201406.0015	C	6/20/14 13:10	AN	500
300.PO.IMP.201408.0021	A	8/27/14 10:30	AN	50
300.PO.IMP.201408.0021	B	8/27/14 10:30	AN	1000
300.PO.IMP.201408.0021	D	8/27/14 11:31	AN	1000
300.PO.IMP.201408.0021	E	8/27/14 12:31	AN	2000
300.PO.IMP.201408.0021	C	8/27/14 14:31	AN	1000
300.PO.IMP.201406.0025	A	6/20/14 12:32	AN	400
300.PO.IMP.201406.0025	D	6/20/14 13:10	AN	1000
300.PO.IMP.201406.0025	E	6/20/14 14:10	AN	500
300.PO.IMP.201406.0025	B	6/20/14 15:00	AN	500
300.PO.IMP.201406.0025	C	6/20/14 15:50	AN	500

6.3.2.1. Pengujian Validitas Hasil *Discovery*

Pada bagian ini akan dijelaskan pengujian hasil *discovery event log* menjadi model proses bisnis. Pengecekan dengan membandingkan model awal yang dibuat pada YAWL pada

Gambar 6.21 harus sama dengan hasil proses *discovery*. Perbandingan dilakukan dengan membandingkan hubungan antar aktivitas dan percabangan yang digunakan. Hubungan antar aktivitas pada model YAWL dapat dilihat pada Tabel 6.11. Hasil proses *discovery* dapat dilihat pada Gambar 6.23 dan Hubungan antar aktivitas pada model hasil proses *discovery* dapat dilihat pada Tabel 6.12.



Gambar 6.23 Bentuk Model Hasil *Discovery*

Tabel 6.11 Hubungan Aktivitas Model YAWL Gambar 6.21

Input	Split atau Join	Output
A	OR Split	D, B
D		E
E, B	OR Join	C

Tabel 6.12 Hubungan Aktivitas Model Hasil *Discovery* Gambar 6.23

Input	Split atau Join	Output
A	OR Split	D, B
D		E
E, B	OR Join	C

Dari Tabel 6.11 dan Tabel 6.12 dapat dilihat bahwa kedua tabel tersebut sama hal ini menandakan bahwa model hasil *discovery* benar.

6.3.2.2. Pengujian Validitas Hasil Perhitungan Data untuk Optimasi

Pada bagian ini akan dijelaskan pengujian hasil perhitungan data untuk optimasi. Pengecekan dilakukan dengan membandingkan perhitungan secara manual dengan menggunakan Excel dan kemudian hasilnya dibandingkan dengan hasil perhitungan menggunakan program. Data langkah perhitungan Excel dapat dilihat pada Tabel 6.13, hasil perhitungan Excel dapat dilihat pada Tabel 6.14, hasil perhitungan program dapat dilihat pada Tabel 6.15.

Tabel 6.13 Langkah Perhitungan Excel

Case ID	Activity	TimeStamp	Originator	Cost	Durasi Per Case									
300.PO.IMP.201406.0024	A	6/20/14 12:32	AN	400	0.634166667		Aktivitas	Rata-rata Dur	SD Dur		0.63	2.828888889	3	2	2.115764
300.PO.IMP.201406.0024	D	6/20/14 13:10	AN	1500	2.828888889		A	0.57	0.42		1	1	0.634166667	2	0.8175
300.PO.IMP.201406.0024	B	6/20/14 15:00	AN	1000	3		B	2.12	1.65		0	1	4.011388889	1.666667	1.753611
300.PO.IMP.201406.0024	E	6/20/14 16:00	AN	2000	2		C	1.43	0.61		0.63		0.833333333		1.033542
300.PO.IMP.201406.0024	C	6/20/14 18:00	AN	1200	2.115763889		D	1.61	1.06						
300.PO.IMP.201406.0015	A	6/20/14 11:32	AN	1000	1.000833333		E	1.89	0.19						
300.PO.IMP.201406.0015	B	6/20/14 12:32	AN	400	0.634166667			Rata-rata Cos	SD Cost						
300.PO.IMP.201406.0015	C	6/20/14 13:10	AN	500	0.8175		A	463	394.49		400	1500	1000	2000	1200
300.PO.IMP.201408.0021	A	8/27/14 10:30	AN	50	0.003055556		B	725	320.16		1000	1000	400	2000	500
300.PO.IMP.201408.0021	B	8/27/14 10:30	AN	1000	4.011388889		C	800	355.90		50	1000	1000	500	1000
300.PO.IMP.201408.0021	D	8/27/14 11:31	AN	1000	1		D	1167	288.68		400		500		500
300.PO.IMP.201408.0021	E	8/27/14 12:31	AN	2000	2		E	1500	866.03						
300.PO.IMP.201408.0021	C	8/27/14 14:31	AN	1000	1.753611111		Aktivitas	Normal Dur	Crash Dur	Normal Cost	Crash Cost	Time Slope	Cost Slope		
300.PO.IMP.201406.0025	A	6/20/14 12:32	AN	400	0.634166667		A	0.57	0.15	462.50	856.99	0.42	939.26		
300.PO.IMP.201406.0025	D	6/20/14 13:10	AN	1000	1		B	2.12	0.47	725.00	1045.16	1.65	194.04		
300.PO.IMP.201406.0025	E	6/20/14 14:10	AN	500	1.666666667		C	1.43	0.82	800.00	1155.90	0.61	583.44		
300.PO.IMP.201406.0025	B	6/20/14 15:00	AN	500	0.833333333		D	1.61	0.55	1166.67	1455.35	1.06	272.34		
300.PO.IMP.201406.0025	C	6/20/14 15:50	AN	500	1.033541667		E	1.89	1.70	1500.00	2366.03	0.19	4558.05		

Tabel 6.14 Hasil Perhitungan Excel

Aktivitas	Normal Dur	Crash Dur	Normal Cost	Crash Cost	Time Slope	Cost Slope
A	0.57	0.15	462.50	856.99	0.41	952
B	2.12	0.46	725.00	1045.16	1.65	193
C	1.43	0.82	800.00	1155.90	0.61	586
D	1.61	0.55	1166.67	1455.34	1.06	273
E	1.89	1.70	1500.00	2366.03	0.19	4500

Tabel 6.15 Hasil Perhitungan Program

Activity	NormalDuration	CrashDuration	NormalCost	CrashCost	TimeSlope	CostSlope
A	0.57	0.15	462.5	856.99	0.42	939.26
B	2.12	0.47	725	1045.16	1.65	194.04
C	1.43	0.82	800	1155.9	0.61	583.44
D	1.61	0.55	1166.67	1455.35	1.06	272.34
E	1.89	1.7	1500	2366.03	0.19	4558.05

Dari Tabel 6.14 dan Tabel 6.15 dapat dilihat bahwa kedua tabel tersebut sama hal ini menandakan bahwa model hasil perhitungan benar.

6.3.2.3. Pengujian Validitas Hasil Optimasi (Minimum Durasi dan Biaya Tambahan)

Pada bagian ini akan dijelaskan pengujian hasil optimasi (minimum durasi dan biaya tambahan). Pengecekan dilakukan dengan membandingkan hasil optimasi jika menggunakan LINGO dan kemudian hasilnya dibandingkan dengan hasil optimasi menggunakan program. Data hasil optimasi LINGO dapat dilihat pada Gambar 6.24, hasil optimasi program dapat dilihat pada Gambar 6.25.

```
Global optimal solution found.
Objective value:                              1905.100
Infeasibilities:                              0.000000
Total solver iterations:                             12
Elapsed runtime seconds:                           0.05

Model Class:                                       LP

Total variables:                     11
Nonlinear variables:                  0
Integer variables:                    0

Total constraints:                   23
Nonlinear constraints:                0

Total nonzeros:                      39
Nonlinear nonzeros:                   0


                     Variable           Value
                           XA        0.4200000
                           XD        1.060000
                           XB        0.000000
                           XE        0.1900000
                           XC        0.6100000
                           YA        0.000000
                           YD        0.1500000
                           YB        0.2800000
                           YE        0.7000000
                           YC        2.400000
                      YFINISH        3.220000
```

Gambar 6.24 Hasil Optimasi LINGO

Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
A	0.57	0.15	0.42	394.49
D	1.61	0.55	1.06	288.68
B	2.12	2.12	0	0
E	1.89	1.7	0.19	866.03
C	1.43	0.82	0.61	355.9
Result	5.5	3.22	2.28	1905.1

Gambar 6.25 Hasil Optimasi Program

Dari Gambar 6.24 dan Gambar 6.25 dapat dilihat bahwa kedua menghasilkan nilai yang sama, hal ini menandakan bahwa hasil optimasi perhitungan benar.

6.4.3. Hasil Uji Terhadap Data Studi Kasus

Pada bagian ini akan diperlihatkan hasil *running* sistem dengan masukkan data studi kasus seperti yang ada pada subbab 6.2.

6.3.3.1. *Hasil Data Studi Kasus* Event Log Purchase Order *Bahan Baku Benang*

Data studi kasus ini telah dijelaskan pada subbab 6.3.1. dalam susbab ini akan diperlihatkan hasil dari *running* sistem menggunakan studi kasus tersebut. Pada Gambar 6.26 menunjukkan hasil proses *discovery* dari *event log purchase order,* kemudian hasil penyimpanan dari proses *discovery* ditunjukkan pada Tabel 6.16. Kesamaan antara model dan hasil *discovery* juga ditunjukkan oleh Tabel 6.17. Untuk data optimasi hasil perhitungan ditunjukkan pada Gambar 6.27 dan hasil optimasi ditunjukkan pada Gambar 6.28.

Hasil proses *discovery* pada sistem yang ditunjukkan Gambar 6.26 sama dengan model awal *event log* pada gambar Gambar 6.1.

Dari hasil optimasi pada Gambar 6.28 diketahui bahwa durasi awal proses bisnis adalah 152.34 jam kemudian dipercepat hingga 84.69 jam dengan biaya tambahan sebesar 336760.76 rupiah.

Gambar 6.26 Hasil Model Proses *Discovery Event Log* Studi Kasus *Purchase Order*

Tabel 6.16 Hasil *Save* Model Proses *Discovery Event Log* ke Excel Studi Kasus *Purchase Order*

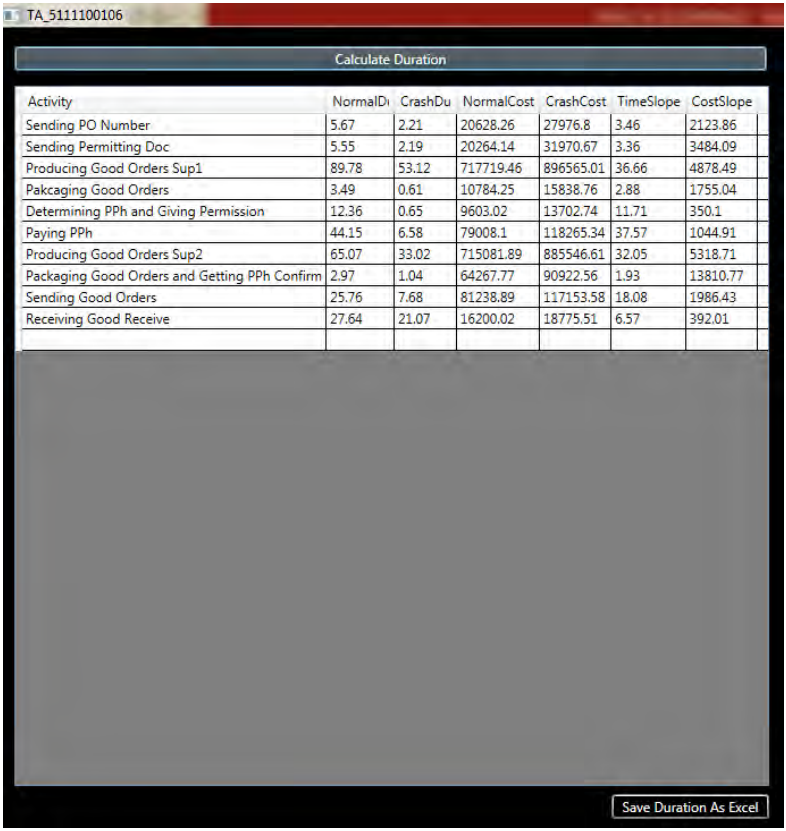
Input	Split/Join	Output		OneLoop	TwoLoop
{start}		Sending PO Number			
Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,			
Producing Good Orders Sup1		Pakcaging Good Orders			
Sending Permitting Doc		Determining PPh and Giving Permission			
Determining PPh and Giving Permission	AND Split	Paying PPh, Producing Good Orders Sup2,			
Producing Good Orders Sup2, Paying PPh,	AND Join	Packaging Good Orders and Getting PPh Confirm			
Packaging Good Orders and Getting PPh Confirm, Pakcaging Good Orders,	OR Join	Sending Good Orders			
Sending Good Orders		Receiving Good Receive			
Receiving Good Receive		{end}			

Tabel 6.17 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

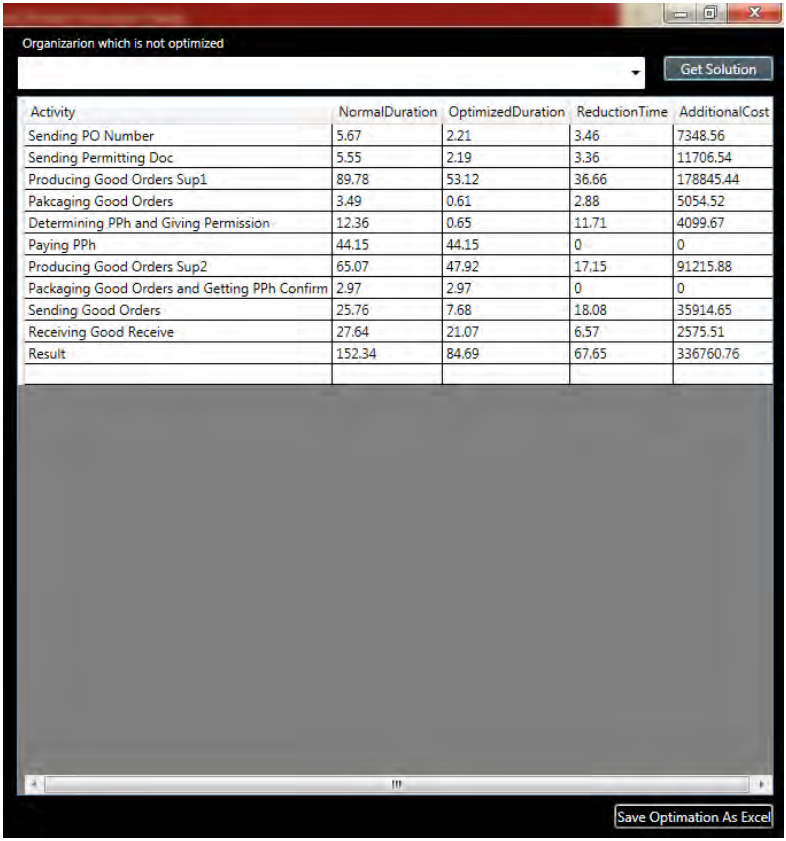
Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
{Start}		Sending PO Number	{Start}		Sending PO Number	Sama
Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,	Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,	Sama
Producing Good Orders Sup1		Pakcaging Good Orders	Producing Good Orders Sup1		Pakcaging Good Orders	Sama
Sending Permitting Doc		Determining PPh and Giving Permission	Sending Permitting Doc		Determining PPh and Giving Permission	Sama
Determining PPh and	AND Split	Paying PPh, Producing	Determining PPh and	AND Split	Paying PPh, Producing	

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
Giving Permission		Good Orders Sup2,	Giving Permission		Good Orders Sup2,	
Producing Good Orders Sup2, Paying PPh	AND Join	Packaging Good Orders and Getting PPh Confirm	Producing Good Orders Sup2, Paying PPh	AND Join	Packaging Good Orders and Getting PPh Confirm	Sama
Packaging Good Orders and Getting PPh Confirm, Pakcaging Good Orders	OR Join	Sending Good Orders	Packaging Good Orders and Getting PPh Confirm, Pakcaging Good Orders	OR Join	Sending Good Orders	Sama
Sending Good Orders		Receiving Good Receive	Sending Good Orders		Receiving Good Receive	Sama

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
Receiving Good Receive		{end}	Receiving Good Receive		{end}	Sama



Gambar 6.27 Hasil Data Optimasi *Event Log* Studi Kasus *Purchase Order*



Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
Sending PO Number	5.67	2.21	3.46	7348.56
Sending Permitting Doc	5.55	2.19	3.36	11706.54
Producing Good Orders Sup1	89.78	53.12	36.66	178845.44
Pakcaging Good Orders	3.49	0.61	2.88	5054.52
Determining PPh and Giving Permission	12.36	0.65	11.71	4099.67
Paying PPh	44.15	44.15	0	0
Producing Good Orders Sup2	65.07	47.92	17.15	91215.88
Packaging Good Orders and Getting PPh Confirm	2.97	2.97	0	0
Sending Good Orders	25.76	7.68	18.08	35914.65
Receiving Good Receive	27.64	21.07	6.57	2575.51
Result	152.34	84.69	67.65	336760.76

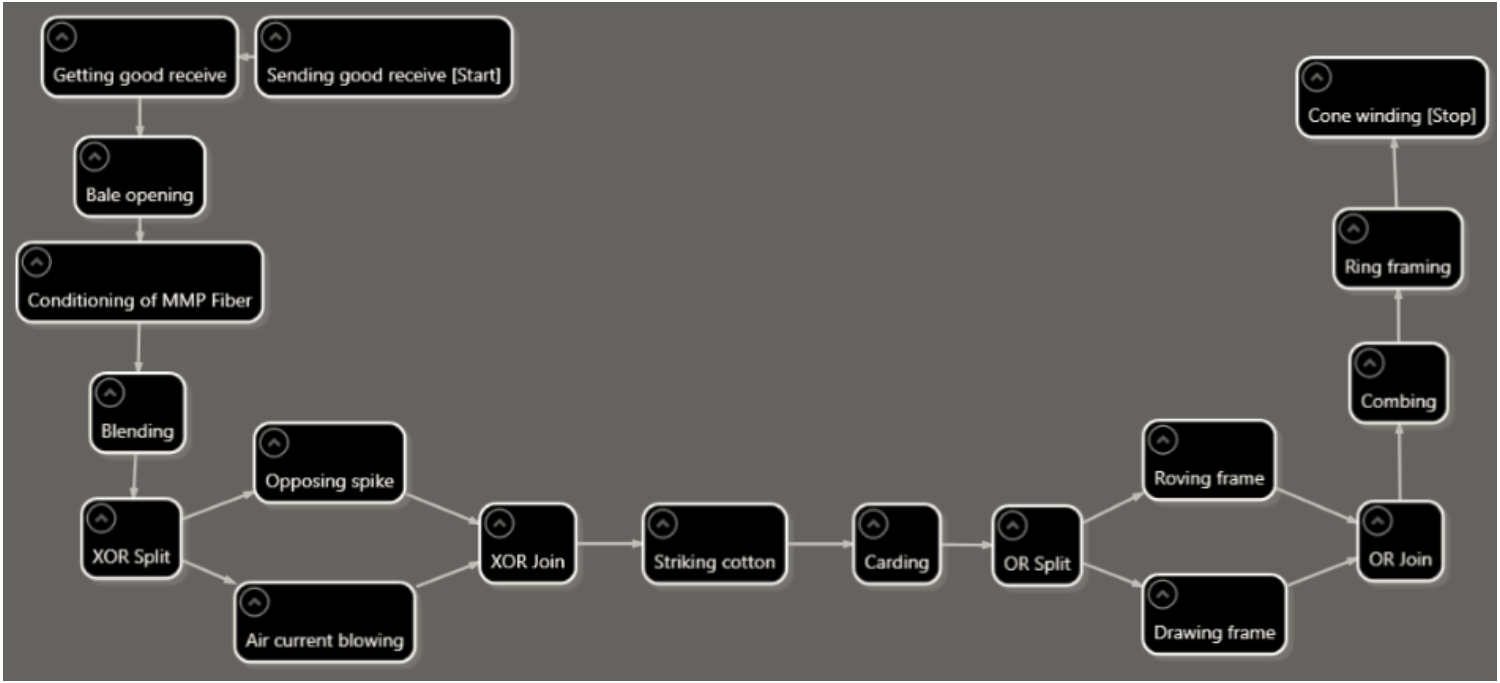
Gambar 6.28 Hasil Optimasi *Event Log* Studi Kasus *Purchase Order*

6.3.3.2. *Hasil Data Studi Kasus* Event Log *Produksi Benang*

Data studi kasus ini telah dijelaskan pada subbab 6.3.2. dalam subbab ini akan diperlihatkan hasil dari *running* sistem menggunakan studi kasus tersebut. Pada Gambar 6.29 menunjukkan hasil proses *discovery* dari *event log* produksi, kemudian hasil penyimpanan dari proses *discovery* ditunjukkan pada Tabel 6.19. Kesamaan antara model dan hasil *discovery* juga ditunjukkan oleh Tabel 6.18. Untuk data optimasi hasil perhitungan ditunjukkan pada Gambar 6.30 dan hasil optimasi ditunjukkan pada Gambar 6.31.

Hasil proses *discovery* pada sistem yang ditunjukkan Gambar 6.29 sama dengan model awal *event log* pada gambar Gambar 6.5.

Dari hasil optimasi pada Gambar 6.31 diketahui bahwa durasi awal proses bisnis adalah 81.44 jam kemudian dipercepat hingga 50.78 jam dengan biaya tambahan sebesar 533944.52 rupiah.



Gambar 6.29 Hasil Model Proses *Discovery Event Log* Studi Kasus Produksi

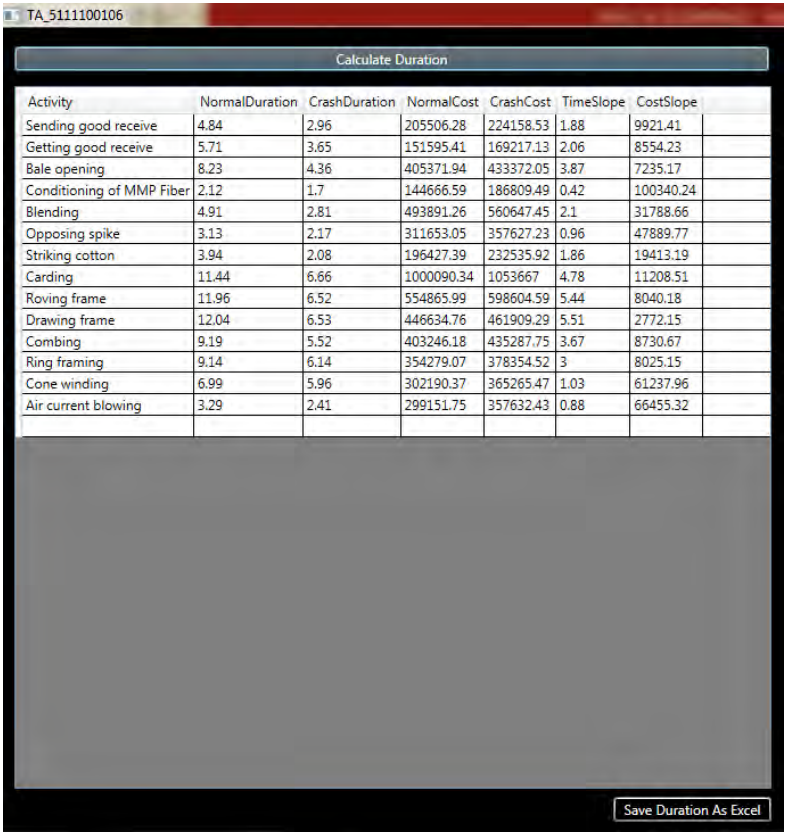
Tabel 6.18 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
{Start}		Sending good receive	{Start}		Sending good receive	Sama
Sending good receive		Getting good receive	Sending good receive		Getting good receive	Sama
Getting good receive		Bale opening	Getting good receive		Bale opening	Sama
Bale opening		Conditioning of MMP Fiber	Bale opening		Conditioning of MMP Fiber	Sama
Conditioning of MMP Fiber		Blending	Conditioning of MMP Fiber		Blending	
Blending	XOR Split	Opposing spike, Air current blowing,	Blending	XOR Split	Opposing spike, Air current blowing,	Sama
Opposing spike, Air	XOR Join	Striking cotton	Opposing spike, Air	XOR Join	Striking cotton	Sama

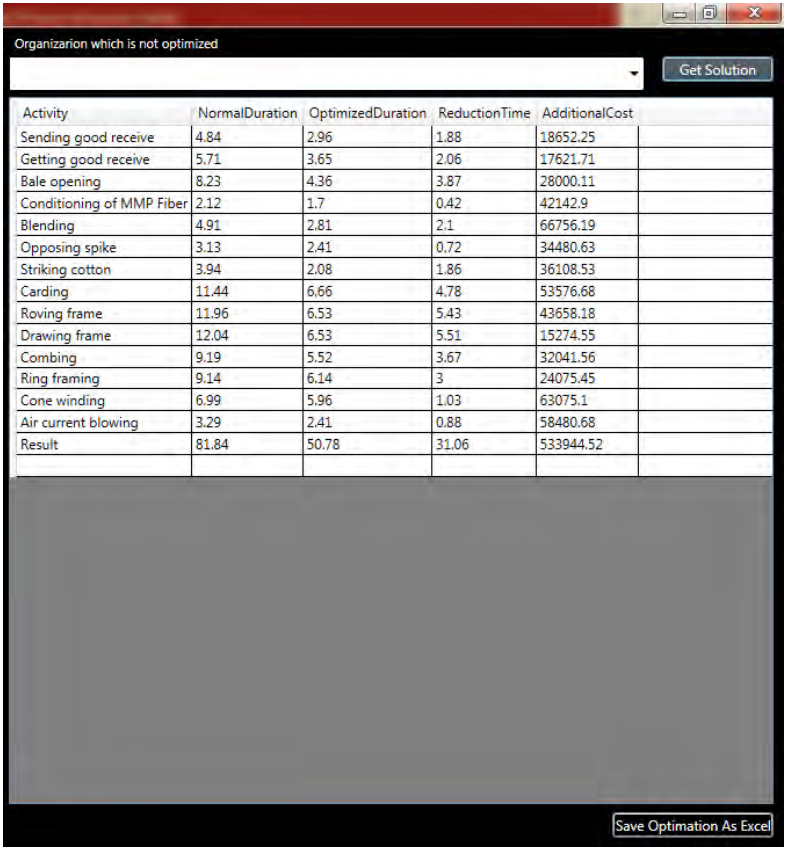
Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
current blowing,			current blowing,			
Striking cotton		Carding	Striking cotton		Carding	Sama
Carding	OR Split	Roving frame, Drawing frame,	Carding	OR Split	Roving frame, Drawing frame,	Sama
Drawing frame, Roving frame,	OR Join	Combing	Drawing frame, Roving frame,	OR Join	Combing	Sama
Combing		Ring framing	Combing		Ring framing	Sama
Ring framing		Cone winding	Ring framing		Cone winding	Sama
Cone winding		{end}	Cone winding		{end}	Sama

Tabel 6.19 Hasil *Save* Model Proses *Discovery Event Log* ke Excel Studi Kasus Produksi

Input	Split/Join	Output		OneLoop	TwoLoop
{start}		Sending good receive			
Sending good receive		Getting good receive			
Getting good receive		Bale opening			
Bale opening		Conditioning of MMP Fiber			
Conditioning of MMP Fiber		Blending			
Blending	XOR Split	Opposing spike, Air current blowing,			
Opposing spike, Air current blowing,	XOR Join	Striking cotton			
Striking cotton		Carding			
Carding	OR Split	Roving frame, Drawing frame,			
Drawing frame, Roving frame,	OR Join	Combing			
Combing		Ring framing			
Ring framing		Cone winding			
Cone winding		{end}			



Gambar 6.30 Hasil Data Optimasi *Event Log* Studi Kasus Produksi



Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
Sending good receive	4.84	2.96	1.88	18652.25
Getting good receive	5.71	3.65	2.06	17621.71
Bale opening	8.23	4.36	3.87	28000.11
Conditioning of MMP Fiber	2.12	1.7	0.42	42142.9
Blending	4.91	2.81	2.1	66756.19
Opposing spike	3.13	2.41	0.72	34480.63
Striking cotton	3.94	2.08	1.86	36108.53
Carding	11.44	6.66	4.78	53576.68
Roving frame	11.96	6.53	5.43	43658.18
Drawing frame	12.04	6.53	5.51	15274.55
Combing	9.19	5.52	3.67	32041.56
Ring framing	9.14	6.14	3	24075.45
Cone winding	6.99	5.96	1.03	63075.1
Air current blowing	3.29	2.41	0.88	58480.68
Result	81.84	50.78	31.06	533944.52

Gambar 6.31 Hasil Optimasi *Event Log* Studi Kasus Produksi

6.5 Evaluasi Sistem dengan Sistem Lain

Evaluasi dilakukan dengan membandingkan kinerja sistem yang dikembangkan dengan sistem lain. Perbandingan dilakukan dengan menggunakan hasil yang didapatkan pada hasil uji studi kasus.

6.5.1. Evaluasi terhadap Data *Event Log Purchace Order*

Rincian evaluasi ditulis pada Tabel 6.20 dan Tabel 6.21. Tabel 6.20 merincinkan evaluasi *process discovery* antarsistem. Dan Tabel 6.21 merincikan evaluasi optimasi antarsistem. Pada evaluasi ini terdapat 2 sistem yaitu sistem A dan sistem B. Sistem A adalah sistem yang dikembangkan. Sistem B adalah sistem yang dikembangkan pada buku Tugas Akhir dengan judul "Optimasi Waktu Produksi Berdasarkan Process Mining dari Activity Lifespan".

Tabel 6.20. Evaluasi Sistem dengan Sistem Lain terhadap *Process Discovery* Proses Bisnis *Purchase Order*

Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Sending PO number	Mulai	*OR-Split*	Mulai	*OR-Split*
Sending permitting document	*OR-Split*	*Sequence*	*OR-Split*	*Sequence*
Producing good orders sup1	*OR- Split*	*Sequence*	*OR- Split*	*Sequence*
Pakcaging good orders	*Sequence*	*OR-Join*	*Sequence*	*OR-Join*
Determining PPh and giving permission	*Sequence*	*AND-Split*	*Sequence*	*AND-Split*
Paying PPh	*AND-Split*	*AND-Join*	*AND-Split*	*AND-Join*
Producing good orders sup2	*AND-Split*	*AND-Join*	*AND-Split*	*AND-Join*
Packaging good orders and getting PPh confirm	*AND-Join*	*OR-Join*	*AND-Join*	*OR-Join*
Sending good orders	*OR-Join*	*Sequence*	*OR-Join*	*Sequence*

Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Receiving good receive	*Sequence*	Selesai	*Sequence*	Selesai

Tabel 6.21. Evaluasi Sistem dengan Sistem Lain terhadap Optimasi Proses Bisnis *Purchase Order*

Aktivitas	Sistem A	Sistem B
Biaya Tambahan	336760.76	336760.76
Durasi	84.69 jam	84.69 jam

6.5.2. Evaluasi terhadap Data *Event Log* Produksi Benang

Rincian evaluasi ditulis pada Tabel 6.22 dan Tabel 6.23. Tabel 6.22 merincinkan evaluasi *process discovery* antarsistem. Dan Tabel 6.23 merincikan evaluasi optimasi antarsistem. Pada evaluasi ini terdapat 2 sistem yaitu sistem A dan sistem B. Sistem A adalah sistem yang dikembangkan. Sistem B adalah sistem yang dikembangkan pada buku Tugas Akhir dengan judul "Optimasi Waktu Produksi Berdasarkan Process Mining dari Activity Lifespan".

Tabel 6.22. Evaluasi Sistem dengan Sistem Lain terhadap *Process Discovery* Proses Bisnis Produksi Benang

Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Sending good receive	Mulai	*Sequence*	Mulai	*Sequence*
Getting good receive	*Sequence*	*Sequence*	*Sequence*	*Sequence*
Bale opening	*Sequence*	*Sequence*	*Sequence*	*Sequence*

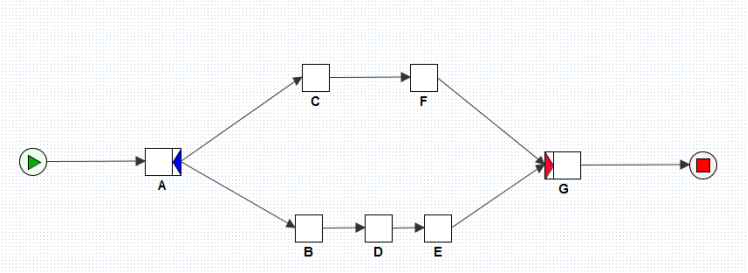
Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Conditioning of MMF Fiber	*Sequence*	*Sequence*	*Sequence*	*Sequence*
Blending	*Sequence*	*XOR-Split*	*Sequence*	*XOR-Split*
Opposing spikes	*XOR-Split*	*XOR-Join*	*XOR-Split*	*XOR-Join*
Air current blowing	*XOR-Split*	*XOR-Join*	*XOR-Split*	*XOR-Join*
Striking cotton	*XOR-Join*	*Sequence*	*XOR-Join*	*Sequence*
Carding	*Sequence*	*OR-Split*	*Sequence*	*OR-Split*
Drawing frame	*OR-Split*	*OR-Join*	*OR-Split*	*OR-Join*
Roving frame	*OR-Split*	*OR-Join*	*OR-Split*	*OR-Join*
Combing	*OR-Join*	*Sequence*	*OR-Join*	*Sequence*
Ring framing	*Sequence*	*Sequence*	*Sequence*	*Sequence*
Cone winding	*Sequence*	Selesai	*Sequence*	Selesai

Tabel 6.23. Evaluasi Sistem dengan Sistem Lain terhadap Optimasi Proses Bisnis Produksi Benang

Aktivitas	Sistem A	Sistem B
Biaya Tambahan	533944.52	533944.52
Durasi	50.78 jam	50.78 jam

6.6 Evaluasi *Process Discovery* dengan *Event Log* Mengandung *Noise*

Terdapat suatu model:



Gambar 6.32 Model AND YAWL

Dari model YAWL tersebut dibangkitkan suatu *event log* sebagai berikut:

L1 = {(ACFBDEG), (ACBFDEG), (ACBDFEG), (ACBDEFG), (ABDECFG), (ABCDEFG), (ABCFDEG), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Hubungan antar aktivitas dari model Gambar 6.32 dapat dilihat pada Tabel 6.24.

Tabel 6.24. Hubungan Aktivitas Model Gambar 6.32

Input	Split / Join	Output
{Start}		A
A	AND Split	C, B
C		F
B		D
D		E
E, F	AND Join	G
G	XOR Join	{end}

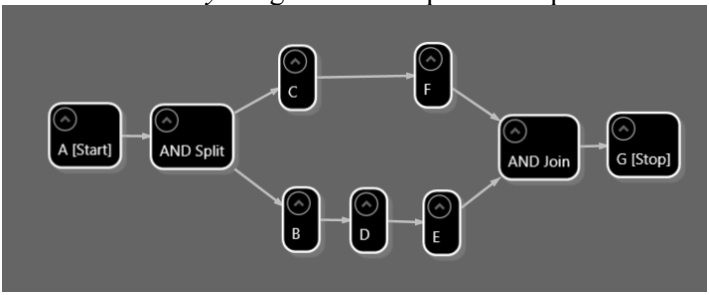
- **Noise 40%**

Dari *event log* L1 yang memiliki jumlah *case* 10, dirubah menjadi *event log* dengan *noise* 40%.

Event log pada L1 dirubah menjadi *event log*

L1' = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Hasil *discovery* L' dengan menggunakan sistem dapat dilihat pada Gambar 6.33 dan perbandingan hubungan aktivitas antara hasis *discovery* dengan model dapat dilihat pada Tabel 6.25.



Gambar 6.33 Hasil *Discovery* dengan *Noise* 40%

Tabel 6.25 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
{Start}		A	{Start}		A	Sama
A	AND Split	C, B	A	AND Split	C, B	Sama
C		F	C		F	Sama
B		D	B		D	Sama
D		E	D		E	Sama
E, F	AND Join	G	E, F	AND Join	G	Sama

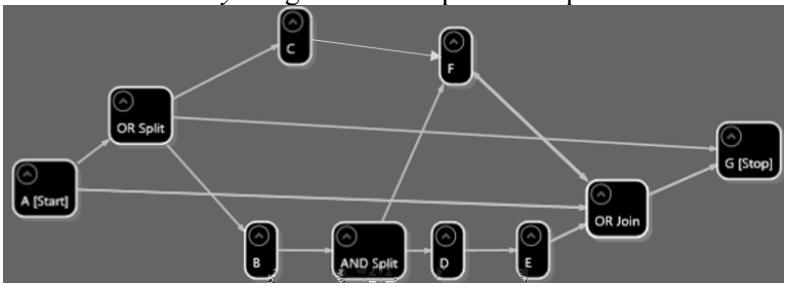
Dari hasil Tabel 6.25 diketahui bahwa dengan kandungan *noise* 40% model masih dapat di-*discovery* sesuai dengan model aslinya.

- **Noise 50%**

Untuk *event log* yang jumlah *noise* pada *event log* L1 50%, sebagai contohnya adalah L1".

L1" = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCD)}

Hasil *discovery* L1" dengan menggunakan sistem dapat dilihat pada Gambar 6.34 dan perbandingan hubungan aktivitas antara hasis *discovery* dengan model dapat dilihat pada Tabel 6.26.



Gambar 6.34 Hasil *Discovery* dengan *Noise* 50%

Tabel 6.26 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
{Start}		A	{Start}		A	Sama
A	AND Split	C, B	A	OR Split	C, B,G	Beda
C		F	C		F	Sama
B		D	B	AND Split	D, F	Beda
D		E	D		E	Sama
E, F	AND Join	G	A, E, F	OR Join	G	Beda

Dari hasil Tabel 6.26 diketahui bahwa dengan kandungan *noise* 50% hasil *discovery* berbeda dengan model aslinya. Sehingga hasil *discovery* gagal.

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

7.1. Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi, dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Modifikasi terhadap *heuristic miner* dengan penentuan *threshold* yang otomatis dapat menghasilkan model sesuai dengan model proses bisnis yang sebenarnya.
2. Modifikasi terhadap *heuristic miner* dapat memodelkan bentuk *parallel split* dan *join* OR.
3. Data optimasi dapat dihitung dengan menggunakan rata-rata dan standar deviasi dari setiap aktivitas yang ada pada *event log* (ditunjukkan pada subbap 3.5).
4. Perangkat lunak berhasil memodelkan proses bisnis yang mengandung OR.
5. Proses optimasi pada proses bisnis *purchase order* menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 44.23% dengan tambahan biaya 19.41%.
6. Proses optimasi pada proses bisnis produksi benang menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 37.65% dengan tambahan biaya 12.1%.
7. *Process discovery* berhasil dengan kandungan *noise* 40% dalam *event log* sedangkan kandungan *noise* 50% dalam *event log* membuat *process discovery* gagal.
8. Perangkat lunak dapat mendapatkan durasi proses bisnis yang paling minimum dan biaya tambahan yang paling minimum.

9. Perangkat lunak berhasil menjalankan fitur-fitur yang ada dengan baik yaitu memasukkan *event log*, men-*discover* bisnis proses model, menghitung data optimasi, optimasi biaya tambahan dan percepatan *makespan* (durasi proses bisnis).

7.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan.

1. Penentuan *threshold* dapat ditentukan secara otomatis oleh sistem maupun dapat dimasukkan sendiri oleh pengguna, hal ini dilakukan untuk memberikan fasilitas pengguna yang sudah ahli dibidang penentuan model proses bisnis.
2. Penyerderhanaan pengimplementasian agar *running* sistem menjadi lebih cepat.
3. Fitur memasukkan data pada sistem dapat dikembangkan sehingga dapat menerima dokumen selain format Excel.
4. Fitur menyimpan data pada sistem dapat dikembangkan sehingga dapat menyimpan dokumen selain format Excel.

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

7.1. Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi, dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Modifikasi terhadap *heuristic miner* dengan penentuan *threshold* yang otomatis dapat menghasilkan model sesuai dengan model proses bisnis yang sebenarnya.
2. Modifikasi terhadap *heuristic miner* dapat memodelkan bentuk *parallel split* dan *join* OR.
3. Data optimasi dapat dihitung dengan menggunakan rata-rata dan standar deviasi dari setiap aktivitas yang ada pada *event log* (ditunjukkan pada subbap 3.5).
4. Perangkat lunak berhasil memodelkan proses bisnis yang mengandung OR.
5. Proses optimasi pada proses bisnis *purchase order* menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 44.23% dengan tambahan biaya 19.41%.
6. Proses optimasi pada proses bisnis produksi benang menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 37.65% dengan tambahan biaya 12.1%.
7. *Process discovery* berhasil dengan kandungan *noise* 40% dalam *event log* sedangkan kandungan *noise* 50% dalam *event log* membuat *process discovery* gagal.
8. Perangkat lunak dapat mendapatkan durasi proses bisnis yang paling minimum dan biaya tambahan yang paling minimum.

9. Perangkat lunak berhasil menjalankan fitur-fitur yang ada dengan baik yaitu memasukkan *event log*, men-*discover* bisnis proses model, menghitung data optimasi, optimasi biaya tambahan dan percepatan *makespan* (durasi proses bisnis).

7.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan.

1. Penentuan *threshold* dapat ditentukan secara otomatis oleh sistem maupun dapat dimasukkan sendiri oleh pengguna, hal ini dilakukan untuk memberikan fasilitas pengguna yang sudah ahli dibidang penentuan model proses bisnis.
2. Penyerderhanaan pengimplementasian agar *running* sistem menjadi lebih cepat.
3. Fitur memasukkan data pada sistem dapat dikembangkan sehingga dapat menerima dokumen selain format Excel.
4. Fitur menyimpan data pada sistem dapat dikembangkan sehingga dapat menyimpan dokumen selain format Excel.

LAMPIRAN A

1. Halaman Proses *Discovery*

- Kode Sumber Xaml

```
<Window x:Class="WpfApplication2.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:graphsharp="clr-
namespace:GraphSharp.Controls;assembly=GraphSharp.Controls"
xmlns:local="clr-namespace:WpfApplication2"
xmlns:zoom="clr-
namespace:WPFExtensions.Controls;assembly=WPFExtensions"
Title="TA_5111100106" Height="416" Width="679">
<Window.Resources>
   <DataTemplate x:Key="demoTemplate" DataType="{x:Type
   local:Main}">
   <StackPanel Orientation="Horizontal" Margin="5">
   <TextBlock Text="{Binding Path=activity}"
   Foreground="White" />
   </StackPanel>
   </DataTemplate>
   <Style TargetType="{x:Type graphsharp:VertexControl}">
   <Setter Property="Template">
   <Setter.Value>
   <ControlTemplate TargetType="{x:Type
   graphsharp:VertexControl}">
    <Border BorderBrush="White" Background="Black"
    BorderThickness="2" CornerRadius="10,10,10,10"
    Padding="{TemplateBinding Padding}">
   <StackPanel Orientation="Vertical">
   <Expander IsExpanded="True">
   <ContentPresenter Content="{TemplateBinding Vertex}"
   ContentTemplate="{StaticResource demoTemplate}"/>
   </Expander>
   </StackPanel>
   <Border.Effect>
   <DropShadowEffect BlurRadius="2" Color="LightGray"
   Opacity="0.3" Direction="315"/>
   </Border.Effect>
   </Border>
   </ControlTemplate>
   </Setter.Value>
   </Setter>
   </Style>
   <Style TargetType="{x:Type graphsharp:EdgeControl}">
   <Style.Resources>
   <ToolTip x:Key="ToolTipContent">
```

```
    <StackPanel>
    <TextBlock FontWeight="Bold" Text="Edge Information"/>
    <TextBlock Text="{Binding ID}"/>
    </StackPanel>
    </ToolTip>
    </Style.Resources>
    <Setter Property="ToolTip" Value="{StaticResource
    ToolTipContent}"/>
    </Style>
    </Window.Resources>
    <Grid  Background="Black">
    <Grid.RowDefinitions>
    <RowDefinition Height="Auto" MinHeight="70"/>
    <RowDefinition/>
    </Grid.RowDefinitions>
    <zoom:ZoomControl  Grid.Row="1"  Zoom="0.2"
    ZoomBoxOpacity="0.5" Background="#ff656565">
    <local:PocGraphLayout x:Name="graphLayout" Margin="10"
    Graph="{Binding Path=Graph}"
        LayoutAlgorithmType="{Binding
    Path=LayoutAlgorithmType}"
        OverlapRemovalAlgorithmType="FSA"
        HighlightAlgorithmType="Simple" Grid.Row="1" />
    </zoom:ZoomControl>
    <Button Content="Discovery" HorizontalAlignment="Left"
    Margin="22,42,0,0" VerticalAlignment="Top" Width="75"
    Click="Button Click 1" Height="22" Background="Black"
    BorderBrush="Gray" Foreground="White"/>
    <Button Content="Save Discovery As Excel"
    HorizontalAlignment="Right" Margin="0,0,5,6"
    VerticalAlignment="Bottom" Width="137"
    Click="Button_Click_2" Height="25" Background="Black"
    BorderBrush="Gray" Foreground="White" Grid.Row="1"/>
    <Button Content="Browse File" HorizontalAlignment="Left"
    Margin="291,10,0,0" VerticalAlignment="Top" Width="74"
    Click="Button_Click_3" Height="auto" Background="Black"
    BorderBrush="Gray" Foreground="White"/>
    <TextBox Name="namaFile" HorizontalAlignment="Left"
    Height="auto" Margin="22,10,0,0" TextWrapping="Wrap"
    Text="" VerticalAlignment="Top" Width="255"/>
    <Button Content="Get Duration"
    HorizontalAlignment="Left" Margin="115,42,0,0"
    VerticalAlignment="Top" Width="86"
    Click="Button_Click_4" Height="22" Background="Black"
    BorderBrush="Gray" Foreground="White"/>
    </Grid>
</Window>
```

Kode Sumber A. 1 Xaml *Discovery*

- Kode Sumber Fungsi *Button Browse*

```
private void Button_Click_3(object sender,
RoutedEventArgs e)
{
   OpenFileDialog openFileDialog = new OpenFileDialog();
   openFileDialog.Filter = "Excel Files(*.xlsx)|*.xlsx";
   if (openFileDialog.ShowDialog() == true)
   {
      namaFile.Text = openFileDialog.FileName; ;
   }
}
```

Kode Sumber A. 2 *Buttun Browse*

- Kode Sumber Fungsi *Button Discovery*

```
private void Button_Click_1(object sender,
RoutedEventArgs e)
{
  if (namaFile.Text == "")
  {
    MessageBoxResult result = MessageBox.Show("There is
    no file to discover");
  }
  else
  {
    Main vm = new Main();
    vm.viewing(namaFile.Text);
    this.DataContext = vm;
  }
}
```

Kode Sumber A. 3 *Button Discovery*

- Kode Sumber Fungsi *Button Get Duration*

```
private void Button_Click_4(object sender,
RoutedEventArgs e)
{
   if (namaFile.Text == "")
   {
     MessageBoxResult result = MessageBox.Show("There is
     no file to generate duration");
   }
   else
   {
     Optimation window = new Optimation(namaFile.Text);
     window.ShowDialog();
   }
}
```

Kode Sumber A. 4 *Button Get Duration*

- Kode Sumber Fungsi *Button Save Discovery As* Excel

```
private void Button_Click_2(object sender,
RoutedEventArgs e)
{
  if (namaFile.Text == "")
  {
    MessageBoxResult result =MessageBox.Show("There
    is no file to save");
  }
  else
  {
    SaveFileDialog saveFileDialog = new
    SaveFileDialog();
    saveFileDialog.Filter = "Excel Files
    (*.xlsx)|*.xlsx";
  if (saveFileDialog.ShowDialog() == true)
  {
    vm.saveAs(namaFile.Text,
    saveFileDialog.FileName);
  }
  }
}
```

Kode Sumber A. 5 *Button Save Discovery As Excel*

2. Kode Sumber Halaman Optimasi

- Kode Sumber Xaml

```
<Window x:Class="WpfApplication2.Optimation"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pres
entation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="TA_5111100106" Height="344" Width="679">
    <Grid Background="Black">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="2*" />
        </Grid.ColumnDefinitions>
         <Button Content="Calculate Duration"
         HorizontalAlignment="Stretch"
         Margin="10,10,10,0" VerticalAlignment="Top"
         Width="auto"
         RenderTransformOrigin="0.856,0.272"
         Click="Button_Click_1" Background="Black"
         BorderBrush="Gray" Foreground="White"
         Grid.Column="0"/>
         <DataGrid Height="auto"
         HorizontalAlignment="Stretch"
         Margin="10,37,8,37" Name="dataGrid1"
         VerticalAlignment="Stretch" Width="auto"
         Background="Gray" Grid.Column="0">
         </DataGrid>
```

```
        <Button Content="Save Duration As Excel"
        HorizontalAlignment="Right" Margin="0,0,8,8"
        VerticalAlignment="Bottom" Width="139"
        Click="Button Click 2" Background="Black"
        BorderBrush="Gray" Foreground="White"/>
        <TextBox Name="mathText"
        HorizontalAlignment="Stretch" Height="auto"
        Margin="14,37,10,10" TextWrapping="Wrap"
        VerticalAlignment="Stretch" Width="auto"
        Background="Gray" Grid.Column="1"/>
        <Button Content="Optimal Solution for
        Additional Cost and Project Duration"
        HorizontalAlignment="Stretch"
        Margin="14,10,10,0" VerticalAlignment="Top"
        Width="auto" Click="Button_Click_5"
        Background="Black" BorderBrush="Gray"
        Foreground="White" Grid.Column="1"/>
    </Grid>
</Window>
```

Kode Sumber A. 6 *Xaml* Optimasi

- Kode Sumber *Button Calculate Duration*

```
private void Button_Click_1(object sender,
RoutedEventArgs e)
{
  vm.clear();
  vm.GetDuration(namaFile);
  List<ActivityOptimisation> opData = new
  List<ActivityOptimisation>();
  ActivityOptimisation check = new
  ActivityOptimisation() { Activity = ""};
  foreach (string key in vm.GetnormalDuration().Keys)
  {
    if (!(check.Activity.Contains(key)))
    {
       check = (new ActivityOptimisation() { Activity =
       key,
       NormalDuration = vm.GetnormalDuration()[key],
       CrashDuration =
        vm.GetcrashDuration().First(kvp =>
        kvp.Key.Equals(key)).Value,
                                  NormalCost =
        vm.GetnormalCost().First(kvp =>
        kvp.Key.Equals(key)).Value,
                                  CrashCost =
        vm.GetCrashcost().First(kvp =>
        kvp.Key.Equals(key)).Value,
```

```
                                    TimeSlope =
            vm.GettimeSlope().First(kvp =>
            kvp.Key.Equals(key)).Value,
                                    CostSlope =
            vm.GetcostSlope().First(kvp =>
            kvp.Key.Equals(key)).Value });
             opData.Add(check);
        }
    }
    this.dataGrid1.ItemsSource = opData;
}
```

Kode Sumber A. 7 *Button Calculate Duration*

- Kode Sumber *Button Save Duration As Excel*

```
private void Button_Click_2(object sender,
RoutedEventArgs e)
{
    SaveFileDialog saveFileDialog1 = new
    SaveFileDialog();
    saveFileDialog1.Filter = "Excel Files
    (*.xlsx)|*.xlsx";
    if (saveFileDialog1.ShowDialog() == true)
    {
       vm.SaveDurationAsExcel(saveFileDialog1.FileName);
    }
}
```

Kode Sumber A. 8 *Button Save Duration As Excel*

- Kode Sumber *Button Optimal Solution for Additional Cost and Project Duration*

```
private void Button_Click_5(object sender,
RoutedEventArgs e)
{
      vm.clear();
      vm.GenerateMathModel(namaFile);
      mathText.Text = "Duration before optimization :";
      mathText.Text += vm.GetMakespan().ToString();
      mathText.Text +=
      vm.SolveMathModel().After("===Solution
      Details===").Replace("YFINISH", "Project Duration
      Time").Replace("AdditionalCost", "Additional Cost
      Project").Replace("X", "Crash Time Activity ");
}
```

Kode Sumber A. 9 *Button Optimal Solution for Additional Cost and Project Duration*

[Halaman ini sengaja dikosongkan]

OPTIMASI KINERJA PADA CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL INFORMATIKA INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA

Nama	**: Fitrianing Haryadita**
NRP	**: 5111100106**
Jurusan	**: Teknik Informatika – FTIf ITS**
Dosen Pembimbing I	**: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Dosen Pembimbing II	**: Adhatus Solichah Ahmadiyah, S.Kom., M.Sc.**

Abstrak

Optimasi cross-organizational business process adalah salah satu masalah yang harus dipecahkan. Untuk mengoptimasi kinerja dalam cross-organizational business process yang pertama dilakukan adalah melakukan process discovery terhadap model proses bisnis dari event log. Banyak algoritma process discovery yang telah diterapkan seperti Alpha, Alpha ++ dan Heuristic Miner, tetapi tidak dapat men-discover parallel OR. Oleh kerena itu Tugas Akhir ini memodifikasi Heuristic Miner dengan menggunakan interval threshold untuk men-discover parallel XOR, AND, dan OR. Interval threshold ditentukan berdasarkan rata-rata dari positive dependency measure dalam dependency matriks.

Setelah mendapatkan model dari cross-organizational business process event log, kemudian dilakukan optimasi kinerja dengan mendapatkan durasi minimum proses bisnis dan biaya tambahan yang minimum. CPM crashing project adalah salah astu metode yang digunakan untuk time–cost optimization. Tetapi CPM crashing project memerlukan beberapa data untuk melakukan percepatan durasi proses bisnis tetapi pada realitanya banyak data yang berbentuk single timestamp event log yang tidak memiliki

data yang dibutuhkan untuk CPM crashing project. Oleh karena itu dalam Tugas akhir ini menggunakan perhitungan rata-rata durasi eksekusi dan biaya setiap aktivitas dari setiap case untuk menentukan data optimasi yang digunakan untuk CPM crashing project. Kemudian linear programming digunakan untuk mendapatkan durasi minimum dan biaya tambahan minimum dari proses bisnis.

Hasil uji coba menunjukkan bahwa modified Heuristic Miner dapat discover OR split dan join, dan linear programming dengan data optimasi yang telah dihitung sebelumnya dapat melakukan optimasi terhadap cross-organizational business process dengan mendapatkan minimum durasi dan minimum biaya tambahan yang digunakan.

Kata kunci: Process Discovery, Discovery Parallel Activity OR, Modified Heuristic Miner, Time–cost Optimization, Single Timestamp Event Log, Cross-Organizaional Business Process Model, Linear Programming

PERFORMANCE OPTIMIZATION IN CROSS-ORGANIZATIONAL BUSINESS PROCESS MODEL INFORMATIKA INSTITUT TEKNOLOGI SEPULUH NOPEMBER SURABAYA

Nama	**: Fitrianing Haryadita**
NRP	**: 5111100106**
Jurusan	**: Teknik Informatika – FTIf ITS**
Dosen Pembimbing I	**: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Dosen Pembimbing II	**: Adhatus Solichah Ahmadiyah, S.Kom., M.Sc.**

Abstract

In cross-organizational business process model performance optimization is one of the problem which should be solve. To optimize the cross-organizational business process model the first thing to do is discover the business process from event log. Many algorithms have been employed for process discovery, such as Alpha, Alpha ++ and Heuristic Miner but they cannot discover processes containing parallel OR. Therefor in this undergraduate thesis represents the modified Heuristic Miner which utilizes the threshold intervals to discover parallel XOR, AND, and OR. The threshold intervals are determined based on average positive dependency measure in dependency matrix.

After getting the model of cross-organizational business process event log then optimize the model to get the minimum duration of process and minimum additional cost which is needed. CPM crashing project is one of method to solve the time–cost optimization. But CPM crashing project need some data to speeding up the process business. But in reality there are a lot of data which is represent using single timestamp event log which does not provide data to do CPM crashing project. Therefor this undergraduate thesis represents a method to get data which is use to crashing project. The data is got from averaging time execution

of each activity in case in event log. Then to crashing the project this undergraduate thesis use linear programming to get minimum duration and minimum additional cost.

The results show that the modified Heuristic Miner can discover OR split and join, and using linear programming use the data which is calculate, this undergraduate thesis can optimize the performance of cross-organizational business process by getting minimum makspan and minimum additional cost.

Keywords: Process Discovery, Discovery Parallel Activity OR, Modified Heuristic Miner, Time–cost Optimization, Single Timestamp Event Log, Cross-Organizaional Business Process Model, Linear Programming

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Segala puji bagi Allah SWT, Tuhan semesta alam yang telah melimpahkan rahmat dan hidayah-Nya kepada penulis, sehingga tugas akhir berjudul "Optimasi Kinerja pada Cross-Organizational Business Process Model" ini dapat selesai sesuai dengan waktu yang telah ditentukan.

Pengerjaan tugas akhir ini menjadi sebuah sarana untuk penulis memperdalam ilmu yang telah didapatkan selama menempuh pendidikan di kampus perjuangan Institut Teknologi Sepuluh Nopember Surabaya, khususnya dalam disiplin ilmu Teknik Informatika. Terselesaikannya buku tugas akhir ini tidak terlepas dari bantuan dan dukungan semua pihak. Pada kesempatan kali ini penulis ingin mengucapkan terima kasih kepada:

1. Bapak, Ibu, kakak dan keluarga yang selalu memberikan dukungan penuh untuk menyelesaikan Tugas Akhir ini.
2. Bapak Riyanarto Sarno dan Ibu Adhatus Solichah selaku dosen pembimbing yang telah bersedia meluangkan waktu untuk memberikan petunjuk selama proses pengerjaan Tugas Akhir ini.
3. Bapak, Ibu dosen Jurusan Teknik Informatika ITS yang telah banyak memberikan ilmu dan bimbingan yang tak ternilai harganya bagi penulis.
4. Seluruh staf dan karyawan FTIf ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
5. Segenap dosen rumpun mata kuliah Manajemen Informasi.
6. Rekan-rekan laboratorium Manajemen Informasi.
7. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak kekurangan. Dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Juni 2015

DAFTAR ISI

[Halaman ini segaja dikosongkan]

DAFTAR TABEL

[Halam ini sengaja dikosongkan]

NOMENKLATUR

$A =>_w B$	: nilai *dependency* dari aktivitas A ke B.
$\|A >_w B\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$\|B >_w A\|$	: frekuensi aktivitas B yang diikuti secara langsung oleh A.
$A =>_w A$	: nilai *dependency Length One Loop*
$\|A >_w A\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh A.
$max\{\|A >_w x\|\| x \in e\}$	: frekuensi aktivitas A yang diikuti oleh aktivitas x dimana aktivitas x merupakan bagian dari aktivitas yang ada dari event log.
$A =>_{2w} B$	: nilai *dependency Length Two Loop*
$\|A \gg_w B\|$	: frekuensi dari trace dalam bentuk ABA muncul dalam log
$\|B \gg_w A\|$	: frekuensi dari trace dalam bentuk BAB muncul dalam log
$A =>_w B \wedge C$	: *parallel measure* antara B dan C dimana *split* terjadi di A.
$\|B >_w C\|$	: frekuensi aktivitas B yang diikuti secara langsung oleh C.

$\|C >_w B\|$	: frekuensi aktivitas C yang diikuti secara langsung oleh B.
$\|A >_w B\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$\|A >_w C\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh C.
Z	: nilai fungsi tujuan.
C_i	: sumbangan per unit kegiatan, untuk masalah maksimisasi C_i menunjukkan keuntungan atau penerimaan per unit, sementara dalam kasus minimisasi menunjukkan biaya per unit.
X_i	: banyaknya kegiatan i, dimana i = 1, 2, 3, … w. berarti disini terdapat w variabel keputusan.
a_{ij}	: banyaknya sumberdaya i yang dikonsumsi sumberdaya j.
b_i	: jumlah sumber daya i (i = 1, 2, …, w)
g	: macam batasan sumber atau fasilitas yang tersedia.
h	: macam kegiatan yang menggunakan sumber atau fasilitas tersebut.
RBT	: *Relative-to-best threshold*
$Avg\ PDM$	: rata-rata positive dependency measure yang lebih dari 0 pada *matrix dependency*

DM	: *dependency measure*
$DM_{a=>b}$	: *dependency measure* antar aktivitas dalam dependency matriks.
$SD\ PDM$	: standar deviasi *positive dependency measure* pada *matrix dependency*
POT	: *Positive observations threshold*
$f_{a=>b}$	: frekuensi antar aktivitas pada matriks frekuensi.
DT	: *dependency threshold*
$A =>_w B \wedge C$	: *parallel measure* antara aktivitas pada percabangan dari A yaitu B dan C.
$\|B \gg>_w C\|$	: *undirect and direct followed frequency* aktivitas B dan C
$\|C \gg>_w B\|$	: *undirect and direct followed frequency* aktivitas C dan B
$\|A >_w B\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$\|A >_w C\|$	: frekuensi aktivitas A yang diikuti secara langsung oleh C.
$\|B \gg> not_w C\|$	: frekuensi eksekusi aktivitas B yang tidak diikuti aktivitas C.
$\|C \gg> not_w B\|$	: frekuensi eksekusi aktivitas C yang tidak diikuti aktivitas B.
PM	: *Parallel Measure*
Limit PDM	: nilai minimum dari *positive dependency measure* pada *matrix dependency*

pt	: aktivitas pada *parallel split* dan *join*
n_s	*: noise*
m	: jumlah *case* dalam *event log*
j_t	: frekuensi *trace* dalam *event log*
t	*: trace* pada *event log*
t_{ns}	: bagian *trace* tang dipotong
m	: jumlah *case*
n	: jumlah *complete trace* dari model
X_i	: *time slope* yang terjadi untuk penyelesaian aktivit ke – i dimana i ∈ (variable keputusan)
T_{ni}	: durasi normal aktivitas ke – i dimana i ∈
T_{ci}	: *crash* duration aktivitas i dimana i ∈
S_i	: *Cost slope* aktivitas ke – i dimana i ∈
C_{ni}	: *cost* normal aktivitas ke – i dimana i ∈
C_{ci}	: *cost crash* aktivitas ke – i dimana i ∈
ed	: durasi eksekusi
$et_{activity_{output}}$	: waktu eksekusi *output* aktivitas
$et_{activity}$	: waktu eksekusi aktivitas
ed_k	: waktu eksekusi aktivitas pada *case* ke-k
$cost_k$	: biaya aktivitas pada *case* ke-k
h	: banyaknya aktivitas dieksekusi dalam *event log*

	(banyaknya *case* yang mengandung aktivitas)
j	: banyaknya aktivitas pada proses bisnis
j	: banyaknya aktivitas pada proses bisnis
Y_i	: variabel *start time* aktivitas ke i
Y_{pi}	: variabel *start time predecessor* aktivitas ke i
X_{pi}	: variabel *time slope predecessor* aktivitas ke i
K	: *normal time predecessor* aktivitas ke i
D	: maksimum durasi pada *critical path* pada proses bisnis
Y'_{Finish}	: durasi proses bisnis paling minimum setelah dimampatkan
Y_{Finish}	: durasi hasil akhir durasi *crashing* proses bisnis
P	: *place* pada *Petri Net*
T	: transisi (aktivitas) pada *Petri Net*
M	: *message* pada *Petri Net*

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

(2015, Maret 20). (Wikipedia) Retrieved 2 April, 2015, from http://en.wikipedia.org/wiki/Linear_programming

Cnude, S. (2014). Improving the quality of the Heuristics Miner in ProM 6.2.

Elmabrouk, O. M. (2011). A Linear Programming Technique for the Optimization of the Activities in Maintenance Projects. *International Journal of Engineering & Technology IJET-IJENS, 11*, 24-29.

Goedertier, S., De Weerdt, J., Martens, D., Vanthienen, J., & Baesens, B. (2011). Process Discovery in Event Log: An Application in the Telecom Industry. *Applied Soft Computing, 11*(2), 1697-1710.

Larson, W. E., & Gray, F. C. (2006). In *Project Management The Managerial Process Fitfth Edition* (pp. 171-180). Oregon State University: McGraw-Hill Irwin.

Loreto, D. A. (2012). *Sampling and Abstract Interpretation for Tackling Noise in Process Discovery*. Barcelona: UNIVERSITAT POLITÈCNICA DE CATALUNYA.

Prawira, B. (2014, Agustus 12). *Pixelbali*. (Pixelbali) Retrieved April 2, 2015, from http://pixelbali.com/informasi-teknologi/critical-path-method.html

Sarno, R., Ginardi, H., Pamungkas, E. W., & Sunaryono, D. (2013). Clustering of ERP Business Process Fragments. *Proceeding IEEE International conference on computer, control, informatics, and its applications*, 319-324.

Sarno, R., Indita, P. L., Ginadi, H., Sunaryono, D., & Mukhlash, I. (2013). Decision Mining for Multi Choice *Workflow* Patterns. *International Conference on Computer, Control, Informatics ad Its Applications*, 337-342.

Taha, H. A. (n.d.). Operations research: an introduction. In *Operations research: an introduction* (pp. 12-20). New Jersey: Pearson.

van der Aalst, W. (2011). In *Process Mining - Discovery, Conformance and Enhancement of Business Processes* (pp. 95-125). Netherlands: Springer.

van der Aalst, W. (2013, Oktober). *Process Mining: Beyond Business Intelligence*. Retrieved Januari 2015, 21, from www.processmining.org

van der Aalst, W., Adriansyah, A., & van Dongen, B. (2011). Causal Nets: A Modeling Language Tailored Towards Process Discovery. *In J.P. Katoen and B. Koenig, editors, 22nd International Conference on Concurrency Theory (CONCUR 2011)*, 28-42.

van der Aalst, W., Schonenberg, M., & Song, M. (2011). Time Prediction Based on Process Mining. *Information System Sciencedirect*, 450-475.

Wang, J., He, T., Wen, L., Wu, N., ter Hofstede, A., & Su, J. (2010). A behavioral similarity measure between labeled Petri nets based on principal transition sequences.

Weber, P. (2013). Principled Approach to Mining From Noisy Log. *IEEE Symposium on Computational Intelligence and Data Mining*.

Weijters, A., van der Aalst, W., & de Medeiros, A. (2006). Process mining with the Heuristics Miner-algorithm. *BETA Working Paper Series, WP 166, Eindhoven University of Technology*.

Wen, L., Aalst, W. v., Jianmin, W., & Sun, J. (2012). Mining Process Models with Non-Free-Choice Construct. *School of Software Tsinghua University,100084, Beijing,China*.

Weske, M. (2011). Business Process Management- Concepts, Languages, Architectures. *Springer*, 128-135.

Wicaksono, S., Atastina, I., & Kurniati, A. P. (2014). Evaluasi Proses Bisnis ERP dengan Menggunakan Process Mining (Studi Kasus : Goods Receipt (GR) Lotte Mart Bandung). *Informatika Telkom University*.

Zeng, Q., Sun, S. X., Duan, H., Liu, C., & Wang, H. (2013). Cross-organizational collaborative *workflow* mining from a multi-source log. *Sciencedirect*, 1280-1301.

BIODATA PENULIS

Fitrianing Haryadita, lahir di Klaten pada tanggal 17 April 1993. Penulis menempuh pendidikan mulai dari SDN Ponggok (1999-2005), SMPN 2 Klaten (2005-2008), SMAN 1 Klaten (2008-2011) dan S1 Teknik Informatika ITS (2011-2015).

Selama masa kuliah, penulis aktif dalam organisasi yang ada di lingkungan kampus ITS yaitu Himpunan Mahasiswa Teknik Computer-Informatika(HMTC) dan Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi(BEM FTIf). Penulis dapat dihubungi melalui *email*: fitrianing11@mhs.if.its.ac.id.

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan Tugas Akhir, dan sistematika penulisan.

1.1. Latar Belakang

Cross-organizational business process merupakan suatu proses bisnis yang melibatkan lebih dari satu organisasi untuk menjalankannya. Dalam menjalankan c*ross-organizational business process*, mengoptimalkan proses bisnis penting untuk mempercepat durasi yang dibutuhkan tetapi hal ini mengakibatkan penambahan biaya pengeluaran yang dibutuhkan. Sehingga diperlukan upaya untuk meminimumkan durasi dari proses bisnis tapi dengan menggunakan biaya tambahan yang minimum. Dalam mengoptimasi diperlukan untuk mengetahui model dari proses bisnis untuk proses bisnis dari perusahaan berkembang suatu proses yaitu *process mining. Process mining* memiliki salah satu tujuannya yaitu membentuk model dari suatu proses bisnis. *Process mining* sendiri memiliki tiga proses utama yaitu *process discovery, conformance checking,* dan *enchanment.*

Process discovery sendiri merupakan salah satu dari rangkaian *process mining* dimana tujuan dari proses ini adalah untuk membentuk model dengan upaya menggali informasi dari data yang tercatat dalam suatu *event log*. Terdapat banyak algoritma yang digunakan dalam *process discovery*. Alpha ++ merupakan salah satu dari algoritma yang digunakan dalm process discovery. Algoritma ini menggunakan prinsip pemodelan *Petri net,* dengan pemodelan ini algoritma ini dapat memodelkan beberapa jenis proses yang ada pada *event log* seperti *short loop, non free choice,* dan proses paralel (Wen, Aalst, Jianmin, & Sun, 2012). Namun algoritma ini tidak dapat men-*discover event log* yang mengandung *noise* (Weber, 2013) dan karena menggunakan prinsip *Petri net* maka hanya dapat men-*discover* proses paralel

dengan menggunakan percabangan AND dan XOR. Oleh karena itu munculah algoritma baru yaitu *heuristic miner*.

Heuristic miner merpakan jenis pemodelan dengan menggunakan prinsip *Causal-net* (C-net) dalam modelnya. *Heuristic miner* merupakan salah satu algoritma yang dapat menangani event log yang mengandung *noise* (Weber, 2013). *Noise* dalam *event log* ditangani dengan penggunakan *threshold* yang ditentukan untuk memodelkan proses bisnis. Namun dengan cara ini masih terdapat kekurangan yaitu model dapat menjadi *overfitting* dan *underfitting* ketika salah dalam menentukan *threshold*-nya, dan juga pada algoritma ini belum dapat menentukan percabangan yang menggunakan OR, padahal banyak model yang menggunakan jenis percabangan ini. Selain tidak dapat men-*discover* percabangan OR *heuristic miner* juga termasuk algoritma yang tidak mudah untuk digunakan kerena penggunaan *threshold* yang harus ditentukan secara manual oleh pengguna. Sehingga hanya pengguna yang ahli dalam *process discovery* yang dapat menggunakannya. *Discover* percabangan OR dan penentuan *threshold* yang digunakan pada *heuristic miner* merupakan permasalahan pertama yang dipecahkan.

Selain pemodelan proses bisnis dari *event log*, terdapat permasalahan yang kedua yaitu dalam menentukan optimasi kinerja dalam proses bisnis, yang dimaksud kinerja disisni adalah terjadi percepatan waktu durasi dari proses bisnis. Karena hal tersebut dibutuhkan suatu mekanisme untuk optimasi biaya tambahan yang dibutuhkan untuk mencapai percepatan maksimal dari sebuah proses bisnis, dengan kata lain mencari biaya tambahan yang paling minimum untuk kemungkinan percepatan yang paling maksimal dalam istilahnya hal ini disebut dengan *Time-Cost Optimization.*

Berdasarkan permasalahan pertama dan kedua di atas Tugas Akhir ini akan memberikan solusi tentang dua cara yaitu memodifikasi dalam algoritma *heuristic miner* sehingga dapat menentukan *threshold* yang tepat dan dapat memodelkan percabangan OR dalam model yang di-*discover* dan menentukan

threshold yang digunakan secra otomatis sehinggal model dapat di-*discovery* oleh orang awam sehingga ter-*discover* model yang ideal dan untuk optimasi karena *event log* hanya memiliki *single timestamp* maka akan dilakukan perhitungan untuk durasi dan biaya yang diperlukan untuk optimasi dan menggunakan pembentukan model matematika *linear programming* akan dicari durasi terpendek dari proses bisnis dengan penambahan biaya yang paling minimum.

Event log yang diolah dalam tugas akhir ini adalah *event log* dari kerjasama antar departemen produksi dari PT. Toray Industries Indonesia untuk proses produksi benang, dan juga *event log* dari kerjasama antara PT. Toray Industries Indonesia dengan *supplier* dan Bea dan Cukai dalam melakukan *purchase order* bahan baku dari pembuatan benang.

1.2. Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana memodifikasi *heuristic miner* agar dapat memodelkan proses bisnis dengan *threshold* otomatis?
2. Bagaimana memodifikasi *heuristic miner* agar dapat memodelkan paralel OR?
3. Bagaimana menghitung data durasi dan biaya dari *single timestamp event log*?
4. Bagaimana cara mendapatkan optimasi durasi terendah dengan minimum biaya tambahan?

1.3. Batasan Permasalahan

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Bahasa pemrograman yang digunakan adalah C#.
2. Aplikasi yang dikembangkan aplikasi *dekstop*.
3. Data masukan berupa *event log* dalam bentuk Excel.
4. Data keluaran proses *discovery* dalam bentuk *graph*.

5. Data keluaran proses *discovery* hanya dapat disimpan dalam bentuk tabel Excel.
6. Data keluaran data optimasi dalam bentuk tabel dan dapat disimpan dalam Excel.
7. Data keluaran optimasi berupa biaya tambahan paling optimal, durasi proyek paling minimal, dan *crashed duration* untuk tiap aktivitas.

1.4. Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Menentukan *threshold heuristic miner* sehingga dapat membentuk model proses bisnis yang ideal tanpa harus memasukkan secara manual.
2. Memodifikasi *heuristic miner* sehingga dapat men-*discover* bentuk percabangan OR.
3. Melakukan perhitungan untuk data optimasi (durasi normal, durasi *crash*, biaya normal, biaya *crash*) yang diperlukan ketika melakukan percepatan dengan metode CPM *project crashing.*
4. Mendapatkan nilai durasi dan tambahan biaya yang paling optimal, yaitu durasi yang minimum dengan biaya yang minimum.

1.5. Manfaat

Manfaat Keilmuan

Manfaat yang dihasilkan dari pengerjaan Tugas Akhir ini adalah terbentuknya suatu metode yang dapat men-*discover* model yang mengandung paralel OR dan terbebas dari *noise*, selain mendapatkan model yang mengandung OR dalam Tugas Akhir ini juga didapatkan suatu upaya optimasi yang dilakukan pada *cross-organizational business process.* Dengan mendapatkan model yang mengandung OR maka model dapat mengembalikan *event log* sesuai dengan *event log* yang digali. Pemodelan OR dilakukan dengan memodifikasi algoritma yang sudah ada yaitu *heuristic miner*, hal ini dilakukan karena selain untuk mendapatkan model

paralel OR diharapkan model hasil *discovery* juga terbebas dari *noise* yang terdapat pada *event log*. Pemodelan paralel OR merupakan salah satu kontribusi utama yang ada dalam tugas akhir ini karena yang penulis ketahui pemodelan OR dengan memodifikasi *heuristic miner* belum pernah dilakukan. Selain pemodelan OR terdapat manfaat lainnya yaitu optimasi dari *single timestamp event log* dan hubungan optimasi dengan pola *cross-organizational business process.* Untuk optimasi dari *single timestamp event log* diperlukan perhitungan data durasi. Dalam Tugas Akhir ini data durasi dihitung dengan memodifikasi metode *time prediction* dari Van Der Aalst karena durasi yang dihitung dari *time prediction* merupakan durasi per *trace* sedangkan yang diperlukan untuk optimasi adalah durasi per aktivitas. Hubungan optimasi dengan pola *cross-organizational business process* menghubungkan bentuk dari pola dengan keterkaitan antar organisasi jika terdapat salah satu organisasi yang tidak melakukan optimasi.

Manfaat Praktis

Manfaat dari proses *discovery* sendiri adalah agar dapat menemukan proses yang sebenarnya terjadi, untuk penemuan paralel OR dimaksudkan utntuk mendapatkan model yang benar-benar dapat mengembalikan data yang ada dalam *event log*. Dari model proses yang didapatkan dapat dilihat bagaimana model proses bisnis yang sesuai untuk SOP ke depannya, hal ini dapat dilihat dengan medapatkan model secara periodik dari *event log.* Sedangkan untuk optimasinya sendiri lebih membantu memprediksi *crash time* dan biaya yang dibutuhkan walaupun data berupa *single tumestamp event log* yang tidak memiliki data durasi maupun *crash time.* Pola hubungan optimasi dengan *cross-organizational business process* dapat dimanfaatkan untuk menentukan organisasi mana saja yang akan diptimasi jika terdapat keterbatasan optimasi dari organisasi yang berhubungan dengan proses bisnis tersebut.

1.6. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas

Akhir ini yaitu:

1. **Studi literatur**

   Dalam pembuatan Tugas Akhir ini telah dipelajari tentang hal-hal yang dibutuhkan sebagai ilmu penunjang dalam penyelesaiannya. Pertama adalah tentang struktur utama dari *Heuristic Miner*, CPM *Crashing Project*, dan Optimasi dengan *Linear Programming*. Kemudian adalah menentukan batasan-batasan pemetaan. Selain itu, juga dibantu beberapa literatur lain yang dapat menunjang proses penyelesaian Tugas Akhir ini.

2. **Pemodifikasian Metode**

   Pada tahap ini penulis menjabarkan cara pemecahan masalah yang terdapat dalam rumusan masalah. Selain itu penulis juga menjabarkan tentang modifikasi yang telah dilakuakan pada algoritma sebelumnya sebagai salah satu kontribusi dalam Tugas Akhir ini.

3. **Analisis dan Perancangan Sistem**

   Aktor yang menjadi pelaku adalah pengguna perangkat lunak yang dibangun oleh penulis. Kemudian beberapa kebutuhan fungsional dari sistem ini adalah sebagai berikut :

   a. Melakukan proses *discovery* dari *file event log* menjadi model proses bisnis;
   b. Melakukan perhitungan untuk penentuan data optimasi berupa durasi dan biaya.
   c. Melakukan optimasi minimum durasi proses bisnis dan minimum biaya tambahan yang dibutuhkan.

4. **Implementasi**

   Pada tahap ini dilakukan pembuatan elemen perangkat lunak yang merupakan implementasi dari rancangan yang telah dibuat sebelumnya.

5. Pengujian dan evaluasi

Pada tahap ini dilakukan pengujian terhadap elemen perangkat lunak dengan menggunakan data uji yang telah dipersiapkan pada jurnal. Pengujian dan evaluasi perangkat dilakukan untuk mengevaluasi jalannya perangkat lunak, mengevaluasi fitur utama, mengevaluasi fitur-fitur tambahan, mencari kesalahan yang timbul pada saat perangkat lunak aktif, mengadakan perbaikan jika ada kekurangan. Tahapan-tahapan dari pengujian adalah sebagai berikut:

a. pencocokan hasil *discovery* perangkat lunak dengan model *cross-organizational* dan *single-organizational* yang sudah ada;
b. pengujian ketahanan perangkat lunak dalam menangani *noise* pada proses *discovery*; dan
c. pencocokan hasil uji sistem dengan hasil uji pada eksperimen yang telah dilakukan.

6. Penyusunan buku Tugas Akhir

Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan Tugas Akhir.

1.7. Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan Tugas Akhir ini.

Bab III Metode Pemecahan Masalah

Bab ini membahas cara penulis memecahkan masalah yang ada. Penjelasan tentang algoritma yang dikembangkan penulis dan langkah-langkahnya sehingga dapat memecahkan masalah yang ada.

Bab IV Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan alur, proses dan perancangan antarmuka pada perangkat lunak.

Bab V Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak perangkat lunak dan implementasi fitur-fitur penunjang perangkat lunak.

Bab VI Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan serta evaluasi terhadap fitur-fitur perangkat lunak.

Bab VII Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan bab tambahan yang berisi daftar istilah yang penting pada aplikasi ini.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir. Teori-teori tersebut meliputi *Process Mining, Process Discovery*, model proses bisnis, *Heuristic Miner Algorithm*, *Causal Net* (C-net), *Noise*, proses paralel, *Critical Path Method* (CPM), *Linear Programming*.

2.1 Model Proses Bisnis

Proses bisnis merupakan sekumpulan aktivitas yang dibuat untuk menghasilkan keluaran spesifik dengan tujuan tertentu (Wang, et al., 2010). Dari model tersebut juga dapat diketahui informasi dimana dan kapan suatu aktivitas dilakukan, kondisi awal sebelum aktivitas dilakukan, kondisi akhir setelah aktivitas dilakukan, serta masukan dan keluaran yang jelas. Adapun ciri-ciri dari proses bisnis itu sendiri adalah sebagai berikut :

1. Mempunyai tujuan tertentu
2. Mempunyai masukan yang spesifik.
3. Mempunyai keluaran yang spesifik.
4. Memanfaatkan *resource*.
5. Memiliki aktivitas yang dapat dieksekusi dengan urutan tertentu.
6. Dapat melibatkan lebih dari satu organisasi.

Model proses bisnis merupakan representasi dari proses bisnis. Sehingga sebuah model proses bisnis harus secara jelas mendefinisikan setiap ciri-ciri yang harus dimiliki oleh suatu proses bisnis. *Unified Modeling Language* (UML) merupakan salah satu representasi dasar dari proses bisnis. Saat ini representasi dari model proses bisnis itu sendiri sudah banyak berkembang dan banyak jenisnya. Mulai dari *Causal Net*, UML, *Business* BPEL, *Business Process Model and Notation* (BPMN), EPC, PNML, dan masih banyak lagi. Tetapi, masing-masing jenis tersebut juga memiliki kegunaan dan fungsi sendiri-sendiri.

2.2 *Event Log*

Dalam *process mining* untuk menganalisis suatu *business process* digunakan *event log* dari *business process* tersebut sebagai acuannya. *Event log* didefinisikan sebagai suatu set proses eksekusi yang mengambil dari data aktivitas proses bisnis yang dilakukan dalam konteks tertentu (Wen, Aalst, Jianmin, & Sun, 2012). Atau dengan kata lain merupakan catatan dari eksekusi aktivitas dalam suatu proses bisnis, catatan eksekusi ini dapat menyimpan data berupa waktu dilaksanakannya suatu aktivitas, *resource* yang melaksanakan aktivitas, dan lain-lain sesuai dengan kebutuhan dari perusahaan yang menghidupkan *event log*-nya. Dalam *event log* dapat terdiri dari berbagai macam *case, trace,* dan *activity* (van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, 2011).

➨ *Case* dan *Trace*

Case merupakan suatu kasus tertentu yang ada pada *event log*. Kasus tertentu tersebut dapat berupa suatu kasus dalam memproduksi suatu barang tertentu, karena *event log* dapat terdiri dari catatan dari proses eksekusi pembuatan banyak barang atau proses eksekusi dari banyak kasus proses. Sedangkan *trace* merupakan alur dari aktivitas yang dijalankan dalam suatu proses.

Sebagai gambarannya misal dalam suatu *event log*:

$$L = [\langle a, c, d\rangle^{45}, \langle b, c, d\rangle^{42}, \langle a, c, e\rangle^{38}, \langle b, c, e\rangle^{22}]$$

Dalam *event log* tersebut:

1. Terdapat 4 *trace* yaitu (a,c,d), (b,c,d), (a,c,d), (b,c,e)
2. Terdapat 147 *case* karena (a,c,d) dilakukan sebanyak 45 kali, (b,c,d) sebanyak 42 kali, (a,c,e) sebanyak 38 kali, dan (b,c,e) sebanyak 22 kali.

- Activity

Merupakan bagian dari case yang merupakan sub proses dalam pembuatan suatu barang atau dalam suatu proses tertentu.

Misal dalam event log

$$L = [\langle a, c, d\rangle^{45}, \langle b, c, d\rangle^{42}, \langle a, c, e\rangle^{38}, \langle b, c, e\rangle^{22}]$$

Memiliki lima aktivitas yaitu {a, b, c, d, e}.

2.3 Noise

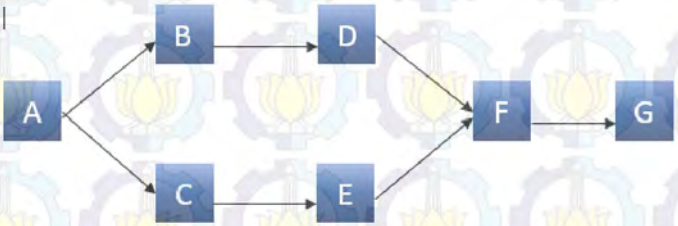
Noise dedifinisikan sebagai '*outlier'* atau event yang jarang terjadi atau exeptional event. Maka diasumsikan bahwa model yang di-*mining* tidak harus mengandung *noise* yang menyebabkan model berantakan. *Noise* sendiri dapat berupa *traceted trace* yang memiliki frekuensi yang jarang terjadi dalam *event log* (Weber, 2013; van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, 2011).

Menurut cara pembentukannya terdapat tiga jenis *noise* yaitu (Loreto, 2012).

- Menghapus *head* dari *trace* aktivitas

Pada cara pembuatan *noise* ini dilakukan dengan menghapus *head* dari *trace* yang ada pada *event log*. Aktivitas yang dihapus dapat berupa aktivitas pertama atau serangkaian aktivitas awal dari *trace* yang ada pada *event log*.

Contohnya adalah sebagai berikut:



Gambar 2.1 Proses Bisnis

Gambar 2.1 akan menghasikan complete trace event log sebagai berikut:

Tabel 2.1 *Event Log* Tanpa *Noise* Gambar 2.1

No. Trace	Trace Utuh
1	ABDCEFG
2	ABCEDFG
3	ABCDEFG
4	ACEBDFG
5	ACBEDFG
6	ACBDEFG

Pada *event log* pada Tabel 2.1 terdapat penghapusan pada aktivitas pertama pada *trace* 1 dan terdapat penghapusan serangkaian aktivitas awal pada *trace* 4, sehingga *trace* 1 dan 4 menjadi noise dengan jenis pembentukan dengan penghapusan *head* dari *trace. Event log* yang mengandung *noise* adalah sebagai berikut:

Tabel 2.2 *Event Log* dengan *Noise* Hasil Perpotongan *Head*

No. Trace	Trace
1	BDCEFG
2	ABCEDFG
3	ABCDEFG
4	EBDFG
5	ACBEDFG
6	ACBDEFG

➨ Menghapus tail dari trace aktivitas

Pada cara pembuatan *noise* ini dilakukan dengan menghapus *head* dari *trace* yang ada pada *event log*. Aktivitas yang dihapus dapat berupa aktivitas pertama atau serangkaian aktivitas awal dari *trace* yang ada pada *event log.*

Contoh yang digunakan adalah Gambar 2.1 Proses Bisnis dengan hasil *event log* pada Tabel 2.1.

Pada *event log* pada Tabel 2.1 terdapat penghapusan pada aktivitas terakhir pada *trace* 1 dan terdapat penghapusan serangkaian aktivitas akhir pada *trace* 4, sehingga *trace* 1 dan 4 menjadi noise dengan jenis pembentukan dengan penghapusan *tail* dari *trace. Event log* yang mengandung *noise* adalah sebagai berikut:

Tabel 2.3 *Event Log* dengan *Noise* Hasil Perpotongan *Tail*

No. Trace	Trace
1	ABDCEF
2	ABCEDFG
3	ABCDEFG
4	ACEBD
5	ACBEDFG
6	ACBDEFG

➨ Menghapus bagian acak dari tubuh trace aktivitas

Pada cara pembuatan *noise* ini dilakukan dengan menghapus *body* dari *trace* yang ada pada *event log*. Aktivitas yang dihapus dapat berupa seluruh *body* dari *trace* atau salah satu aktivitas *body* dari *trace* yang ada pada *event log*.

Cara pembentukan *noise* ini akan menghasilkan *noise* yang melanggar aturan hubungan antar aktivitas yang ada pada model misal sebenarnya antar aktivitas tidak terhubung, tiba-tiba terdapat hubungan antar aktivitas.

Contoh yang digunakan adalah Gambar 2.1 Proses Bisnis dengan hasil *event log* pada Tabel 2.1.

Pada *event log* pada Tabel 2.1 terdapat penghapusan seluruh *body* pada *trace* 1 dan terdapat penghapusan satu aktivitas *body* pada *trace* 4, sehingga *trace* 1 dan 4 menjadi *noise* dengan jenis pembentukan dengan penghapusan *body* dari *trace. Event log* yang mengandung *noise* adalah sebagai berikut:

Tabel 2.4 *Event Log* dengan *Noise* Hasil Perpotongan *Body*

No. Trace	Trace
1	AG
2	ABCEDFG

No. Trace	Trace
3	ABCDEFG
4	ACEBDG
5	ACBEDFG
6	ACBDEFG

Banyak algoritma yang tidak dapat menangangani masalah *noise* seperti *alpha*, *alpha plus* maupun *alpha plus plus* tidak dapat mengananinya (Wen, Aalst, Jianmin, & Sun, 2012). *Heuristic miner* merupakan algoritma yang dapat mengani *noise* dengan menggunakan *thershold* yang telah ditentukan (Weber, 2013).

Dalam Tugas Akhir ini akan dicari *limitation* dari *heuristic miner* dalam menangani *noise* yang ada pada *event log.*

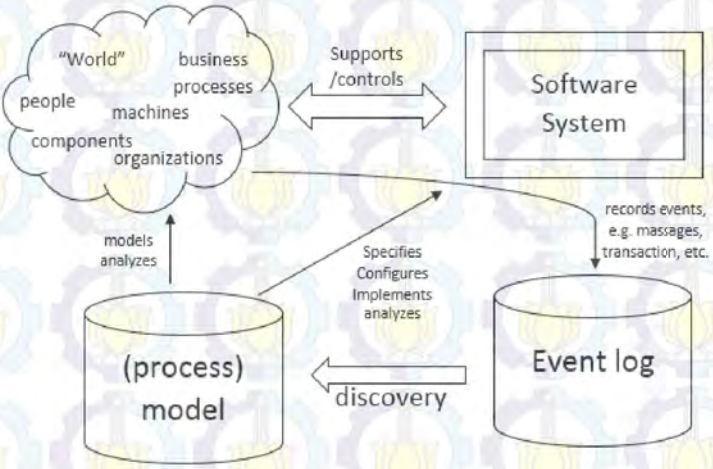
2.4 *Process Mining*

Process Mining merupakan suatu teknologi yang relatif masih baru dalam kaitannya dengan BPM – *Business Process Management* (van der Aalst, Process Mining: Beyond Business Intelligence, 2013). BPM sendiri bertujuan untuk mendapatkan model dengan cara mengamati perilaku proses bisnis di suatu organisasi. Pada *process mining,* pengamatan dilakukan terhadap proses bisnis yang telah tercatat dalam suatu *event log*. Dengan cara ini diharapkan akan ditemukan struktur proses baru yang sebelumnya tidak disadari sedang terjadi. Berdasarkan siklus yang konsisten serta frekuensi aliran informasi yang terjadi maka dapat diketahui apakah selama ini proses bisnis yang diterapkan oleh sistem informasi telah sesuai dengan pedoman yang dimiliki oleh organisasi ataukah sebaliknya. Berbagai manfaat bisa didapat dengan adanya *Process Mining,* seperti untuk mengetahui bagaimanakah proses yang sebenarnya terjadi. Mengetahui apakah proses yang berjalan sudah sesuai dengan model yang dirancang sebelumnya. Mengetahui di tahapan manakah terjadi perlambatan proses. Selain itu yang cukup menarik bahwasannya *Process Mining* juga mampu melakukan prediksi atas jumlah keterlambatan yang mungkin timbul serta membuat rancangan model seperti apa yang lebih tepat guna menyelesaikan permasalahan. *Event log* sebagai sumber data dari teknik *Process Mining* dirasa tepat karena

umumnya *log* sebuah sistem informasi berisi data dari berbagai kasus yang dieksekusi organisasi. Data yang dicatat umumnya berupa waktu mulai dan selesainya pekerjaan di suatu bagian, siapa saja pelakunya, dan lain sebagainya (Wicaksono, Atastina, & Kurniati, 2014).

2.5 *Process Discovery*

Process discovery merupakan salah satu proses yang paling menantang dari rangkaian *process mining*. Tujuan dari proses ini adalah untuk membentuk model dengan cara menggali informasi dari data yang tercatat dalam suatu *event log* (Goedertier, De Weerdt, Martens, Vanthienen, & Baesens, 2011). Dalam struktur proses bisnis, model dapat dianggap sebagai *graph* untuk mengandung satu set *node* dihubungkan dengan *edge* (Sarno, Ginardi, Pamungkas, & Sunaryono, 2013).



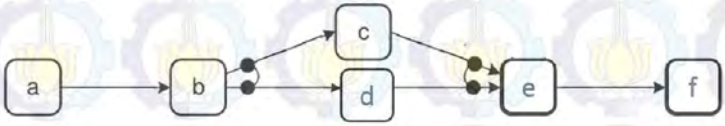
Gambar 2.2 Skema *Process Discovery*

Adapun algoritma yang digunakan dalam *Process Discovery* adalah algoritma *alpha miner*, *alpha plus miner*, *alpha plus plus miner*, *heuristics miner* (van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, 2011). Tiap algoritma memiliki pendekatan yang berbeda-beda dalam menganalisis proses yang terjadi. Dalam Tugas Akhir ini

akan mengembangkan algoritma *heuristics miner* untuk melakukan *process discovery* karena algoritma ini merupakan algoritma yang dapat menangani adanya *noise* dalam *event log*.

2.6 *Causal Net* (C-Net)

Causal Net (C-Net) adalah suatu *graph* yang menggambarkan suatu proses bisnis, dimana *nodes* dari *graph* menggambarkan aktivitas yang terjadi pada proses bisnis dan *arcs* atau *edges* pada *gaph* menggambarkan hubungan antar aktivitas yang ada pada proses bisnis. Dalam C-Net setiap aktivitas memiliki serangkaian *input bindings* dan *output bindings* (van der Aalst, Adriansyah, & van Dongen, Causal Nets: A Modeling Language Tailored Towards Process Discovery, 2011). Salah satu contoh dari causal net adalah sebagai berikut:



Gambar 2.3 Contoh *Causal Net*

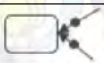
Pada contoh C-Net pada Gambar 2.3 aktivitas a memiliki *input bindings* kosong sehingga a merupakan *start activity* dan a memiliki *output bindings:* {b}, ini berarti setelah aktivitas a diikuti oleh aktivitas b. Aktivitas b memiliki *input bindings:*{a} dan *output bindings:* {c,d}, karena *output bindings* yang dimiliki b lebih dari satu maka menggunakan *split model* yang dimiliki C-Net, *input bindings* dan *output bindings* untuk C-Net begitu seterusnya sehingga akan menghasilkan *Causal Matrix*. Untuk Gambar 2.3 *Causal Matrix*-nya adalah sebagai berikut:

Tabel 2.5 *Causal Matrix* Gambar 2.3

Input	Aktivitas	Output
{}	a	b
a	b	c,d
b	c	e
b	d	e
c, d	e	f
e	f	{}

Bentuk model *split* dan *join* yang dimiliki C-Net adalah sebagai berikut (van der Aalst, Adriansyah, & van Dongen, Causal Nets: A Modeling Language Tailored Towards Process Discovery, 2011):

Tabel 2.6 C-Net *Split* dan *Join*

C-Net Split Model	C-Net Join Model
 XOR-Split	 XOR-Join
 AND-Split	 AND-Join
 OR-Split	 OR-Join

2.7 Jenis *Split* dan *Join*

Dalam proses bisnis dalam alurnya dimungkinkan untuk adanya percabangan sehingga membentuk suatu pilihan ataupun suatu proses paralel. Proses paralel sendiri terdapat dua jenis *split join* AND dan *split join* OR, sedangkan untuk pilihan menggunakan *split* dan *join* XOR (Weske, 2011). Sifat-sifat untuk masing-masing *split* dan *join* adalah sebagai berikut:

- *Single Choice* XOR

Single Choice XOR terjadi jika titik dalam proses alur kerja di mana satu cabang dibagi menjadi dua atau lebih tetapi trace hanya dapat memilih salah satu cabang saja. Semua jenis pemodelan dapat memodelkan XOR oleh karena itu hampir semua algoritma dapat memodelkan XOR (Weske, 2011).

- *Parallel* AND

Parallel AND terjadi jika *parallel split pattern* muncul. *Parallel split pattern* didefinisikan sebagai mekanisme yang memungkinkan dua kegiatan yang berbeda dilakukan secara

bersamaan. Sifat dasar dari pola ini sendiri adaah semua aktivitas yang ada di percabangan harus dijalankan, baik itu dijalankan secara bersamaan atau secara bergantian (Weske, 2011).

- *Conditional* OR

Conditional OR digunakan ketika *multiple choice pattern* muncul. *Multiple choice pattern* pemilihan satu atau lebih aktivitas dalam percabangan untuk dijalankan. Dalam *multiple choice pattern* satu aktivitas dapat dijalankan sendiri tanpa harus menjalankan aktivitas lain yang ada dipercabangan, atau juga dapat menjalankan beberapa aktivitas baik secara bersamaan maupun tidak (Weske, 2011).

Dalam *process mining* banyak algoritma *process discovery* yang tidak dpat mngidentifikasi atau memodelkan secara benar suatu *event log* yang mengandung *multiple choice pattern* (Sarno, Indita, Ginadi, Sunaryono, & Mukhlash, 2013). Hal ini dikarenakan kebanyakan algoritma terutama algoritma yang menggunakan pemodelan dengan *Petri Net* tidak memiliki aturan dalam memodelkan OR, hal ini dikarenakan bentuk pemodelan dari *Petri Net* sendiri memamng tidak dapat memodelkan bentuk *split* dan *join* OR (Weske, 2011).

2.8 *Heuristic Miner Algorithm*

Heuristic miner meupakan salah satu algoritma yang digunakan untuk melakukan *process discovery* dalam *process mining*. *Heuristic miner* menggunakan model C-Net dalam merepresentasikan model proses bisnis yang dihasilkan. Algoritma ini menggunakan frekuensi dari suatu hubungan aktivitas dan urutannya untuk membangun sebuah model proses bisnis. Ide dasar dari algoritma ini adalah suatu urutan atau hubungan dari aktivitas yang jarang terjadi dalam *event log* tidak harus dimasukkan dalam model. Penggunaan frekuensi dalam pembentukan model membuat algoritma ini lebih akurat dibandingkan dengan algoritma yang lainnya selain itu juga membuat algoritma ini dapat menangani *noise*, walaupun belum ada algoritma yang data sepenuhnya menangani masalah *noise* (van der Aalst, Process Mining -

Discovery, Conformance and Enhancement of Business Processes, 2011).

Dalam *heuristic miner* tedapat tiga langkah utama dalam pembentukan model dari proses bisnis (Weijters, van der Aalst, & de Medeiros, 2006), yaitu:

➨ *Mining Dependency Graph*

Langkah awal dari *heutistic miner* adalah membuat suatu *dependency graph* yang merupakan hasil dari perhitungan dari frekuensi-frekuensi dari hungungan antar aktivitas yang ada. Pertama yang dilakukan adalah menghitung frekuensi dari hubungan masing masing aktivitas yang terhubung. Kemudian dari frekuensi tersebut akan menentukan hubungan ketergantungan dari masing-masing aktivitas atau dengan kata lain dapat dikatakan aktivitas yang dapat saling dihubungkan.

Perhitungan dari *dependency measure* yang digunakan untuk menentukan hubungan antar aktivitas dalam model dapat dilihat pada Persamaan 2.1.

$$A =>_w B = \left(\frac{|A>_w B| - |B>_w A|}{|A>_w B| + |B>_w A| + 1}\right) \quad (2.1)$$

Keterangan:

$A =>_w B$: nilai *dependency* dari aktivitas A ke B.

$|A >_w B|$: frekuensi aktivitas A yang diikuti secara langsung oleh B.

$|B >_w A|$: frekuensi aktivitas B yang diikuti secara langsung oleh A.

Hasil dari Persamaan 2.1 menghasilkan suatu matriks yaitu matriks *dependency measure* yang akan digunakan untuk membentuk *dependency graph*. A. Weijters dan Sofie Cnude (Weijters, van der Aalst, & de Medeiros, 2006; Cnude, 2014) menjelaskan bahwa dalam pemilihan *dependency measure* untuk pembentukan *dependency graph, heuristic miner* menggunakan beberapa jenis *threshold* yaitu:

a. *Relative-to-best threshold* (RBT)

Threshold ini menunjukkan bahwa suatu *edge* atau hubungan akan digunakakan (yaitu untuk memasukkan

edge/hubungan antar aktivitas ke dalam *control-flow network*) jika perbedaan antara nilai *dependency measure* yang dihitung untuk *edge* dan nilai terbesar dari *dependency measure* yang ada pada matriks *dependency graph* lebih rendah dari nilai parameter ini.

b. *Positive observations threshold* (POT)
Threshold ini mengontrol jumlah minimum berapa kali aktivitas memiliki ketergantungan hubungan dengan aktivitas lain kegiatan, dua: relasi dianggap ketika banyak frekuensi hubungan ini berada di atas nilai atau sama dengan parameter ini.

c. *Dependency threshold* (DT)
Threshold ini berguna untuk mengabaikan semua hubungan yang nilai *dependency measure* berada dibawah nilai parameter. Dengan kata lain nilai *dependency measure* yang digunakan harus lebih besar atau sama dengan nilai *dependency threshold.*

Salah satu kekurangan dari *heuristic miner* adalah *threshold* yang tidak ditentukan sehingga pengguna harus menentukannya sendiri. Dengan kata lain pengguna harus mengerti betul dengan cara kerja algoritma ini supaya dapat menentukan *threshold* yang dapat mengahsilkan proses bisnis yang ideal. Hal ini mengakibatkan pengguna awam kesulitan dalam menggunakan algoritma ini, oleh karena itu dalam Tugas Akhir ini akan memodifikasi algorima ini sehingga pengguna awam juga dapat menggunakan algoritma ini.

➨ *Mining Short Loop (Length One Loop dan Length Two Loop)*

Dalam proses, dimungkinkan untuk menjalankan kegiatan yang sama beberapa kali. Jika ini terjadi, ini biasanya mengacu pada *Loop* yang terjadi pada model. Untuk *Long Loop* sudah dapat diatasi dengan menggunakan *dependency measure* tetapi untuk *Short Loop* diperlukan pengecekan tersendiri.

Rumus untuk *Length One Loop:*

$$A =>_w A = \left(\frac{|A>_w A|}{\max\{|A>_w x|\,|\, x \in e\}}\right) \qquad (2.2)$$

Keterangan:
$A =>_w A$: nilai *dependency Length One Loop*
$|A >_w A|$: frekuensi aktivitas A yang diikuti secara langsung oleh A.
$\max\{|A >_w x|| \; x \in e\}$: frekuensi aktivitas A yang diikuti oleh aktivitas x dimana aktivitas x merupakan bagian dari aktivitas yang ada dari event log.

Hasil dari Persamaan 2.2 menghasilkan suatu matriks yaitu matriks *length one loop* yang digunakan untuk membentuk *one loop* pada *dependency graph*.

Rumus untuk *Length Two Loop:*

$$A =>_{2w} B = \left(\frac{|A \gg_w B| + |B \gg_w A|}{|A \gg_w B| + |B \gg_w A| + 1}\right) \quad (2.3)$$

Keterangan:
$A =>_{2w} B$: nilai *dependency Length Two Loop*
$|A \gg_w B|$: frekuensi dari *trace* dalam bentuk ABA muncul dalam log
$|B \gg_w A|$: frekuensi dari *trace* dalam bentuk BAB muncul dalam log

Hasil dari Persamaan 2.3 menghasilkan suatu matriks yaitu matriks *length two loop* yang digunakan untuk membentuk *two loop* pada *dependency graph.*

- *Mining Process Parallel*

Untuk mendapatkan aktivitas paralel pada algoritma ini menggunakan *parallel measure* untuk menentukan suatu aktivitas tersebut paralel atau tidak.

Sebelum menghitung nilai dari *parallel measure* ditentukan terlebig dahulu proses mana saja yang paralel dengan menggunakan *causal* matriks seperti pada Tabel 2.5.

Jika *input output*-nya lebih dari satu aktivitas maka ada kemungkinan paralel maka dihitung nilai *parallel measure* dari aktivitas tersebut.

Persamaan 2. 4 merupakan rumus untuk penentuan dari *parallel measure*.

$$A =>_w B \wedge C = \left(\frac{|B >_w C| + |C >_w B|}{|A >_w B| + |A >_w C| + 1}\right) \quad (2.4)$$

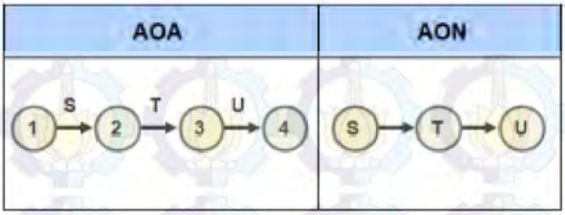
Keterangan:
$A =>_w B \wedge C$: *parallel measure* antara B dan C dimana *split* terjadi di A.
$|B >_w C|$: frekuensi aktivitas B yang diikuti secara langsung oleh C.
$|C >_w B|$: frekuensi aktivitas C yang diikuti secara langsung oleh B.
$|A >_w B|$: frekuensi aktivitas A yang diikuti secara langsung oleh B.
$|A >_w C|$: frekuensi aktivitas A yang diikuti secara langsung oleh C.

Kekurangan dari *heuristic miner* yang lainnya adalah hanya dapat men-*discover* jenis *split* dan *join* XOR dan AND, sementara untuk OR tidak bisa walaupun sebenarnya C-Net dapat memodelkan bentuk *split* dan *join* OR. Oleh karena itu dalam Tugas Akhir ini akan memodifikasi algoritma ini sehingga dapat men-*discovery split* dan *join OR.*

2.9 *Critical Path Method*

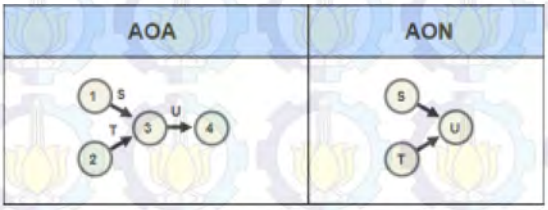
Critical Path Method (CPM) adalah teknik untuk menganalisa jaringan kegiatan/aktivitas-aktivitas ketika menjalankan proyek dalam rangka memprediksi durasi total. *Critical path* sendiri dalam sebuah proyek adalah jalur terpanjang dalam *network diagram* (Prawira, 2014).

Critical path didapatkan dari sebuah diagram jaringan (*network diagram*) yang memperlihatakan hubungan dan urutan aktitivas-aktivitas dalam suatu proyek. Secara umum network diagram digambarkan menggunakan *activity on node* (AON) dan *activity on arrow* (AOA) (Larson & Gray, 2006). Pada AON, aktivitas proyek direpresentasikan dengan titik (*node*), sementara pada AOA, aktivitas kegiatan direpresentasikan dengan panah (*arrow*). Aktivitas proyek yang mendahului/menjadi syarat dillakukan aktivitas lainnya disebut *predesesor*. Gambar 2.4 sampai dengan gambar Gambar 2.6, menunjukkan hubungan aktivitas dalam proyek menggunakan AOA dan AON.



Gambar 2.4 Bentuk Serial

Pada Gambar 2.4 (bentuk serial), diperlihatkan tiga aktivitas proyek yaitu S, T, dan U. Pada gambar tersebut ditunjukkan bahwa aktivitas S merupakan *predesesor* bagi aktivitas T, sementara aktivitas T menjadi *predesesor* bagi aktivitas U. Pada Gambar 2.5 Bentuk (bentuk *join*), diperlihatkan bahwa aktivitas S dan T menjadi *predesesor* bagi aktivitas U, atau dengan kata lain dapat dikatakan bahwa aktivitas U bisa dilaksanakan jika aktivitas S dan T sudah dilaksanakan terlebih dahulu. Gambar 2.6 (bentuk *split*) memperlihatkan aktivitas S menjadi *predesesor* bagi aktivitas T dan U. Hal ini menggambarkan bahwa aktivitas T dan U bisa dilaksanakan jika aktivitas S telah dilaksanakan terlebih dahulu.



Gambar 2.5 Bentuk *Join*

Gambar 2.6 Bentuk *Split*

Pada Tugas Akhir ini digunakan jenis AON untuk menentukan *critical path* pada jaringan atau proses bisnis yang digunakan yang berguna untuk menentukan durasi dari proses bisnis.

Selain digunakan untuk menentukan durasi dari proses bisnis salah satu metode CPM yaitu *crashing project* digunakan sebagai dasar untuk optimasi durasi. Dasar yang digunakan yaitu perhitungan dari perhitungan *cost slope* dan *time slope* yang akan digunakan untuk menjadi konstanta dalan model matematika dari *linear programming*.

2.10 *Linear Programming*

Linear programming merupakan suatu teknik optimasi dimana variabel pembentuknya merupakan variabel-variabel *linear*. *Linear programming* digunakan untuk melakukan suatu optimasi dalam bentuk hasil keputusan yang maksimum atau minimum dengan menggunakan batasan-batasan tertentu. Pemrograman linier berkaitan dengan pemodelan suatu kasus yang ada pada dunia nyata ke dalam suatu model matematika yang terdiri dari sebuah fungsi tujuan (*objective function*) linier dengan beberapa batasan (*constrain function*) linier. Salah satu bentuk permasalahan yang dipecahkan dengan *linear programming* adalah perencanaan aktivitas untuk mendapatkan hasil optimal (menurut model matematika) diantara semua kemungkinan alternatif yang ada.

Sekilas tentang sejarah *linear programming*, Seorang Matematikawan Rusia L.V. Kantorovich pada 1939 berhasil menemukan pemecaham masalah yang berkaitan dengan program linear. Pada waktu itu Kantorovich bekerja untuk Kantor Pemerintah Uni Soviet. L.V. Kantorovich diberi tugas untuk mengoptimalkan produksi pada industri *plywood*. L.V. Kantorovich kemudian muncul dengan teknik matematis yang diakui sebagai pemrograman linear. Matematikawan Amerika George B. Dantzig mengembangkan pemecahan masalah tersebut, dimana hasil karyanya pada masalah tersebut pertama kali dipublikasikan pada tahun 1947.

Pada setiap masalah, ditentukan variabel keputusan, fungsi tujuan, dan sistem kendala, yang bersama-sama membentuk suatu model matematika dari dunia nyata. Bentuk umum dari program linier adalah (Taha):

- Fungsi Tujuan:
  $Max\ atau\ Min\ Z = \sum_{i=1}^{h} C_i X_i$ (2.5)
- Fungsi Batasan:
  $\sum_{j=1}^{h} a_{ij} X_j \leq atau = atau \geq b_i\ untuk\ i = 1, 2, 3, \ldots, g$ (2.6)

Keterangan:
Z : nilai fungsi tujuan.
C_i : sumbangan per unit kegiatan, untuk masalah maksimisasi C_i menunjukkan keuntungan atau penerimaan per unit, sementara dalam kasus minimisasi menunjukkan biaya per unit.
X_i : banyaknya kegiatan j, dimana i = 1, 2, 3, … n. berarti disini terdapat n variabel keputusan.
a_{ij} : banyaknya sumberdaya i yang dikonsumsi sumberdaya j.
b_i : jumlah sumberdaya i (i = 1, 2, …, m)
g : macam batasan sumber atau fasilitas yang tersedia.
h : macam kegiatan yang menggunakan sumber atau fasilitas tersebut.

Untuk menyelesaikan model matematika dari *linear programming* dalam tugas akhir ini menggunakan metode *simplex*. Metode *simplex* sendiri merupakan salah satu teknik penyelesaian dalam program linear yang digunakan sebagai teknik pengambilan keputusan dalam permasalahan yang berhubungan dengan pengalokasian sumberdaya secara optimal. Metode *simplex* digunakan umtuk mencari nilai optimal dari program linier yang melibatkan banyak *constraint* (pembatas) dan banyak variabel.

2.11 *Cross Organizational Bussiness Process Model*

Ada berbagai macam pola koordinasi pada Antar-Organisasi *workflow* yaitu,

2.11.1 Pola 1 : Koordinasi dengan Aktivitas yang Sinkron

Aktivitas yang dapat dilaksanakan oleh dua organisasi yang berbeda dapat didefinisikan sebagai pola koordinasi antar organisasi. Misalnya, dalam proses bisnis multi-moda transportasi, penandatanganan kontrak transportasi harus diselesaikan oleh *Sender* dan *Consignor* secara bersama-sama. Dengan demikian, penandatanganan kontrak merupakan kegiatan sinkron untuk mengkoordinasikan proses antara *Sender* dan *Consignor*. Informasi yang sama tentang awal dan akhir penandatanganan kontrak dapat direkam. Syarat pola ini adalah sebagai berikut (Zeng, Sun, Duan, Liu, & Wang, 2013):

Definisi 2.(Koordinasi dengan aktivitas yang sinkron) Misal $\Sigma_1= (P_1,T_1;F_1,M_{01})$ and $\Sigma_2=(P_2,T_2;F_2,M_{02})$ menjadi model *workflow* dari dua organisasi jika :

(1) $T1 \cap T2 \neq \emptyset$, dan

(2) $P1 \cap P2 = \emptyset$,

Σ_1 dan Σ_2 adalah koordinasi dengan aktivitas yang sinkron. Jika $\Sigma=(P,T;F,M0)$, maka

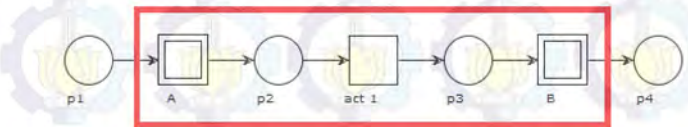
(1) $P=P_1 \cup P_2$;

(2) $T=T_1 \cup T_2$;

(3) $F=F_1 \cup F_2$;

(4) $M_0=M_{01} \cup M_{02}$.

Pada pola koordinasi ini, paling tidak ada satu aktivitas *workflow* dikerjakan oleh dua organisasi (Zeng, Sun, Duan, Liu, & Wang, 2013).



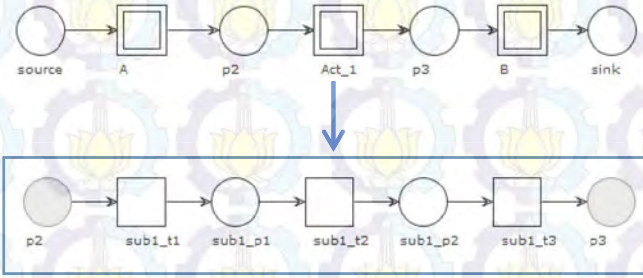
Gambar 2.7 Pola aktivitas sinkron

Aktivitas sinkron digambarkan oleh transisi “act 1” diantara organisasi A dan organisasi B. Aktivitas yang tidak sinkron berada di dalam subproses masing-masing organisasi sehingga aktivitas *private* tidak dapat diketahui oleh organisasi yang lain. Aktivitas yang dapat diketahui oleh organisasi yang lain hanyalah aktivitas yang bersifat sinkron atau dilaksanaan bersama.

Ada beberapa pola aktivitas sinkron yang mungkin terjadi dalam pelaksanaan proses bisnis, antara lain:

- Pola hubungan *Capasity sharing*

  Bentuk kerjasama *capasity sharing* adalah bentuk yang diasumsikan mempunyai pengendalian terpusat (*controlized control*), dimana sebuah urutan kasus dikendalikan oleh sebuah organisasi yang melintasi beberapa organisasi. Pelaksanaan pekerjaan didistribusikan kepada orgainsasi lain yang menjalankannya.



Gambar 2.8 Hubungan *Capasity sharing*

Act_1 adalah aktivitas yang akan di-*share* dari organisasi A sebagai pusat kontrol aktivitas Act_1 ke organisasi B sebagai penerima aktivitas. Act_1 adalah aktivitas subproses yang didalamnya terdiri dari aktivitas-aktivitas yang berurutan.

- Pola hubungan *chained execution*

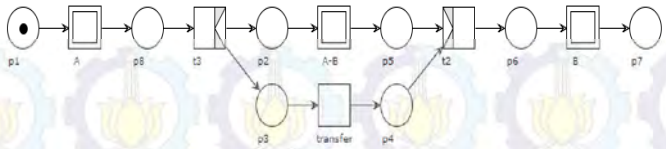
  Bentuk kerjasama *chained execution* adalah suatu proses dipilah berdasarkan subproses (*disjoint* subproses) dimana pelaksanaannya dilakukan oleh beberapa organisasi secara berurutan. Kerjasamanya ini membutuhkan *partner* yang mengawali atau mentransfer urutan sebuah kasus sampai pada akhirnya semua pekerjaan dapat diselesaikan. Berlawanan dengan *capacity sharing*, pengendalian urutan pekerjaan (*workflow*) didistribusikan melawati berbagai organisasi.

Gambar 2.9 Hubungan *chained execution*

  Act1.1, Act1.2, dan Act1.3 adalah aktivitas yang dilaksanakan secara berurutan namun memiliki karakteristik yang berbeda pada masing-masing organisasi. Perbedaan dengan aktivitas sinkron pada umumnya adalah untuk *chained execution*, setiap organisasi melakukan aktivitas yang mirip namun tidak dapat dilakukan secara bersama-sama.

- Pola hubungan *case transfer*

  Bentuk kerjasama organisasi berikutnya adalah *case transfer*. Pada bentuk kerjasama ini setiap organisasi mempunyai deskripsi proses pekerjaan yang sama. Sebuah *case process* instan dapat dilakukan oleh organisasi yang lain untuk menjaga keseimbangan pekerjaan agar tidak terjadi *workload* (kelebihan pekerjaan).

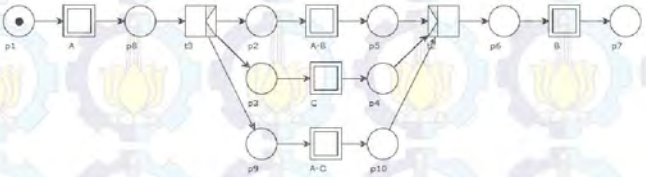


Gambar 2.10 Pola *case transfer*

Organisasi A dan B memiliki hubungan aktivitas 'transfer' yang berisi aktivitas organisasi A yang dilakukan oleh organisasi B.

- Pola hubungan *loosely coupled*

  Bentuk kerjasama organisasi berikutnya adalah *loosely coupled* dimana bentuk kerjasama ini sebuah proses dipotong-potong dalam bagian yang terjadi secara bersamaan. Ada semacam protocol yang mengkomunikasikan dan menghubungkan dengan bagian lain yang terlibat.



Gambar 2.11 Pola *loosely coupled*

Organisasi A adalah *protocol* untuk oragnisasi B dan C yang mengatur aktivitas pada hubungan A-B dan A-C.

2.11.2 Pola 2 : Koordinasi dengan pertukaran pesan

Selama pelaksanaan *workflow*, ada banyak kasus pesan-pesan yang dipertukarkan antara partner yang berbeda. Contoh, setelah *Buyer* melakukan pembayaran, verifikasi pembayaran pesan dikirim ke *Sender*, dan *workflow* dikoordinasikan dengan bertukar pesan verifikasi pembayaran antara *Buyer* dan *Sender* (Zeng, Sun, Duan, Liu, & Wang, 2013).

Definisi 3. (Koordinasi dengan pertukaran pesan) $\Sigma_1=(P_1,T_1;F_1,M_{01})$ dan $\Sigma_2=(P_2,T_2;F_2,M_{02})$ adalah *workflow* model dari dua organisasi, jika :

(1) $P_{M1}\cap P_{M2}\neq\emptyset$,

(2) $P_{L1}\cap P_{L2}=\emptyset$,

(3) $P_{R1}\cap P_{R2}=\emptyset$,

(4) $T_1\cap T_2=\emptyset$,

Σ_1 dan Σ_2 adalah koordinasi dengan pertukaran pesan. $\Sigma=(P,T;F,M_0)$ adalah model yang dibuat oleh Σ_1 dan Σ_2 dengan pertukaran pesan, sehingga

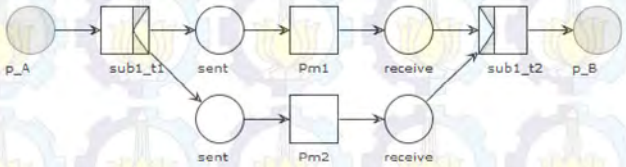
(1) $P=P_1\cup P_2$;

(2) $T=T_1\cup T_2$;

(3) $F=F_1\cup F_2$;

(4) $M_0=M_{01}\cup M_{02}$. (Zeng, Sun, Duan, Liu, & Wang, 2013)

Pada pola koordinasi ini, paling tidak ada satu pertukaran pesan pada *workflow* dikerjakan oleh dua organisasi. Pesan dapat berupa aliran data, form, dan pertukaran pesan dengan *Electronic Data Interchange (EDI)* (Zeng, Sun, Duan, Liu, & Wang, 2013).



Gambar 2.12 Pola pertukaran pesan

Pertukaran pesan berada didalam subproses yang menggambarkan hubungan antar organisasi. “p_A” adalah *place* awal dari organisasi A dan “p_B” adalah place akhir dari organisasi B. Pm1 dan Pm2 adalah pesan yang dikirimkan dari organisasi A ke organisasi B dengan keterangan *place* sebelumnya yaitu *sent* dan *receive*.

2.11.3 Pola 3 : Koordinasi dengan pembagian sumber daya

Karena sumber daya yang sering diakses secara eksklusif oleh *private workflow*, alokasi sumber daya yang penting dalam menghindari konflik sumber daya antar organisasi yang berbeda ketika lebih dari satu *private workflow* memiliki kebutuhan untuk mengakses sumber daya yang sama. Misalnya, dalam proses bisnis multi moda transportasi, sumber daya seperti mobil box, stasiun barang, dan *broker* pabean diakses oleh *private workflow*. Namun, jika sumber daya seperti mobil box dan stasiun barang ditempati oleh suatu kegiatan dalam *private workflow*, kegiatan yang membutuhkan sumber daya yang sama di tempat lain harus menunggu sampai sumber daya dilepaskan. Sehingga sumber daya alokasi dan kontrol konflik sumber daya yang penting untuk dua *workflow* dikoordinasikan dengan sumber daya bersama (Zeng, Sun, Duan, Liu, & Wang, 2013).

Definisi 4. (Koordinasi dengan pembagian sumber daya) $\Sigma_1=(P_1,T_1;F_1,M_{01})$ dan $\Sigma_2=(P_2,T_2;F_2,M_{02})$ adalah *workflow* model dari dua organisasi, jika :

(1) $P_{M1}\cap P_{M2}\neq\emptyset$,

(2) $P_{L1}\cap P_{L2}=\emptyset$,

(3) $P_{R1}\cap P_{R2}=\emptyset$,

(4) $T_1\cap T_2=\emptyset$,

Σ_1 dan Σ_2 adalah koordinasi dengan pembagian sumber daya $\Sigma=(P,T;F,M_0)$ adalah model yang dibuat oleh Σ_1 dan Σ_2 dengan pembagian sumber daya, sehingga

(1) $P=P_1\cup P_2$;

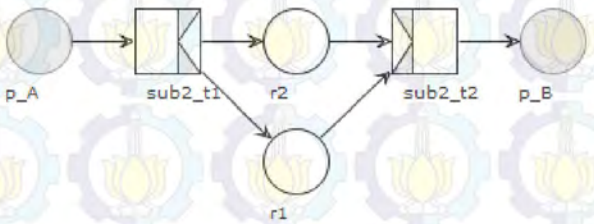
(2) $T=T_1\cup T_2$;

(3) $F=F_1\cup F_2$;

(4) $M_0=M_{01}\cup M_{02}$. (Zeng, Sun, Duan, Liu, & Wang, 2013)

Pada pola ini, paling tidak terdapat satu sumber daya yang dibagikan antara dua organisasi. Pembagian sumber daya ini dapat menggunkaan sistem *cloud-based* sehingga pada masing-masing organisasi, dilakukan perekaman *resource* yang

dibutuhkan untuk memudian di koordinasikan dengan *cloud* untuk organisasi lain (Zeng, Sun, Duan, Liu, & Wang, 2013).



Gambar 2.13 Pola *resource sharing*

Resource tidak dapat digambarkan sebagai transisi karena tidak termasuk aktivitas yang menghubungkan antar organisasi sehingga *resource* digambarkan sebagai place yang menghubungkan antar organisasi didalam subproses hubungan. Pada gambar 3, r1 dan r2 adalah *resource* yang saling digunakan oleh organisasi A dan organisasi B. Pada subproses hubungan antara organisasi B dengan A, maka *place resource* juga harus disertakan.

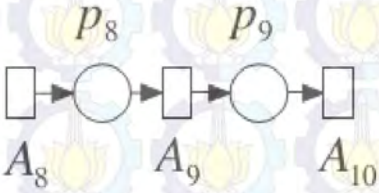
2.11.4 Pola 4 : Koordinasi dengan prosedur yang abstrak

Saat melakukan koordinasi dengan organisasi lain, persyaratan dan output dari *private workflow* dapat dipublikasikan dengan rincian yang dilakukan oleh organisasi lain. Rincian aktivitas tersebut merupakan satu aktifitas oleh organisasi lain. Sehingga antar-organisasi sebetulnya melakukan aktifitas yang sama.

Σ_1 dan Σ_2 adalah koordinasi dengan prosedur yang abstrak $\Sigma=(P,T;F,M_0)$ adalah model yang dibuat oleh Σ_1 dan Σ_2 dengan prosedur yang abstrak, sehingga

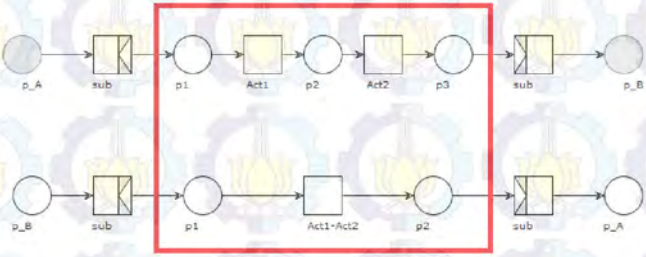
(1) $P=P_1 \cup P_2$;
(2) $T=T_1 \cup T_2$;
(3) $F=F_1 \cup F_2$;
(4) $M_0=M_{01} \cup M_{02}$. (Zeng, Sun, Duan, Liu, & Wang, 2013)

Jika *Consignor* tidak ingin membuat dokumen prosedur bea cukai, tiga kegiatan yang terlibat, yaitu *Generate Declaration Form*, *Declaration Application*, dan *Declaration Acceptance* dapat dikemas sebagai prosedur abstrak yang diterbitkan untuk kepentingan partner. Contoh prosedur abstrak dijelaskan pada Gambar 2.14 yang merepresentasikan kegiatan membuat dokumen prosedur Bea dan Cukai.



Gambar 2.14 Contoh prosedur abstrak

Pada pola prosedur abstrak, terdapat jenis koordinasi lain yaitu *subcontracting*. Pada koordinasi ini sebuah organisasi membagi aktivitas-aktivitas tertentu menjadi subproses kepada organisasi yang lain. Hubungan subcontracting dapat dilakukan untuk mengatasi aktivitas berurutan yang dilakukan oleh dua organisasi yang berbeda secara bersama-sama.



Gambar 2.15 Pola prosedur *subcontracting*

Prosedur abstrak hampir sama dengan aktivitas sinkron, namun pada satu organisasi tidak melakukan semua aktivitas secara

terperinci dan organisasi lain melakukan seluruh rangkaian aktivitas. Pada Gambar 2.15 dijelaskan untuk hubungan organisasi A ke B, organisasi A harus melakukan aktivitas Act1 dan Act2 secara berurutan, sedangkan pada hubungan organisasi B ke organisasi A, organisasi B hanya mekukan satu aktivitas yaitu Act1-Act2 yang dilakukan secara bersama dan tidak terperinci.

BAB III
METODE PEMECAHAN MASALAH

Pada bab ini akan dibahas mengenai metodologi pemecahan masalah yang digunakan sebagai dasar solusi dari pembuatan Tugas Akhir. Metodologi tersebut menerangkan langkah demi langkah proses hingga dapat menghasilkan model dari suatu *event log* permasalahan ini dibahas pada subbab 3.2 sampai dengan 3.4 dan dapat mengasilkan suatu persamaan linear guna untuk mengoptimasi model dari *event log* permasalahan ini akan dibahas pada subbab 3.5.

3.1 Cakupan Permasalahan

Permasalahan utama yang diangkat dalam pembuatan Tugas Akhir ini adalah memodifikasi *heuristic miner algorithm,* menentukan limitasi dari *heuristic miner*, dan bagaimana mengoptimasi waktu *makespan* dari suatu proses bisnis dan biaya optimasi waktu yang diperlukan (total *crash cost*).

Heuristic miner algorithm merupakan salah satu algoritma *process discovery* yang sulit untuk digunakan, hal ini dikarenakan pada *heuristic miner algorithm* tedapat *threshold-trheshold* yang harus ditentukan pengguna. Sementara pengertian dari *threshold* itu sendiri tidak semua pengguna mengerti. Oleh karena itu dalam Tugas Akhir ini yang dimaksud dengan memodifikasi *heuristic miner algorithm* adalah menentukan nilai dari *threshold* yang ada. Selain penentuan *threshold* yang dimaksud dalam memodifikasi *heuristic miner algorithm* disini adalah merubah rumus untuk *mining parallel activity* dan penambahan beberapa interval sehingga dapat men-*discover split* dan *join* OR. Sehingga dalam modifikasi *heuristic miner algorithm* terdapat dua tahap utaha yaitu penentuan nilai *threshold* dan perubahan pada rumus *mining parallel activity*. Selain memodifikasi *heuristic miner algorithm* Tugas Akhir ini juga menentukan limitasi dari *heuristic miner algorithm* baik yang sudah dimodikasi maupun limitasi sebelum dan sesudah modifikasi.

Permasalahan selanjutnya adalah pengoptimasian waktu atau *makespan* dari proses bisnis dan biaya yang dibutuhkan (total *crash cost*). Karena bentuk dari *event log* yang ada hanya memiliki *single timestamp* mengakibatkan aktivitas dalam proses bisnis memiliki durasi yang tidak tentu oleh karena itu langkah awal dalam memecahkan masalah ini adalah menentukan durasi untuk setiap aktivitas yang ada dalam proses bisnis. Selanjutnya untuk pengotimalan waktu atau *makespan* dan biaya yang dibutuhkan diperlukan *crash time* (waktu yang dikurangkan dari normal waktu aktivitas) dan *crash cost* (biaya yang diperlukan untuk pemampatan per aktivitas), maka langkah selanjutnya adalah penentuan durasi (*normal duration*), *crash duration*, dan *crash cost* per aktivitas, yang terakhir adalah pemodelan bentuk linear untuk mendapatkan waktu dan biaya paling optimal.

3.2 Modifikasi *Heuristic Miner Algorithm*

Pada bagian ini dijelaskan tentang bagaimana *heuristic miner algorithm* digunakan dalam Tugas Akhir ini dan bagaimana *heuristic miner algorithm* dimodifikasi dalam Tugas Akhir ini. Tahapan *heuristic miner algorithm* yang telah dikembangkan memiliki kesamaan dengan *heuristic miner algorithm* yang telah dikembangkan sebelumnya hanya saja terdapat tambahan dan perubahan pada tahapan dan rumus yang ada pada tiga tahap utama.

3.2.1 *Mining Dependency Graph*

Sama seperti tahapan yang ada pada *heuristic miner algorithm* yang telah dikembangkan sebelumnya, pada *heuristic miner algorithm* yang telah dimodifikasi memerlukan matriks frekuensi dari hubungan antar aktivitas dalam proses bisnis. Kemudian frekuensi dari matriks frekuensi akan dihitung menggunakan Persamaan 2.1. Modifikasi yang dilakukan pada bagian ini adalah pada penentuan *threshold* yang akan digunakan untuk memilih *dependency measure* pada dependency matriks yang akan digunakan dalam model. Penentuan dari nilai-nilai *threshold* ini dimaksudkan untuk mempermudah pengguna agar

dapat men-*discover* model yang benar. Penentuan dari *threshold-threshold* itu adalah sebagai berikut:

a. Relative-to-best threshold (RBT)
Threshold ini digunakan untuk memilih *dependency measure* yang memiliki selisih dengan maximum *dependency measure* pada *matrix dependency* yang lebih kecil dari nilai *threshold* ini.
- Langkah pertama dalam menghitung *Relative-to-best threshold* (RBT) ini adalah menghitung rata-rata (Avg) dari *positive dependency measure* pada *matrix dependency* (PDM).
- Setelah itu menentukan nilai dari RBT menggunakan Persamaan 3.1.

$$RBT = Avg\ PDM - \left(\frac{SD\ PDM}{2}\right) \quad (3.1)$$

Keterangan:
RBT : *Relative-to-best threshold*
$Avg\ PDM$: rata-rata *positive dependency measure* pada *matrix dependency*
$SD\ PDM$: standar deviasi *positive dependency measure* pada *matrix dependency*
Hubungan antar aktivitas diambil jika memenuhi Persamaan 3.2.

$$Max(DM) - DM_{a=>b} \leq RBT \quad (3.2)$$

Keterangan:
DM : *dependency measure*
$DM_{a=>b}$: *dependency measure* antar aktivitas dalam dependency matriks.

Diambil menggunakan rata-rata karena untuk mengambil nilai yang paling general dari *dependency measure* yang ada.

b. Positive observations threshold (POT)
Threshold ini mengontrol jumlah minimum berapa kali aktivitas memiliki ketergantungan hubungan dengan aktivitas lain. Persamaan 3.3 merupakan rumus untuk penentuan *threshold* ini.

$POT = Min\ (f_{a=>b})$ (3.3)

Hubungan antar aktivitas diambil jika memenuhi Persamaan 3.4.

$f_{a=>b} \geq POT$ (3.4)

Keterangan:

POT : *positive observations threshold.*

$f_{a=>b}$: frekuensi antar aktivitas pada matriks frekuensi.

c. Dependency Threshold (DT)

Threshold ini berguna untuk mengabaikan semua hubungan yang nilai *dependency measure* berada dibawah nilai parameter. Persamaan 3.5 merupakan rumus untuk penentuan *threshold* ini.

$DT = Avg\ PDM - SD\ PDM$ (3.5)

Hubungan antar aktivitas diambil jika memenuhi Persamaan 3.6.

$DM_{a=>b} \geq DT$ (3.6)

Keterangan:

DT : *dependency threshold*

3.2.2 *Mining Short Loop (Length One Loop dan Length Two Loop)*

Pada tahap ini tidak terdapat perubahan. Untuk menghitung *short loop heuristic miner* yang dimodifikasi tetap menggunakan Persamaan 2.2 dan Persamaan 2.3.

3.2.3 *Mining Process Parallel*

Pada *heuristic miner algorithm* yang telah dikembangkan sebelumnya hanya dapat men-*discover split* dan *join* AND dan XOR. Oleh karena itu dalam *heuristic miner algorithm* yang telah dimodifikasi ini akan membuat *heuristic miner algorithm* agar dapat men-*discover* semua jenis *split* dan *join* baik itu AND, XOR, maupun OR. Untuk dapat men-*discover* OR yang pertama dilakukan sama dengan *heuristic miner algorithm* yaitu

membentuk *causal matrix* dari hasil *dependency graph*, tapi dalam memodifikasi ini dilakukan perubahan pada rumus yang digunakan untuk *parallel measure*. Pada persamaan sebelumya pada Persamaan 2.4 hanya mementingkan *direct followed frequency* dari aktivitas pada cabang, padahal sebenarnya untuk *split* dan *join* AND sendiri memeperbolehkan adanya *undirect followed frequency*.

Perubahan dilakukan dengan mengganti frekuensi yang hanya menghitung *direct followed frequency* pada aktivitas percabangan dengan juga menghitung semua *undirect followed frequency* karena sifat dari AND sendiri yang tidak hanya dapat dikakukan secara bersamaan atau *sequence* tapi aktivitas yang ada dalam percabangan dapat dilakukan secara tidak *sequence* yang penting semua aktivitas yang ada dalam percabangan dilakukan.

Untuk perhitungan OR dalam Tugas Akhir ini juga diperhitungkan frekuensi dari aktivitas yang tidak dijalankan secara bersamaan. Rumus untuk *parallel measure* adalah sebagai berikut:

$$A =>_w B \wedge C = \left(\frac{|B \gg>_w C| + |C \gg>_w B|}{|A >_w B| + |A >_w C| + |B >>> not_w C| + |C >>> not_w B| + 1} \right) \quad (3.7)$$

Keterangan:

$A =>_w B \wedge C$: *parallel measure* antara aktivitas pada percabangan dari A yaitu B dan C.

$|B \gg>_w C|$: *undirect and direct followed frequency* aktivitas B dan C

$|C \gg>_w B|$: *undirect and direct followed frequency* aktivitas C dan B

$|A >_w B|$: frekuensi aktivitas A yang diikuti secara langsung oleh B.

$|A >_w C|$: frekuensi aktivitas A yang diikuti secara langsung oleh C.

$|B >>> not_w C|$: frekuensi eksekusi aktivitas B yang tidak diikuti aktivitas C.

$|C >>> not_w B|$: frekuensi eksekusi aktivitas C yang tidak diikuti aktivitas B.

3.2.4 Pemodelan OR, XOR, dan AND

Untuk pemodelan XOR, OR dan AND penentuannya menggunakan interval yang digunakan untuk menentukan letak dari rata-rata *parallel measure* dari aktivitas dengan *input split* yang sama ataupun *output join* yang sama.Yang pertama dilakukan adalah menentukan rata-rata dari *dependency measure* yang masuk dalam *dependency graph* dan rata-rata dari *parallel measure*.

$$Avg\ PDM = \frac{\sum_{i=1}^{n_e} e_i}{n_e} \quad (3.8)$$

$$Avg\ PM = \frac{\sum_{i=1}^{n_{PM}} PM_i}{n_{PM}} \quad (3.9)$$

Interval untuk XOR, OR, dan AND adalah sebagai berikut:

- XOR
  $If\ Avg\ PM\ \leq Limit\ PDM\ then\ XOR$ (3.10)
- OR
  $If\ Limit\ PDM \leq Avg\ PM\ \leq AVG\ PDM\ then\ OR$ (3.11)
- AND
  $If\ Avg\ PDM\ \leq Avg\ PM\ then\ AND$ (3.12)

Keterangan:

$Avg\ PDM$: rata-rata *positive dependency measure* yang lebih dari 0 dari *positive dependency measure*

e_i : *dependency measure* pada *dependency matrix*

n_e : jumlah *dependency measure*

PM : *Parallel measure* dari Persamaan 3.7

$Avg\ PM$: rata-rata dari *parallel measure* yang memiliki induk aktivitas yang sama.

n_{PM} : jumlah *parallel measure* yang memiliki *input* atau *output* yang sama.

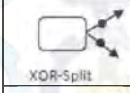
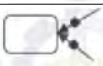
PDM : *positive dependency measure*

$Limit\ PDM$: nilai minimum dari positive *dependency measure* pada *matrix dependency measure.*

3.2.5 Penggambaran OR, XOR, dan AND pada Causal Net

Karena bentuk dari pemodelan untuk OR, XOR, dan AND yang telah dilimiki oleh C-Net kurang informatif dan susah dbaca oleh pengguna awam, maka dalam Tugas Akhir ini modelnya dirubah menjadi sebagai berikut:

Tabel 3.1 C-Net dan *Proposed Parallel Model*

C-Net Model	Proposed Model
	
	
	
	
	
	

3.3 Contoh *Discovery* Menggunakan Modifikasi *Heuristic Miner*

Diberikan *event log* sebagai berikut*:*

$$L = [(ABC), (ABDEC), (ADBEC), (ADEBC), (ABDEDEC)]$$

Sesuai dengan tahapan dari *heuristic miner* yang telah diterangkan pada bagian 3.2 maka yang pertama dilakukan adalah *mining*

dependency graph. Dalam *mining dependency graph* diperlukan matriks frekuensi. Matriks frekuensi dari *event log* L menghasilkan Tabel 3.2.

Tabel 3.2 Matriks Frekuensi *Event Log* L

Aktivitas	A	B	C	D	E
A	0	3	0	2	0
B	0	0	2	2	1
C	0	0	0	0	0
D	0	1	0	0	4
E	0	1	3	1	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.3.

Tabel 3.3 Matriks *Dependency Measure*

Aktivitas	A	B	C	D	E
A	0	0.75	0	0.667	0
B	0	0	0.667	0.25	0
C	0	0	0	0	0
D	0	-0.25	0	0	0.5
E	0	0	0.75	-0.5	0

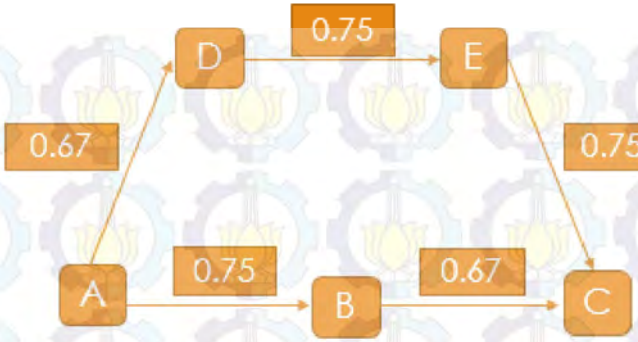
Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RBT = 0.5

PO = 1

DT = 0.42

Dengan menggunakan nilai dari *threshold* tersebut maka didapatkan *dependency graph* sebagi berikut:



Gambar 3.1 *Dependency Graph* dengan *Dependency Measure*

Setelah mendapatkan *dependency graph* dilakukan *mining* terhadap *short loop*. Dari matriks frekuensi pada Tabel 3.2 dan menggunakan Persamaan 2.2 didapatkan matriks *Length One Loop* sebagai berikut:

Tabel 3.4 Matriks *Length One Loop*

Aktivitas	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

Dari matriks *Length One Loop* pada Tabel 3.4 dapat diketahui bahwa model dari *event log* L ini tidak memiliki *Length One Loop*. Sedangkan untuk *Length Two Loop*, yang pertama harus dilakukan adalah menghitung frekuensi *Length Two Loop* sehingga mendapatkan matriks frekuensi *Length Two Loop*.

Tabel 3.5 Matriks Frekuensi *Length Two Loop*

Aktivitas	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

Kemudia menggunakan Persamaan 2.3 mengitung *Length Two Loop* dan membentuk matriks mengitung *Length Two Loop*.

Tabel 3.6 Matriks *Length Two Loop*

Aktivitas	A	B	C	D	E
A	0	0	0	0	0
B	0	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0.667
E	0	0	0	0.667	0

Dari Tabel 3.6 diketahui bahwa model memiliki *Length Two Loop* DED dan EDE, sehingga model menjadi:

Gambar 3.2 *Dependency Graph* dengan *Dependency Measure* dan *Length Two Loop*

Selanjutnya adalah *mining parallel activity*, yang pertama dilakukan dalam *mining parallel activity* adalah menentukan *causal* matriks. *Causal* matriks untuk hasil *dependency graph* untuk contoh terdapat pada Tabel 3.11.

Tabel 3.7 *Causal* Matriks Gambar 3.1

Input	Activity	Output
{}	A	B , D
A	B	C
E, B	C	{}
A	D	E
D	E	C

Dari *causal* matriks pada Tabel 3.7 didapatkan terdapat satu *split* dan *join* yaitu *split* dengan *input* A dan aktivitas pada percabangan adalah B dan D, dan *join* dengan *output C* dengan aktivitas pada percabangan E dan B. Dengan Persamaan 3.7 maka didapatkan *parallel measure:*

$a \rightarrow w^{b \wedge d}$ = 0.57

$c \rightarrow w^{b \wedge e}$ = 0.57

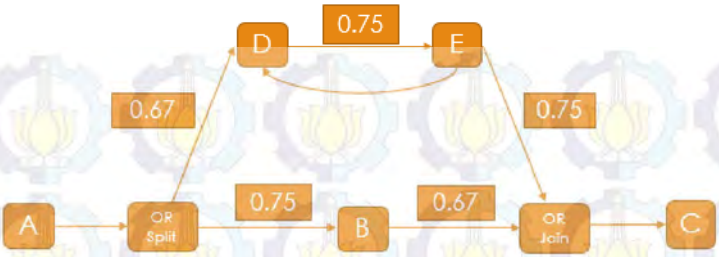
Lalu untuk nilai Avg PM *input* A adalah 0.57 dan Avg PM *output* C adalah 0.57.

Selain Avg PM yang dibutuhkan selanjutnya adalah:

Avg PDM dengan persamaan Persamaan 3.8 adalah sebesar = 0.59

Limit PDM = 0.25

Dengan menggunakan interval dari Persamaan 3.10, Persamaan 3.11, Persamaan 3.12 maka *split* dan *join* yang digunakan model yang di-*discovery* keduanya adalah OR. Sehingga modelnya menjadi sebagai berikut:



Gambar 3.3 Model Akhir dengan *Dependency Measure*

3.4 Limitasi *Heuristic Miner* dalam Menangani *Noise*

Dalam menangani *noise heuristic miner algorithm* memiliki keterbatasan-keterbatasan. Dalam Tugas Akhir ini juga mencari limitasi dari *heuristic miner algorithm.* Limitasi dari *heuristic miner algorithm* dalam menangani *noise* adalah sebagai berikut:

a. Jika *noise* digabungkan masih membentuk *trace* utuh maka *discovery* masih bisa berhasil. Dengan catatan *noise* membentuk *trace* yang hilang. *Noise* terbentuk dari pemotongan jumlah *head* dan *tail* yang sama.

$$If\ n_{s1} + n_{s2} = pt\ then\ discover\ success \quad (3.13)$$

Keterangan:

pt : aktivitas pada paralel *split* dan $join$

n_s: $noise$

Contoh:

Pada model dengan bentuk sesuai dengan Gambar 2.1 yang memiliki *event log* sesuai dengan Tabel **2.1**. Kemudian terbentuk noise dengan pemotongan kepala dan ekor dari dua buah *trace* sehingga *event log* menjadi sebagai berikut:

Tabel 3.8 *Event Log* dengan *Noise* Perpotongan Kepala dan Ekor

No. Trace	Trace Utuh
1	ABDCEFG
2	ABCEDFG
3	ABCDEFG

No. Trace	Trace Utuh
4	ACEBDFG
5	EDFG
6	ACBE

Jika *noise* yang terletak pada *trace* kelima dan keenam digabungkan maka akan membentuk salah satu *trace* utuh yang telah hilang yaitu *trace* ACBEDFG.

Jika *event log* tersebut di *discovery* menggunakan *heuristic miner* maka langkah-langkahnya adalah sebagai berikut:

- *Mining Dependency Graph*

  Matriks frekuensi dari *event log* pada Tabel 3.8 menghasilkan Tabel 3.9.

Tabel 3.9 Matriks Frekuensi Tabel 3.8

Aktivitas	A	B	C	D	E	F	G
A	0	3	2	0	0	0	0
B	0	0	2	2	1	0	0
C	0	1	0	1	3	0	0
D	0	0	1	0	1	3	0
E	0	1	0	2	0	2	0
F	0	0	0	0	0	0	5
G	0	0	0	0	0	0	0

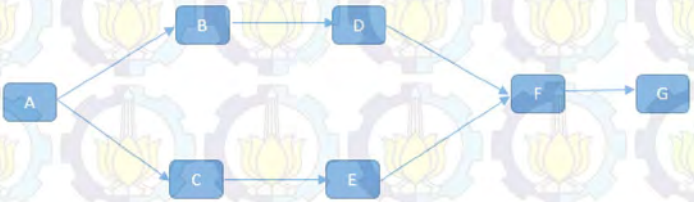
Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.10.

Tabel 3.10 Matriks *Dependency Measure* Tabel 3.8

Aktivitas	A	B	C	D	E	F	G
A	0	0.75	0.67	0	0	0	0
B	0	0	0.25	0.67	0	0	0
C	0	-0.25	0	0	0.75	0	0
D	0	0	0	-0.67	-0.25	0.75	0
E	0	0	0	0.25	0	0.67	0
F	0	0	0	0	0	0	0.83
G	0	0	0	0	0	0	0

Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RBT = 0.72
PO = 1
DT = 0.41



Gambar 3.4 *Dependency Graph* Tabel 3.8

- Mining Short Loop
  Model untuk Tabel 3.8 tidak memiliki *short loop*.
- Mining Parallel Activity

Tabel 3.11 Causal Matriks

Input	Activity	Output
{}	A	B, C
A	B	D
A	C	E
B	D	F
C	E	F
D, E	F	G
F	G	{}

Dari *causal* matriks pada Tabel 3.11 didapatkan terdapat satu *split* dan *join* yaitu *split* dengan *input* A dan aktivitas pada percabangan adalah B dan C, dan join dengan *output* F dengan aktivitas pada percabangan D dan E. Dengan Persamaan 3.7 maka didapatkan *parallel measure:*

$A => w^{B\hat{}C}$ = 0.83
$F => w^{D\hat{}E}$ = 0.83

Nilai Avg PM *input* A adalah 0.83 dan Avg PM *output* F adalah 0.83.

Avg PDM = 0.62
Limit PDM = 0.25

Menurut interval pada Persamaan 3.9, Persamaan 3.10, Persamaan 3.11 maka split dan join yang digunakan model yang di-*discovery* keduanya adalah AND.

Gambar 3.5 Model Akhir dari Tabel 3.8

b. Untuk *noise* tidak membentuk suatu *trace* yang utuh jika *noise* saling digabungkan, minimum *trace* utuh yang dibutuhkan agar proses *discovery* berhasil adalah sebagai berikut:

Untuk jumlah *case* genap:

$$j_t = \frac{m}{2} + 1 \quad (3.14)$$

Untuk jumlah *case* genap:

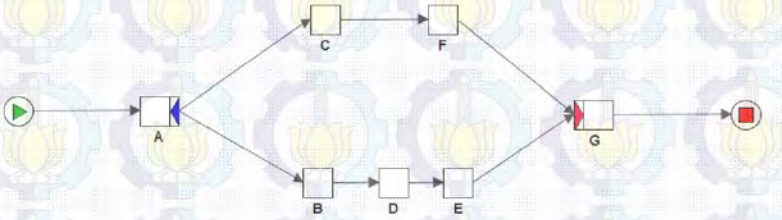
$$j_t = \frac{m+1}{2} \quad (3.15)$$

Keterangan:

m : jumlah *case* dalam *event log*

j_t : frekuensi *trace* dalam *event log*

Contoh pembuktian:
Terdapat suatu model:



Gambar 3.6 Model AND YAWL

Dari model YAWL tersebut dibangkitkan suatu *event log* sebagai berikut:

L1 = {(ACFBDEG), (ACBFDEG), (ACBDFEG), (ACBDEFG), (ABDECFG), (ABCDEFG), (ABCFDEG), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Dari *event log* L1 yang memiliki jumlah *case* 10, maka sesuai dengan yang ada pada Persamaan 3.14 dan Persamaan 3.15, maka jumlah *trace* utuh minimal yang diperlukan adalah 6 *trace* untuk dapat men-*discover event log* diatas walaupun terdapat *noise*. Maka untuk membuktikannya *event log* pada L1 akan dimasukkan *noise* yang pertama sesuai dengan Persamaan 3.14 dan Persamaan 3.15, dan yang satu melanggar jumlah minimum *trace* utuh yang dianjurkan pada Persamaan 3.14 dan Persamaan 3.15.

Contoh pertama:

Event log pada L1 dirubah menjadi *event log*

L1’ = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Event log L1’ memiliki jumlah *trace* utuh sesuai dengan persamaan Persamaan 3.14 dan Persamaan 3.15.

Dari *event log* L1' didapatkan perhitungan *heuristic miner algorithm* secagai berikut:

- *Mining Dependency Graph*
  Matriks frekuensi dari *event log* L1' menghasilkan Tabel 3.12.

Tabel 3.12 Matriks Frekuensi L1'

Aktivitas	A	B	C	D	E	F	G
A	0	5	2	0	0	0	1
B	0	0	2	4	0	1	0
C	0	2	0	2	1	3	0
D	0	0	2	0	4	2	0
E	0	0	0	0	0	3	4
F	0	0	0	2	3	0	4
G	0	0	0	0	0	0	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.13.

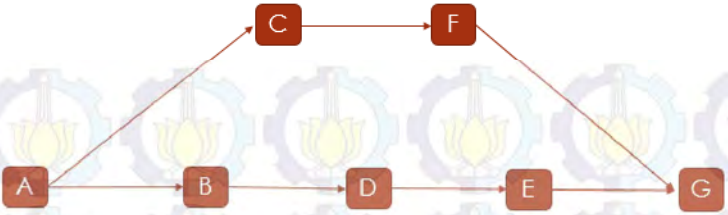
Tabel 3.13 Matriks *Dependency* Measure L1'

Aktivitas	A	B	C	D	E	F	G
A	0	0.83	0.67	0	0	0	0.5
B	0	0	0	0.8	0	0.5	0
C	0	0	0	0	0	0.75	0
D	0	0	0	0	0.8	0	0
E	0	0	0	0	0	0	0.8
F	0	0	0	0	0	0	0.8
G	0	0	0	0	0	0	0

Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RTB = 0.65
PO = 1
DT = 0.59

Maka *dependency graph*-nya adalah sebagai berikut:



Gambar 3.7 *Dependency Graph* L1'

Untuk *short loop* model event log ini tidak memilikinya karena hasil matriks *length one loop* dan *two loop* semuanya 0.

Setelah itu melakukan *mining parallel activity* untuk itu memerkukan *causal* matriks.

Tabel 3.14 *Causal* Matriks L1'

Input	Activity	Output
{}	A	B, C
A	B	D
A	C	F
B	D	E
D	E	G
C	F	G
F, E	G	{}

Dari *causal* matriks pada Tabel 3.14 didapatkan terdapat satu *split* dan *join* yaitu *split* dengan *input* A dan aktivitas pada percabangan adalah B dan C, dan *join* dengan *output* G dengan aktivitas pada percabangan F dan E. Dengan Persamaan 3.7 maka didapatkan *parallel measure:*

$A => w^{B\hat{}C}$ = 0.77

$G => w^{F\hat{}E}$ = 0.8

Nilai Avg PM *input* A adalah 0.77 dan Avg PM *output* G adalah 0.8.

Avg PDM = 0.62

Limit PDM = 0

Menurut interval pada Persamaan 3.9, Persamaan 3.10, Persamaan 3.11 maka split dan join yang digunakan model yang di-*discovery* keduanya adalah AND.

Gambar 3.8 Model Akhir Hasil *Discover* L1'

Untuk *event log* yang jumlah trace utuhnya tidak sesuai dengan jumlah minimal pada Persamaan 3.14 dan Persamaan 3.15, sebagai contohnya adalah L1".

L1" = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCD)}

Dari *event log* L1" didapatkan perhitungan *heuristic miner algorithm* secagai berikut:

Matriks frekuensi dari *event log* L1" menghasilkan Tabel 3.15.

Tabel 3.15 Matriks Frekuensi L1"

Aktivitas	A	B	C	D	E	F	G
A	0	5	2	0	0	0	1
B	0	0	2	4	0	1	0
C	0	2	0	2	1	3	0
D	0	0	2	0	4	1	0
E	0	0	0	0	0	3	3
F	0	0	0	2	2	0	4
G	0	0	0	0	0	0	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.16.

Tabel 3.16 Matriks *Dependency Measure* L1"

Aktivitas	A	B	C	D	E	F	G
A	0	0.83	0.67	0	0	0	0.5

Aktivitas	A	B	C	D	E	F	G
B	0	0	0	0.8	0	0.5	0
C	0	0	0	0	0	0.75	0
D	0	0	0	0	0.8	-0.25	0
E	0	0	0	0	0	0.167	0.75
F	0	0	0	0.25	-0.167	0	0.8
G	0	0	0	0	0	0	0

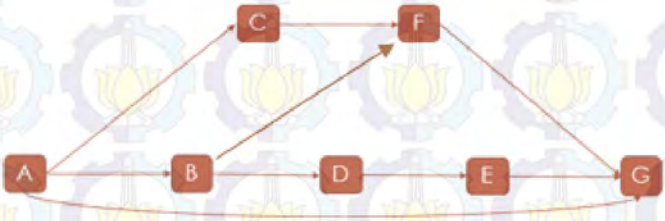
Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RTB = 0.5

PO = 1

DT = 0.39

Maka *dependency graph*-nya adalah sebagai berikut:



Gambar 3.9 *Dependency Graph* L1"

Dari *dependency graph*-nya sudah terjadi kesalahan *discovery* yaitu adanya hubungan antar aktivitas A dan G, B dan F yang sebenarnya pada model aslinya tidak terdapat hubungan antara A dan G, B dan F. Dengan ini maka proses *discovery* L1" gagal.

c. Jika jumlah *case* yang ada dari suatu *event log* sama dengan jumlah *complete trace*-nya (tanpa duplikasi), jumlah *complete trace* dari model tersebut senilai P_1^n, dan *noise* yang terdapat pada *event log* berupa *noise* yang terbentuk karena penghilangan *body* dari *trace* maka akan terjadi kegagalan *discovery*.

$$if\ m = n\ \&\&\ n = P_1^p\ \&\&\ t_{ns} = body\ t$$
$$then\ discover\ not\ success \qquad (3.16)$$

Keterangan:

t : $trace$ pada $event\ log$

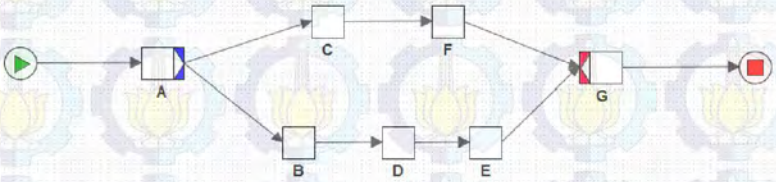
t_{ns}: bagian dari $trace$ yang dipotong

m : jumlah $case$ dalam $event\ log$

n : jumlah $complete\ trace$ dari model

Contohnya adalah sebagai berikut:

Suatu model XOR YAWL sebagai berikut:



Gambar 3.10 Model XOR YAWL

Dari model tersebut dibangkitkan *complete event log* sebagai berikut:

L2 = {(ACFG), (ABDEG)}

Kemudian dari *event log* ditambahkan *noise* menjadi: L2' = {(ACFG), (ABDEG), (AG)}

Dari *event log* L1" didapatkan perhitungan *heuristic miner algorithm* secagai berikut:

Matriks frekuensi dari *event log* L1" menghasilkan Tabel 3.17.

Tabel 3.17 Matriks Frekuensi L2'

Aktivitas	A	B	C	D	E	F	G
A	0	1	1	0	0	0	1
B	0	0	0	1	0	0	0
C	0	0	0	0	0	1	0
D	0	0	0	0	1	0	0
E	0	0	0	0	0	0	1
F	0	0	0	0	0	0	1
G	0	0	0	0	0	0	0

Lalu membentuk matriks *dependency measure* dengan menggunakan perhitungan pada Persamaan 2.1 menghasilkan Tabel 3.18.

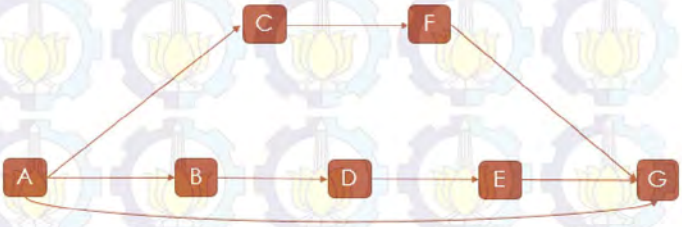
Tabel 3.18 Matriks *Dependency Measure* L2'

Aktivitas	A	B	C	D	E	F	G
A	0	0.5	0.5	0	0	0	0.5
B	0	0	0	0.5	0	0	0
C	0	0	0	0	0	0.5	0
D	0	0	0	0	0.5	0	0
E	0	0	0	0	0	0	0.5
F	0	0	0	0	0	0	0.5
G	0	0	0	0	0	0	0

Dengan menggunakan Persamaan 3.1, Persamaan 3.3, Persamaan 3.5, maka didapatkan nilai *threshold* sebagai berikut:

RTB = 0.5
PO = 1
DT = 0.5

Maka *dependency graph*-nya adalah sebagai berikut:



Gambar 3.11 *Dependency Graph* L2'

Dari *dependency graph*-nya sudah terjadi kesalahan *discovery* yaitu adanya hubungan antar aktivitas A dan G yang sebenarnya pada model aslinya tidak terdapat hubungan antara A dan G. Dengan ini maka proses *discovery* L2' gagal.

Hal ini mengakibatkan bagaimanapun nilai dari *Relative to the best threshold* maupun *dependency threshold* dan *positive observation threshold* AG akan tetap memiliki hubungan sehingga proses *discovery* salah.

Untuk mengatasi masalah ini maka paling tidak harus terdapat duplikasi untuk masing-masing *trace* pada *event log*. Dan jumlah frekuensi dari *noise* harus lebih kecil dari jumlah frekuensi dari *trace* utuh yang ada dalam *event log*.

3.5 Optimasi Waktu dan Biaya Proses Bisnis

Optimasi yang dilakukan dalam Tugas Akhir ini adalah optimasi antara tambahan biaya yang harus dibayarkan untuk memampatkan waktu eksekusi (*makespan*) dari proses bisnis. Kasus ini juga dapat disebut dengan *Time-cost Tradeoff Prolem.* Tugas akhir ini menggunakan metode *Crasing Project* yang ada pada CPM. *Crashing Project* sendiri merupakan suatu metode untuk mempersingkat lamanya waktu proyek dengan mengurangi waktu dari satu atau lebih aktivitas proyek yang penting menjadi kurang dari waktu normal aktivitas (Elmabrouk, 2011).

Terdapat dua nilai waktu yang ditunjukkan tiap aktifitas dalam suatu jaringan kerja saat terjadi c*rashing project* yaitu:

1. *Normal Duration*
   Waktu yang dibutuhkan untuk menyelesaikan suatu aktivitas atau kegiatan dengan sumber daya normal yang ada tanpa adanya biaya tambahan lain dalam sebuah proyek.
2. *Crash Duration*
   Waktu yang dibutuhkan suatu proyek dalam usahanya mempersingkat waktu yang durasinya lebih pendek dari normal duration.

Untuk membuat model matematika yang digunakan untuk optimasi diperlukan *time*. *Time slope* sendiri merupakan percepatan maksimum yang dapat dilakukan untuk *crashing project.*

$$X_i = T_{ni} - T_{ci} \quad (3.17)$$

Keterangan:

X_i : *Time slope* aktivitas ke – i dimana i ∈

T_{ni} : durasi normal aktivitas ke – i dimana i ∈

T_{ci} : *crash* duration aktivitas ke – i dimana i ∈

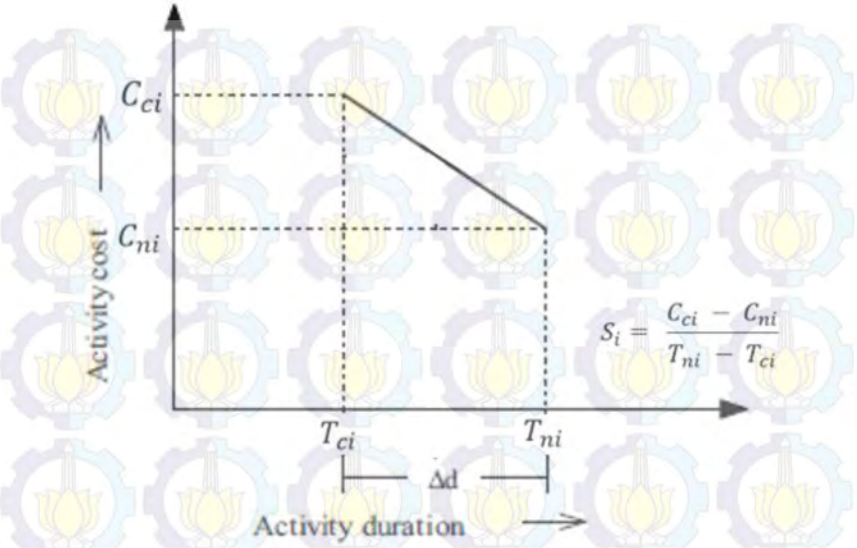
Crashing Project juga menyebabkan perubahan pada elemen biaya yaitu:

1. *Normal Cost*
   Biaya yang dikeluarkan dengan penyelesaian proyek dalam waktu normal. Perkiraan biaya ini adalah pada saat perencanaan dan penjadwalan bersamaan dengan penentuan waktu normal.
2. *Crash Cost*
   Biaya yang dikeluarkan dengan penyelesaian proyek dalam jangka waktu sebesar durasi *crash*-nya. Biaya setelah di *crashing* akan menjadi lebih besar dari biaya normal.

Perhitungan *Time-Cost TradeOff* diutamakan pada kegiatan-kegiatan yang memiliki nilai *cost slope* terendah. *Cost slope* merupakan perbandingan antara pertambahan biaya dengan percepatan waktu penyelesaian proyek. Persamaan 3.18 adalah sebagai berikut:

$$S_i = \frac{C_{ci} - C_{ni}}{T_{ni} - T_{ci}} \quad (3.18)$$

Keterangan:
S_i : *cost slope* aktivitas ke – i dimana i ∈
C_{ni} : *cost* normal aktivitas ke – i dimana i ∈
C_{ci} : *cost crash* aktivitas ke – i dimana i ∈
T_{ni} : durasi normal aktivitas ke – i dimana i ∈
T_{ci} : *crash duration* aktivitas ke- i dimana i ∈



Gambar 3.12 *Cost Slope*

3.5.1 Penentuan Durasi Normal dan *Cost* Normal

Penentuan durasi dengan rata-rata sudah dikembangkan oleh Van Der Aalst, tetapi yang yang dicari merupakan durasi dari eksekusi satu *trace* dalam *event log* (van der Aalst, Schonenberg, & Song, Time Prediction Based on Process Mining, 2011). Oleh karena itu dalam Tugas Akhir ini dikembangkan metode untuk menentukan durasi normal dan biaya normal dari setiap aktivitas dengan merata-rata setiap durasi dan biaya dari aktivitas pada setiap *case* yang ada pada *event log*. Durasi eksekusi diperoleh dengan mengurangkan *output* dari aktivitas dengan aktivitas (aktivitas yaitu A memiliki aktivitas B sebagai *output* maka eksekusi durasi A adalah waktu eksekusi B mengurangkan dengan waktu pelaksanaan A).

$$if\ activity_{output} = true\ then$$
$$ed = et_{activity_{output}} - et_{activity} \quad (3.19)$$

Keterangan:
ed : durasi eksekusi per *case*
$et_{activity_{output}}$: waktu eksekusi *output* dari aktivitas
$et_{activity}$: waktu eksekusi aktivitas

Setelah mendapatkan semua eksekusi durasi setiap aktivitas pada semua kasus di *event log* maka rata-rata pelaksanaan durasi per aktivitas dihitung untuk menentukan durasi normal.

$$T_{ni} = \frac{\sum_{i=1}^{h} ed_i}{h} \quad (3.20)$$

Setelah menghitung durasi dari menghitung biaya normal setiap aktivitas. *Event log* berisi biaya pelaksanaan setiap aktivitas dalam setiap *case*, oleh karena itu untuk mendapatkan biaya normal setiap aktivitas digunakan perhitungan biaya rata-rata aktivitas dalam setiap kasus.

$$C_{ni} = \frac{\sum_{i=1}^{h} cost_i}{h} \quad (3.21)$$

Keterangan:
T_{ni} : durasi normal setiap aktivitas ke-i dimana i ∈
ed_q : waktu eksekusi aktivitas pada *case* ke-q dimana q = 1, 2, 3, … w
C_{ni} : *cost* normal setiap aktivitas ke-i dimana i ∈
$cost_q$: biaya aktivitas pada *case* ke-i ke-q dimana q = 1, 2, … w
h : banyaknya aktivitas dieksekusi dalam *event log* (banyaknya *case* yang mengandung aktivitas)

3.5.2 Penentuan *Crash Time* dan *Crash Cost*

Dalam menentukan *crash time* dan *crash cost* dari setiap aktivitas diambil dengan mengurangi normal durasi dengan standar deviasi waktu eksekusi masing-masing aktivitas untuk *crash time* dan menambahkan biaya normal dengan standar deviasi biaya dari masih-masing aktivitas untuk *crash cost*.

3.5.3 Hubungan Cross-Organizational dengan Optimasi

Pola pada *cross-organizational business process* mempengaruhi optimasi jika terdapat salah satu organisasi yang ada pada pola tidak melakukan optimasi. Salah satu organisasi tidak dapat melakukan optimasi dapat dikarenakan banyak hal bisa jadi dikarenakan organisasi tersebuat memiliki *workload* yang sudah tidak dapat diganggu ataupun karena organisasi utama tidak mampu mengoptimasi seluruh organisasi yang diajak kerjasama sehingga mengharuskan untuk memilih organisasi yang tidak dioptimasi dan yang dioptimasi. Dari pola hubungan *cross-organizational business process* dapat dibedakan lagi dari bentuk polanya yaitu kelompok pola yang sekuensial dan paralel. Yang termasuk dalam pola sekuensial adalah pola *cross-organizational business process:*

- *Capacity sharing*
- *Chain execution*
- *Message exchange*

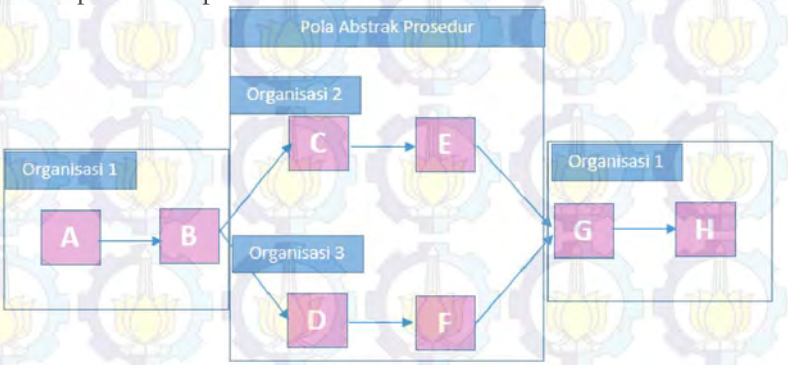
Yang termasuk dalam pola paralel adalah pola *cross-organizational business process:*

- *Case transfer*
- *Loosely couple*
- *Astrack procedure*

Untuk *cross-organizational business process* yang sekuensial, jika salah satu organisasi tidak dioptimasi maka hanya berpengaruh terhadap organisasi itu sendiri dan tidak berpengaruh terhadap *crash time* yang dapat dilakukan oleh organisasi lain yang terdapat pada pola tersebut.

Sedangkan untuk pola paralel jika salah satu organisasi yang terdapat pada pola tersebut tidak dioptimasi maka berpengaruh terhadap organisasi lain yang ada pada pola tersebut. Contohnya pada *cross-organizational business process* dengan pola *abstract procedure* memiliki 3 organisasi yang ada dalam proses bisnis

dengan satu organisasi utama dan 2 organisasi yang masuk dalam pola *abstract procedure* model dari *cross-organizational business process* dapat dilihat pada Gambar 3.13.



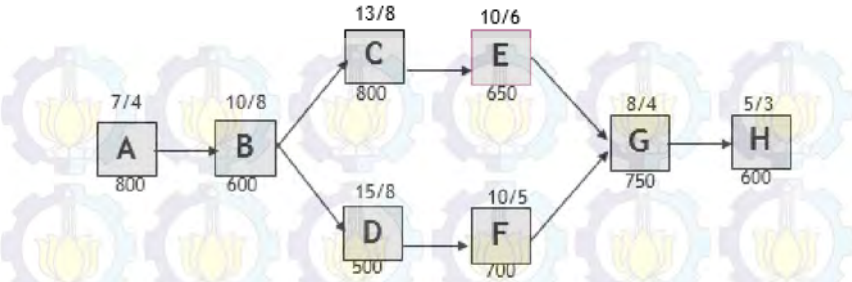
Gambar 3.13 Model *Pola Abstract Procedure*

Model pada Gambar 3.13 memiliki data untuk optimasi seperti pada Tabel 3.19.

Tabel 3.19 Data Optimasi Gambar 3.13

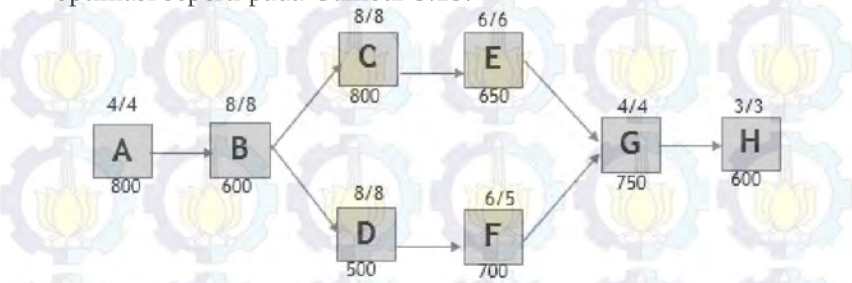
Aktivitas	Normal Dur	Crashed dur	Time Slope	Cost Slope
A	7	4	3	800
B	10	8	2	600
C	13	8	5	800
D	15	8	7	500
E	10	6	4	650
F	10	5	5	700
G	8	4	4	750
H	5	3	2	600

Jika model pada Gambar 3.13 digambarkan dengan nilai data optimasinya maka menjadi seperti pada Gambar 3.14.



Gambar 3.14 Model Dengan Nilai Data Optimasi

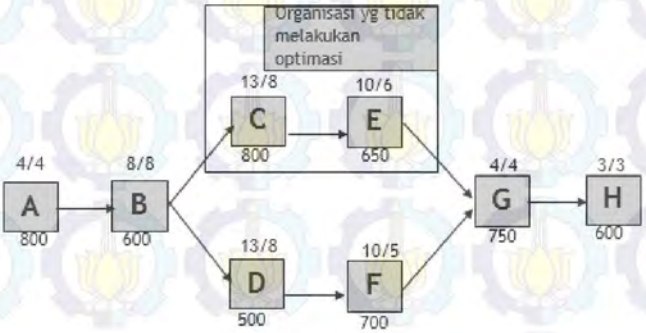
Dengan CPM maka didapatkan *critical path* A-B-D-F-G-H dengan *makespan* 55. Jika semua organisasi dapat dioptimasi maka hasil optimasi seperti pada Gambar 3.15.



Gambar 3.15 Model Optimasi dengan Semua Organisasi Dioptimasi

Maka didapatkan hasil durasi setelah dioptimasi adalah 33. Jika organisasi 2 dalam model Gambar 3.13 merupakan organisasi yang tidak dapat dioptimasi. Maka hasil optimasi model Gambar 3.13 akam menjadi seperti pada Gambar 3.16 dengan hasil optimasi 42. Hal ini dikarenakan aktivitas pada organisasi 3 yaitu D dan F tidak dapat dioptimasi secara optimal, karena organisasi 2 tidak dioptimasi maka batas untuk optimasi organisasi 3 adalah durasi yang dimiliki oleh organisasi 2. Dapat dilihat pada aktivitas D yang seharusnya dapat dioptimasi menjadi 8 jika semua organisasi dioptimasi tapi karena organisasi 2 tidak dioptimasi maka hanya bisa dioptimasi menjadi 13, begitu pula organisasi F yang seharusnya dapat dioptimasi menjadi 6 karena organisasi 2 tidak

dioptimasi maka F tetap memiliki nilai durasi 10. Hal ini menunjukkan bahwa untuk pola paralel *cross-organizational business process* jika salah satu organisasi tidak dioptimasi maka mempengaruhi organisasi lain yang ada pada pola tersebut.



Gambar 3.16 Model Optimasi dengan Organisasi 2 Tidak Dioptimasi

3.5.4 Pemodelan Fungsi Linear untuk Optimasi

Dalam pemodelan fungsi matematika untuk model optimasi ini menggunakan fungsi objektif dan beberapa fungsi batasan. Fungsi objektf untuk pemodelan matematika disini menggunakan fungsi objektif maksimum dua fungsi objektif yang pertama adalah fungsi untuk mencari waktu durasi *crash* proyek yang paling minimum. Untuk mencari durasi *crash* proses bisnis yang paling minimum maka fungsi objektif yang digunakanan adalah maksimal dari biaya *crashing* proses bisnis.

- Fungsi Objektif 1

$$MAXIMIZE \sum_{i=1}^{j} S_i X_i \qquad (3.22)$$

Keterangan:

j : banyaknya aktivitas pada proses bisnis
S_i : *cost slope* aktivitas ke – i dimana i ∈
X_i : *time slope* yang terjadi untuk penyelesaian aktivitas ke – i dimana i ∈ (variable keputusan)

Sedangkan untuk fungsi batasannya adalah sebagai berikut:

- Fungsi batasan
- *Maximum reduction constrain*

$$X_i \leq T_{ni} - T_{ci} \; for \; all \; i \tag{3.23}$$

2. *Non-negative constraint*

$$X_i \geq 0 \; for \; all \; i \tag{3.24}$$

$$Y_i \geq 0 \; for \; all \; i \tag{3.25}$$

Keterangan:

Y_i : variabel *start time* aktivitas ke – i dimana i ∈

3. *Start time constraint*

$$Y_i - Y_{pi} + X_{pi} \geq K \; for \; all \; i \tag{3.26}$$

Keterangan:

Y_{pi} : variabel *start time predecessor* aktivitas ke – i dimana i ∈ {Aktivitas}

X_{pi} : variabel *time slope predecessor* aktivitas ke – i dimana i ∈ {Aktivitas}

K : nilai *normal time predecessor* aktivitas ke – i dimana i ∈ {Aktivitas}

4. *Project duration constraint*

$$Y'_{Finish} - D \leq 0 \tag{3.27}$$

Keterangan:

D : maksimum durasi pada critical path pada proses bisnis

Y'_{Finish}: durasi proses bisnis paling minimum setelah dimampatkan

Kemudian untuk mencari biaya tambahan yang paling minimum untuk durasi proses bisnis yang paling minimum dilakukan dengan merubah fungsi objektif dan *project duration constraint,* sedangkan untuk *maximum reduction constrain, non-negative constraint,* dan *start time constraint* tetap sama.

- Fungsi Objektif 2

$$MINIMIZE \; \sum_{i=1}^{j} S_i X_i \tag{3.28}$$

Keterangan:

j : banyaknya aktivitas pada proses bisnis

S_i : *cost slope* aktivitas ke – i dimana i ∈

X_i : *time slope* yang terjadi untuk penyelesaian aktivit ke – i dimana i ∈ (variable keputusan)

- *Project duration constraint*

$$Y_{Finish} - Y'_{Finish} \leq 0 \quad (3.29)$$

Keterangan:

D : maksimum durasi pada critical path pada proses bisnis

Y_{Finish} : durasi hasil akhir durasi *crashing* proses bisnis

Y'_{Finish} : durasi proses bisnis paling minimum setelah dimampatkan

3.7 Contoh Optimasi Waktu dan Biaya

Tabel 3.20 menunjukkan *single timestamps event log* yang akan digunakan untuk menghitung data optimasi. *Single timestamps* adalah adanya satu waktu untuk setiap aktivitas pada *event log.*

Tabel 3.20 Event Log

Case ID	Activity	TimeStamp	Originator	Cost
300.PO.IMP.201406.0024	A	6/20/2014 8:32	AN	400
300.PO.IMP.201406.0024	D	6/20/2014 13:42	AN	1500
300.PO.IMP.201406.0024	B	6/20/2014 23:41	AN	1000
300.PO.IMP.201406.0024	E	6/21/2014 8:16	AN	2000
300.PO.IMP.201406.0024	C	6/21/2014 10:46	AN	1200
300.PO.IMP.201406.0015	A	6/21/2014 16:57	AN	1000
300.PO.IMP.201406.0015	B	6/21/2014 18:09	AN	400
300.PO.IMP.201406.0015	C	6/22/2014 5:15	AN	500
300.PO.IMP.201408.0021	A	6/22/2014 10:51	AN	50
300.PO.IMP.201408.0021	B	6/22/2014 23:42	AN	1000
300.PO.IMP.201408.0021	D	6/22/2014 23:42	AN	1000
300.PO.IMP.201408.0021	E	6/23/2014 4:48	AN	2000
300.PO.IMP.201408.0021	C	6/23/2014 16:44	AN	1000
300.PO.IMP.201406.0025	A	6/23/2014 23:26	AN	400
300.PO.IMP.201406.0025	D	6/24/2014 4:19	AN	1000
300.PO.IMP.201406.0025	E	6/24/2014 7:48	AN	500
300.PO.IMP.201406.0025	B	6/25/2014 1:08	AN	500
300.PO.IMP.201406.0025	C	6/25/2014 3:02	AN	500
300.PO.IMP.201408.0026	A	6/25/2014 5:11	AN	500
300.PO.IMP.201408.0026	B	6/25/2014 12:45	AN	1000
300.PO.IMP.201408.0026	D	6/25/2014 12:45	AN	1000
300.PO.IMP.201408.0026	E	6/26/2014 0:12	AN	2000
300.PO.IMP.201408.0026	D	6/26/2014 4:48	AN	1000
300.PO.IMP.201408.0026	E	6/26/2014 15:16	AN	100
300.PO.IMP.201408.0026	C	6/26/2014 22:49	AN	1000

Dari Tabel 3.20 didapatkan dengan cara yang telah disebutkan pada 3.5.1 dan 3.5.2 maka didapatkan data seperti pada Tabel 3.21.

Tabel 3.21 Tabel data Optimasi

Activity	Normal Duration	Crassh Duration	Normal Cost	Cash Cost	Time Slope	Cost Slope
A	1.66	0.42	470	812.05	1.24	275.85
D	1.97	0.73	1100	1323.61	1.24	180.33
B	2.9	0.65	680	974.96	2.25	131.09
E	2.13	1.01	1320	2261.81	1.12	840.9
C	1.89	0.95	840	1160.94	0.94	341.43

Model matematika untuk mencari *crashing* durasi proses bisnis paling minimum menggunakan Persamaan 3.22 sampai Persamaan 3.29:

- Fungsi objektif 1

  $MAXIMUM = (275.85*X_A + 180.33*X_D + 131.09*X_B + 804.9*X_E + 341.43*X_C);$

- Fungsi batasan

1. Maximum reduction constrain

   $X_A<=1.24;\ X_D<=1.24;\ X_B<=2.25;\ X_E<=1.12;\ X_C<=0.94;$

2. Non-negative constraint

   $X_A>=0;\ X_D>=0;\ X_B>=0;\ X_E>=0;\ X_C>=0;$
   $Y_A>=0;\ Y_D>=0;\ Y_B>=0;\ Y_E>=0;\ Y_C>=0;$

3. Start time constraint

   $Y_D - Y_A + X_A >= 1.66;\ Y_B - Y_A + X_A >= 1.66;\ Y_E - Y_D + X_D >= 1.97;\ Y_C - Y_E + X_E >= 2.13;\ Y_C - Y_B + X_B >= 2.9;\ Y_{FINISH} - Y_C + X_C >= 1.89;$

4. Project duration constraint

   $Y'_{FINISH} - 7.65 <= 0$

Diketahui *crashing* durasi proses bisnis paling minimum adalah 3.11.

Model matematika untuk mencari biaya tambahan minimum:

- Fungsi objektif 2

  $MINIMUM = (275.85*X_A + 180.33*X_D + 131.09*X_B + 804.9*X_E + 341.43*X_C);$

- Fungsi batasan

2. Maximum reduction constrain

   $XA<=1.24;\ XD<=1.24;\ XB<=2.25;\ XE<=1.12;\ XC<=0.94;$

5. Non-negative constraint

   $X_A>=0;\ X_D>=0;\ X_B>=0;\ X_E>=0;\ X_C>=0;$
   $Y_A>=0;\ Y_D>=0;\ Y_B>=0;\ Y_E>=0;\ Y_C>=0;$

6. Start time constraint

   $Y_D - Y_A + X_A >= 1.66;\ Y_B - Y_A + X_A >= 1.66;\ Y_E - Y_D + X_D >= 1.97;\ Y_C - Y_E + X_E >= 2.13;\ Y_C - Y_B + X_B >= 2.9;\ Y_{FINISH} - Y_C + X_C >= 1.89;$

7. Project duration constraint

   $Y_{FINISH} - 3.11 <= 0$

Dari perhitungan menggunakan metode *simplex linear programming* didapatkan biaya minimum tambahan sebesar 1980.48 dan durasi project menjadi 3.1 jam dari yang sebelumnya 5.5 jam.

BAB IV
ANALISIS DAN PERANCANGAN SISTEM

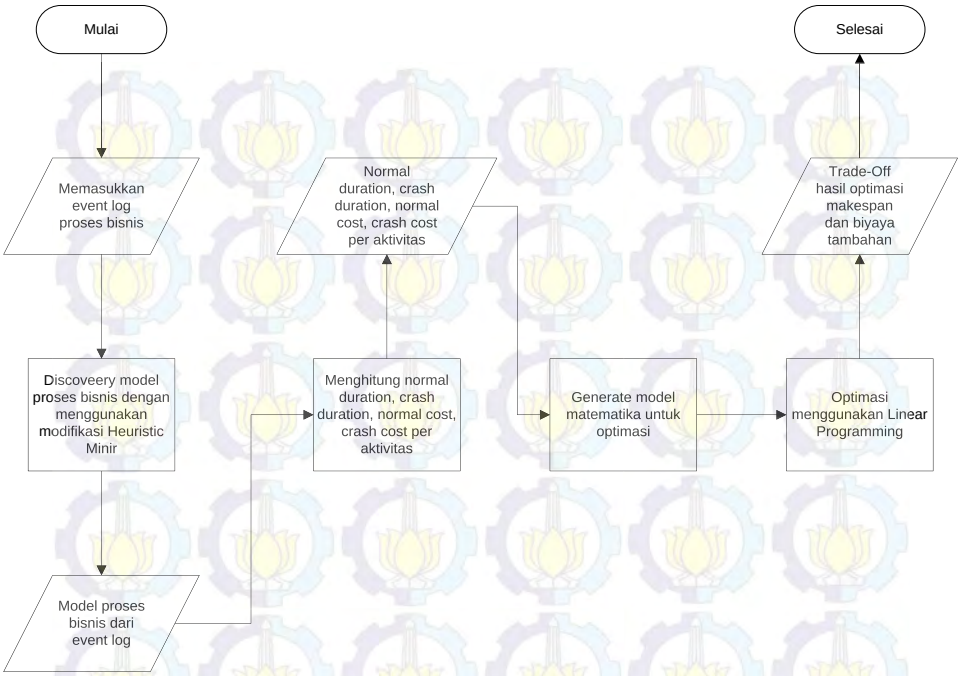
Bab ini membahas tahap analisis permasalahan dan perancangan Tugas Akhir. Analisis permasalahan membahas permasalahan yang yang diangkat dalam pengerjaan Tugas Akhir. Solusi yang ditawarkan oleh penulis juga dicantumkan pada tahap permasalahan analisis ini. Analisis kebutuhan mencantumkan kebutuhan-kebutuhan yang diperlukan perangkat lunak. Selanjutnya dibahas mengenai perancangan sistem yang dibuat. Perancangan direpresentasikan dengan diagram UML (*Unified Modelling Language*).

4.1 Analisis

Tahap analisis dibagi menjadi beberapa bagian antara lain cakupan permasalahan, deskripsi umum sistem, kasus penggunaan sistem, dan kebutuhan perangkat lunak.

4.2 Deskripsi Umum Sistem

Perangkat lunak yang akan dibangun dapat menghasilkan model dari proses bisnis, durasi normal, durasi *crash*, biaya normal dan biaya *crash* per aktivitas yang ada dalam proses bisnis, dan biaya tambahan dan *makespan* yang paling optimum untuk melakukan percepatan durasi proses bisnis. Proses pemodelan proses bisnis tersebut membutuhkan data atau *event log* proses bisnis yang sudah ada dan berjalan. Dari *event log* yang digunakan sebagai masukan dilakukan serangkaian proses hingga menghasilkan keluaran yang diharapkan. Gambar 4.1 merupakan alur pemrosesan dan bentuk arsitektur perangkat secara sederhana.



Gambar 4.1. Alur Sistem

Pada fase *discovery*, pengguna akan memilih *file event log* yang akan di-*discovery* bentuk proses modelnya. Setelah pengguna memilih *file event log* yang di-*discovery* bentuk proses modelnya hal yang paling awal dilakukan adalah melakukan proses pada *file event log* yang dipilih dengan menggunakan teknik *reading from* Excel *file* menggunakan Bytescout Spreadsheet. Setiap elemen-elemen *case id* pada *file* Excel *event log* diidentifikasi *trace-trace* yang terdapat pada *event log* dan juga frekuensi *trace* yang ada pada event log. Setelah mendapatkan *trace* dan masing-masing frekuensinya, kemudian menggunakan modifikasi dari *heuristic miner* dicari hubungan dari masing-masing aktivitas dari *event log*, sehingga dapat mengasilkan model dari *event log* yang dipilih oleh pengguna.

Setelah mendapatkan model dari *event log* yang dimasukkan, selanjutnya memasuki tahap perhitungan data optimasi berupa durasi normal, durasi *crash*, biaya normal, dan biaya *crash*. Seperti yang telah dijelaskan pada 3.5.1 dan 3.5.2 penentuan data optimasi dilakukan dengan rata-rata dan standar deviasi. Sesuai dengan *time prediction* yang dikembangkan oleh Van Der Aalst rata-rata yang diperoleh harus didapatkan dari eksekusi antar waktu aktivitas yang saling berhubungan oleh karena itu sebelum mencari data optimasi diharuskan melalui proses *discovery* terlebih dahulu.

Selanjutnya adalah fase optimasi yang akan mendapatkan nilai optimum dari percepatan *makespan* dengan biaya tambahan yang paling minimum dengan menggunakan *linear programming* sesuai yang dijelaskan pada 3.5.3.

4.3 Spesifikasi Kebutuhan Perangkat Lunak

Bagian ini berisi semua kebutuhan perangkat lunak yang diuraikan secara rinci dalam bentuk diagram kasus, diagram urutan, dan diagram aktivitas. Masing-masing diagram menjelaskan perilaku atau sifat dari sistem ini. Kebutuhan perangkat lunak dalam sistem ini mencakup kebutuhan fungsional saja. Pada bab ini juga dijelaskan tentang spesifikasi terperinci pada masing-masing kebutuhan fungsional. Rincian spesifikasi dari kasus penggunaan disajikan dalam bentuk tabel.

4.4 Kebutuhan Fungsional

Kebutuhan fungsional berisi kebutuhan utama yang harus dipenuhi oleh sistem agar dapat bekerja dengan baik. Kebutuhan fungsional mendefinisikan layanan yang harus disediakan oleh sistem, bagaimana reaksi terhadap masukan, dan apa yang harus dilakukan sistem pada situasi khusus. Daftar kebutuhan fungsional dapat dilihat pada Tabel 4.1.

Tabel 4.1 Daftar Kebutuhan Fungsional Perangkat Lunak

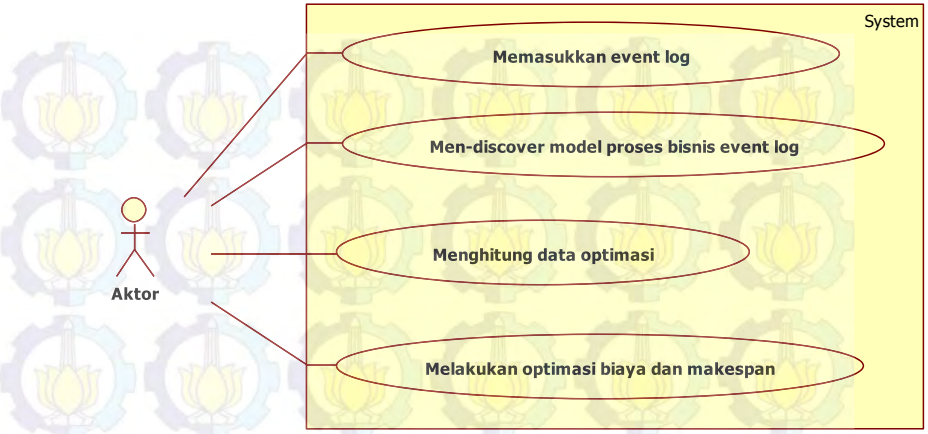
Kode Kebutuhan	Kebutuhan Fungsional	Deskripsi
TA-F0001	Memasukkan event log	Pengguna dapat memasukkan *event log* dalam bentuk Excel
TA-F0002	Men-*disover* bisnis proses model	Pengguna dapat men-*disover* bisnis proses model sehingga mengetahui bentuk *graph* bisnis proses dan hubungan antar aktivitas.
TA-F0003	Menghitung data optimasi	Pengguna dapat menghitung data optimasi berupa durasi normal, durasi *crash*, biaya normal dan biaya *crash* per-aktivitas yang ada dalam proses bisnis
TA-F0004	Optimasi biaya tambahan dan percpatan *makespan*	Pengguna dapat mengentahui hasil optimasi yaitu berupa biaya tambahan dan *makespan* percepatan yang minimum dari proses bisnis.

4.5 Aktor

Aktor mendefinisikan entitas-entitas yang terlibat dan berinteraksi langsung dengan sistem. Entitas ini bisa berupa manusia maupun sistem atau perangkat lunak yang lain. Penulis mendefinisikan aktor untuk sistem ini yaitu pengguna dari aplikasi yang dibangung oleh penulis.

4.6 Kasus Penggunaan

Kasus-kasus penggunaan dalam sistem ini akan dijelaskan secara rinci pada subbab ini. Kasus penggunaan secara umum akan digambarkan oleh salah satu model UML, yaitu diagram kasus penggunaan. Rincian kasus penggunaan berisi spesifikasi kasus penggunaan, diagram aktivitas, dan diagram urutan untuk masing-masing kasus penggunaan. Diagram kasus penggunaan dapat dilihat pada Gambar 4.2. Daftar kode diagram kasus penggunaan sistem dapat dilihat pada Tabel 4.2.



Gambar 4.2 Diagram Kasus Penggunaan Sistem

Tabel 4.2 Daftar Kode Diagram Kasus Penggunaan

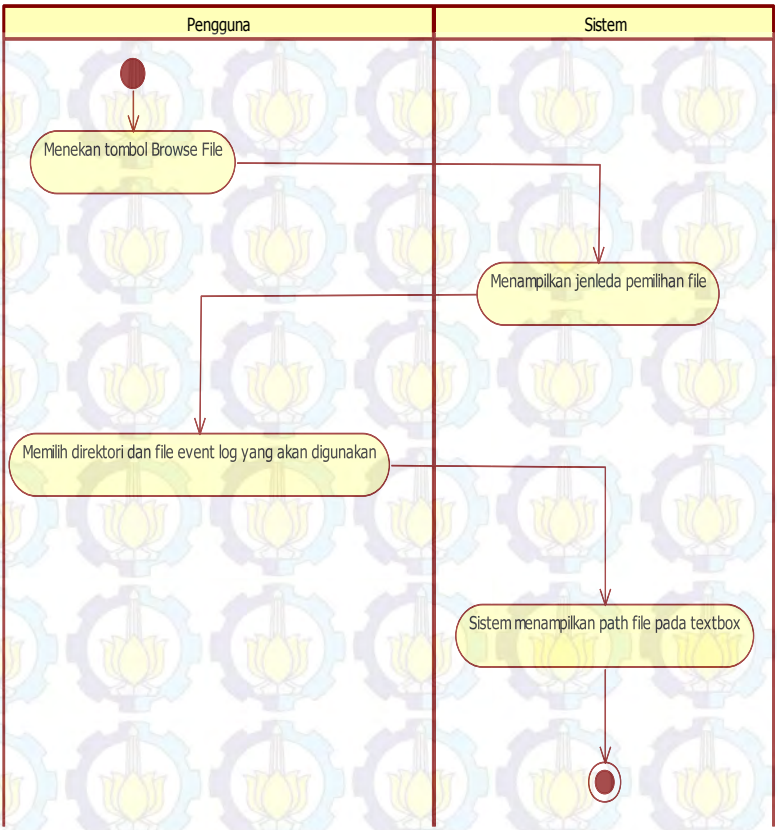
Kode Kasus Penggunaan	Nama
TA-UC0001	Memasukkan data event log
TA-UC0002	Men-*discover* model proses bisnis
TA-UC0003	Menghitung data optimasi
TA-UC0004	Melakukan optimasi biaya dan *makespan*

4.6.1 Memasukkan dan Membaca Data Event Log

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk memasukkan data yang berupa *event log* proses bisnis dalam notasi Excel. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.3 dan diagram aktivitas dari kasus penggunaan ini bisa dilihat pada Gambar 4.3.

Tabel 4.3 Spesifikasi Kasus Penggunaan Memasukkan dan Membaca *Data Event Log*

Nama	Menampilkan similarity metric
Kode	TA-UC0001
Deskripsi	Memasukkan data event log.
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Browse File* dan memilih *file* Excel *event log* yang akan digunakan
Aktor	Pengguna
Kondisi Awal	*Event log* yang akan diproses belum ada
Aliran:	
-Kejadian Normal	1. Pengguna menekan tombol *browse* 2. Pengguna memilih *file* Excel *event log* yang terdapat dalam direktori 3. Sistem mengambil *path* direktori dan mengambil *file* Excel *event log* ke dalam sistem
-Kejadian Alternatif	Tidak ada
Kondisi Akhir	Sistem menampilkan *path file* Excel *event log* dalam *textbox* pada sistem
Kebutuhan Khusus	Tidak ada



Gambar 4.3 Diagram Aktivitas Memasukkan dan Membaca Data *Event Log*

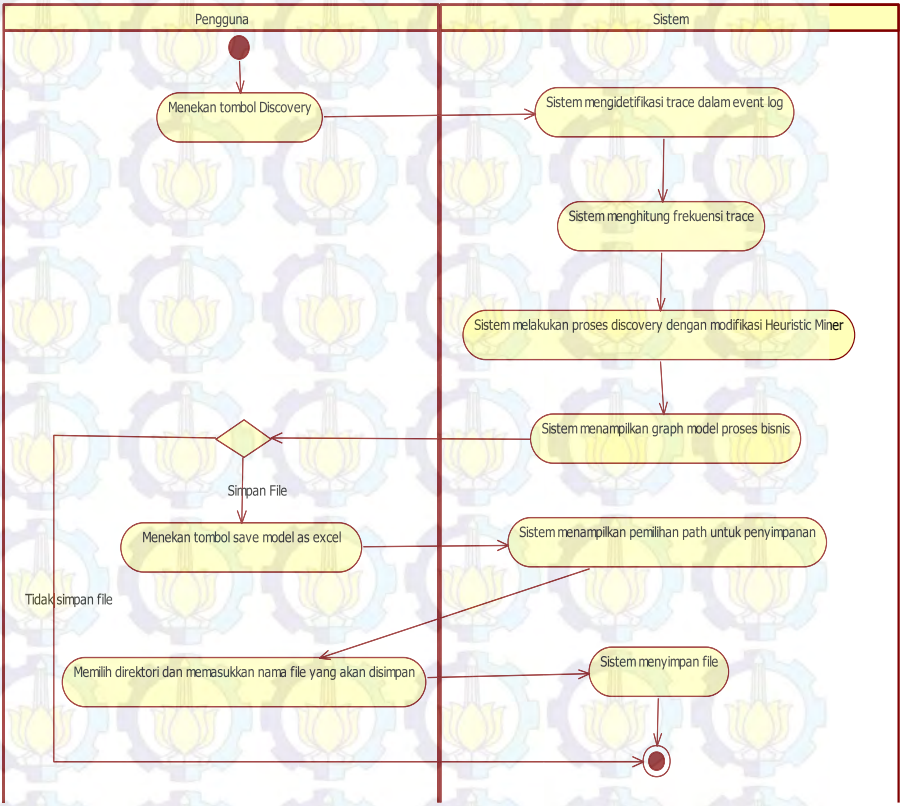
4.6.2 Men-*discover* Model Proses Bisnis

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk melakukan proses *discovery* pada *event log* yang telah dimasukkan oleh pengguna. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.4 dan diagram aktivitas dan diagram urutan dari kasus penggunaan ini bisa dilihat pada Gambar 4.4.

Tabel 4.4 Spesifikasi Kasus Penggunaan Men-*discover* Model Proses Bisnis

Nama	Men-*discover* model proses bisnis
Kode	TA-UC0002
Deskripsi	Pengguna dapat men-*disover* bisnis proses model sehingga mengetahui bentuk *graph* bisnis proses dan hubungan antar aktivitas.
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Discover* dan kemudian untuk menyimpan hasil dalam bentuk Excel pengguna menekan tombol *Save Model As* Excel
Aktor	Pengguna
Kondisi Awal	*Path file* Excel *event log* dalam *textbox* sudah terisi
Aliran:	
- **Kejadian Normal**	1. Pengguna menekan tombol *Discover.* 2. Sistem mengolah *file event log* dan menampilkan bentuk model proses bisnis dari *event log.* 3. Pengguna menekan tombol *Save Model As* Excel. 4. Pengguna memilih direktori penyimpanan dan memasukkan nama *file.* 5. Sistem mengolah data sehingga berubah dalam bentuk tabel Excel dan menyimpannya sesuai dengan nama dan direktori yang telah dipilih pengguna.
- **Kejadian Alternatif**	Pengguna tidak menyimpan hasil proses *discovery.* Maka sistem tidak akan menyimpan hasil proses *discovery.*

Kondisi Akhir	Sistem menampilkan *graph* model proses bisnis dan menyimpannya dalam bentuk tabel di Excel jika pengguna ingin menyimpannya.
Kebutuhan Khusus	Tidak ada



Gambar 4.4 Diagram Aktivitas Men-*discover* Model Proses Bisnis

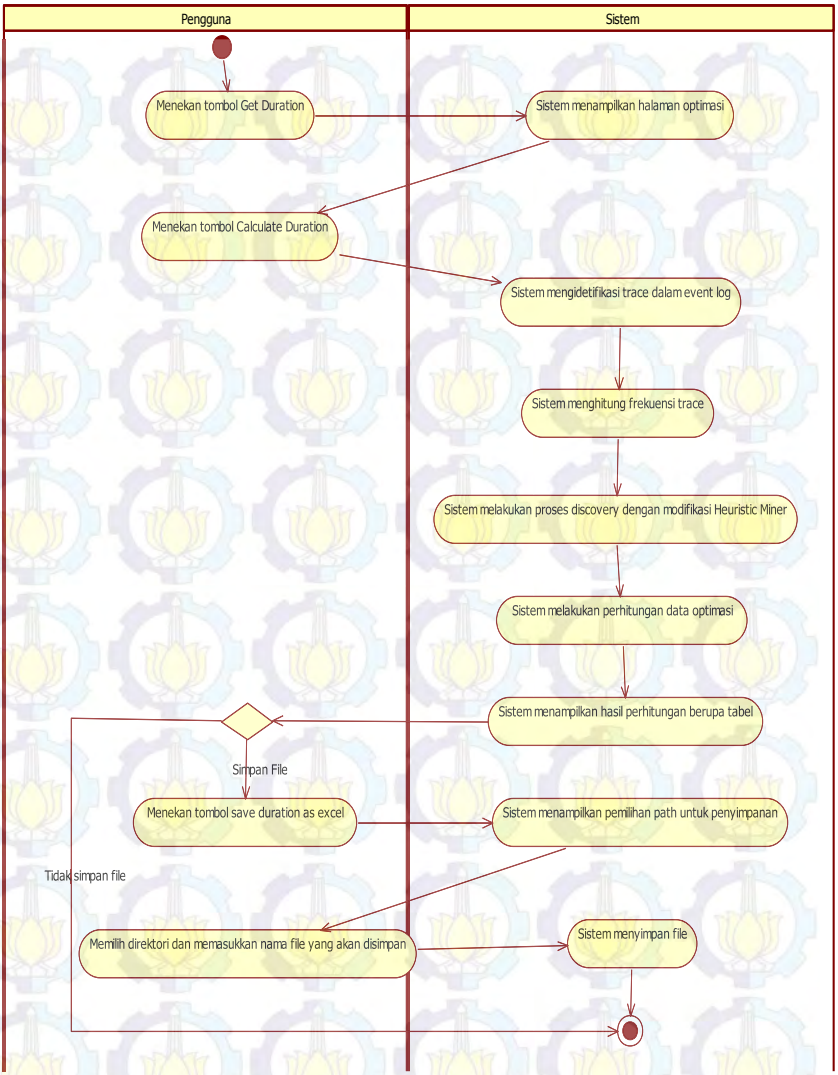
4.6.3 Menghitung Data Optimasi

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk mulai melakukan perhitungan data optimasi dari *event log* yang telah dipilih. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.5. Diagram aktivitas dari kasus penggunaan ini bisa dilihat pada Gambar 4.5.

Tabel 4.5. Spesifikasi Kasus Penggunaan Menghitung Data Optimasi

Nama	Menghitung data optimasi
Kode	TA-UC0003
Deskripsi	Pengguna dapat menghitung data optimasi berupa durasi normal, durasi *crash*, biaya normal dan biaya *crash* per aktivitas yang ada dalam proses bisnis
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Get Duration* untuk pindah ke halaman optimasi dan kemudian untuk menghitung data optimasi pengguna menekan tombol *Calculate Duration*, kemudian untuk menyimoan hasil dalam bentuk Excel pengguna menekan tombol *Save Duration As* Excel
Aktor	Pengguna
Kondisi Awal	*Path file* Excel *event log* dalam *textbox* sudah terisi
Aliran:	
-Kejadian Normal	1. Pengguna menekan tombol *Get Duration.* 2. Sistem menampilkan halaman optimasi. 3. Pengguna menekan tombol *Calculate Duration.* 4. Sistem mengolah *file event log* dengan fungsi *discovery* dan fungsi perhitungan data optimasi. 5. Sistem menampilkan hasil perhitungan data optimasi dalam bentuk tabel. 6. Pengguna menekan tombol *Save Duration As* Excel. 7. Pengguna memilih direktori penyimpanan dan memasukkan nama *file.*

	8. Sistem menyimpan hasil perhitungan sesuai dengan nama dan direktori yang telah dipilih pengguna.
-Kejadian Alternatif	Pengguna tidak menyimpan hasil perhitungan. Maka sistem tidak akan menyimpan hasil perhitungan.
Kondisi Akhir	Sistem menampilkan data optimasi dalam bentuk table
Kebutuhan Khusus	Tidak ada



Gambar 4.5. Diagram Aktivitas Menghitung Data Optimasi

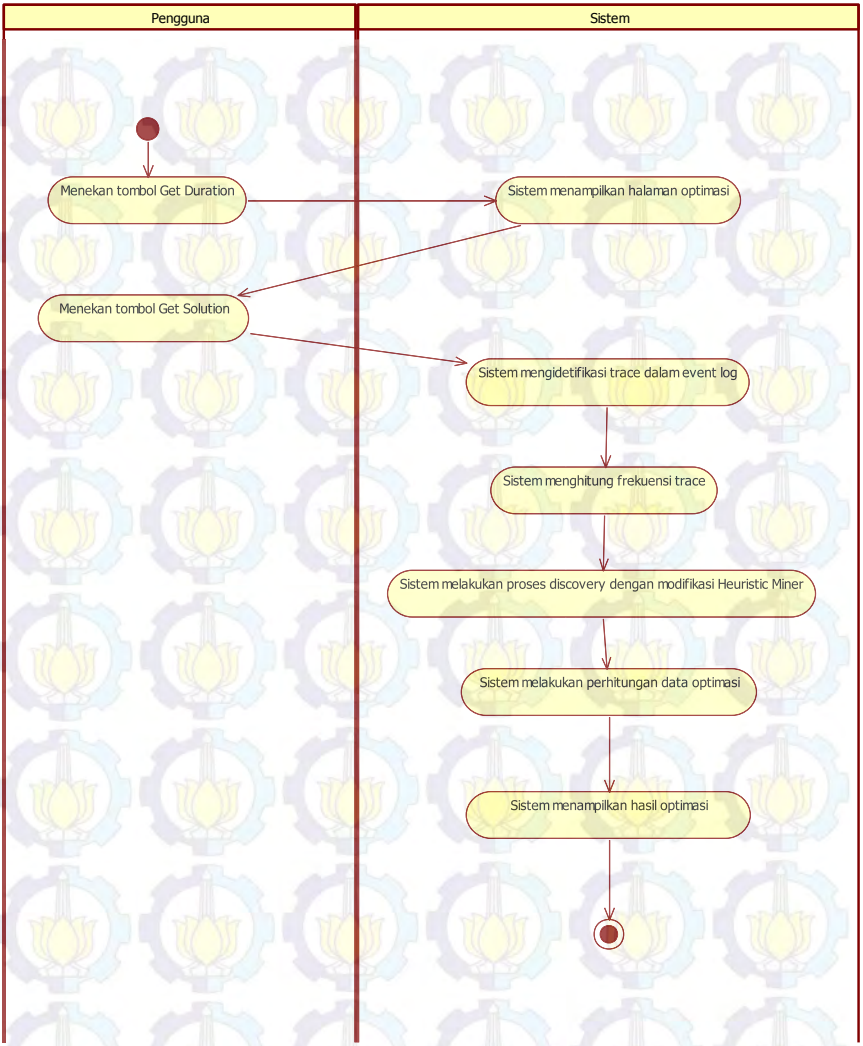
4.6.4 Melakukan Optimasi Biaya dan *Makespan*

Pada kasus penggunaan ini, sistem akan menerima perintah dari pengguna untuk mulai melakukan optimasi biaya dan *makespan* dari event log yang telah dipilih. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.6. Diagram aktivitas urutan dari kasus penggunaan ini bisa dilihat pada Gambar 4.6.

Tabel 4.6. Spesifikasi Kasus Penggunaan Melakukan Optimasi Biaya dan *Makespan*

Nama	Melakukan optimasi biaya dan *makespan*
Kode	TA-UC0004
Deskripsi	Pengguna dapat mengentahui hasil optimasi yaitu berupa biaya tambahan dan *makespan* percepatan yang minimum dari proses bisnis.
Tipe	Fungsional
Pemicu	Pengguna menekan tombol *Get Duration* untuk pindah ke halaman optimasi dan kemudian untuk menghitung data optimasi pengguna menekan tombol *Get Solution*.
Aktor	Pengguna
Kondisi Awal	*Path file* Excel *event log* dalam *textbox* sudah terisi
Aliran:	
-Kejadian Normal	1. Pengguna menekan tombol *Get Duration.* 2. Sistem menampilkan halaman optimasi. 3. Pengguna menekan tombol *Get Solution.* 4. Sistem mengolah *file event log* dengan fungsi fungsi perhitungan data optimasi, *generate* model matematika, dan fungsi *solve* model matematika. 5. Sistem menampilkan hasil optimasi berupa biaya tambahan minimum, aktivitas yang dimampatkan, dan durasi proyek yang telah dimampatkan.
-Kejadian Alternatif	1. Pengguna dapat memilih organisasi yang tidak dioptimasi pada *dropdown* dengan label *Organization which is not optimized.*

	2. Pengguna dapat menyimpan hasil optimasi pada Excel.
Kondisi Akhir	Sistem menampilkan hasil optimasi berupa biaya tambahan minimum dan durasi yang telah dimampatkan. Hasil optimasi dapat tersimpan dalam Excel.
Kebutuhan Khusus	Tidak ada



Gambar 4.6. Diagram Aktivitas Melakukan Optimasi Biaya dan *Makespan*

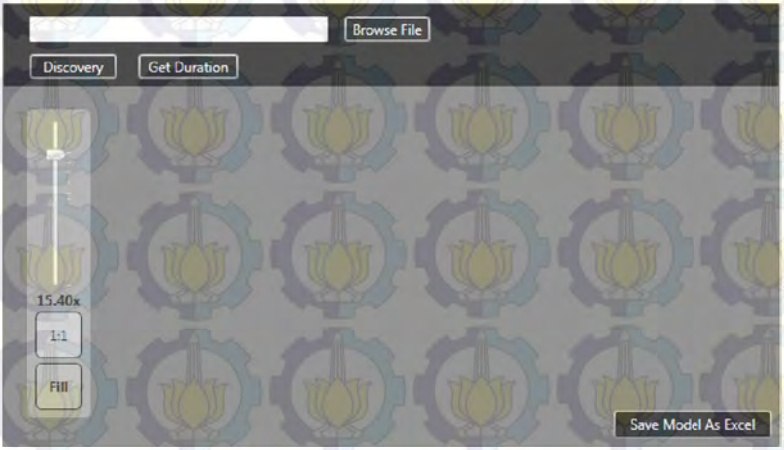
4.7 Perancangan Sistem

Penjelasan tahap perancangan perangkat lunak dibagi menjadi beberapa bagian yaitu perancangan proses analisis dan perancangan antarmuka.

4.8 Perancangan Antarmuka Pengguna

Bagian ini membahas mengenai perancangan antarmuka pada sistem. Terdapat dua kelas *view* yang masing-masing kelas memiliki fungsionalnya masing-masing. Kedua kelas tersebut merupakan form dan membutuhkan interaksi dari pengguna.

4.8.1 Halaman *Process Discovery*



Gambar 4.7 Rancangan Halaman *Process Discovery*

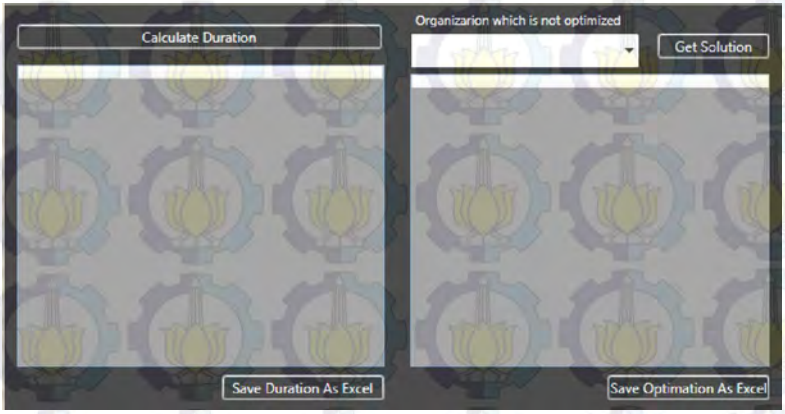
Halaman ini merupakan halam pertama yang muncul ketika aplikasi dijalankan. Pada halaman ini terdapat beberapa parameter yang harus diisi dan dibutuhkan untuk pemrosesan selanjutnya. Parameter tersebut antara lain *path* direktori tempat *event log* berada. Setelah mendapat path sistem akan membuka *file event log* yang telah dipilih. Setelah *file* terbuka maka sistem akan melakukan proses *discovery* atau bisa langsung melakukan optimasi tergantung kebutuhan dari pengguna. Rancangan halaman

utama ini dapat dilihat pada Gambar 4.7. Penjelasan mengenai atribut-atribut yang terdapat pada halaman ini bisa dilihat pada Tabel 4.7.

Tabel 4.7 Spesifikasi Atribut Antarmuka *Process Discovery*

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Jenis Masukan / Keluaran
1	Tombol *Browse File*	ActionButton	Mencari *file* Excel yang akan dikonversi.	*Action*
2	Teks Box	TextBox	Menampilkan *path* dari *file event log* yang telah dipilih	*String*
3	Tombol *Discovery*	ActionButton	Memproses *file event log* untuk mendapatkan model dari proses bisnis.	*Action*
4	Tombol *Get Duration*	ActionButton	Menuju halaman optimasi	*Action*
5	Tombol *Save Model As* Excel	ActionButton	Menyimpan hasil proses *discovery* dalam bentuk Excel	*Action*
6	*Scroll bar*	Action	Mengatur ukuran graph hasil *discovery*	*Action*

4.8.2 Halaman Optimasi



Gambar 4.8 Rancangan Antarmuka Optimasi

Halaman ini akan muncul ketika pengguna ingin melakukan optimasi tambahan biaya dan *makespan* percepatan yang dilakukan pada proses bisnis. Halaman ini akan menampilkan tombol-tombol yang digunakan untuk melakukan optimasi. Rancangan halaman optimasi ini dapat dilihat pada Gambar 4.8. Penjelasan mengenai atribut-atribut yang terdapat pada halaman ini bisa dilihat pada Tabel 4.8.

Tabel 4.8 Spesifikasi Atribut Antarmuka Optimasi

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Jenis Masukan / Keluaran
1	Tombol *Calculate Duration*	ActionButton	Memproses *file event log* dan melakukan perhitungan data optimasi	*Action*
2	Tombol *Save Duration As* Excel	ActionButton	Menyimpan hasil proses perhitungan data optimasi dalam bentuk Excel	*Action*
3	*Dropdown Organization*	Combox	Memilih organisasi yang tidak dioptimasi	*String*

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan	Jenis Masukan / Keluaran
	which is optimized			
4	Tombol *Get Solution*	ActionButton	Memproses *file event log* dan menghitung hasil optimasi minimum dutasi proyek dan minimum biaya tambahan yang dibutuhkan.	*Action*

[Halaman ini sengaja dikosongkan]

BAB V
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bab ini berisi proses implementasi dari setiap kelas pada semua modul. Bahasa pemrograman yang digunakan adalah bahasa pemrograman C#.

5.1 Lingkungan Implementasi

Lingkungan implementasi merupakan lingkungan dimana sistem ini dibangun, dimana akan dijelaskan mengenai kebutuhan perangkat yang diperlukan untuk membangun sistem ini. Lingkungan implementasi dibagi menjadi dua, yaitu lingkungan implementasi terhadap perangkat keras dan lingkungan implementasi terhadap perangkat lunak.

5.1.1 Perangkat Keras

Implementasi dilakukan pada sebuah Laptop dengan spesifikasi sebagai berikut:

- Merk : Lenovo
- Seri : Idea Pad Z470
- Processor : Processor Intel Core i3-2310M CPU @ 2.10GHz
- RAM : 4 GB

5.1.2 Perangkat Lunak

Suatu sistem atau perangkat lunak belum tentu bisa berdiri sendiri, dan sistem ini membutuhkan perangkat lunak lain yang dapat mendukung fungsionalitasnya, yang antara lain:

1. Sistem Operasi Windows

   Sistem operasi yang digunakan adalah Microsoft Windows 7 Professional 32-bit.
2. Microsoft Visual Studio

   Kakas bantu yang digunakan dalam mengembangkan sistem ini adalah Microsoft Visual Studio 2012 dengan bahasa pemrograman C#.

3. Framework .NET
   Pembuatan GUI menggunakan WPF, dibutuhkan Framework .NET 4.5.
4. Microsoft Solver Foundation
   Microsoft Solver Foundation mereupakan salah satu *reference* dari Microsoft yang digunakan untuk melakukan perhitungan aplikasi matematika. Dalam penggunaaan Microsoft Solver Foundation membutuhkan Framework .NET minimum .NET 4.0.
5. Bytescout Spreadsheet
   Bytescout Spreadsheet mereupakan salah satu *reference* yang dapat digunakan untuk membaca file Excel.

5.2 Penjelasan Implementasi

Pada subbab ini dijelaskan implementasi setiap metode yang dijelaskan pada bab metode pemecahan masalah, sehingga terbentuk suatu perangkat lunak yang mengimplementasi metode-metode pada bab metode pemecahan masalah.

5.2.1 Implementasi Heuristic Miner

Pada bagian ini mengimplementasi *heuristic miner* agar dapat melakukan proses *discovery* terhadap *event log* agar mendaoatkan model proses busnis yang terjadi. Proses dalam implementasi ini terdapat dua jenis yaitu:

- Pertama proses pembacaan *file event log* yang sekaligus melakukan pengidentifikasin *trace* yang terkandung dalam *event log* dan frekuensinya. Hal ini dapat dilihat pada Kode Sumber 5.1 dan Kode Sumber 5.2.

```
public void HeuristicMiner(string name)
{
document.LoadFromFile(name);
Worksheet worksheetNode = document.Workbook.Worksheets[0];
int i = 1;
int j = 0;
while (Convert.ToString(worksheetNode.Cell(i, 0)) != "")
{
```

```
  Cell currentCellCaseID = worksheetNode.Cell(i, 0);
  Cell currentCellCaseID1 = worksheetNode.Cell(i + 1, 0);
  Cell currentCellCaseID2 = worksheetNode.Cell(i - 1, 0);
  Cell currentCellActivity = worksheetNode.Cell(i, 1);
  Cell currentCellTime1 = worksheetNode.Cell(i, 2);
  Cell currentCellTime2 = worksheetNode.Cell(i + 1, 2);
  Cell oroginator = worksheetNode.Cell(i, 3);
  Cell currentCost = worksheetNode.Cell(i, 4);
  Cell Time = worksheetNode.Cell(i, 5);
  double ab = Convert.ToDouble(currentCellTime1.Value);
  double ac = Convert.ToDouble(currentCellTime2.Value);
  DateTime time1 = DateTime.FromOADate(ab);
  DateTime time2 = DateTime.FromOADate(ac);
  if (currentCellCaseID1.Value == currentCellCaseID.Value)
  {
    timeGap = time2 - time1;
    Time.Value = timeGap;
  }
  if (currentCellCaseID1.Value == currentCellCaseID.Value &&
((timeGap >= TimeSpan.Zero) || currentCellTime2.Value ==
currentCellTime1.Value))
{if(!(durationCase.ContainsKey(currentCellCaseID.Value.ToString
()())))
   {
   durationCase.Add(currentCellCaseID1.Value.ToString(), new
   List<double>());
    }
   activity.Add(currentCellActivity.Value.ToString());
}
  if (currentCellCaseID2.Value == currentCellCaseID.Value)
  {
    if
(!activity.Contains(currentCellActivity.Value.ToString()))
    {
      activity.Add(currentCellActivity.Value.ToString());
    }}
  if (!listAct.Contains(currentCellActivity.Value.ToString()))
  {
    listAct.Add(currentCellActivity.Value.ToString());
    input.Add(currentCellActivity.Value.ToString(), new
    List<string>());
    output.Add(currentCellActivity.Value.ToString(), new
    List<string>());
  }
  if (currentCellCaseID1.Value != currentCellCaseID.Value)
  {
    if (j == 0)
    {
      int w = 0;
      int t = activity.Count();
```

```
        myTrace.Add(j, new List<string>());
        myTrace[j] = activity.GetRange(w, t);
        freq.Add(0);
         j++;
            }
           if (j > 0)
           {
          int count = 0;
          for (int m = 0; m < myTrace.Count(); m++)
          {
         bool equal = myTrace[m].SequenceEqual(activity);
         if (equal == true)
         {
            count++;
         }
         }
         if (count == 0)
         {
             int w = 0;
             int t = activity.Count();
             myTrace[j] = activity.GetRange(w, t);
             freq.Add(0);
             j++;
         }
         }
         for (int m = 0; m < myTrace.Count(); m++)
         {
         bool equal = myTrace[m].SequenceEqual(activity);
         if (equal == true)
            {
              freq[m]++;
         }
         }
         activity.Clear();
    }
   i++;
}
document.Close();
```

Kode Sumber 5.1 Fungsi `HeuristicMiner()` bagian Membaca Data dan Melihat *Trace*

```
for (int m = 0; m < myTrace.Count(); m++)
{
   Console.WriteLine("trace: " + (m + 1));
   for (int z = 0; z < myTrace[m].Count(); z++)
   {
     Console.WriteLine(myTrace[m][z]);
     if (z < myTrace[m].Count() - 1)
```

```
      {
      matriksFrekuensi[myTrace[m][z], myTrace[m][z + 1]] +=
freq[m];
   }
   if ((z < myTrace[m].Count() - 2) &&
myTrace[m][z].Equals(myTrace[m][z + 2]))
   {
      matriksFreTwoLoop[myTrace[m][z], myTrace[m][z + 1]] +=
freq[m];
   }
}
```

Kode Sumber 5.2 Fungsi `HeuristicMiner()` bagian Menghitung Frekuensi *Trace*

- Yang kedua adalah bagian perhitungan matriks yang digunakan untuk melakukan proses *discovery* sehingga menghasilkan model proses bisnis. Pengimplementasian dapat dilihat pada Kode Sumber 5.3 sampai dengan Kode Sumber 5.6.

```
for (int m = 0; m < myTrace.Count(); m++)
{
   for (int z = 0; z < myTrace[m].Count(); z++)
   {
      if (z < myTrace[m].Count() - 1)
      {
         matriksDependency[myTrace[m][z], myTrace[m][z + 1]] =
         (double)((matriksFrekuensi[myTrace[m][z], myTrace[m][z +
         1]]) - (matriksFrekuensi[myTrace[m][z + 1],
         myTrace[m][z]])) /
         (double)((matriksFrekuensi[myTrace[m][z], myTrace[m][z +
         1]]) + (matriksFrekuensi[myTrace[m][z + 1],
         myTrace[m][z]]) + 1);
      }
      if ((z < myTrace[m].Count() - 1) &&
      myTrace[m][z].Equals(myTrace[m][z + 1]))
      {
         matriksOneLoop[myTrace[m][z], myTrace[m][z + 1]] =
         (double)(matriksFrekuensi[myTrace[m][z], myTrace[m][z +
         1]]) /
         (double)((matriksFrekuensi[myTrace[m][z], myTrace[m][z +
         1]]) + 1);
      }
      if ((z < myTrace[m].Count() - 2) &&
      myTrace[m][z].Equals(myTrace[m][z + 2]))
      {
      matriksTwoLoop[myTrace[m][z], myTrace[m][z + 1]] =
      (double)((matriksFreTwoLoop[myTrace[m][z], myTrace[m][z +
      1]]) + (matriksFreTwoLoop[myTrace[m][z + 1],
```

```
    myTrace[m][z]])) /
    (double)((matriksFreTwoLoop[myTrace[m][z], myTrace[m][z +
    1]]) + (matriksFreTwoLoop[myTrace[m][z + 1], myTrace[m][z]])
    + 1);
    }}}
```

Kode Sumber 5.3 Fungsi `HeuristicMiner()` bagian Perhitungan Matriks *Dependency* dan *ShortLoop*

```
RTB = matriksDependency.avg() -
(matriksDependency.standradDeviation() / 2);
DT = matriksDependency.avg() -
matriksDependency.standradDeviation();
for (int m = 0; m < myTrace.Count(); m++)
{
  for (int z = 0; z < myTrace[m].Count(); z++)
  {
    if (z < myTrace[m].Count() - 1 && (matriksDependency.maxi()
    - matriksDependency[myTrace[m][z], myTrace[m][z + 1]]) <=
    RTB && (matriksDependency[myTrace[m][z], myTrace[m][z + 1]]
    >= DT)
    || (z < myTrace[m].Count() - 1 &&
    (Math.Round((matriksDependency.maxi() -
    matriksDependency[myTrace[m][z], myTrace[m][z + 1]]), 2) <=
    Math.Round(RTB, 2) &&
    Math.Round(matriksDependency[myTrace[m][z], myTrace[m][z +
    1]], 2) >= Math.Round(DT, 2))))
    {
      if (!(output[myTrace[m][z]].Contains(myTrace[m][z +
      1])))
      {
      output[myTrace[m][z]].Add(myTrace[m][z + 1]);
      }
      if (!(input[myTrace[m][z + 1]].Contains(myTrace[m][z])))
      {
      input[myTrace[m][z + 1]].Add(myTrace[m][z]);
      }
    }
     if (z < myTrace[m].Count() - 1 &&
     matriksFrekuensi[myTrace[m][z], myTrace[m][z + 1]] > 0 &&
     matriksDependency[myTrace[m][z], myTrace[m][z + 1]] >= 0)
     {
       if (minPDM > matriksDependency[myTrace[m][z],
       myTrace[m][z + 1]])
       {
       minPDM = matriksDependency[myTrace[m][z], myTrace[m][z +
       1]];
       }
```

```
    }}}
```

Kode Sumber 5.4 Fungsi `HeuristicMiner()` bagian Penentuan *Causal* Matriks dan *Dependency Graph*

```
for (int m = 0; m < myTrace.Count(); m++)
{
  for (int z = 0; z < myTrace[m].Count(); z++)
  {
    if ((z < myTrace[m].Count() - 1) &&
    output[myTrace[m][z]].Count() > 1)
    {
      if (!output[myTrace[m][z]].Except(myTrace[m]).Any())
      {
        matriksParallel[myTrace[m][z], myTrace[m][z + 1]] +=
      freq[m];
      }
      if (!(!output[myTrace[m][z]].Except(myTrace[m]).Any()))
      {
        nonParallelSplit[myTrace[m][z]]++;
      }
    }
    if ((z < myTrace[m].Count() - 1) && (input[myTrace[m][z +
    1]].Count() > 1))
    {
       if (!input[myTrace[m][z + 1]].Except(myTrace[m]).Any())
       {
         matriksParallel[myTrace[m][z + 1], myTrace[m][z]] +=
       freq[m];
       }
       if (!(!input[myTrace[m][z +
       1]].Except(myTrace[m]).Any()))
       {
         nonParallelJoin[myTrace[m][z + 1]]++;
       }
    }
  }
}
foreach (string key in input.Keys)
{
    if (input[key].Count() == 0)
    {
      start = key;
    }
    if (output[key].Count() == 0)
    {
      end = key;
    }
    for (int m = 0; m < input[key].Count; m++)
```

```
    {
      if (m < (input[key].Count) - 1 && matriksParallel[key,
      input[key][m]] >= 0 && matriksParallel[key, input[key][m +
      1]] >= 0 && input[key].Count > 1)
      {
      matriksParallel[input[key][m], input[key][m + 1]] =
      matriksParallel[key, input[key][m]];
      matriksParallel[input[key][m + 1], input[key][m]] =
      matriksParallel[key, input[key][m + 1]];
      PMinput[key].Add((double)(matriksParallel[input[key][m],
      input[key][m + 1]] + matriksParallel[input[key][m + 1],
      input[key][m]]) /
       (double)(nonParallelJoin[key] +
      matriksFrekuensi[input[key][m], key] +
      matriksFrekuensi[input[key][m + 1], key] + 1));
      matriksParallel[key, input[key][m]] = 0;
      matriksParallel[key, input[key][m + 1]] = 0;
    }
  }
}
foreach (string key in output.Keys)
{
  for (int m = 0; m < output[key].Count; m++)
  {
    if (m < (output[key].Count) - 1 && matriksParallel[key,
    output[key][m]] >= 0 && matriksParallel[key, output[key][m +
    1]] >= 0 && output[key].Count > 1)
    {
      matriksParallel[output[key][m], output[key][m + 1]] =
      matriksParallel[key, output[key][m]];
      matriksParallel[output[key][m + 1], output[key][m]] =
      matriksParallel[key, output[key][m + 1]];
      PMoutput[key].Add((double)(matriksParallel[output[key][m],
      output[key][m + 1]] + matriksParallel[output[key][m + 1],
      output[key][m]]) /
      (double)(nonParallelSplit[key] + matriksFrekuensi[key,
      output[key][m]] + matriksFrekuensi[key, output[key][m +
      1]] + 1));
      matriksParallel[key, output[key][m]] = 0;
      matriksParallel[key, output[key][m + 1]] = 0;
    }
  }
}
```

Kode Sumber 5.5 Fungsi `HeuristicMiner()` bagian Perhitungan Matriks Frekuensi *Parallel* dan *Parellel Measure*

```
foreach (string key in PMinput.Keys)
```

```
{
  if (PMinput[key].Count > 0)
  {
    avgPMJoin[key] = (double)PMinput[key].Sum() /
    (double)PMinput[key].Count();
  }
}
foreach (string key in PMoutput.Keys)
{
  if (PMoutput[key].Count > 0)
  {
    avgPMSplit[key] = (double)PMoutput[key].Sum() /
    (double)PMoutput[key].Count();
  }
}
foreach (string key in avgPMSplit.Keys)
{
  if (avgPMSplit[key] <= minPDM)
  {
     split.Add(key, "XOR Split");
  }
  if (avgPMSplit[key] > minPDM && avgPMSplit[key] <=
  matriksDependency.avg())
  {
     split.Add(key, "OR Split");
  }
  if (avgPMSplit[key] > matriksDependency.avg())
  {
     split.Add(key, "AND Split");
  }
}
foreach (string key in avgPMJoin.Keys)
{
  if (avgPMJoin[key] <= minPDM)
  {
    join.Add(key, "XOR Join");
  }
  if (avgPMJoin[key] > minPDM && avgPMJoin[key] <=
  matriksDependency.avg())
  {
    join.Add(key, "OR Join");
  }
  if (avgPMJoin[key] > matriksDependency.avg())
  {
    join.Add(key, "AND Join");
  }
}}
```

Kode Sumber 5.6 Fungsi `HeuristicMiner()` bagian Penentuan *Split* dan *Join* untuk Paralel

5.2.2 Implementasi Perhitungan Data untuk Optimasi

Pada bagian ini mengimplementasi perhitungan dari optimasi data. Pengimplementasian ini bertujuan untuk mendapatkan nilai dari durasi normal, biaya normal, durasi *crash*, biaya *crash*, dan juga *cost slope* dan *time slope* yang akan digunakan dalam pembentukan model matematika *linear programming*. Pengimplementasiannya dapat dilihat pada Kode Sumber 5.7.

```
public void GetDuration(string name)
{
HeuristicMiner(name);
List<string> temp = new List<string>();
Worksheet worksheetNode = document.Workbook.Worksheets[0];
int o = 1;
while (Convert.ToString(worksheetNode.Cell(o, 0)) != "")
{
  Cell currentCellCaseID = worksheetNode.Cell(o, 0);
  Cell currentCellCaseID1 = worksheetNode.Cell(o + 1, 0);
  Cell currentCellCaseID2 = worksheetNode.Cell(o - 1, 0);
  Cell currentCellActivity = worksheetNode.Cell(o, 1);
  Cell currentCellActivity1 = worksheetNode.Cell(o + 1, 1);
  Cell Time = worksheetNode.Cell(o, 7);
  Cell currentCost = worksheetNode.Cell(o, 4);
  Cell currentCellTime1 = worksheetNode.Cell(o, 2);
  if (currentCellCaseID1.Value == currentCellCaseID.Value)
  {
    int m = o;
    double minim = 10000000;
    Cell currentCellTime2 = worksheetNode.Cell(m + 1, 2);
    double ab = Convert.ToDouble(currentCellTime1.Value);
    double ac = Convert.ToDouble(currentCellTime2.Value);
    double lalala = 0;
    DateTime time1 = DateTime.FromOADate(ab);
    DateTime time2 = DateTime.FromOADate(ac);
    Cell currentCellCaseID3 = worksheetNode.Cell(m + 1, 0);
    Cell currentCellActivity3 = worksheetNode.Cell(m + 1, 1);
    string hihihi = currentCellCaseID.Value.ToString();
    string hahaha = currentCellActivity.Value.ToString();
    while (hihihi == currentCellCaseID3.Value.ToString())
    {
        if
        (output[hahaha].Contains(currentCellActivity3.Value.ToString
        ()) && output[hahaha].Count() > 1)
        {
            timeGap = time2 - time1;
```

```
            lalala = (ac - ab) / (double)0.0416666666666667;
            if (lalala < minim)
            {
              duration[currentCellActivity.Value.ToString()].Add(lalala
              );
              durationCase[currentCellCaseID.Value.ToString()].Add(lala
              la);
              Time.Value = lalala;
              minim = lalala;
            }
            temp.Add(hahaha);
          }
          if
          (output[hahaha].Contains(currentCellActivity3.Value.ToString
          ()) && output[hahaha].Count() < 2)
          {
              timeGap = time2 - time1;
              lalala = (ac - ab) / (double)0.0416666666666667;
              Time.Value = lalala;

              duration[currentCellActivity.Value.ToString()].Add(lalal
              a);

              durationCase[currentCellCaseID.Value.ToString()].Add(lal
              ala);
              temp.Add(hahaha);
          }
          m++;
          currentCellCaseID3 = worksheetNode.Cell(m + 1, 0);
          currentCellActivity3 = worksheetNode.Cell(m + 1, 1);
          currentCellTime2 = worksheetNode.Cell(m + 1, 2);
          ac = Convert.ToDouble(currentCellTime2.Value);
          time2 = DateTime.FromOADate(ac);
          if (currentCellCaseID3.Value == null)
          {
              break;
          }
      }
      cost[currentCellActivity.Value.ToString()].Add(Convert.ToDouble(c
      urrentCost.Value));
      }
      if (currentCellCaseID2.Value == currentCellCaseID.Value)
      {
         if (!(temp.Contains(currentCellActivity.Value.ToString())))
         {

           cost[currentCellActivity.Value.ToString()].Add(Convert.ToD
           ouble(currentCost.Value));
```

```
            duration[currentCellActivity.Value.ToString()].Add(durat
            ionCase[currentCellCaseID.Value.ToString()].Average());
            Time.Value =
            duration[currentCellActivity.Value.ToString()];
            temp.Add(currentCellActivity.Value.ToString());
           }
         }
     if (currentCellCaseID1.Value != currentCellCaseID.Value)
     {
        temp.Clear();
     }
     o++;
}

document.Close();
foreach (string key in duration.Keys)
{
   normalDuration.Add(key, Math.Round((duration[key].Sum() /
   duration[key].Count()), 2));
   double sumOfSquaresOfDiffDur =
   Math.Round(duration[key].Select(val => (val -
   duration[key].Average()) * (val -
   duration[key].Average())).Sum(), 2);
   double timeSlope = Math.Sqrt(sumOfSquaresOfDiffDur /
   (duration[key].Count() - 1));
   durationSlope.Add(key, Math.Round(timeSlope, 2));
}
foreach (string key in normalDuration.Keys)
{
   crashDuration.Add(key, Math.Round((normalDuration[key] -
   durationSlope[key]), 2));
}

foreach (string key in cost.Keys)
{
   normalCost.Add(key, Math.Round(cost[key].Average(), 2));
   double sumOfSquaresOfDiffCost = Math.Round(cost[key].Select(val
   => (val - cost[key].Average()) * (val -
   cost[key].Average())).Sum(), 2);
   double slopeCost = Math.Sqrt(sumOfSquaresOfDiffCost /
   (cost[key].Count() - 1));
   sdCost.Add(key, Math.Round(slopeCost, 2));
}

foreach (string key in normalCost.Keys)
{
   Crashcost.Add(key, Math.Round((normalCost[key] + sdCost[key]),
   2));
```

```
    costSlope.Add(key, Math.Round((Crashcost[key] - normalCost[key])
    / (durationSlope[key]), 2));
}
        }
```

Kode Sumber 5.7 Fungsi `GetDuration()` untuk Menghitung Data Optimasi

5.2.3 Implementasi Pemecahan Optimasi Durasi dan Biaya Tambahan Minimum

Pada bagian ini mengimplementasi cara untuk pengoptimasian durasi dan biaya tambahan percepatan proses bisnis. Pengimplementasian ini bertujuan untuk mendapatkan nilai durasi proyek paling minimal dan biaya tambahan yang minimal pula untuk peminimalan durasi tersebut. Dalam pengimplementasiannya terdapat dua fungsi yang digunakan yaitu:

- Yang pertama adalah fungsi `GenerateMathModel()` dalam fungsi ini yang pertama dilakukan adalah memanggil fungsi `GetDuration()` untuk mendapatkan data optimasi dan kemudian menginisialisasi kelas CPM dan memanggil fungsi-fungsi yang ada di dalalamnya yaitu `CPMethod()` untuk mengetahui *critical path* dan fungsi `GetDistance()` untuk mendapatkan durasi proses bisnis. Fungsi `GenerateMathModel()` implemetasinya dapat dilihat pada Kode Sumber 5.8 dan pengimplementasian kelas CPM dapat dilihat pada Kode Sumber 5.9 sampai Kode Sumber 5.11.

```
public void GenerateMathModel(string m)
{
GetDuration(m);
CPM cpm1 = new CPM();
foreach (string act in listAct)
{
  if (!cpm1.GetDistance().ContainsKey(act))
  {
    cpm1.GetDistance().Add(act, new KeyValuePair<List<string>,
    double>(new List<String> { "" }, -1.0));
  }
}
cpm1.CPMethod(start, end, output, normalDuration);
List<string> criticalPath = cpm1.getCriticalAct();
```

```
obj = "(" + Math.Round(costSlope[start], 2) + "*X" +
start.Replace(" ", string.Empty);
foreach (string act in listAct)
{
  if (criticalPath.Contains(act))
  {
    obj = obj + " + " + Math.Round(costSlope[act], 2) + "*X" +
    act.Replace(" ", string.Empty);
  }
  if (!criticalPath.Contains(act) && act != start)
  {
    obj = obj + " + " + Math.Round(costSlope[act], 2) + "*X" +
    act.Replace(" ", string.Empty);
  }
  maxRed = maxRed + "X" + act.Replace(" ", string.Empty) + "<="
  + Math.Round(durationSlope[act], 2) + "; ";
  nonNegX = nonNegX + "X" + act.Replace(" ", string.Empty) +
  ">=" + 0 + "; ";
  nonNegY = nonNegY + "Y" + act.Replace(" ", string.Empty) +
  ">=" + 0 + "; ";
  if (act != end)
  {
    durCons = durCons + "X" + act.Replace(" ", string.Empty) +
    "+";
  }
  if (input[act].Count != 0 && input[act].Count < 2)
  {
    startCons = startCons + "Y" + act.Replace(" ",
    string.Empty) + " - " + "Y" + input[act][0].Replace(" ",
    string.Empty) + " + X" + input[act][0].Replace(" ",
    string.Empty) + " >= " +
    Math.Round(normalDuration[input[act][0]], 2) + "; ";
  }
  if (input[act].Count != 0 && input[act].Count > 1)
  {
    for (int g = 0; g < input[act].Count(); g++)
    {
      startCons = startCons + "Y" + act.Replace(" ",
      string.Empty) + " - " + "Y" + input[act][g].Replace(" ",
      string.Empty) + " + X" + input[act][g].Replace(" ",
      string.Empty) + " >= " +
      Math.Round(normalDuration[input[act][g]], 2) + "; ";
    }
  }
}
obj = obj + "); ";
startCons = startCons + "YFINISH - Y" + end.Replace(" ",
string.Empty) + " + X" + end.Replace(" ", string.Empty) + " >= "
+ Math.Round(normalDuration[end], 2) + "; ";
```

```
durCons = "YFINISH - " +
Math.Round(cpm1.GetDistance()[end].Value, 2).ToString() + " <= "
+ 0;
makespan=Math.Round(cpm1.GetDistance()[end].Value,
2).ToString();
}
```

Kode Sumber 5.8 Fungsi `GenerateMathModel()` untuk Menghitung Data Optimasi

```
public void CPMethod(string start, string finish,
Dictionary<string, List<string>> output, Dictionary<string,
double> normalDuration)
{
start1 = start;
end1 = finish;
distances[start] = new KeyValuePair<List<string>, double>(new
List<String> { "" }, normalDuration[start]);
List<String> successors = new List<string>();
successors.Add(start);
while (successors.Count() > 0)
{
  current = successors[0];
  if (current == finish)
  {
    successors.RemoveAt(0);
    if (successors.Count() > 0)
        current = successors[0];
    else
        break;
  }

  successors.RemoveAt(0);
  for (int g = 0; g < output[current].Count(); g++)
  {
   if (distances[current].Value +
   normalDuration[output[current][g]] >
   distances[output[current][g]].Value)
   {
     distances[output[current][g]] = new
     KeyValuePair<List<string>, Double>(new List<string> {
     current }, distances[current].Value +
     normalDuration[output[current][g]]);
     if (!successors.Contains(output[current][g]))
       successors.Add(output[current][g]);
   }

   else if (distances[current].Value +
   normalDuration[output[current][g]] ==
   distances[output[current][g]].Value)
```

```
    {
        if (!distances[output[current][g]].Key.Contains(current))
            distances[output[current][g]].Key.Add(current);
        if (!successors.Contains(output[current][g]))
            successors.Add(output[current][g]);
    }
  }
}
}
```

Kode Sumber 5.9 Fungsi `CPMethod()` pada Kelas CPM untuk Mendapatkan *critical path*

```
public void GetPath(string current, String path)
{
String pathTemp = start1;
while (current != start1)
{
  if (distances[current].Key.Count() > 1)
  {
    for (int i = 1; i < distances[current].Key.Count(); i++)
    {
      GetPath(distances[current].Key[i], path + current);
      path += current + "->";
      current = distances[current].Key[0];
    }
  }
}
paths.Add(path);
}
```

Kode Sumber 5.10 Fungsi `GetPath()` pada Kelas CPM untuk Mendapatkan *critical path*

```
public List<string> getCriticalAct()
{
string[] result = null;
GetPath(end1, "");
foreach (var path in paths)
{
  result = path.Split(new string[] { "->" },
  StringSplitOptions.None);
  foreach (string a in result)
  {
    if (!(criticalAct.Contains(a)))
    {
      criticalAct.Add(a);
    }
  }
```

```
}


return criticalAct;
}
```

Kode Sumber 5.11 Fungsi `getCtiticalAct()` pada Kelas CPM untuk Mendapatkan Aktivitas yang masuk *critical path*

- Yang kedua merupakan fungsi yang digunakan untuk menyelesaikan model matematika dari *linear programming*. Penyelesaian ini digunakan untuk mendapatkan durasi proses bisnis yang paling minimum dengan biaya yang digunakan untuk meminimumkan durasi juga minimum. Penyelesaian ini dilakukan dengan menggunakan metode *simplex* yang ada pada Microsoft Solver Foundation. Pengimplementasiannya dapat dilihat Kode Sumber 5.12.

```
public string SolveMathModel()
{
string [] constrain1, constrain2;
string goal, constrain3;
Dictionary<string,Decision> dec= new
Dictionary<string,Decision>();
SolverContext context = SolverContext.GetContext();
context.ClearModel();
Model model = context.CreateModel();
foreach (string act in listAct)
{
  dec.Add("X" + act.Replace(" ", string.Empty), new
  Decision(Domain.RealNonnegative, "X" + act.Replace(" ",
  string.Empty)));
  dec.Add("Y" + act.Replace(" ", string.Empty), new
  Decision(Domain.RealNonnegative, "Y" + act.Replace(" ",
  string.Empty)));
}
Decision YFINISH = new Decision(Domain.RealNonnegative,
"YFINISH");
foreach (string key in dec.Keys)
{
  model.AddDecision(dec[key]);
}
model.AddDecision(YFINISH);
goal = GetMathModel()[0].Substring(0, GetMathModel()[0].Length -
2);
```

```
constrain1 = GetMathModel()[1].Substring(0,
GetMathModel()[1].Length - 2).Split(new string[] { "; " },
StringSplitOptions.None);
constrain2 = GetMathModel()[2].Substring(0,
GetMathModel()[2].Length - 2).Split(new string[] { "; " },
StringSplitOptions.None);
constrain3 = GetMathModel()[3];
int con1 = constrain1.Length - 1;
foreach (string a in constrain1)
{
  if (con1 >= 0)
  {
    model.AddConstraint("MaxRed" + con1, a);
  }
  con1--;
}
int con2 = constrain2.Length - 1;
foreach (string a in constrain2)
{
  if (con2 >= 0)
  {
    model.AddConstraint("StartConstraint" + con2, a);
  }
  con2--;
}
model.AddConstraint("DurationConstraint", constrain3);
model.AddGoal("AdditionalCost", GoalKind.Maximize, goal);
Solution solution = context.Solve(new SimplexDirective());
Report report = solution.GetReport();
string minimumMakespan1 = report.ToString().After("YFINISH:");
double minimumMakespan2 = Convert.ToDouble(minimumMakespan1);
foreach (Goal a in model.Goals)
{
  model.RemoveGoal(a);
  break;
}
foreach (Constraint a in model.Constraints)
{
  if (a.Expression.ToString().Equals(constrain3))
  {
    model.RemoveConstraint(a);
    break;
  }
}
model.AddGoal("AdditionalCost", GoalKind.Minimize, goal);
model.AddConstraints("DurationConstraint", YFINISH -
minimumMakespan2 <= 0);
Solution solution2 = context.Solve(new SimplexDirective());
Report report2 = solution2.GetReport();
string solutionMath = report2.ToString();
```

```
return solutionMath;
}
```

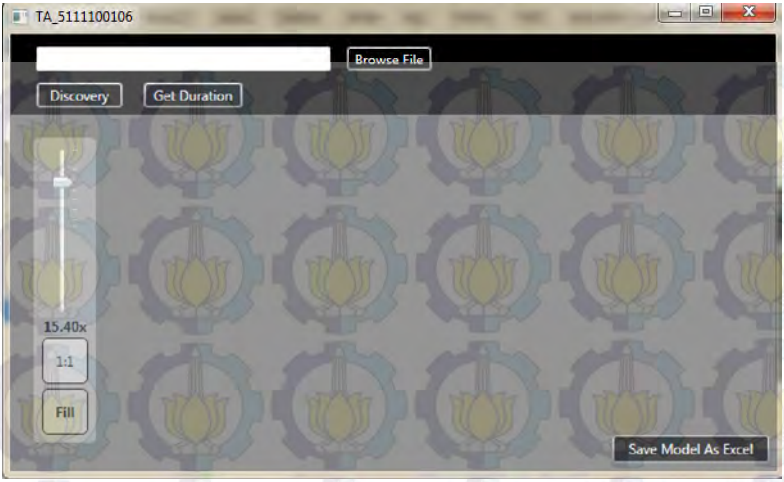
Kode Sumber 5.12 Fungsi `SolveMathModel()` Digunakan unruk Menyelesaikan Optimasi

5.3 Implementasi Antar Muka

Implementasi tampilan antarmuka pengguna pada Visual Studio 2012 dengan menggunakan jenis tampilan WPF. Berikut ini akan dijelaskan mengenai implementasi tampilan antarmuka pengguna yang terdapat pada perangkat lunak.

5.3.1 Halaman Proses *Discovery*

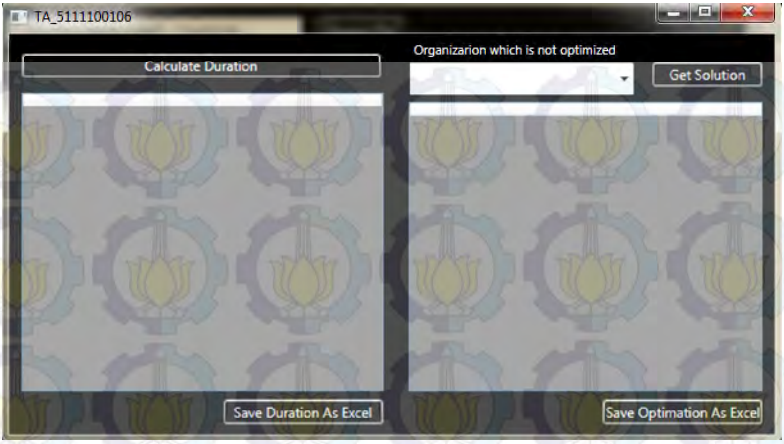
Halaman ini merupakan halam pertama yang muncul ketika aplikasi dijalankan. Pada halaman ini terdapat beberapa parameter yang harus diisi dan dibutuhkan untuk pemrosesan selanjutnya. Parameter tersebut antara lain *path* direktori tempat *event log* berada. Setelah mendapat *path* sistem akan membuka *file event log* yang telah dipilih. Setelah *file* terbuka maka sistem akan melakukan proses *discovery* atau bisa langsung melakukan optimasi tergantung kebutuhan dari pengguna. Selain itu pada halaman ini juga terdapat fungsi *Get Duration* yang digunakan untuk menuju pada halaman optimasi. Hasil implementasi pada gambar ini dapat dilihat pada Gambar 5.1. Sementara untuk implementasi kode sumber pada halaman ini dapat dilihat pada lampiran A.



Gambar 5.1 Hasil Implementasi Halaman *Process Discovery*

5.3.2 Halaman Proses Optimasi

Halaman ini akan muncul ketika pengguna menekan tombol *Get Duration* pada halaman proses *discovery*, hal ini dilakukan ketika pengguna ingin melakukan optimasi tambahan biaya dan *makespan* percepatan yang dilakukan pada proses bisnis. Halaman ini akan menampilkan tombol-tombol yang digunakan untuk melakukan optimasi. Hasil implementasi pada gambar ini dapat dilihat pada Gambar 5.2. Sementara untuk implementasi kode sumber pada halaman ini dapat dilihat pada lampiran A.



Gambar 5.2 Hasil Implementasi Halaman Optimasi

[Halaman ini sengaja dikosongkan]

BAB VI
PENGUJIAN DAN EVALUASI

Bab ini membahas hasil dan pembahasan pada aplikasi yang dikembangkan. Pada bab ini akan dijelaskan tentang data yang digunakan, hasil yang didapatkan dari penggunaan perangkat lunak dan uji coba yang dilakukan pada perangkat lunak yang telah dikerjakan untuk menguji apakah fungsionalitas aplikasi telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya.

6.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi pembuatan sistem pada tugas akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
   a. Prosesor: Intel® Core™ i3 CPU @ 2.10GHz
   b. Memori(RAM): 4 GB
   c. Tipe sistem: 32-bit sistem operasi
2. Perangkat lunak
   a. Sistem operasi: Windows 7 Professional
   b. Perangkat pengembang: Microsoft Visual Studio 2012

6.2 Tahapan Uji Coba

Dalam uji coba yang dilakukan dalam tugas akhir ini memiliki beberapa tahapan yang dijelaskan pada subbab ini.

6.2.1 Memasukkan Data *Event Log*

Yang pertama kali dilakukan untuk tahapan uji coba adalah memasukkan *event log* pada program. *Event log* yang digunakan harus mengandung informasi sebagai berikut:

- *Case* Id

  Case merupakan suatu kasus tertentu yang ada pada *event log*. Kasus tertentu tersebut dapat berupa suatu kasus

dalam memproduksi suatu barang tertentu, karena *event log* dapat terdiri dari catatan dari proses eksekusi pembuatan banyak barang atau proses eksekusi dari banyak kasus proses.

- Aktivitas yang dijalankan pada proses bisnis.
- Waktu dijalankannya aktiviast dalam suatu *case* dalam proses bisnis (*timestamp*)
- Originator pelaksana aktivitas dalam setiap *case* pada proses bisnis.
- Biaya setiap aktitivas pada setiap *case* dalam proses bisnis.

Data masukkan yang digunakan dalam program ini memiliki ekstensi Excel. Contoh bentuk data masukkan yang digunakan terdapat pada. Dalam perangkat lunak yang dikembangkan bagian ini dilakukan dengan menekan timbol *Browse.*

Tabel 6.1 Contoh Format Data Masukkan

Case ID	Activity	Time Stamp	Organisator	Cost
PP1	Sending good receive	6/20/2014 8:32	Purchase Department	243303
PP1	Getting good receive	6/20/2014 13:42	Spinning Department	138941
PP1	Bale opening	6/20/2014 23:41	Spinning Department	508599
PP1	Conditioning of MMP Fiber	6/21/2014 8:16	Spinning Department	63706
PP1	Blending	6/21/2014 10:46	Spinning Department	496301
PP1	Opposing spike	6/21/2014 16:57	Blowing Department	352814
PP1	Striking cotton	6/21/2014 18:09	Blowing Department	125149
PP1	Carding	6/21/2014 19:15	Spinning Department	957335
PP1	Roving frame	6/22/2014 10:51	Framing Department	516512
PP1	Drawing frame	6/22/2014 16:28	Framing Department	455652
PP1	Combing	6/22/2014 23:42	Spinning Department	394780
PP1	Ring framing	6/23/2014 4:48	Spinning Department	359985
PP1	Cone winding	6/23/2014 16:44	Spinning Department	319064
PP2	Sending good receive	6/23/2014 23:26	Purchase Department	212893
PP2	Getting good receive	6/24/2014 4:19	Spinning Department	157720
PP2	Bale opening	6/24/2014 7:48	Spinning Department	405832
PP2	Conditioning of MMP Fiber	6/25/2014 1:08	Spinning Department	137116
PP2	Blending	6/25/2014 3:02	Spinning Department	450040
PP2	Air current blowing	6/25/2014 5:11	Blowing Department	269791
PP2	Striking cotton	6/25/2014 8:25	Blowing Department	186637
PP2	Carding	6/25/2014 12:45	Spinning Department	1079023
PP2	Drawing frame	6/26/2014 0:12	Framing Department	460565
PP2	Combing	6/26/2014 4:48	Spinning Department	417705
PP2	Ring framing	6/26/2014 15:16	Spinning Department	360005
PP2	Cone winding	6/26/2014 22:49	Spinning Department	313302
PP3	Sending good receive	6/27/2014 5:19	Purchase Department	180588
PP3	Getting good receive	6/27/2014 11:40	Spinning Department	159531
PP3	Bale opening	6/27/2014 20:00	Spinning Department	413071

6.2.2 Melakukan Proses *Discovery*

Setelah memasukkan *event log*, proses selanjutnya yang dilakukan pada *event log* adalah melakukan proses *discovery*. Proses *discovery* dilakukan dengan algoritma modifikasi *heuristic miner* yang telah dijelaskan pada subbab 3.2. Proses *discovery* dimaksudkan untuk mengetahui hubungan antar aktivitas. Hubungan antar aktivitas digunakan untuk proses selanjutnya yaitu untuk melakuan perhitungan durasi yang digunakan untuk proses optimasi. Para perangkat lunak yang dikembangkan proses ini dilakukan dengan menekan tombol *Discovery.*

6.2.3 Melakukan Perhitungan Data Optimasi

Perhitungan data optimasi ditujukan untuk mendapatkan data optimasi berupa durasi normal, durasi *crash*, biaya normal, biaya *crash*, dan *cost slope.* Hubungan antar aktivitas digunakan untuk mencari durasi per *case* setiap aktivitas yang ada, oleh karena itu sebelum melakukan proses ini diperlukan proses *discovery* untuk mengetahui hubungan antar aktivitas. Perhitungan data untuk optimasi telah dijelaskan pada subbab 3.5. Pada perangkat lunak yang dikembangkan proses ini dilakukan dengan yang pertama menekan tombol *Get Duration* pada halaman awal, dan kemudian menekan tombol *Calculate Duration.*

6.2.4 Melakukan Optimasi dengan Linear Programming

Setelah mendapatkan data optimasi yang kemudian dilakukan adalah memetakan data optimasi menjadi model matematika. Model matematika pertama menggunakan fungsi objektif *maximize* hal ini dikarenakan fungsi *maximize* pada biaya tambahan menghasilkan durasi proses bisnis yang paling minimal. Untuk model matematika yang kedua menggunakan fungsi objektif *minimize* karena fungsi *minimize* digunakan untuk menghasilkan biaya tambahan yang paling minimum dengan durasi batasan menggnakan durasi yang dihasilkan pada fungsi objektif model matematika yang pertama. Pada perangkat lunak yang dikembangkan proses ini dilakukan dengan yang pertama menekan tombol *Get Duration* pada halaman awal, dan kemudian menekan

tombol *Get Solution.* Pada proses optimasi ini pengguna juga dapat memilih organisasi yang tidak dioptimasi pada *dropdown* dengan label *Organization which is not optimized.*

6.3 Data Studi Kasus

Data uji yang digunakan dalam tugas akhir terdapat tiga jenis data uji yaitu untuk data *cross-organizational* didapatkan dari data *event log purchace order* bahan pembuatan benang PT Toray Industries Indonesia dan data *event log* produksi benang dari PT Toray Industries Indonesia, sedangkan untuk data *single-organization* yang digunakan untuk pembuktian *noise* diambil dari model yang dibuat menggunakan YAWL yang kemudian dibangkitkan *event log*-nya yang kemudian dirubah-rubah sehingga mengandung *noise.*

6.3.1 Data *Event Log Purchace Order* Bahan Pembuatan Benang PT Toray Industries Indonesia

Data *event log purchase order* bahan pembuatan benang dari PT Toray Industries Indonesia merupakan jenis *event* log yang melibatkan lebih dari satu departemen atau lebih dari satu organisasi. Dalam data ini melibatkan empat departmen yaitu *Purchase Order*, *Supplier*, dan Bea dan Cukai. Dalam data ini *Supplier* terdapat dua jenis yaitu *Supplier* kawasan berikat dan *Supplier* dari kawasan non berikat. Hal ini terjadi karena perbedaan alur proses bisnis antara *Supplier* kawasan berikat dan *Supplier* dari kawasan non berikat, kerana PT Toray Industries Indonesia merupakan perusahaan kawasan berikat maka ketika PT Toray Industries Indonesia memembeli bahan baku *Supplier* dari kawasan non berikat maka *Supplier* tersebut harus mengurus perijinan dan pembayaran pajak terhadap Bea dan Cukai, tetapi jika PT Toray Industries Indonesia memembeli bahan baku *Supplier* dari kawasan berikat maka dapat langsung diproses tanpa harus melakukan perijinan terhadap Bea dan Cukai. Adapun aktivitas dari masing-masing departemen adalah sebagai berikut:

1. Aktivitas pada departemen *Purchase*
   - *Sending PO Number*
   - *Receiving Good Receive*
2. Aktivitas pada *Supplier*
   - *Supplier* Berikat
     - *Producing Good Orders Sup1*
     - *Pakcaging Good Orders*
   - *Supplier* Non Berikat
     - *Sending Permitting Doc*
     - *Paying PPh*
     - *Producing Good Orders Sup2*
     - *Packaging Good Orders and Getting PPh Confirm*
   - Aktivitas yang dilakukan kedua jenis *Supplier*
     - *Sending Good Orders*
3. Aktivitas pada Bea dan Cukai
   - *Determining PPh and Giving Permission*

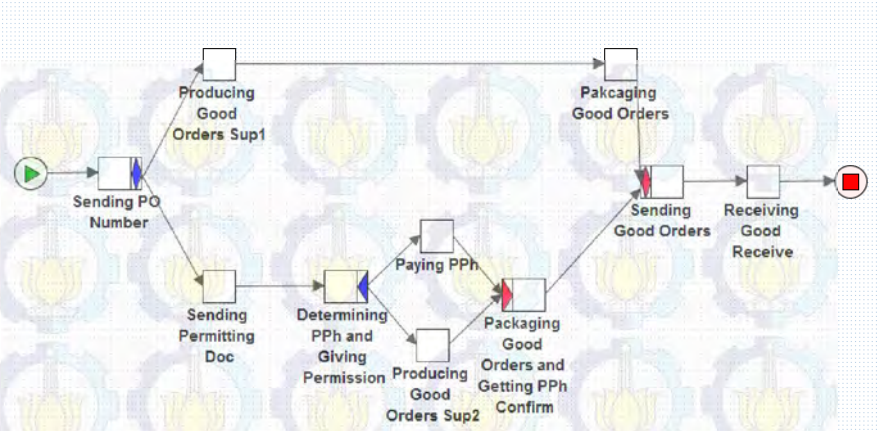
Bentuk *event log* dari data ini diperlihatkan pada Tabel 6.2.

Tabel 6.2 *Event Log* Data *Purchase Order*

Case Id	Activity	Time Stamp	Organisator	Cost
PO1	Sending PO Number	1/20/2014 8:32	Purchasing Department	14117
PO1	Sending Permitting Doc	1/20/2014 12:03	Supplier	34664
PO1	Producing Good Orders Sup1	1/20/2014 20:43	Supplier	738443
PO1	Pakcaging Good Orders	1/22/2014 21:57	Supplier	12063
PO1	Determining PPh and Giving Permission	1/23/2014 10:01	Beacukai	8291
PO1	Paying PPh	1/23/2014 14:10	Supplier	40600
PO1	Producing Good Orders Sup2	1/23/2014 15:11	Supplier	610468
PO1	Packaging Good Orders and Getting PPh Confirm	1/25/2014 7:53	Supplier	29462
PO1	Sending Good Orders	1/25/2014 8:51	Supplier	128884
PO1	Receiving Good Receive	1/26/2014 17:05	Purchasing Department	16950

Pada Tabel 6.2 menunjukan salah satu *case* dari *event log* data produksi. Sebenarnya dalam event log data produksi terkandung 100 *case* produksi, dengan 45 jenis *trace*. Pada kolom aktivitas terlihat tedapat perbedaan warna, perbedaan ini menunjukkan bahwa aktivitas itu dikerjakan oleh organisasi yang berbeda.

Model dari *event log* data prodroduksi dapat dilihat pada Gambar 6.1. Hubungan antar aktivitas dari model data purchase dapat dilihat pada Tabel 6.3.



Gambar 6.1 Model Data *Purchase Order*

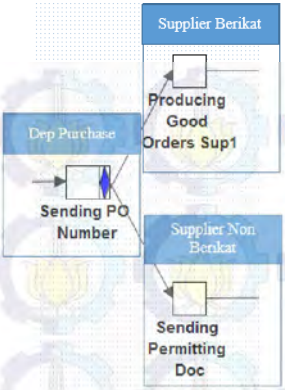
Tabel 6.3 Hubungan Antar Aktivitas dari Gambar 6.1

Input	Split / Join	Output
{Start}		Sending PO Number
Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,
Producing Good Orders Sup1		Pakcaging Good Orders
Sending Permitting Doc		Determining PPh and Giving Permission
Determining PPh and Giving Permission	AND Split	Paying PPh, Producing Good Orders Sup2,
Producing Good Orders Sup2, Paying PPh	AND Join	Packaging Good Orders and Getting PPh Confirm
Packaging Good Orders and Getting	OR Join	Sending Good Orders

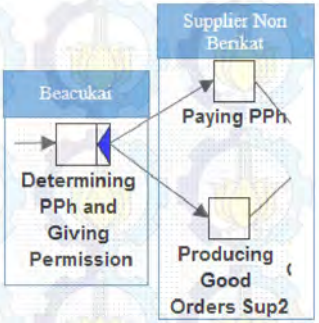
Input	Split / Join	Output
PPh Confirm, Pakcaging Good Orders		
Sending Good Orders		Receiving Good Receive
Receiving Good Receive		{end}

Jenis pola koordinasi *cross-organizational* yang ada pada data *event log* produksi adalah sebagau berikut:

1. Pola Koordinasi Pertukaran Pesan
   Pola ini terjadi antara departemen *Purchase Supplier* dan *Supplier* Bea dan Cukai. *Purchase Supplier* pada saat aktivitas *sending po number* dan *getting good receive* saat itu terjadi pertukaran pesan berupa pemesanan barang oleh *Purchase* terhadap *Supplier* jika *Supplier* Berikat maka *Supplier* akan langsung memproduksi pesanan tapi jika *Supplier* Non Berikat maka *Supplier* akan mengurus perijinan pada Bea dan Cukai, mengurus perijinan ini juga masuk dalam alur pertukaran pesan antara *Supplier* dan Bea dan Cukai. Alur pertukaran pesan *Purchase Supplier* dapat dilihat pada Gambar 6.2 dan alur pertukaran pesan *Supplier* Bea dan Cukai dapat dilihat pada Gambar 6.3.

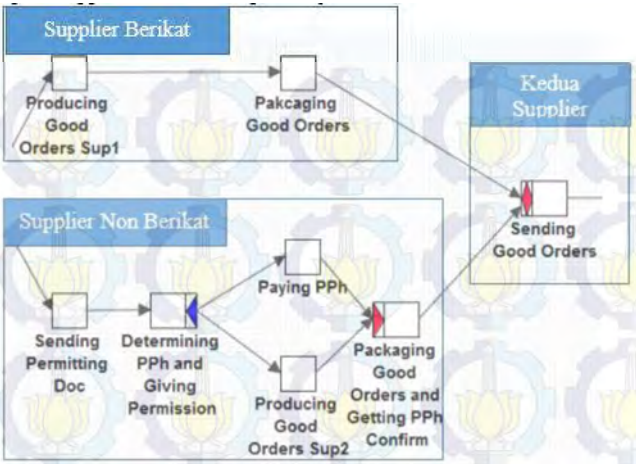


Gambar 6.2 Alur Pertukaran Pesan *Purchase Supplier* Data *Purchase*



Gambar 6.3 Alur Pertukaran Pesan *Supplier* Bea dan Cukai Data *Purchase*

2. Pola Koordinasi Prosedur Abstrak
   Pola ini terjadi antara departemen antara *Supplier* Berikat dan Non Berikat, hal ini dikarenakan sebenarnya keduanya melakukan proses yang sama yaitu memproduksi dan mengirimkan barang ke PT Toray Industries Indonesia tapi pada *Supplier* Non Berikat prosesnya lebih rinci karena belum memiliki perijinan dari Bea dan Cukai. Alur prosedur abstrak dapat dilihat pada Gambar 6.4.



Gambar 6.4 Alur Prosedur Abstrak *Supplier* Berikat dan Von Berikat Data *Purchase*

6.3.1 Data *Event Log* Produksi Benang dari PT Toray Industries Indonesia

Data *event log* produksi benang dari PT Toray Industries Indonesia merupakan jenis *event* log yang melibatkan lebih dari satu departemen atau lebih dari satu organisasi. Dalam data ini melibatkan empat departmen yaitu *Purchase*, *Spinning*, *Blowing*, dan *Framming*. Keempat departemen ini memiliki kewenangan masing-masing seperti untuk departemen *Purchase* memiliki kewenangan dalam mengirimkan bahan baku pada departemen *Spinning*, sedangkan *Spinning* memiliki kewenangan untuk pemintalan benang sehingga menjadi bentuk *cone* atau gulungan besar yang siap dipasarkan, dalam pemintalan benang sehingga sampai siap untuk dipasarkan. Departemen *Spinning* melakukan kerjasama dengan dua departemen lain yaitu *Blowing* dan *Framming* dalam memproduksi benang. Departemen *Blowing* berperan dalam pembersihan bahan benang yang akan diproses sedangkan *Framming* berperan dalam pemilihan kualitas serat yang baik. Adapun aktivitas dari masing-masing departemen adalah sebagai berikut:

1. Aktivitas pada departemen *Purchase*
   - *Sending good receive*
2. Aktivitas pada departemen *Spinning*
   - *Getting good receive*
   - *Bale Opening*
   - *Conditioning of MMP Fiber*
   - *Blending*
   - *Carding*
   - *Combing*
   - *Ring framing*
   - *Cone winding*
3. Aktivitas pada departemen *Blowing*
   - *Opposing spike*
   - *Air current blowing*
   - *Striking cotton*
4. Aktivitas pada departemen *Framming*
   - *Drawing frame*
   - *Roving frame*

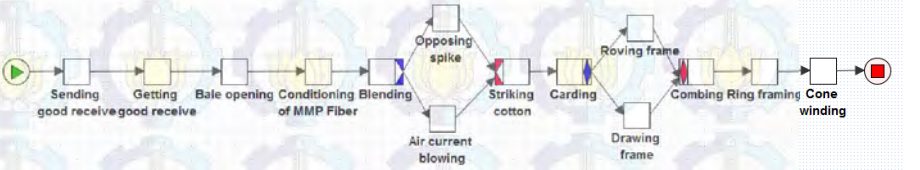
Bentuk *event log* dari data ini diperlihatkan pada Tabel 6.4.

Tabel 6.4 *Event Log* Data Produksi

Case ID	Activity	Time Stamp	Organisator	Cost
PP1	Sending good receive	6/20/2014 8:32	Purchase Department	243303
PP1	Getting good receive	6/20/2014 13:42	Spinning Department	138941
PP1	Bale opening	6/20/2014 23:41	Spinning Department	508599
PP1	Conditioning of MMP Fiber	6/21/2014 8:16	Spinning Department	63706
PP1	Blending	6/21/2014 10:46	Spinning Department	496301
PP1	Opposing spike	6/21/2014 16:57	Blowing Department	352814
PP1	Striking cotton	6/21/2014 18:09	Blowing Department	125149
PP1	Carding	6/21/2014 19:15	Spinning Department	957335
PP1	Roving frame	6/22/2014 10:51	Framing Department	516512
PP1	Drawing frame	6/22/2014 16:28	Framing Department	455652
PP1	Combing	6/22/2014 23:42	Spinning Department	394780
PP1	Ring framing	6/23/2014 4:48	Spinning Department	359985
PP1	Cone winding	6/23/2014 16:44	Spinning Department	319064

Pada Tabel 6.4 menunjukan salah satu *case* dari *event log* data produksi. Sebenarnya dalam event log data produksi terkandung 50

case produksi, dengan 8 jenis *trace*. Pada kolom aktivitas terlihat tedapat perbedaan warna, perbedaan ini menunjukkan bahwa aktivitas itu dikerjakan oleh organisasi yang berbeda.
Model dari *event log* data prodroduksi dapat dilihat pada Gambar 6.5. Hubungan antar aktivitas dari model data purchase dapat dilihat pada Tabel 6.5.



Gambar 6.5 Model Data Produksi

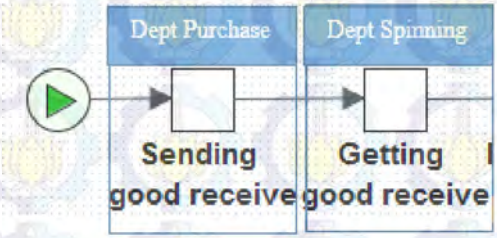
Tabel 6.5 Hubungan Antar Aktivitas dari Gambar 6.5

Input	Split / Join	Output
{Start}		Sending good receive
Sending good receive		Getting good receive
Getting good receive		Bale opening
Bale opening		Conditioning of MMP Fiber
Conditioning of MMP Fiber		Blending
Blending	XOR Split	Opposing spike, Air current blowing,
Opposing spike, Air current blowing,	XOR Join	Striking cotton
Striking cotton		Carding
Carding	OR Split	Roving frame, Drawing frame,
Drawing frame, Roving frame,	OR Join	Combing

Input	Split / Join	Output
Combing		Ring framing
Ring framing		Cone winding
Cone winding		{end}

Jenis pola koordinasi *cross-organizational* yang ada pada data *event log* produksi adalah sebagau berikut:

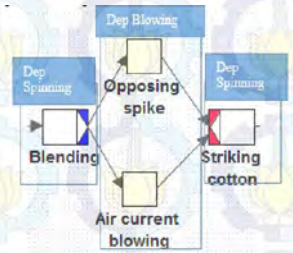
3. Pola Koordinasi Pertukaran Pesan
   Pola ini terjadi antara departemen *Purchase* dan *Spinning* pada saat aktivitas *sending* dan *getting good receive* saat itu terjadi pertukaran pesan berupa *good receive* apa saja yang diterima dan dikirimkan, pertukaran peran ini terjadi untuk melakukan pengecekan barang keluar dan masuk. Alur pertukaran pesan dapat dilihat pada Gambar 6.6.



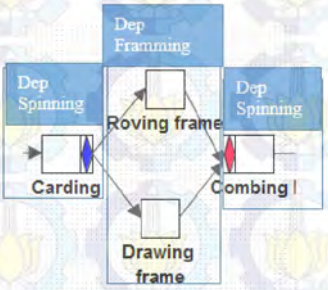
Gambar 6.6 Alur Pertukaran Pesan Data Produksi

4. Pola Koordinasi Aktivitas Sinkron
   Dalam data ini terdapat dua jenis pola hubungan aktivitas sinkron yaitu:
   - Pola hubungan *Capacity Sharing*
     Pola hubungan ini terjadi antara departemen *Spinning Blowing* dan *Spinning Framming*. Pada *Spinning Blowing* walaupun departemen *Blowing* yang melakukan pembersihan namun departemen *Spinning* yang mengatur bahan yang akan dibersihkan, begitu pula dengan

departemen *Framming,* walaupaun yang melakukan pemilihan bahan adalah departemen *Framming* namun keputusan untuk memilih bentuk pemilihan yang akan dilakukan tetap oleh departemen *Spinning.* Jadi departemen *Spinning* yang memegang control walaupun pengerjaannya dilakukan oleh departemen *Blowing* dan *Framming.* Alur hubungan *capacity sharing Spinning Blowing* akan diperlihatkan pada Gambar 6.7 dan *Spinning Framming* akan diperlihatkan pada Gambar 6.8.



Gambar 6.7 Alur *Capacity Sharing Spinning Blowing* Data Produksi

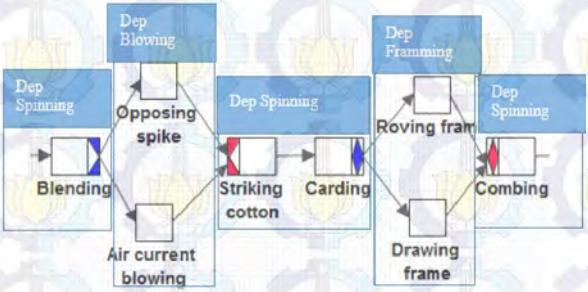


Gambar 6.8 Alur *Capacity Sharing Spinning Framming* Data Produksi

- Pola hubungan *Chain Execution*

  Pola hubungan ini terjadi antara departemen *Spinning, Blowing,* dan *Framming.* Karena departemen ini harus

berjalan berurutan prosesnya. Alur proses tersebut dapat dilihat pada Gambar 6.9.



Gambar 6.9 Alur *Chain Execution*

6.4 Uji Kebenaran dan Hasil Uji Coba

Uji coba pada sistem ini mengacu pada pengujian *Blackbox* untuk menguji apakah fungsionalitas sistem telah berjalan sebagaimana mestinya. Pengujian mengacu pada setiap fitur yang telah diimplementasikan.

6.4.1. Pengujian Fungsionalitas

Pengujian fungsionalitas sistem dilakukan secara mandiri dengan menyiapkan sejumlah skenario sebagai tolak ukur keberhasilan pengujian. Pengujian fungsionalitas dilakukan dengan mengacu pada kasus penggunaan yang telah dijelaskan pada subbab 4.6. Pengujian pada kebutuhan fungsionalitas dapat dijabarkan pada subbab berikut.

6.4.1.1 Pengujian Fitur Memasukkan *Event Log*

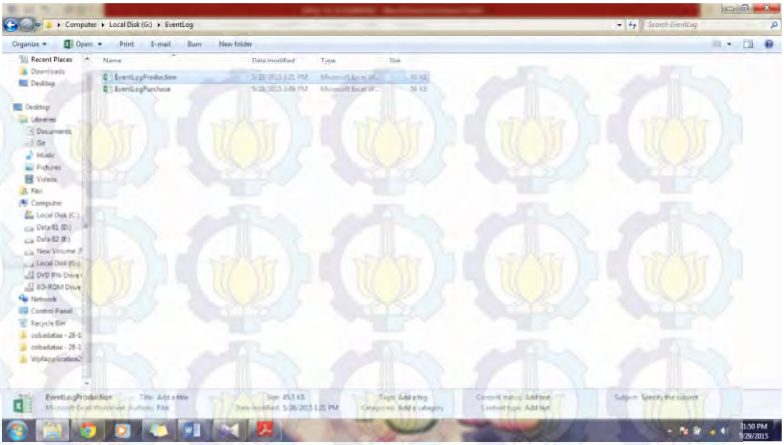
Pengujian fitur memasukkan *file event log* dilakukan dengan melakukan impor pada direktori yang di dalamnya ada *file event log*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.6.

Tabel 6.6 Pengujian Fitur Memasukkan Data *Event Log*

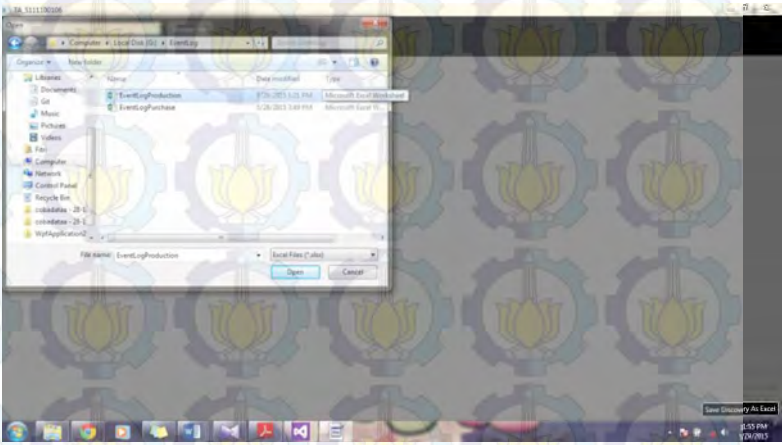
ID	**TA-UJ.UC0001**
Referensi Kasus Penggunaan	TA-UC0001

Nama	Pengujian fitur memasukkan *file event log*
Tujuan Pengujian	Menguji fitur untuk memasukkan *file event log* dengan memilih direktori tempat *file Petri Net* berada
Skenario 1	Pengguna memilih direktori tempat *file event log* berada dan sistem akan mengambil *file* tersebut
Kondisi Awal	• Sistem sudah dijalankan pada dan sistem menampilkan halaman Proses *Discovery* • Sistem menampilkan jendela pencari direktori
Data Uji	Data uji menggunakan direktori yang dibuat sendiri dan *file event log* yang dibuat di dalamnya
Langkah Pengujian	Pengguna memilih *file* yang sesuai pada jendela pencarian
Hasil Yang Diharapkan	Sistem mampu memasukkan *file event log* dengan menyimpan *path* dari *file* tersebut
Hasil Yang Didapat	*Path* dari *file event log* dapat digunakan untuk proses selanjutnya
Hasil Pengujian	100% Berhasil
Kondisi Akhir	*Path* dari *file event log* telah didapatkan oleh sistem

Dalam uji ini digunakan *file* EventLogProduction.xlsx seperti yang terlihat pada Gambar 6.10. Kemudian pengguna akan memilih *file event log* tersebut saat sistem menampilkan halaman Proses *Discovery*. Proses tersebut dapat dilihat pada Gambar 6.11. Kemudian untuk *file* yang telah berhasil diimpor dapat dilihat pada Gambar 6.12. Terisinya *textbox* dengan *path file* tersebut dapat disimpulkan bahwa sistem telah mampu mengambil *file event log* untuk proses selanjutnya.



Gambar 6.10 *File* EventLogProduction.xlsx pada Direktori



Gambar 6.11. Pengguna Memilih *File Event Log*

Gambar 6.12 *File Event Log* yang Telah Berhasil Diimpor

6.4.1.2 Pengujian Fitur Men-*discover* Proses Bisnis

Pengujian fitur *discover* proses bisnis dilakukan dengan melakukan pengisian atribut-atribut yang tersedia pada halaman konfigurasi. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.7.

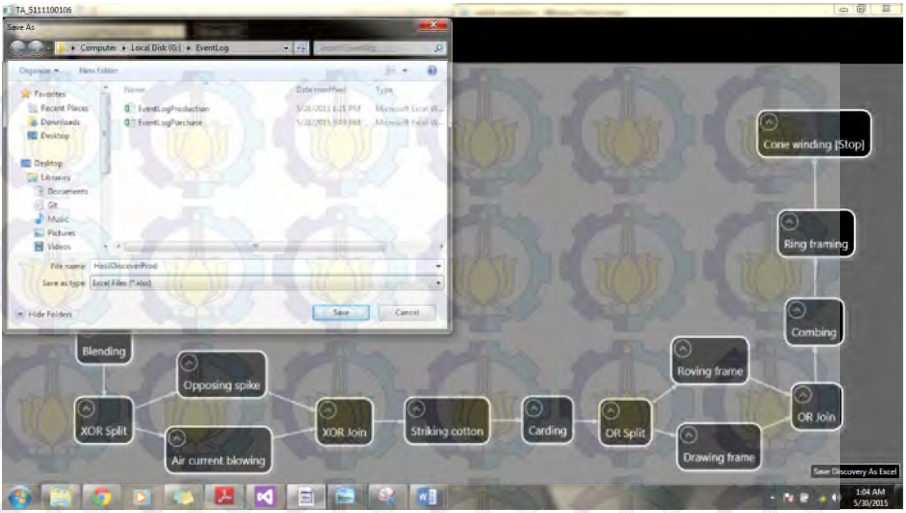
Tabel 6.7 Pengujian Fitur Konfigurasi BPMN

ID	**TA-UJ.UC0002**
Referensi Kasus Penggunaan	TA-UC0002
Nama	Pengujian fitur men-*discover* proses bisnis
Tujuan Pengujian	Menguji fitur untuk melakukan *discover* proses bisnis dari *event log* yang dimasukkan
Skenario 1	Pengguna menekan tombol *Discovery* setelah *textbox* terisi *path file*

Skenario 2	Pengguna menekan tombol *Save Discovery as* Excel setelah menekan tombol *Discovery*
Kondisi Awal	• Sistem sudah dijalankan dan sistem menampilkan halaman proses *discovery*
Data Uji	Data uji file *event log* yang telah dimasukkan
Langkah Pengujian	Setelah memasukkan *file* pengguna menekan tombol *Discovery*, setelah itu pengguna menekan tombol *Save Discovery as* Excel
Hasil Yang Diharapkan	Sistem mampu men-*discovery event log*
Hasil Yang Didapat	*Graph* model proses bisnis dari *event log* dan *file* Excel yang berisi tabel *input output* dan jenis paralel yang berhubungan dengan aktivitas.
Hasil Pengujian	100% Berhasil
Kondisi Akhir	*Graph* model proses bisnis *File* Excel tabel dari *graph*

Dalam uji ini digunakan *file* hasil dari masukan *event log* pada proses sebelumnya. Kemudian pengguna akan menekan tombol D*iscovery* dan kemudian menekan *Save Discovery as* Excel untuk menyimpan hasil *discovery* dalam bentuk Excel. Kemudian untuk *file* yang telah hasil *discovery* dalam bentuk *graph* dapat dilihat pada Gambar 6.13, proses penyimpanan dapat dilihat pada Gambar 6.14 dan Gambar 6.15 menunjukkan isi dari *file* Excel yang disimpan dari hasil proses *discovery*.

Gambar 6.13. Hasil *Discovery* EventLogProduction.xlsx



Gambar 6.14. Penyimpanan *File* Hasil *Discovery*

Input	Split/Join	Output	OneLoop	TwoLoop
{start}		Sending good receive		
Sending good receive		Getting good receive		
Getting good receive		Bale opening		
Bale opening		Conditioning of MMP Fiber		
Conditioning of MMP Fiber		Blending		
Blending	XOR Split	Opposing spike, Air current blowing,		
Opposing spike, Air current blowing,	XOR Join	Striking cotton		
Striking cotton		Carding		
Carding	OR Split	Roving frame, Drawing frame,		
Drawing frame, Roving frame,	OR Join	Combing		
Combing		Ring framing		
Ring framing		Cone winding		
Cone winding		{end}		

Gambar 6.15. Isi *File* Hasil *Discovery*

6.4.1.3 Pengujian Fitur Menghitung Data Optimasi

Pengujian menghitng data optimasi dilakukan dengan menekan tombol *Get Duration* sehingga menampilkan halaman optimasi kemudian pada halaman optimasi pengguna menekan tombol *Calculate Duration*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.8.

Tabel 6.8. Pengujian Fitur Menghitung Data Optimasi

ID	**TA-UJ.UC0003**
Referensi Kasus Penggunaan	TA-UC0003
Nama	Pengujian fitur menghitung data optimasi
Tujuan Pengujian	Menguji fitur untuk melakukan perhitungan data optimasi
Skenario 1	Pengguna menekan tombol *Calculate Duration*
Skenario 2	Pengguna menekan tombol *Save Duration as* Excel setelah menekan tombol *Calculate Duration*
Kondisi Awal	• Sistem sudah dijalankan pada dan sistem menampilkan halaman optimasi
Data Uji	Data uji file *event log* yang telah dimasukkan
Langkah Pengujian	Pengguna menekan tombol *Calculate Duration* pada halaman optimasi
Hasil Yang Diharapkan	Sistem mampu menghasilkan perhitungan data optimasi berupa *normal duration, crash duration, normal cost, crash cost, time slope,* dan *cost slope*
Hasil Yang Didapat	Data optimasi berupa *normal duration, crash duration, normal cost, crash cost, time slope, dan cost slope*
Hasil Pengujian	100% Berhasil
Kondisi Akhir	Tabel data optimasi berupa *normal duration, crash duration, normal cost, crash cost, time slope, dan cost slope* baik tampilan pada sistem maupun hasil penyimpanan dalam bentuk Excel

Dalam uji ini digunakan *file* hasil dari masukan *event log* pada proses sebelumnya. Kemudian pengguna akan menekan tombol *Calculate Duration* dan kemudian menekan *Save Duration as* Excel untuk menyimpan hasil perhitungan data optimasi dalam

bentuk Excel. Hasil perhitungan data optimasi beserta halaman optimasi secara penuh dapat dilihat pada Gambar 6.16, hasil perhitungan data optimasinya sendiri dapat dilihat pada Gambar 6.17 dan proses penyimpanan data optimasi Gambar 6.18.

Gambar 6.16. Halaman Optimasi dan Hasil Perhitungan Data Optimasi

Activity	NormalDuration	CrashDuration	NormalCost	CrashCost	TimeSlope	CostSlope
Sending good receive	4.84	2.96	205506.28	224158.53	1.88	9921.41
Getting good receive	5.71	3.65	151595.41	169217.13	2.06	8554.23
Bale opening	8.23	4.36	405371.94	433372.05	3.87	7235.17
Conditioning of MMP Fiber	2.12	1.7	144666.59	186809.49	0.42	100340.24
Blending	4.91	2.81	493891.26	560647.45	2.1	31788.66
Opposing spike	3.13	2.17	311653.05	357627.23	0.96	47889.77
Striking cotton	3.94	2.08	196427.39	232535.92	1.86	19413.19
Carding	11.44	6.66	1000090.34	1053667	4.78	11208.51
Roving frame	11.96	6.52	554865.99	598604.59	5.44	8040.18
Drawing frame	12.04	6.53	446634.76	461909.29	5.51	2772.15
Combing	9.19	5.52	403246.18	435287.75	3.67	8730.67
Ring framing	9.14	6.14	354279.07	378354.52	3	8025.15
Cone winding	6.99	5.96	302190.37	365265.47	1.03	61237.96
Air current blowing	3.29	2.41	299151.75	357632.43	0.88	66455.32

Gambar 6.17. Hasil Perhitungan Data Optimasi

Gambar 6.18. Penyimpanan *File* Data Optimasi

6.4.1.4 Pengujian Fitur Melakukan Optimasi Biaya dan *Makespan*

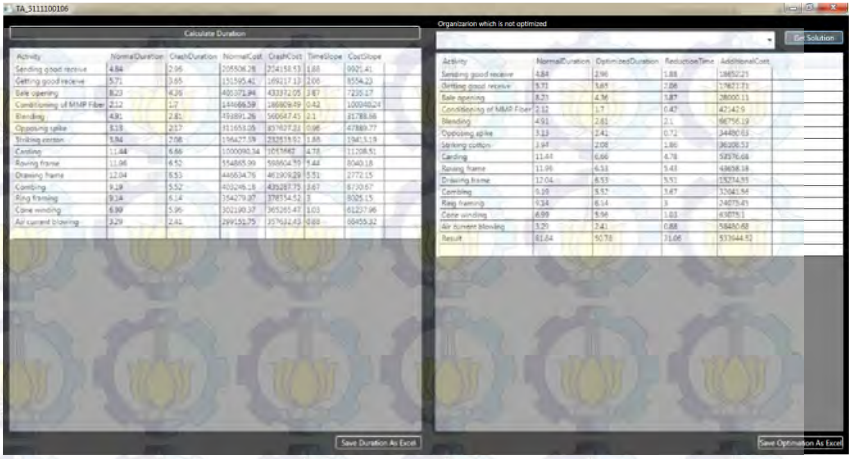
Pengujian menghitng data optimasi dilakukan dengan menekan tombol *Get Duration* sehingga menampilkan halaman optimasi kemudian pada halaman optimasi pengguna menekan tombol *Get Solution*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.9.

Tabel 6.9. Pengujian Fitur Melakukan Optimasi Biaya dan *Makespan*

ID	**TA-UJ.UC0004**
Referensi Kasus Penggunaan	TA-UC0004
Nama	Pengujian fitur melakukan optimasi biaya dan makespan
Tujuan Pengujian	Menguji fitur untuk melakukan optimasi biaya dan makespan
Skenario 1	Pengguna menekan tombol *Get Solution*
Kondisi Awal	• Sistem sudah dijalankan pada dan sistem menampilkan halaman optimasi

Data Uji	Data uji file *event log* yang telah dimasukkan
Langkah Pengujian	Pengguna menekan tombol *Get Solution* pada halaman optimasi
Hasil Yang Diharapkan	Sistem mampu menghasilkan perhitungan durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit
Hasil Yang Didapat	Durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit
Hasil Pengujian	100% Berhasil
Kondisi Akhir	Durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit

Dalam uji ini digunakan *file* hasil dari masukan *event log* pada proses sebelumnya. Kemudian pengguna akan menekan tombol *Get Solution*. Hasil perhitungan durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit beserta halaman optimasi secara penuh dapat dilihat pada Gambar 6.19, hasil perhitungan durasi proses bisnis yang paling kecil dan biaya tambahan yang paling sedikit sendiri dapat dilihat pada Gambar 6.20.



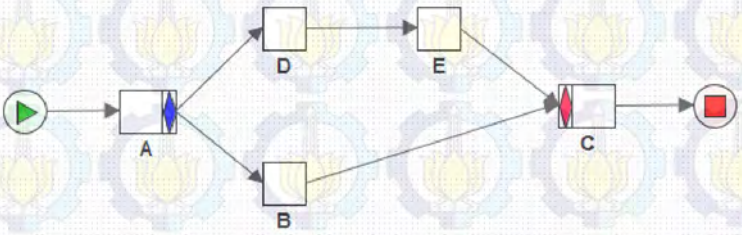
Gambar 6.19. Halaman Optimasi dan Durasi Proses Bisnis yang Paling Kecil dan Biaya Tambahan yang Paling Sedikit

Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
Sending good receive	4.84	2.96	1.88	18652.25
Getting good receive	5.71	3.65	2.06	17621.71
Bale opening	8.23	4.36	3.87	28000.11
Conditioning of MMP Fiber	2.12	1.7	0.42	42142.9
Blending	4.91	2.81	2.1	66756.19
Opposing spike	3.13	2.41	0.72	34480.63
Striking cotton	3.94	2.08	1.86	36108.53
Carding	11.44	6.66	4.78	53576.68
Roving frame	11.96	6.53	5.43	43658.18
Drawing frame	12.04	6.53	5.51	15274.55
Combing	9.19	5.52	3.67	32041.56
Ring framing	9.14	6.14	3	24075.45
Cone winding	6.99	5.96	1.03	63075.1
Air current blowing	3.29	2.41	0.88	58480.68
Result	81.84	50.78	31.06	533944.52

Gambar 6.20. Hasil Optimasi Berupa Durasi Proses Bisnis yang Paling Kecil dan Biaya Tambahan yang Paling Sedikit

6.4.2. Pengujian Validitas Hasil

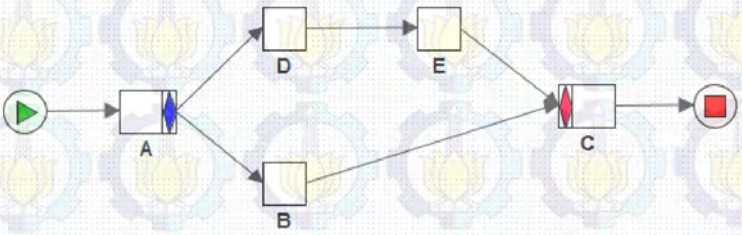
Pada bagian ini akan dijelaskan pengujian validitas hasil sistem. Uji coba dilakukan dengan bantuan kakas YAWL, Excel, dan LINGO. Dalam pengujian ini yang pertama dilakukan adalah membentuk model dengan menggunakan YAWL, model dapat dilihat pada Gambar 6.21. Setelah itu dibangkitkan *event log* untuk model tersebut, *event log* yang dibangkitkan dari model YAWL kemudian dirubah dalam bentuk Excel dan ditambahkan atribut pendukung yang tidak terdapat dalam *event log* hasil dari YAWL seperti *cost* teteapi banyak *trace*, aktivitas, dan *timestamp* tetap sama sesuai dengan *event log* yang dibangkitkan dari YAWL. *Event log* dari YAWL dapat dilihat pada Gambar 6.22 dan *event log* yang sidah dirubah dalam bentuk Excel dapat dilihat pada Tabel 6.10.



Gambar 6.21 Model YAWL

6.4.2. Pengujian Validitas Hasil

Pada bagian ini akan dijelaskan pengujian validitas hasil sistem. Uji coba dilakukan dengan bantuan kakas YAWL, Excel, dan LINGO. Dalam pengujian ini yang pertama dilakukan adalah membentuk model dengan menggunakan YAWL, model dapat dilihat pada Gambar 6.21. Setelah itu dibangkitkan *event log* untuk model tersebut, *event log* yang dibangkitkan dari model YAWL kemudian dirubah dalam bentuk Excel dan ditambahkan atribut pendukung yang tidak terdapat dalam *event log* hasil dari YAWL seperti *cost* teteapi banyak *trace*, aktivitas, dan *timestamp* tetap sama sesuai dengan *event log* yang dibangkitkan dari YAWL. *Event log* dari YAWL dapat dilihat pada Gambar 6.22 dan *event log* yang sidah dirubah dalam bentuk Excel dapat dilihat pada Tabel 6.10.



Gambar 6.21 Model YAWL

```
<trace>
    <string key="concept:name" value="300.PO.IMP.201408.0021"/>
    <event>
        <string key="Operation" value="A"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="A"/>
        <date key="time:timestamp" value="2014-08-27T10:30:42+07:00"/>
    </event>
    <event>
        <string key="Operation" value="B"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="B"/>
        <date key="time:timestamp" value="2014-08-27T10:30:53+07:00"/>
    </event>
    <event>
        <string key="Operation" value="D"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="D"/>
        <date key="time:timestamp" value="2014-08-27T11:31:34+07:00"/>
    </event>
    <event>
        <string key="Operation" value="E"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="E"/>
        <date key="time:timestamp" value="2014-08-27T12:31:34+07:00"/>
    </event>
    <event>
        <string key="Operation" value="C"/>
        <string key="lifecycle:transition" value="complete"/>
        <string key="concept:name" value="C"/>
        <date key="time:timestamp" value="2014-08-27T14:31:34+07:00"/>
    </event>
```

Gambar 6.22 Bentuk *Event Log* YAWL

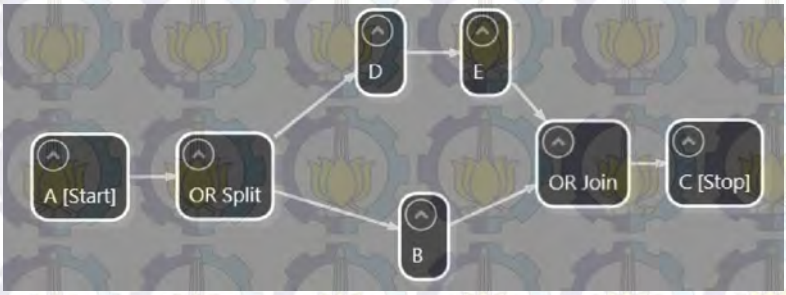
Tabel 6.10 Bentuk *Event Log* YAWL yang Dirubah Excel

Case ID	Activity	TimeStamp	Originator	Cost
300.PO.IMP.201406.0024	A	6/20/14 12:32	AN	400
300.PO.IMP.201406.0024	D	6/20/14 13:10	AN	1500
300.PO.IMP.201406.0024	B	6/20/14 15:00	AN	1000
300.PO.IMP.201406.0024	E	6/20/14 16:00	AN	2000
300.PO.IMP.201406.0024	C	6/20/14 18:00	AN	1200
300.PO.IMP.201406.0015	A	6/20/14 11:32	AN	1000
300.PO.IMP.201406.0015	B	6/20/14 12:32	AN	400
300.PO.IMP.201406.0015	C	6/20/14 13:10	AN	500
300.PO.IMP.201408.0021	A	8/27/14 10:30	AN	50
300.PO.IMP.201408.0021	B	8/27/14 10:30	AN	1000
300.PO.IMP.201408.0021	D	8/27/14 11:31	AN	1000
300.PO.IMP.201408.0021	E	8/27/14 12:31	AN	2000
300.PO.IMP.201408.0021	C	8/27/14 14:31	AN	1000
300.PO.IMP.201406.0025	A	6/20/14 12:32	AN	400
300.PO.IMP.201406.0025	D	6/20/14 13:10	AN	1000
300.PO.IMP.201406.0025	E	6/20/14 14:10	AN	500
300.PO.IMP.201406.0025	B	6/20/14 15:00	AN	500
300.PO.IMP.201406.0025	C	6/20/14 15:50	AN	500

6.3.2.1. Pengujian Validitas Hasil *Discovery*

Pada bagian ini akan dijelaskan pengujian hasil *discovery event log* menjadi model proses bisnis. Pengecekan dengan membandingkan model awal yang dibuat pada YAWL pada

Gambar 6.21 harus sama dengan hasil proses *discovery*. Perbandingan dilakukan dengan membandingkan hubungan antar aktivitas dan percabangan yang digunakan. Hubungan antar aktivitas pada model YAWL dapat dilihat pada Tabel 6.11. Hasil proses *discovery* dapat dilihat pada Gambar 6.23 dan Hubungan antar aktivitas pada model hasil proses *discovery* dapat dilihat pada Tabel 6.12.



Gambar 6.23 Bentuk Model Hasil *Discovery*

Tabel 6.11 Hubungan Aktivitas Model YAWL Gambar 6.21

Input	Split atau Join	Output
A	OR Split	D, B
D		E
E, B	OR Join	C

Tabel 6.12 Hubungan Aktivitas Model Hasil *Discovery* Gambar 6.23

Input	Split atau Join	Output
A	OR Split	D, B
D		E
E, B	OR Join	C

Dari Tabel 6.11 dan Tabel 6.12 dapat dilihat bahwa kedua tabel tersebut sama hal ini menandakan bahwa model hasil *discovery* benar.

6.3.2.2. Pengujian Validitas Hasil Perhitungan Data untuk Optimasi

Pada bagian ini akan dijelaskan pengujian hasil perhitungan data untuk optimasi. Pengecekan dilakukan dengan membandingkan perhitungan secara manual dengan menggunakan Excel dan kemudian hasilnya dibandingkan dengan hasil perhitungan menggunakan program. Data langkah perhitungan Excel dapat dilihat pada Tabel 6.13, hasil perhitungan Excel dapat dilihat pada Tabel 6.14, hasil perhitungan program dapat dilihat pada Tabel 6.15.

Tabel 6.13 Langkah Perhitungan Excel

Case ID	Activity	TimeStamp	Originator	Cost	Durasi Per Case
300.PO.IMP.201406.0024	A	6/20/14 12:32	AN	400	0.634166667
300.PO.IMP.201406.0024	D	6/20/14 13:10	AN	1500	2.828888889
300.PO.IMP.201406.0024	B	6/20/14 15:00	AN	1000	3
300.PO.IMP.201406.0024	E	6/20/14 16:00	AN	2000	2
300.PO.IMP.201406.0024	C	6/20/14 18:00	AN	1200	2.115763889
300.PO.IMP.201406.0015	A	6/20/14 11:32	AN	1000	1.000833333
300.PO.IMP.201406.0015	B	6/20/14 12:32	AN	400	0.634166667
300.PO.IMP.201406.0015	C	6/20/14 13:10	AN	500	0.8175
300.PO.IMP.201408.0021	A	8/27/14 10:30	AN	50	0.003055556
300.PO.IMP.201408.0021	B	8/27/14 10:30	AN	1000	4.011388889
300.PO.IMP.201408.0021	D	8/27/14 11:31	AN	1000	1
300.PO.IMP.201408.0021	E	8/27/14 12:31	AN	2000	2
300.PO.IMP.201408.0021	C	8/27/14 14:31	AN	1000	1.753611111
300.PO.IMP.201406.0025	A	6/20/14 12:32	AN	400	0.634166667
300.PO.IMP.201406.0025	D	6/20/14 13:10	AN	1000	1
300.PO.IMP.201406.0025	E	6/20/14 14:10	AN	500	1.666666667
300.PO.IMP.201406.0025	B	6/20/14 15:00	AN	500	0.833333333
300.PO.IMP.201406.0025	C	6/20/14 15:50	AN	500	1.033541667

Aktivitas	Rata-rata Dur	SD Dur
A	0.57	0.42
B	2.12	1.65
C	1.43	0.61
D	1.61	1.06
E	1.89	0.19

Aktivitas	Rata-rata Cos	SD Cost
A	463	394.49
B	725	320.16
C	800	355.90
D	1167	288.68
E	1500	866.03

Aktivitas	Normal Dur	Crash Dur	Normal Cost	Crash Cost	Time Slope	Cost Slope
A	0.57	0.15	462.50	856.99	0.42	939.26
B	2.12	0.47	725.00	1045.16	1.65	194.04
C	1.43	0.82	800.00	1155.90	0.61	583.44
D	1.61	0.55	1166.67	1455.35	1.06	272.34
E	1.89	1.70	1500.00	2366.03	0.19	4558.05

0.63	2.828888889	3	2	2.115764					
1	1	0.634166667	2	0.8175					
0	1	4.011388889	1.666667	1.753611					
0.63		0.833333333		1.033542					
400	1500	1000	2000	1200					
1000	1000	400	2000	500					
50	1000	1000	500	1000					
400		500		500					

Tabel 6.14 Hasil Perhitungan Excel

Aktivitas	Normal Dur	Crash Dur	Normal Cost	Crash Cost	Time Slope	Cost Slope
A	0.57	0.15	462.50	856.99	0.41	952
B	2.12	0.46	725.00	1045.16	1.65	193
C	1.43	0.82	800.00	1155.90	0.61	586
D	1.61	0.55	1166.67	1455.34	1.06	273
E	1.89	1.70	1500.00	2366.03	0.19	4500

Tabel 6.15 Hasil Perhitungan Program

Activity	NormalDuration	CrashDuration	NormalCost	CrashCost	TimeSlope	CostSlope
A	0.57	0.15	462.5	856.99	0.42	939.26
B	2.12	0.47	725	1045.16	1.65	194.04
C	1.43	0.82	800	1155.9	0.61	583.44
D	1.61	0.55	1166.67	1455.35	1.06	272.34
E	1.89	1.7	1500	2366.03	0.19	4558.05

Dari Tabel 6.14 dan Tabel 6.15 dapat dilihat bahwa kedua tabel tersebut sama hal ini menandakan bahwa model hasil perhitungan benar.

6.3.2.3. Pengujian Validitas Hasil Optimasi (Minimum Durasi dan Biaya Tambahan)

Pada bagian ini akan dijelaskan pengujian hasil optimasi (minimum durasi dan biaya tambahan). Pengecekan dilakukan dengan membandingkan hasil optimasi jika menggunakan LINGO dan kemudian hasilnya dibandingkan dengan hasil optimasi menggunakan program. Data hasil optimasi LINGO dapat dilihat pada Gambar 6.24, hasil optimasi program dapat dilihat pada Gambar 6.25.

```
Global optimal solution found.
Objective value:                              1905.1
Infeasibilities:                              0.000000
Total solver iterations:                           12
Elapsed runtime seconds:                         0.05

Model Class:                                       LP

Total variables:                   11
Nonlinear variables:                0
Integer variables:                  0

Total constraints:                 23
Nonlinear constraints:              0

Total nonzeros:                    39
Nonlinear nonzeros:                 0

                     Variable           Value
                           XA        0.4200000
                           XD        1.060000
                           XB        0.000000
                           XE        0.1900000
                           XC        0.6100000
                           YA        0.000000
                           YD        0.1500000
                           YB        0.2800000
                           YE        0.7000000
                           YC        2.400000
                      YFINISH        3.220000
```

Gambar 6.24 Hasil Optimasi LINGO

Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
A	0.57	0.15	0.42	394.49
D	1.61	0.55	1.06	288.68
B	2.12	2.12	0	0
E	1.89	1.7	0.19	866.03
C	1.43	0.82	0.61	355.9
Result	5.5	3.22	2.28	1905.1

Gambar 6.25 Hasil Optimasi Program

Dari Gambar 6.24 dan Gambar 6.25 dapat dilihat bahwa kedua menghasilkan nilai yang sama, hal ini menandakan bahwa hasil optimasi perhitungan benar.

6.4.3. Hasil Uji Terhadap Data Studi Kasus

Pada bagian ini akan diperlihatkan hasil *running* sistem dengan masukkan data studi kasus seperti yang ada pada subbab 6.2.

6.3.3.1. *Hasil Data Studi Kasus* Event Log Purchase Order *Bahan Baku Benang*

Data studi kasus ini telah dijelaskan pada subbab 6.3.1. dalam susbab ini akan diperlihatkan hasil dari *running* sistem menggunakan studi kasus tersebut. Pada Gambar 6.26 menunjukkan hasil proses *discovery* dari *event log purchase order*, kemudian hasil penyimpanan dari proses *discovery* ditunjukkan pada Tabel 6.16. Kesamaan antara model dan hasil *discovery* juga ditunjukkan oleh Tabel 6.17. Untuk data optimasi hasil perhitungan ditunjukkan pada Gambar 6.27 dan hasil optimasi ditunjukkan pada Gambar 6.28.

Hasil proses *discovery* pada sistem yang ditunjukkan Gambar 6.26 sama dengan model awal *event log* pada gambar Gambar 6.1.

Dari hasil optimasi pada Gambar 6.28 diketahui bahwa durasi awal proses bisnis adalah 152.34 jam kemudian dipercepat hingga 84.69 jam dengan biaya tambahan sebesar 336760.76 rupiah.

Gambar 6.26 Hasil Model Proses *Discovery Event Log* Studi Kasus *Purchase Order*

Tabel 6.16 Hasil *Save* Model Proses *Discovery Event Log* ke Excel Studi Kasus *Purchase Order*

Input	Split/Join	Output	OneLoop	TwoLoop
{start}		Sending PO Number		
Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,		
Producing Good Orders Sup1		Pakcaging Good Orders		
Sending Permitting Doc		Determining PPh and Giving Permission		
Determining PPh and Giving Permission	AND Split	Paying PPh, Producing Good Orders Sup2,		
Producing Good Orders Sup2, Paying PPh,	AND Join	Packaging Good Orders and Getting PPh Confirm		
Packaging Good Orders and Getting PPh Confirm, Pakcaging Good Orders,	OR Join	Sending Good Orders		
Sending Good Orders		Receiving Good Receive		
Receiving Good Receive		{end}		

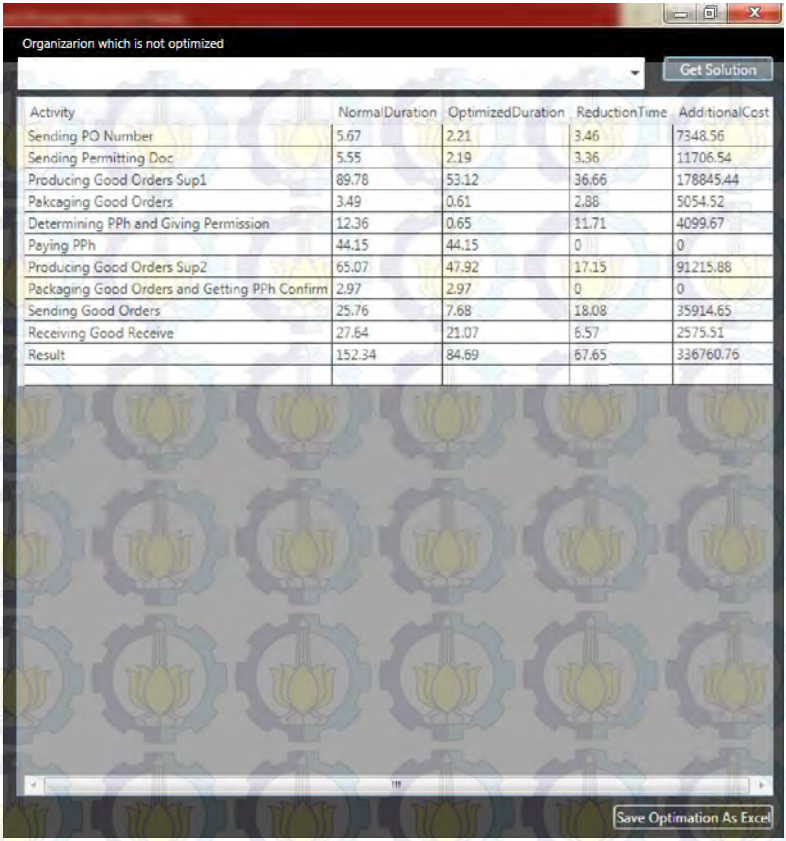
Tabel 6.17 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			
Input	Split / Join	Output	Input	Split / Join	Output	Kesamaan
{Start}		Sending PO Number	{Start}		Sending PO Number	Sama
Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,	Sending PO Number	OR Split	Sending Permitting Doc, Producing Good Orders Sup1,	Sama
Producing Good Orders Sup1		Pakcaging Good Orders	Producing Good Orders Sup1		Pakcaging Good Orders	Sama
Sending Permitting Doc		Determining PPh and Giving Permission	Sending Permitting Doc		Determining PPh and Giving Permission	Sama
Determining PPh and	AND Split	Paying PPh, Producing	Determining PPh and	AND Split	Paying PPh, Producing	

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
Giving Permission		Good Orders Sup2,	Giving Permission		Good Orders Sup2,	
Producing Good Orders Sup2, Paying PPh	AND Join	Packaging Good Orders and Getting PPh Confirm	Producing Good Orders Sup2, Paying PPh	AND Join	Packaging Good Orders and Getting PPh Confirm	Sama
Packaging Good Orders and Getting PPh Confirm, Pakcaging Good Orders	OR Join	Sending Good Orders	Packaging Good Orders and Getting PPh Confirm, Pakcaging Good Orders	OR Join	Sending Good Orders	Sama
Sending Good Orders		Receiving Good Receive	Sending Good Orders		Receiving Good Receive	Sama

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
Receiving Good Receive		{end}	Receiving Good Receive		{end}	Sama

Gambar 6.27 Hasil Data Optimasi *Event Log* Studi Kasus *Purchase Order*



Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
Sending PO Number	5.67	2.21	3.46	7348.56
Sending Permitting Doc	5.55	2.19	3.36	11706.54
Producing Good Orders Sup1	89.78	53.12	36.66	178845.44
Pakcaging Good Orders	3.49	0.61	2.88	5054.52
Determining PPh and Giving Permission	12.36	0.65	11.71	4099.67
Paying PPh	44.15	44.15	0	0
Producing Good Orders Sup2	65.07	47.92	17.15	91215.88
Packaging Good Orders and Getting PPh Confirm	2.97	2.97	0	0
Sending Good Orders	25.76	7.68	18.08	35914.65
Receiving Good Receive	27.64	21.07	6.57	2575.51
Result	152.34	84.69	67.65	336760.76

Gambar 6.28 Hasil Optimasi *Event Log* Studi Kasus *Purchase Order*

6.3.3.2. *Hasil Data Studi Kasus* Event Log *Produksi Benang*

Data studi kasus ini telah dijelaskan pada subbab 6.3.2. dalam subbab ini akan diperlihatkan hasil dari *running* sistem menggunakan studi kasus tersebut. Pada Gambar 6.29 menunjukkan hasil proses *discovery* dari *event log* produksi, kemudian hasil penyimpanan dari proses *discovery* ditunjukkan pada Tabel 6.19. Kesamaan antara model dan hasil *discovery* juga ditunjukkan oleh Tabel 6.18. Untuk data optimasi hasil perhitungan ditunjukkan pada Gambar 6.30 dan hasil optimasi ditunjukkan pada Gambar 6.31.

Hasil proses *discovery* pada sistem yang ditunjukkan Gambar 6.29 sama dengan model awal *event log* pada gambar Gambar 6.5.

Dari hasil optimasi pada Gambar 6.31 diketahui bahwa durasi awal proses bisnis adalah 81.44 jam kemudian dipercepat hingga 50.78 jam dengan biaya tambahan sebesar 533944.52 rupiah.

Gambar 6.29 Hasil Model Proses *Discovery Event Log* Studi Kasus Produksi

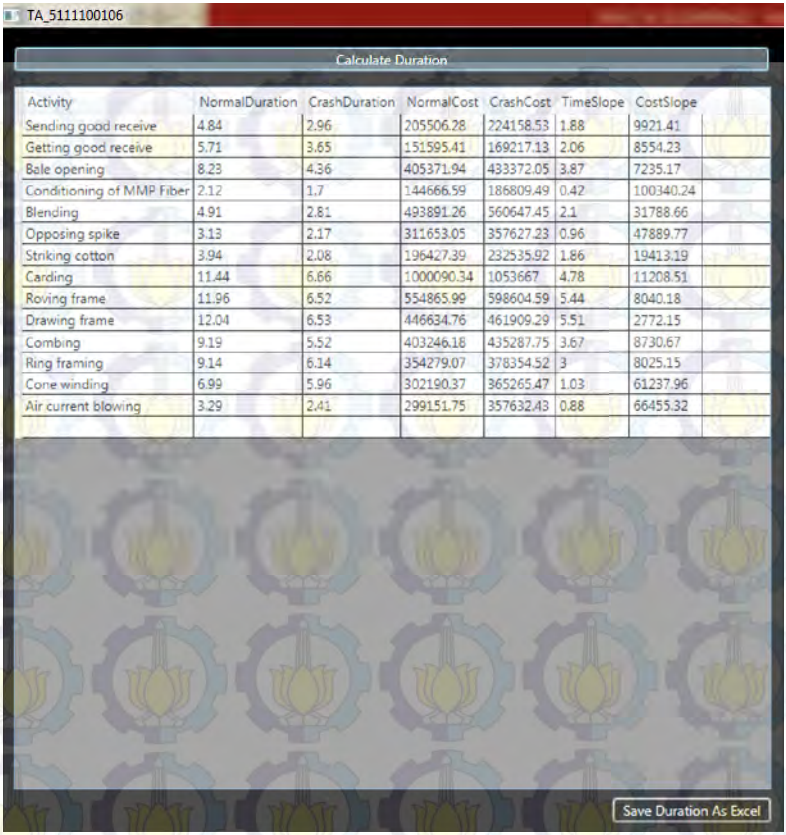
Tabel 6.18 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			
Input	Split / Join	Output	Input	Split / Join	Output	Kesamaan
{Start}		Sending good receive	{Start}		Sending good receive	Sama
Sending good receive		Getting good receive	Sending good receive		Getting good receive	Sama
Getting good receive		Bale opening	Getting good receive		Bale opening	Sama
Bale opening		Conditioning of MMP Fiber	Bale opening		Conditioning of MMP Fiber	Sama
Conditioning of MMP Fiber		Blending	Conditioning of MMP Fiber		Blending	
Blending	XOR Split	Opposing spike, Air current blowing,	Blending	XOR Split	Opposing spike, Air current blowing,	Sama
Opposing spike, Air	XOR Join	Striking cotton	Opposing spike, Air	XOR Join	Striking cotton	Sama

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
current blowing,			current blowing,			
Striking cotton		Carding	Striking cotton		Carding	Sama
Carding	OR Split	Roving frame, Drawing frame,	Carding	OR Split	Roving frame, Drawing frame,	Sama
Drawing frame, Roving frame,	OR Join	Combing	Drawing frame, Roving frame,	OR Join	Combing	Sama
Combing		Ring framing	Combing		Ring framing	Sama
Ring framing		Cone winding	Ring framing		Cone winding	Sama
Cone winding		{end}	Cone winding		{end}	Sama

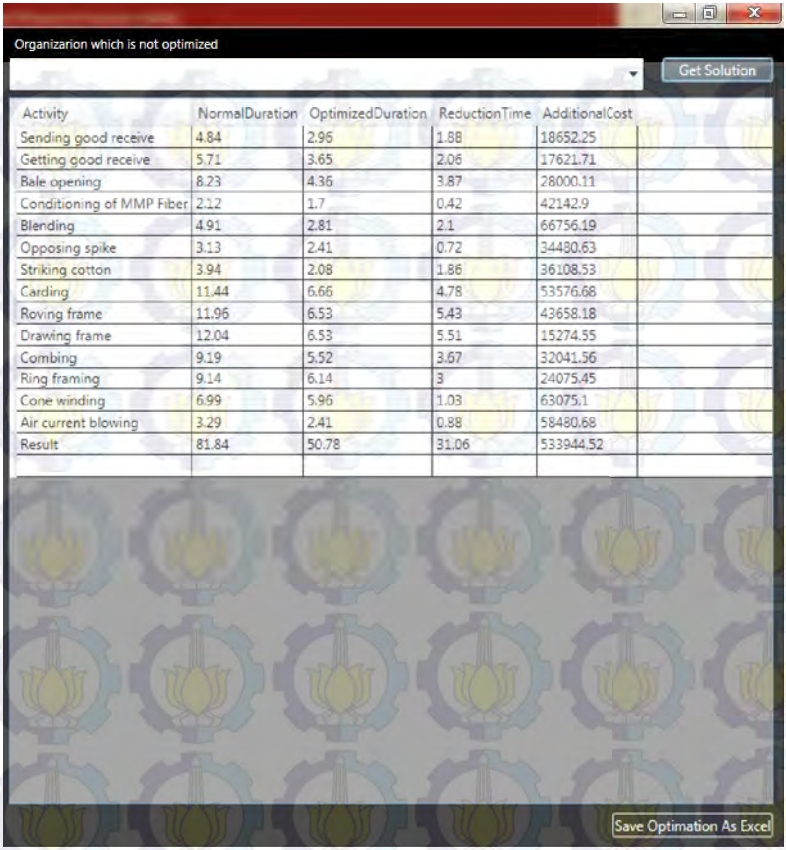
Tabel 6.19 Hasil *Save* Model Proses *Discovery Event Log* ke Excel Studi Kasus Produksi

Input	Split/Join	Output	OneLoop	TwoLoop
{start}		Sending good receive		
Sending good receive		Getting good receive		
Getting good receive		Bale opening		
Bale opening		Conditioning of MMP Fiber		
Conditioning of MMP Fiber		Blending		
Blending	XOR Split	Opposing spike, Air current blowing,		
Opposing spike, Air current blowing,	XOR Join	Striking cotton		
Striking cotton		Carding		
Carding	OR Split	Roving frame, Drawing frame,		
Drawing frame, Roving frame,	OR Join	Combing		
Combing		Ring framing		
Ring framing		Cone winding		
Cone winding		{end}		



Activity	NormalDuration	CrashDuration	NormalCost	CrashCost	TimeSlope	CostSlope
Sending good receive	4.84	2.96	205506.28	224158.53	1.88	9921.41
Getting good receive	5.71	3.65	151595.41	169217.13	2.06	8554.23
Bale opening	8.23	4.36	405371.94	433372.05	3.87	7235.17
Conditioning of MMP Fiber	2.12	1.7	144666.59	186809.49	0.42	100340.24
Blending	4.91	2.81	493891.26	560647.45	2.1	31788.66
Opposing spike	3.13	2.17	311653.05	357627.23	0.96	47889.77
Striking cotton	3.94	2.08	196427.39	232535.92	1.86	19413.19
Carding	11.44	6.66	1000090.34	1053667	4.78	11208.51
Roving frame	11.96	6.52	554865.99	598604.59	5.44	8040.18
Drawing frame	12.04	6.53	446634.76	461909.29	5.51	2772.15
Combing	9.19	5.52	403246.18	435287.75	3.67	8730.67
Ring framing	9.14	6.14	354279.07	378354.52	3	8025.15
Cone winding	6.99	5.96	302190.37	365265.47	1.03	61237.96
Air current blowing	3.29	2.41	299151.75	357632.43	0.88	66455.32

Gambar 6.30 Hasil Data Optimasi *Event Log* Studi Kasus Produksi



Activity	NormalDuration	OptimizedDuration	ReductionTime	AdditionalCost
Sending good receive	4.84	2.96	1.88	18652.25
Getting good receive	5.71	3.65	2.06	17621.71
Bale opening	8.23	4.36	3.87	28000.11
Conditioning of MMP Fiber	2.12	1.7	0.42	42142.9
Blending	4.91	2.81	2.1	66756.19
Opposing spike	3.13	2.41	0.72	34480.63
Striking cotton	3.94	2.08	1.86	36108.53
Carding	11.44	6.66	4.78	53576.68
Roving frame	11.96	6.53	5.43	43658.18
Drawing frame	12.04	6.53	5.51	15274.55
Combing	9.19	5.52	3.67	32041.56
Ring framing	9.14	6.14	3	24075.45
Cone winding	6.99	5.96	1.03	63075.1
Air current blowing	3.29	2.41	0.88	58480.68
Result	81.84	50.78	31.06	533944.52

Gambar 6.31 Hasil Optimasi *Event Log* Studi Kasus Produksi

6.5 Evaluasi Sistem dengan Sistem Lain

Evaluasi dilakukan dengan membandingkan kinerja sistem yang dikembangkan dengan sistem lain. Perbandingan dilakukan dengan menggunakan hasil yang didapatkan pada hasil uji studi kasus.

6.5.1. Evaluasi terhadap Data *Event Log Purchace Order*

Rincian evaluasi ditulis pada Tabel 6.20 dan Tabel 6.21. Tabel 6.20 merincinkan evaluasi *process discovery* antarsistem. Dan Tabel 6.21 merincikan evaluasi optimasi antarsistem. Pada evaluasi ini terdapat 2 sistem yaitu sistem A dan sistem B. Sistem A adalah sistem yang dikembangkan. Sistem B adalah sistem yang dikembangkan pada buku Tugas Akhir dengan judul "Optimasi Waktu Produksi Berdasarkan Process Mining dari Activity Lifespan".

Tabel 6.20. Evaluasi Sistem dengan Sistem Lain terhadap *Process Discovery* Proses Bisnis *Purchase Order*

Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Sending PO number	Mulai	*OR-Split*	Mulai	*OR-Split*
Sending permitting document	*OR-Split*	*Sequence*	*OR-Split*	*Sequence*
Producing good orders sup1	*OR- Split*	*Sequence*	*OR- Split*	*Sequence*
Pakcaging good orders	*Sequence*	*OR-Join*	*Sequence*	*OR-Join*
Determining PPh and giving permission	*Sequence*	*AND-Split*	*Sequence*	*AND-Split*
Paying PPh	*AND-Split*	*AND-Join*	*AND-Split*	*AND-Join*
Producing good orders sup2	*AND-Split*	*AND-Join*	*AND-Split*	*AND-Join*
Packaging good orders and getting PPh confirm	*AND-Join*	*OR-Join*	*AND-Join*	*OR-Join*
Sending good orders	*OR-Join*	*Sequence*	*OR-Join*	*Sequence*

Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Receiving good receive	*Sequence*	Selesai	*Sequence*	Selesai

Tabel 6.21. Evaluasi Sistem dengan Sistem Lain terhadap Optimasi Proses Bisnis *Purchase Order*

Aktivitas	Sistem A	Sistem B
Biaya Tambahan	336760.76	336760.76
Durasi	84.69 jam	84.69 jam

6.5.2. Evaluasi terhadap Data *Event Log* Produksi Benang

Rincian evaluasi ditulis pada Tabel 6.22 dan Tabel 6.23. Tabel 6.22 merincinkan evaluasi *process discovery* antarsistem. Dan Tabel 6.23 merincikan evaluasi optimasi antarsistem. Pada evaluasi ini terdapat 2 sistem yaitu sistem A dan sistem B. Sistem A adalah sistem yang dikembangkan. Sistem B adalah sistem yang dikembangkan pada buku Tugas Akhir dengan judul "Optimasi Waktu Produksi Berdasarkan Process Mining dari Activity Lifespan".

Tabel 6.22. Evaluasi Sistem dengan Sistem Lain terhadap *Process Discovery* Proses Bisnis Produksi Benang

Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Sending good receive	Mulai	*Sequence*	Mulai	*Sequence*
Getting good receive	*Sequence*	*Sequence*	*Sequence*	*Sequence*
Bale opening	*Sequence*	*Sequence*	*Sequence*	*Sequence*

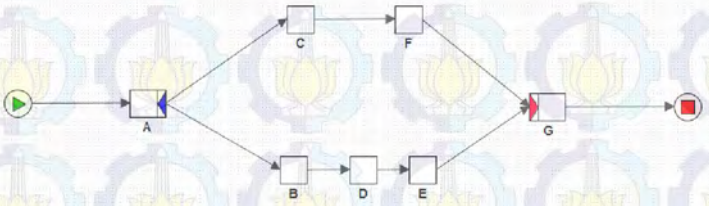
Aktivitas	Sistem A		Sistem B	
	Relasi Masuk	Relasi Keluar	Relasi Masuk	Relasi Keluar
Conditioning of MMF Fiber	*Sequence*	*Sequence*	*Sequence*	*Sequence*
Blending	*Sequence*	*XOR-Split*	*Sequence*	*XOR-Split*
Opposing spikes	*XOR-Split*	*XOR-Join*	*XOR-Split*	*XOR-Join*
Air current blowing	*XOR-Split*	*XOR-Join*	*XOR-Split*	*XOR-Join*
Striking cotton	*XOR-Join*	*Sequence*	*XOR-Join*	*Sequence*
Carding	*Sequence*	*OR-Split*	*Sequence*	*OR-Split*
Drawing frame	*OR-Split*	*OR-Join*	*OR-Split*	*OR-Join*
Roving frame	*OR-Split*	*OR-Join*	*OR-Split*	*OR-Join*
Combing	*OR-Join*	*Sequence*	*OR-Join*	*Sequence*
Ring framing	*Sequence*	*Sequence*	*Sequence*	*Sequence*
Cone winding	*Sequence*	Selesai	*Sequence*	Selesai

Tabel 6.23. Evaluasi Sistem dengan Sistem Lain terhadap Optimasi Proses Bisnis Produksi Benang

Aktivitas	Sistem A	Sistem B
Biaya Tambahan	533944.52	533944.52
Durasi	50.78 jam	50.78 jam

6.6 Evaluasi *Process Discovery* dengan *Event Log* Mengandung *Noise*

Terdapat suatu model:



Gambar 6.32 Model AND YAWL

Dari model YAWL tersebut dibangkitkan suatu *event log* sebagai berikut:

L1 = {(ACFBDEG), (ACBFDEG), (ACBDFEG), (ACBDEFG), (ABDECFG), (ABCDEFG), (ABCFDEG), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Hubungan antar aktivitas dari model Gambar 6.32 dapat dilihat pada Tabel 6.24.

Tabel 6.24. Hubungan Aktivitas Model Gambar 6.32

Input	Split / Join	Output
{Start}		A
A	AND Split	C, B
C		F
B		D
D		E
E, F	AND Join	G
G	XOR Join	{end}

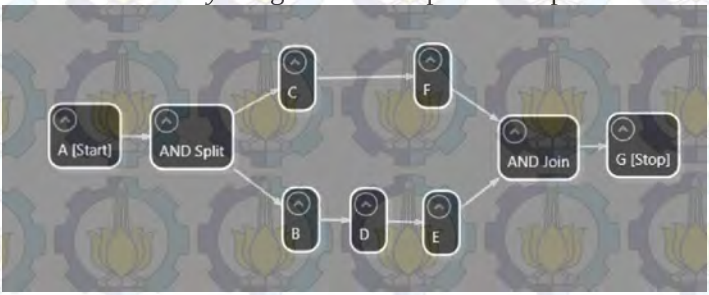
- **Noise 40%**

Dari *event log* L1 yang memiliki jumlah *case* 10, dirubah menjadi *event log* dengan *noise* 40%.

Event log pada L1 dirubah menjadi *event log*

L1’ = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCDFEG)}

Hasil *discovery* L’ dengan menggunakan sistem dapat dilihat pada Gambar 6.33 dan perbandingan hubungan aktivitas antara hasis *discovery* dengan model dapat dilihat pada Tabel 6.25.



Gambar 6.33 Hasil *Discovery* dengan *Noise* 40%

Tabel 6.25 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
{Start}		A	{Start}		A	Sama
A	AND Split	C, B	A	AND Split	C, B	Sama
C		F	C		F	Sama
B		D	B		D	Sama
D		E	D		E	Sama
E, F	AND Join	G	E, F	AND Join	G	Sama

Dari hasil Tabel 6.25 diketahui bahwa dengan kandungan *noise* 40% model masih dapat di-*discovery* sesuai dengan model aslinya.

- **Noise 50%**

Untuk *event log* yang jumlah *noise* pada *event log* L1 50%, sebagai contohnya adalah L1”.

L1” = {(AG), (ACBFDEG), (DFEG), (ACBDEFG), (ABDECFG), (CDEFG), (ABCFD), (ABDCEFG), (ABDCFEG), (ABCD)}

Hasil *discovery* L1” dengan menggunakan sistem dapat dilihat pada Gambar 6.34 dan perbandingan hubungan aktivitas antara hasis *discovery* dengan model dapat dilihat pada Tabel 6.26.

Gambar 6.34 Hasil *Discovery* dengan *Noise* 50%

Tabel 6.26 Perbandingan Hubungan Aktivitas Model dan Hasil *Discovery*

Hubungan Aktivitas Model			Hubungan Aktivitas Hasil *Discovery*			Kesamaan
Input	Split / Join	Output	Input	Split / Join	Output	
{Start}		A	{Start}		A	Sama
A	AND Split	C, B	A	OR Split	C, B,G	Beda
C		F	C		F	Sama
B		D	B	AND Split	D, F	Beda
D		E	D		E	Sama
E, F	AND Join	G	A, E, F	OR Join	G	Beda

Dari hasil Tabel 6.26 diketahui bahwa dengan kandungan *noise* 50% hasil *discovery* berbeda dengan model aslinya. Sehingga hasil *discovery* gagal.

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

7.1. Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi, dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Modifikasi terhadap *heuristic miner* dengan penentuan *threshold* yang otomatis dapat menghasilkan model sesuai dengan model proses bisnis yang sebenarnya.
2. Modifikasi terhadap *heuristic miner* dapat memodelkan bentuk *parallel split* dan *join* OR.
3. Data optimasi dapat dihitung dengan menggunakan rata-rata dan standar deviasi dari setiap aktivitas yang ada pada *event log* (ditunjukkan pada subbap 3.5).
4. Perangkat lunak berhasil memodelkan proses bisnis yang mengandung OR.
5. Proses optimasi pada proses bisnis *purchase order* menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 44.23% dengan tambahan biaya 19.41%.
6. Proses optimasi pada proses bisnis produksi benang menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 37.65% dengan tambahan biaya 12.1%.
7. *Process discovery* berhasil dengan kandungan *noise* 40% dalam *event log* sedangkan kandungan *noise* 50% dalam *event log* membuat *process discovery* gagal.
8. Perangkat lunak dapat mendapatkan durasi proses bisnis yang paling minimum dan biaya tambahan yang paling minimum.

9. Perangkat lunak berhasil menjalankan fitur-fitur yang ada dengan baik yaitu memasukkan *event log*, men-*discover* bisnis proses model, menghitung data optimasi, optimasi biaya tambahan dan percepatan *makespan* (durasi proses bisnis).

7.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan.

1. Penentuan *threshold* dapat ditentukan secara otomatis oleh sistem maupun dapat dimasukkan sendiri oleh pengguna, hal ini dilakukan untuk memberikan fasilitas pengguna yang sudah ahli dibidang penentuan model proses bisnis.
2. Penyerderhanaan pengimplementasian agar *running* sistem menjadi lebih cepat.
3. Fitur memasukkan data pada sistem dapat dikembangkan sehingga dapat menerima dokumen selain format Excel.
4. Fitur menyimpan data pada sistem dapat dikembangkan sehingga dapat menyimpan dokumen selain format Excel.

BAB VII
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

7.1. Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi, dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Modifikasi terhadap *heuristic miner* dengan penentuan *threshold* yang otomatis dapat menghasilkan model sesuai dengan model proses bisnis yang sebenarnya.
2. Modifikasi terhadap *heuristic miner* dapat memodelkan bentuk *parallel split* dan *join* OR.
3. Data optimasi dapat dihitung dengan menggunakan rata-rata dan standar deviasi dari setiap aktivitas yang ada pada *event log* (ditunjukkan pada subbap 3.5).
4. Perangkat lunak berhasil memodelkan proses bisnis yang mengandung OR.
5. Proses optimasi pada proses bisnis *purchase order* menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 44.23% dengan tambahan biaya 19.41%.
6. Proses optimasi pada proses bisnis produksi benang menghasilkan proses bisnis yang lebih optimum dari sebelumnya yaitu waktu berkurang 37.65% dengan tambahan biaya 12.1%.
7. *Process discovery* berhasil dengan kandungan *noise* 40% dalam *event log* sedangkan kandungan *noise* 50% dalam *event log* membuat *process discovery* gagal.
8. Perangkat lunak dapat mendapatkan durasi proses bisnis yang paling minimum dan biaya tambahan yang paling minimum.

9. Perangkat lunak berhasil menjalankan fitur-fitur yang ada dengan baik yaitu memasukkan *event log*, men-*discover* bisnis proses model, menghitung data optimasi, optimasi biaya tambahan dan percepatan *makespan* (durasi proses bisnis).

7.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran-saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan.

1. Penentuan *threshold* dapat ditentukan secara otomatis oleh sistem maupun dapat dimasukkan sendiri oleh pengguna, hal ini dilakukan untuk memberikan fasilitas pengguna yang sudah ahli dibidang penentuan model proses bisnis.
2. Penyerderhanaan pengimplementasian agar *running* sistem menjadi lebih cepat.
3. Fitur memasukkan data pada sistem dapat dikembangkan sehingga dapat menerima dokumen selain format Excel.
4. Fitur menyimpan data pada sistem dapat dikembangkan sehingga dapat menyimpan dokumen selain format Excel.

LAMPIRAN A

1. Halaman Proses *Discovery*

- Kode Sumber Xaml

```
<Window x:Class="WpfApplication2.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/present
ation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:graphsharp="clr-
namespace:GraphSharp.Controls;assembly=GraphSharp.Controls"
xmlns:local="clr-namespace:WpfApplication2"
xmlns:zoom="clr-
namespace:WPFExtensions.Controls;assembly=WPFExtensions"
Title="TA_5111100106" Height="416" Width="679">
<Window.Resources>
   <DataTemplate x:Key="demoTemplate" DataType="{x:Type
   local:Main}">
   <StackPanel Orientation="Horizontal" Margin="5">
   <TextBlock Text="{Binding Path=activity}"
   Foreground="White" />
   </StackPanel>
   </DataTemplate>
   <Style TargetType="{x:Type graphsharp:VertexControl}">
   <Setter Property="Template">
   <Setter.Value>
   <ControlTemplate TargetType="{x:Type
   graphsharp:VertexControl}">
    <Border BorderBrush="White" Background="Black"
    BorderThickness="2" CornerRadius="10,10,10,10"
    Padding="{TemplateBinding Padding}">
   <StackPanel Orientation="Vertical">
   <Expander IsExpanded="True">
   <ContentPresenter Content="{TemplateBinding Vertex}"
   ContentTemplate="{StaticResource demoTemplate}"/>
   </Expander>
   </StackPanel>
   <Border.Effect>
   <DropShadowEffect BlurRadius="2" Color="LightGray"
   Opacity="0.3" Direction="315"/>
   </Border.Effect>
   </Border>
   </ControlTemplate>
   </Setter.Value>
   </Setter>
   </Style>
   <Style TargetType="{x:Type graphsharp:EdgeControl}">
   <Style.Resources>
   <ToolTip x:Key="ToolTipContent">
```

```
    <StackPanel>
    <TextBlock FontWeight="Bold" Text="Edge Information"/>
    <TextBlock Text="{Binding ID}"/>
    </StackPanel>
    </ToolTip>
    </Style.Resources>
    <Setter Property="ToolTip" Value="{StaticResource
    ToolTipContent}"/>
    </Style>
    </Window.Resources>
    <Grid  Background="Black">
    <Grid.RowDefinitions>
    <RowDefinition Height="Auto" MinHeight="70"/>
    <RowDefinition/>
    </Grid.RowDefinitions>
    <zoom:ZoomControl  Grid.Row="1"  Zoom="0.2"
    ZoomBoxOpacity="0.5" Background="#ff656565">
    <local:PocGraphLayout x:Name="graphLayout" Margin="10"
    Graph="{Binding Path=Graph}"
        LayoutAlgorithmType="{Binding
    Path=LayoutAlgorithmType}"
        OverlapRemovalAlgorithmType="FSA"
        HighlightAlgorithmType="Simple" Grid.Row="1" />
    </zoom:ZoomControl>
    <Button Content="Discovery" HorizontalAlignment="Left"
    Margin="22,42,0,0" VerticalAlignment="Top" Width="75"
    Click="Button Click 1" Height="22" Background="Black"
    BorderBrush="Gray" Foreground="White"/>
    <Button Content="Save Discovery As Excel"
    HorizontalAlignment="Right" Margin="0,0,5,6"
    VerticalAlignment="Bottom" Width="137"
    Click="Button_Click_2" Height="25" Background="Black"
    BorderBrush="Gray" Foreground="White" Grid.Row="1"/>
    <Button Content="Browse File" HorizontalAlignment="Left"
    Margin="291,10,0,0" VerticalAlignment="Top" Width="74"
    Click="Button_Click_3" Height="auto" Background="Black"
    BorderBrush="Gray" Foreground="White"/>
    <TextBox Name="namaFile" HorizontalAlignment="Left"
    Height="auto" Margin="22,10,0,0" TextWrapping="Wrap"
    Text="" VerticalAlignment="Top" Width="255"/>
    <Button Content="Get Duration"
    HorizontalAlignment="Left" Margin="115,42,0,0"
    VerticalAlignment="Top" Width="86"
    Click="Button_Click_4" Height="22" Background="Black"
    BorderBrush="Gray" Foreground="White"/>
    </Grid>
</Window>
```

Kode Sumber A. 1 Xaml *Discovery*

- Kode Sumber Fungsi *Button Browse*

```
private void Button_Click_3(object sender,
RoutedEventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog();
    openFileDialog.Filter = "Excel Files(*.xlsx)|*.xlsx";
    if (openFileDialog.ShowDialog() == true)
    {
        namaFile.Text = openFileDialog.FileName; ;
    }
}
```

Kode Sumber A. 2 *Buttun Browse*

- Kode Sumber Fungsi *Button Discovery*

```
private void Button_Click_1(object sender,
RoutedEventArgs e)
{
  if (namaFile.Text == "")
  {
    MessageBoxResult result = MessageBox.Show("There is
    no file to discover");
  }
  else
  {
    Main vm = new Main();
    vm.viewing(namaFile.Text);
    this.DataContext = vm;
  }
}
```

Kode Sumber A. 3 *Button Discovery*

- Kode Sumber Fungsi *Button Get Duration*

```
private void Button_Click_4(object sender,
RoutedEventArgs e)
{
    if (namaFile.Text == "")
    {
      MessageBoxResult result = MessageBox.Show("There is
      no file to generate duration");
    }
    else
    {
      Optimation window = new Optimation(namaFile.Text);
      window.ShowDialog();
    }
}
```

Kode Sumber A. 4 *Button Get Duration*

- Kode Sumber Fungsi *Button Save Discovery As* Excel

```
private void Button_Click_2(object sender,
RoutedEventArgs e)
{
  if (namaFile.Text == "")
  {
    MessageBoxResult result =MessageBox.Show("There
    is no file to save");
  }
  else
  {
    SaveFileDialog saveFileDialog = new
    SaveFileDialog();
    saveFileDialog.Filter = "Excel Files
    (*.xlsx)|*.xlsx";
  if (saveFileDialog.ShowDialog() == true)
  {
    vm.saveAs(namaFile.Text,
    saveFileDialog.FileName);
  }
  }
}
```

Kode Sumber A. 5 *Button Save Discovery As Excel*

2. Kode Sumber Halaman Optimasi

- Kode Sumber Xaml

```
<Window x:Class="WpfApplication2.Optimation"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/pres
entation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="TA_5111100106" Height="344" Width="679">
    <Grid Background="Black">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="2*" />
            <ColumnDefinition Width="2*" />
        </Grid.ColumnDefinitions>
         <Button Content="Calculate Duration"
         HorizontalAlignment="Stretch"
         Margin="10,10,10,0" VerticalAlignment="Top"
         Width="auto"
         RenderTransformOrigin="0.856,0.272"
         Click="Button_Click_1" Background="Black"
         BorderBrush="Gray" Foreground="White"
         Grid.Column="0"/>
         <DataGrid Height="auto"
         HorizontalAlignment="Stretch"
         Margin="10,37,8,37" Name="dataGrid1"
         VerticalAlignment="Stretch" Width="auto"
         Background="Gray" Grid.Column="0">
         </DataGrid>
```

```
        <Button Content="Save Duration As Excel"
        HorizontalAlignment="Right" Margin="0,0,8,8"
        VerticalAlignment="Bottom" Width="139"
        Click="Button Click 2" Background="Black"
        BorderBrush="Gray" Foreground="White"/>
        <TextBox Name="mathText"
        HorizontalAlignment="Stretch" Height="auto"
        Margin="14,37,10,10" TextWrapping="Wrap"
        VerticalAlignment="Stretch" Width="auto"
        Background="Gray" Grid.Column="1"/>
        <Button Content="Optimal Solution for
        Additional Cost and Project Duration"
        HorizontalAlignment="Stretch"
        Margin="14,10,10,0" VerticalAlignment="Top"
        Width="auto" Click="Button_Click_5"
        Background="Black" BorderBrush="Gray"
        Foreground="White" Grid.Column="1"/>
    </Grid>
</Window>
```

Kode Sumber A. 6 *Xaml* Optimasi

- Kode Sumber *Button Calculate Duration*

```
private void Button_Click_1(object sender,
RoutedEventArgs e)
{
  vm.clear();
  vm.GetDuration(namaFile);
  List<ActivityOptimisation> opData = new
  List<ActivityOptimisation>();
  ActivityOptimisation check = new
  ActivityOptimisation() { Activity = ""};
  foreach (string key in vm.GetnormalDuration().Keys)
  {
    if (!(check.Activity.Contains(key)))
    {
       check = (new ActivityOptimisation() { Activity =
       key,
       NormalDuration = vm.GetnormalDuration()[key],
       CrashDuration =
        vm.GetcrashDuration().First(kvp =>
        kvp.Key.Equals(key)).Value,
                                  NormalCost =
        vm.GetnormalCost().First(kvp =>
        kvp.Key.Equals(key)).Value,
                                  CrashCost =
        vm.GetCrashcost().First(kvp =>
        kvp.Key.Equals(key)).Value,
```

```
                                    TimeSlope =
            vm.GettimeSlope().First(kvp =>
            kvp.Key.Equals(key)).Value,
                                    CostSlope =
            vm.GetcostSlope().First(kvp =>
            kvp.Key.Equals(key)).Value });
             opData.Add(check);
        }
      }
      this.dataGrid1.ItemsSource = opData;
}
```

Kode Sumber A. 7 *Button Calculate Duration*

- Kode Sumber *Button Save Duration As Excel*

```
private void Button_Click_2(object sender,
RoutedEventArgs e)
{
    SaveFileDialog saveFileDialog1 = new
    SaveFileDialog();
    saveFileDialog1.Filter = "Excel Files
    (*.xlsx)|*.xlsx";
    if (saveFileDialog1.ShowDialog() == true)
    {
        vm.SaveDurationAsExcel(saveFileDialog1.FileName);
    }
}
```

Kode Sumber A. 8 *Button Save Duration As Excel*

- Kode Sumber *Button Optimal Solution for Additional Cost and Project Duration*

```
private void Button_Click_5(object sender,
RoutedEventArgs e)
{
        vm.clear();
        vm.GenerateMathModel(namaFile);
        mathText.Text = "Duration before optimization :";
        mathText.Text += vm.GetMakespan().ToString();
        mathText.Text +=
        vm.SolveMathModel().After("===Solution
        Details===").Replace("YFINISH", "Project Duration
        Time").Replace("AdditionalCost", "Additional Cost
        Project").Replace("X", "Crash Time Activity ");
}
```

Kode Sumber A. 9 *Button Optimal Solution for Additional Cost and Project Duration*

DAFTAR PUSTAKA

(2015, Maret 20). (Wikipedia) Retrieved 2 April, 2015, from http://en.wikipedia.org/wiki/Linear_programming

Cnude, S. (2014). Improving the quality of the Heuristics Miner in ProM 6.2.

Elmabrouk, O. M. (2011). A Linear Programming Technique for the Optimization of the Activities in Maintenance Projects. *International Journal of Engineering & Technology IJET-IJENS, 11*, 24-29.

Goedertier, S., De Weerdt, J., Martens, D., Vanthienen, J., & Baesens, B. (2011). Process Discovery in Event Log: An Application in the Telecom Industry. *Applied Soft Computing, 11*(2), 1697-1710.

Larson, W. E., & Gray, F. C. (2006). In *Project Management The Managerial Process Fitfth Edition* (pp. 171-180). Oregon State University: McGraw-Hill Irwin.

Loreto, D. A. (2012). *Sampling and Abstract Interpretation for Tackling Noise in Process Discovery.* Barcelona: UNIVERSITAT POLITÈCNICA DE CATALUNYA.

Prawira, B. (2014, Agustus 12). *Pixelbali*. (Pixelbali) Retrieved April 2, 2015, from http://pixelbali.com/informasi-teknologi/critical-path-method.html

Sarno, R., Ginardi, H., Pamungkas, E. W., & Sunaryono, D. (2013). Clustering of ERP Business Process Fragments. *Proceeding IEEE International conference on computer, control, informatics, and its applications*, 319-324.

Sarno, R., Indita, P. L., Ginadi, H., Sunaryono, D., & Mukhlash, I. (2013). Decision Mining for Multi Choice *Workflow* Patterns. *International Conference on Computer, Control, Informatics ad Its Applications*, 337-342.

Taha, H. A. (n.d.). Operations research: an introduction. In *Operations research: an introduction* (pp. 12-20). New Jersey: Pearson.

van der Aalst, W. (2011). In *Process Mining - Discovery, Conformance and Enhancement of Business Processes* (pp. 95-125). Netherlands: Springer.

van der Aalst, W. (2013, Oktober). *Process Mining: Beyond Business Intelligence*. Retrieved Januari 2015, 21, from www.processmining.org

van der Aalst, W., Adriansyah, A., & van Dongen, B. (2011). Causal Nets: A Modeling Language Tailored Towards Process Discovery. *In J.P. Katoen and B. Koenig, editors, 22nd International Conference on Concurrency Theory (CONCUR 2011)*, 28-42.

van der Aalst, W., Schonenberg, M., & Song, M. (2011). Time Prediction Based on Process Mining. *Information System Sciencedirect*, 450-475.

Wang, J., He, T., Wen, L., Wu, N., ter Hofstede, A., & Su, J. (2010). A behavioral similarity measure between labeled Petri nets based on principal transition sequences.

Weber, P. (2013). Principled Approach to Mining From Noisy Log. *IEEE Symposium on Computational Intelligence and Data Mining*.

Weijters, A., van der Aalst, W., & de Medeiros, A. (2006). Process mining with the Heuristics Miner-algorithm. *BETA Working Paper Series, WP 166, Eindhoven University of Technology*.

Wen, L., Aalst, W. v., Jianmin, W., & Sun, J. (2012). Mining Process Models with Non-Free-Choice Construct. *School of Software Tsinghua University,100084, Beijing,China*.

Weske, M. (2011). Business Process Management- Concepts, Languages, Architectures. *Springer*, 128-135.

Wicaksono, S., Atastina, I., & Kurniati, A. P. (2014). Evaluasi Proses Bisnis ERP dengan Menggunakan Process Mining (Studi Kasus : Goods Receipt (GR) Lotte Mart Bandung). *Informatika Telkom University*.

Zeng, Q., Sun, S. X., Duan, H., Liu, C., & Wang, H. (2013). Cross-organizational collaborative *workflow* mining from a multi-source log. *Sciencedirect*, 1280-1301.

BIODATA PENULIS

Fitrianing Haryadita, lahir di Klaten pada tanggal 17 April 1993. Penulis menempuh pendidikan mulai dari SDN Ponggok (1999-2005), SMPN 2 Klaten (2005-2008), SMAN 1 Klaten (2008-2011) dan S1 Teknik Informatika ITS (2011-2015).

Selama masa kuliah, penulis aktif dalam organisasi yang ada di lingkungan kampus ITS yaitu Himpunan Mahasiswa Teknik Computer-Informatika(HMTC) dan Badan Eksekutif Mahasiswa Fakultas Teknologi Informasi(BEM FTIf). Penulis dapat dihubungi melalui *email*: fitrianing11@mhs.if.its.ac.id.