TUGAS AKHIR - KI1502

PROCESS DISCOVERY UNTUK STREAMING EVENT LOG MENGGUNAKAN MODEL MARKOV TERSEMBUNYI

KELLY ROSSA SUNGKONO
NRP 5112 100 199

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

Dosen Pembimbing II
Abdul Munif, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

FINAL PROJECT - KI1502

PROCESS DISCOVERY FOR STREAMING EVENT LOG USING HIDDEN MARKOV MODEL

KELLY ROSSA SUNGKONO
NRP 5112 100 199

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

Supervisor II
Abdul Munif, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2016

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

PROCESS DISCOVERY UNTUK STREAMING EVENT LOG MENGGUNAKAN MODEL MARKOV TERSEMBUNYI

TUGAS AKHIR

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

Oleh
KELLY ROSSA SUNGKONO
NRP. 5112 100 199

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

ABDUL MUNIF, S.Kom., M.Sc. (pembimbing 2)
NIP: 19860823 201504 1 004

SURABAYA
JUNI, 2016

[Halaman ini sengaja dikosongkan]

PROCESS DISCOVERY UNTUK STREAMING EVENT LOG MENGGUNAKAN MODEL MARKOV TERSEMBUNYI

Nama : **Kelly Rossa Sungkono**
NRP : **5112100199**
Jurusan : **Teknik Informatika – FTIf ITS**
Dosen Pembimbing I : **Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Dosen Pembimbing II : **Abdul Munif, S.Kom., M.Sc.**

Abstrak

Process discovery adalah teknik penggalian model proses dari rangkaian aktivitas yang tercatat dalam event log. Saat ini, sistem informasi menghasilkan streaming event log dimana event log dicatat sesuai waktu proses yang terjadi. Online Heuristic Miner adalah algoritma process discovery yang mampu menghasilkan model proses dari streaming event log. Kelemahan dari algoritma Online Heuristic Miner adalah ketidakmampuan mengatasi incomplete trace pada streaming event log. Incomplete trace adalah rangkaian aktivitas pada event log yang terpotong di bagian awal ataupun di bagian akhir. Incomplete trace mengakibatkan proses secara utuh tidak dapat ditampilkan dalam model proses sehingga incomplete trace disebut sebagai noise.

Algoritma yang memanfaatkan Model Markov Tersembunyi digunakan untuk membentuk model proses yang dapat menangani incomplete trace pada streaming event log. Model Markov Tersembunyi dipilih karena model ini banyak digunakan untuk meramalkan data sehingga berguna dalam memperbaiki incomplete trace. Algoritma yang memanfaatkan Model Markov Tersembunyi tersebut terdiri atas gabungan dari metode pembentukan model proses serta metode yang dimodifikasi. Metode yang dimodifikasi adalah metode Baum-Welch, Backward serta Viterbi, dimana pemodifikasian

diperuntukkan untuk mengelompokkan observer pada Model Markov Tersembunyi dan menanggulangi aktivitas yang belum tercatat dalam Model Markov Tersembunyi.

Uji coba dilakukan dengan menggunakan tiga periode process discovery, yaitu setiap 5 detik, setiap 10 detik dan setiap 15 detik. Hasil uji coba menunjukkan bahwa algoritma yang memanfaatkan Model Markov Tersembunyi mampu memperbaiki incomplete trace sehingga tidak ada aktivitas yang hilang dalam model proses. Hasil uji coba juga menunjukkan kualitas model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi lebih baik dibandingkan kualitas model proses dari algoritma Online Heuristic Miner.

Kata kunci: *Incomplete Trace, Kualitas Model Proses, Metode Baum-Welch, Metode Viterbi, Metode Backward, Model Markov Tersembunyi, Process discovery, Streaming event log.*

PROCESS DISCOVERY FOR STREAMING EVENT LOG USING HIDDEN MARKOV MODEL

Student Name	**: Kelly Rossa Sungkono**
NRP	**: 5112100199**
Major	**: Teknik Informatika – FTIf ITS**
Supervisor I	**: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc.,Ph.D.**
Supervisor II	**: Abdul Munif, S.Kom., M.Sc.**

Abstract

Process discovery is a technique for obtaining process model from sequences of activities recorded in the event log. Nowadays, information systems produce a streaming event log wherein it is recorded according to the execution time. Online Heuristic Miner is an algorithm of process discovery which capable of obtaining process models from the streaming event log. Disadvantage using an Online Heuristic Miner algorithm is the inability to overcome the incomplete trace on the streaming event log. The incomplete trace is a truncated trace, either clipped off at the beginning or clipped off at the end. The incomplete trace make a whole process cannot be displayed in the process model, so incomplete trace is noise in process discovery.

An algorithm utilizing Hidden Markov Model is used to obtain a model process that can handle incomplete traces on a streaming event log. Hidden Markov Model is chosen because this model is widely used to predict the data which is useful in recovering incomplete trace. The algorithm utilizing Hidden Markov Model consists of a combination of methods of model process and modified methods. The modified methods are Baum-Welch, Backward and Viterbi methods, where the modification is intended to classify observer on Hidden Markov Model and combat activities that have not been recorded in the Hidden Markov Model

The experiments are using three period of discovery process, i.e. every 5 seconds, every 10 seconds and every 15 seconds. Experimental results show that the algorithm utilizing Hidden Markov Model is capable of recovering incomplete trace, so all activities are displayed in the process model. The experiment results also showed the quality of the process models obtaining by algorithm utilizing Hidden Markov Model is better than those obtaining algorithms Online Heuristic Miner.

Keywords: *Backward Method, Baum-Welch Method, Hidden Markov Model, Incomplete Trace, Invisible Task, Process discovery, Streaming event log, Quality of Process Model, Viterbi Method.*

KATA PENGANTAR

Segala puji syukur penulis kepada Tuhan Yesus Kristus atas penyertaan-Nya, sehingga tugas akhir berjudul "Process discovery untuk Streaming Event Log menggunakan Model Markov Tersembunyi" ini dapat selesai sesuai dengan waktu yang telah ditentukan.

Pengerjaan tugas akhir ini menjadi sebuah sarana untuk penulis memperdalam ilmu yang telah didapatkan di 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. Mama, papa dan keluarga yang selalu memberikan dukungan berupa doa dan nutrisi selama proses pengerjaan Tugas Akhir.
2. Bapak Riyanarto Sarno dan Bapak Abdul Munif selaku dosen pembimbing yang telah bersedia meluangkan waktu selama proses pengerjaan Tugas Akhir.
3. Bapak dan Ibu dosen Jurusan Teknik Informatika ITS yang banyak memberikan ilmu dan bimbingan bagi penulis.
4. Seluruh staf dan karyawan FTIf ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
5. Teman-teman 'Rantau' dan TC Angkatan 2012 yang selalu mendukung selama proses pengerjaan tugas akhir.
6. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak sekali kekurangan. Dengan kerendahan hati, penulis memohon maaf sebesar-besarnya atas kekuranga tersebut.

Surabaya, Juni 2016

[Halaman yang dikosongkan]

DAFTAR ISI

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

DAFTAR SIMBOL

δ	:	nilai dari metode Viterbi.
λ	:	Model Markov Tersembunyi.
π	:	vektor probabilitas *state awal*.
A	:	matrik probabilitas transisi *state*.
B	:	matrik probabilitas *observer*.
O	:	*observer*.
q_t	:	*state* pada waktu=t.
S_i	:	*state i*.
seq_stream	:	rangkaian aktivitas asli dari *streaming event log*.
seq_repair	:	rangkaian aktivitas hasil perbaikan dari *streaming event log*.
t	:	pada rumus PM, t berarti jumlah *trace* yang muncul pada *event log*. Untuk yang lain, t berarti waktu sekarang.
X_0	:	aktivitas awal pada *trace* di *event log*.
X_T	:	aktivitas akhir pada *trace* di *event log*.

[Halaman ini sengaja dikosongkan]

DAFTAR SINGKATAN

CPM	:	*Continous Parsing Measure*.
HMM	:	Model Markov Tersembunyi (*Hidden Markov Model*).
PM	:	*Parsing Measure*.
UML	:	*Unified Modelling Language*.

[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 pengerjaan Tugas Akhir, serta sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Process discovery adalah teknik penggalian model proses bisnis dari beberapa rangkaian aktivitas yang tercatat pada *event log* [1]. Model proses bisnis memiliki dua manfaat utama, yaitu sebagai alat bantu untuk menjelaskan proses yang terjadi pada sistem serta sebagai modal utama dalam menganalisa permasalahan terkait pengaturan rangkaian aktivitas.

Faktanya, sistem infomasi saat ini menghasilkan *streaming event log* dimana keseluruhan *event log* tidak dapat tersimpan di dalam sistem [2]. Selain itu, adanya kemungkinan pertambahan aktivitas di tengah-tengah proses sehingga model proses diharuskan mampu menampilkan pertambahan aktivitas tersebut.

Online Heuristic Miner adalah algoritma *process discovery* untuk *streaming event log* dimana algoritma ini mampu beradaptasi dengan pertambahan aktivitas di tengah proses [2]. Kelemahan dari *Online Heuristic Miner* adalah kemunculan *incomplete trace* pada *streaming event log*. *Incomplete trace* adalah rangkaian aktivitas (*trace*) pada *event log* yang terpotong di bagian awal ataupun di bagian akhir *trace*. *Incomplete trace* dapat mengakibatkan model proses bisnis tidak akurat dikarenakan adanya aktivitas yang tidak terekam pada saat *process discovery*.

Oleh karena itu, algoritma yang memanfaatkan Model Markov Tersembunyi digunakan untuk membangun model proses yang mampu mengatasi kemunculan *incomplete trace*. Model Markov Tersembunyi telah digunakan di berbagai bidang untuk peramalan, seperti *speech recognition* [3], *gene prediction* [4]

ataupun *stock market forecasting* [5]. Akan tetapi, Model Markov Tersembunyi belum pernah dimanfaatkan dalam *process discovery*.

Algoritma yang memanfaatkan Model Markov Tersembunyi terdiri dari metode pembentukan model proses serta metode yang dimodifikasi. Metode yang dimodifikasi adalah metode Baum-Welch [6], metode Viterbi [6] dan metode *Backward* [6]. Metode Baum-Welch digunakan untuk menentukan probabilitas pada Model Markov Tersembunyi sedangkan metode Viterbi dan *Backward* digunakan untuk memperbaiki *incomplete trace*. Metode Viterbi digunakan apabila *incomplete trace* terjadi karena terpotongnya aktivitas di bagian akhir dan metode *Backward* digunakan apabila *incomplete trace* terjadi karena terpotongnya aktivitas di bagian awal. Pemodifikasian metode Viterbi maupun metode *Backward* adalah penambahan aturan untuk menanggulangi aktivitas yang belum termasuk dalam Model Markov Tersembunyi dan pencarian *observer* dari *state* hasil Viterbi dan *Backward*. Pemodifikasian metode Baum-Welch adalah pengaturan kelompok *observer* terhadap *state* Model Markov Tersembunyi. Sedangkan, metode pembentukan model proses digunakan untuk menentukan relasi antar aktivitas sebagai bahan pembentukan model proses yang mampu beradaptasi dengan pertambahan aktivitas di tengah-tengah proses.

Hasil yang diharapkan dari Tugas Akhir ini adalah algoritma yang memanfaatkan Model Markov Tersembunyi mampu memperbaiki *incomplete trace* sehingga menghasilkan model proses yang tepat serta memiliki kualitas yang baik. Kualitas model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi akan dibandingkan dengan algoritma *Online Heuristic Miner*. Kualitas akan diukur berdasarkan 4 sisi yaitu *fitness*, presisi, generalisasi dan *simplicity*.

1.2 Rumusan Permasalahan

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

1. Bagaimana cara algoritma yang memanfaatkan Model Markov Tersembunyi dalam memperbaiki *incomplete trace* pada *streaming event log*?
2. Bagaimana cara algoritma yang memanfaatkan Model Markov Tersembunyi dalam menggali proses (*process discovery*) dari *streaming event log*?
3. Bagaimana hasil evaluasi model proses yang dibentuk oleh algoritma yang memanfaatkan Model Markov Tesembunyi dari sisi *fitness*, presisi, generalisasi dan *simplicity?*

1.3 Batasan Permasalahan

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

1. Data *event log* yang digunakan terdiri dari 100 case.
2. Aktivitas pada *streaming event log* yang digunakan sebagai masukan untuk *process discovery* minimal adalah 5 aktivitas.
3. Setiap aktivitas pada *event log* memiliki catatan waktu mulai pemrosesan aktivitas (*start stamp*) dan catatan waktu selesai pemrosesan aktivitas (*end stamp*).

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah menunjukkan bahwa algoritma yang memanfaatkan Model Markov Tersembunyi dapat memperbaiki *incomplete trace* serta menghasilkan model proses dari *streaming event log*.

1.5 Manfaat

Manfaat dari Tugas Akhir ini adalah sebagai berikut:

1. **Manfaat Keilmuan**
   Manfaat yang dihasilkan dari pengerjaan Tugas Akhir ini adalah terbentuknya suatu algoritma yang dapat menghasilkan model proses bisnis dari *streaming event log* pada suatu perusahaan dengan memanfaatkan Model Markov Tersembunyi. Model proses yang dihasilkan mampu beradaptasi dengan *incomplete trace* dan perkembangan proses.
2. **Manfaat Praktis**
   Process discovery memiliki manfaat, yaitu mendapatkan model proses sebenarnya yang meliputi relasi berupa *sequence*, *XOR, AND* dan *OR* dan perkembangan proses dari *streaming event log*. Model proses bisnis yang dihasilkan mampu membantu perusahaan dalam menganalisa proses bisnis yang terjadi. Evaluasi kualitas model proses akan diukur dari sisi *fitness*, presisi, generalisasi dan *simplicity*.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu sebagai berikut:

1. **Studi literatur**
   Dalam pembuatan Tugas Akhir ini telah dipelajari tentang hal-hal yang dibutuhkan sebagai ilmu penunjang dalam penyelesaiannya. Ilmu penunjang utama pada Tugas Akhir ini adalah struktur utama Model Markov Tersembunyi, metode Baum-Welch, metode *Backward,* metode Viterbi, metode *thread* pada Python serta pendekatan *periodic reset*. Selain itu, terdapat literatur lain yang menunjang proses penyelesaian Tugas Akhir ini.

2. **Pemodifikasian Metode**
   Pada tahap ini penulis menjabarkan cara untuk memecahkan masalah dimana permasalahan telah dijabarkan pada bagian rumusan masalah. Selain itu penulis juga menjabarkan tentang penambahan metode yang memanfaatkan Model Markov Tersembunyi 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 *process discovery* dari file *streaming event log*.
   b. Memperbaiki *incomplete trace* pada *streaming event log* sebelum dilakukan *process discovery*.
   c. Menghasilkan model proses bisnis sebagai hasil akhir dari *process discovery*.
   d. Melakukan perhitungan *fitness,* presisi, generalisasi dan *simplicity* dari model proses bisnis yang dihasilkan.
4. **Implementasi**
   Pada tahap ini dilakukan pembuatan perangkat lunak yang merupakan implementasi dari rancangan pada proses *discovery* untuk *streaming event log* menggunakan algoritma yang memanfaatkan Model Markov Tersembunyi.
5. **Pengujian dan evaluasi**
   Pada tahap ini dilakukan pengujian terhadap elemen perangkat lunak dengan menggunakan data uji yang telah dipersiapkan. Pengujian dan evaluasi perangkat lunak dilakukan untuk mengevaluasi jalannya perangkat lunak, mengevaluasi fitur utama, mengevaluasi fitur-fitur tambahan, mencari kesalahan yang timbul pada saat perangkat lunak aktif, dan mengadakan perbaikan jika ada kekurangan. Tahapan-tahapan dari pengujian adalah sebagai berikut:

a. Konsistensi model proses untuk setiap periode *process discovery*.
b. Hasil *discovery* dalam hal *fitness,* presisi, generalisasi dan *simplicity* untuk mengukur kualitas algoritma yang digunakan.

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 disusun dengan sistematika penulisan sebagai berikut:

Bab I Pendahuluan

Bab ini berisi latar belakang, 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 yang menjadi dasar dari pembuatan Tugas Akhir ini.

Bab III 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 IV Implementasi

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

Bab V 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 VI 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 lampiran yang digunakan untuk menjabarkan hasil pengujian algoritma yang diusulkan.

[Halaman ini sengaja dikosongkan]

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir.

2.1 *Process Discovery*

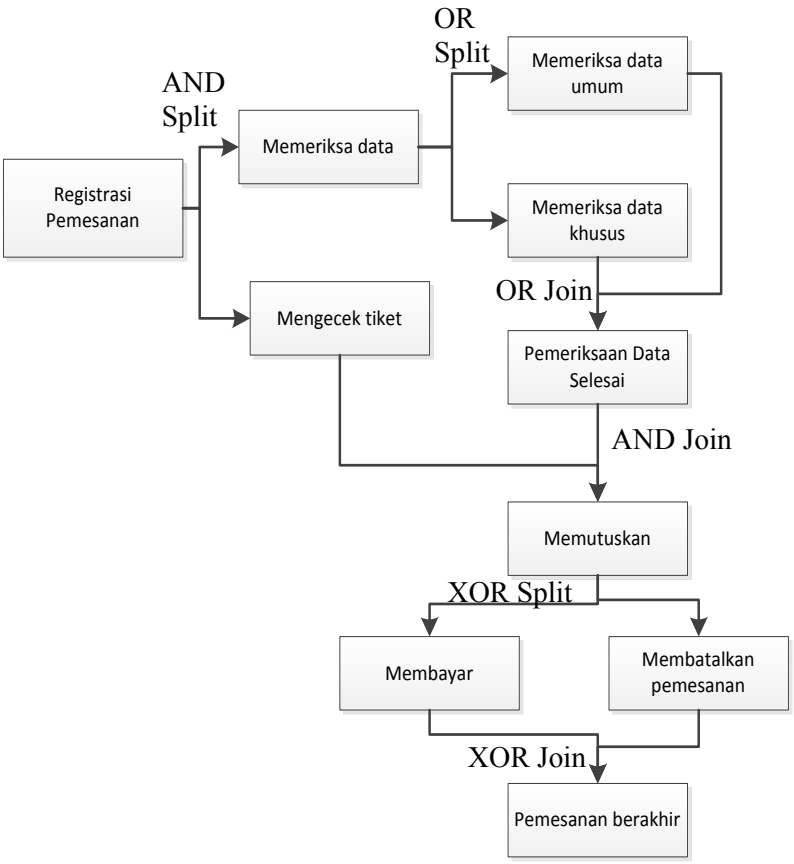
Proses mining adalah suatu ilmu yang menggabungkan antara pembelajaran mesin serta *data mining* pada satu sisi dan memodelkan serta menganalisa proses pada sisi yang lain. Ide dasar dari *process mining* adalah menemukan, memantau serta meningkatkan proses sebenarnya dengan mengambil pengetahuan dari *event log* yang ada pada suatu sistem [1].

Proses mining dibagi menjadi tiga bagian, salah satunya adalah *process discovery*. *Process discovery* adalah suatu teknik yang memanfaatkan *event log* dalam penentuan proses model tanpa menggunakan informasi-informasi lainnya. *Process discovery* yang ada diantaranya algoritma *Alpha#* [7] serta *Heuristics Miner* [8].

Proses model yang benar adalah proses model yang mampu menampilkan konkurensi serta *control-flow pattern* [1]. Konkurensi berarti aktivitas pada proses model tidak ditampilkan secara redundan (tidak ada aktivitas yang sama ditampilkan pada proses model) [1]. *Control-flow pattern* adalah penanda relasi antar aktivitas. Terdapat empat relasi antar aktivitas yaitu relasi *sequence*, relasi XOR, relasi AND and relasi OR [1].

Relasi *sequence* adalah relasi yang menghubungkan satu aktivitas dengan satu aktivitas lainnya. Relasi XOR adalah relasi yang hanya memperbolehkan salah satu aktivitas dalam relasi tersebut terjadi di suatu proses. Relasi OR adalah relasi yang memperbolehkan beberapa aktivitas dalam relasi tersebut terjadi di suatu proses. Sedangkan relasi AND adalah relasi yang yang mengharuskan seluruh aktivitas dalam relasi tersebut terjadi di suatu proses. Apabila aktivitas pilihan pada relasi XOR, OR dan AND terjadi setelah aktivitas yang berhubungan dengannya, maka *control-flow pattern* dari relasi tersebut adalah nama relasi dengan

penambahan kata “Split” dibelakang nama relasi [9]. Sedangkan, apabila aktivitas pilihan pada relasi XOR, OR dan AND terjadi sebelum aktivitas yang berhubungan dengannya, maka *control-flow pattern* dari relasi tersebut adalah nama relasi dengan penambahan kata “Join” di belakang nama relasi [9]. Gambar 2.1 merupakan contoh penggunaan relasi pada proses model.



Gambar 2.1 Contoh Model Proses

2.2 *Streaming Event Log*

Event log merupakan kumpulan proses bisnis yang mencatat setiap kejadian yang berjalan. *Event log* terdiri dari kumpulan proses tunggal, yang biasa disebut sebagai *case* [1]. Setiap *case* menyimpan beberapa *event* dimana setiap *event* menunjukkan sebuah aktivitas yang berjalan. Terdapat beberapa informasi tambahan yang terkait dengan setiap *event* [1]. Salah satu informasi tersebut adalah *timestamp*. *Timestamp* adalah waktu terjadinya suatu kejadian. Definisi dari *event log* dapat dilihat pada Gambar 2.2.

Definisi *Event log*

Event log terdiri dari rangkaian aktivitas (A) dan waktu (TD) yang didefinisikan dengan $L_{A,TD} = (E, C, \alpha, \gamma, \beta, \succ)$ dimana:

- E adalah kumpulan kejadian.
- C adalah kumpulan *case*.
- $\alpha: E \rightarrow A$ adalah fungsi yang menghubungkan setiap kejadian dengan sebuah aktifitas.
- $\gamma: E \rightarrow TD$ adalah fungsi yang menghubungkan setiap kejadian dengan sebuah waktu kejadian (*timestamp*).
- $\beta: E \rightarrow C$ adalah fungsi surjektif yang menghubungkan setiap kejadian dengan sebuah *case*.
- $\succ \subseteq ExE$ adalah *relation succesion*, yang merupakan total suatu kejadian yang dilanjutkan dalam kejadian lain dan termasuk dalam E. $e_2 \succ e_1$ adalah notasi singkat untuk $(e_2, e_1) \in \succ$. Alur dari aktivitas yang dijalankan oleh *case* disebut sebagai *trace*.

Gambar 2.2 Definisi *Event Log*

Sedangkan *streaming event log* merupakan *event log* yang diobservasi satu per satu berdasarkan waktu dilaksanakannya kejadian pada *event log* tersebut [2]. Sehingga, *streaming event log* adalah *event log* yang tercatat secara *real-time*.

2.3 *Noise*

Noise merupakan suatu kejadian yang jarang terjadi sehingga tidak mewakili perilaku sesungguhnya pada *event log* [10]. *Noise*

dapat didefinisikan juga sebagai *outlier* (data di luar dari data sesungguhnya). Kemunculan *noise* sebagai data di *event log* dapat mengakibatkan penggambaran proses utama tidak sesuai.

Noise terdiri dari 3 jenis yang didasarkan pada cara pembentukannya [10]. Tiga jenis *noise* tersebut adalah:

1) Menghapus aktivitas awal dari *trace* aktivitas
   Pembentukan *noise* pada jenis ini adalah penghapusan aktivitas awal dari *trace* yang ada pada *event log*. Contoh dari kasus ini adalah *trace* ABCDE yang hanya terekam sebagai *trace* CDE. Aktivitas A dan B yang merupakan aktivitas awal pada contoh terhapus sehingga *trace* tersebut dianggap *noise*.
2) Menghapus aktivitas akhir dari *trace* aktivitas
   Pembentukan *noise* pada jenis ini adalah penghapusan aktivitas akhir dari *trace* yang ada pada *event log*. Contoh dari kasus ini adalah *trace* ABCDE yang hanya terekam sebagai *trace* ABC. Aktivitas D dan E yang merupakan aktivitas akhir pada contoh terhapus sehingga *trace* tersebut dianggap *noise*.
3) Menghapus bagian acak dari *trace* aktivitas
   Pembentukan *noise* pada jenis ini adalah penghapusan aktivitas secara acak dari *trace* yang ada pada *event log*. Aktivitas yang dihapus berupa keseluruhan *event log* atau salah satu aktivitas dari *trace* yang ada pada *event log*. Contoh dari kasus ini adalah *trace* ABCDE yang terekam sebagai *trace* ABDE. Aktivitas C pada contoh terhapus sehingga *trace* tersebut dianggap *noise*.

Incomplete trace termasuk dalam jenis pertama dan jenis kedua dalam *noise* yaitu penghapusan aktivitas awal atau penghapusan aktivitas akhir.

2.4 *Online Heuristic Miner*

Online Heuristic Miner adalah algoritma *process discovery* yang diperuntukkan untuk *streaming event log* [2]. Algoritma ini akan berjalan secara terus menerus, dimana setiap *event* yang diobservasi diterjemahkan dalam notasi matematika yaitu $e = (c_i, a_i, t_i)$ dimana c_i adalah *case*, a_i adalah aktivitas dan t_i adalah waktu ekseskusi aktivitas. Algoritma ini menyimpan data dalam

bentuk 3 Queue yaitu Queue untuk menyimpan jumlah aktivitas (Q_A), Queue untuk menyimpan aktivitas terakhir yang terjadi pada case tertentu (Q_C), dan Queue untuk menyimpan jumlah relasi antara aktivitas (Q_R). Sedangkan fw_A merupakan bobot jumlah aktivitas dan fw_R merupakan bobot jumlah relasi aktivitas. Pembentukan model (*process discovery*) digunakan dengan memanfaatkan algoritma *Heuristic Miner* menggunakan data yaitu Q_A dan Q_R. Adapun langkah-langkah algoritma dipaparkan pada Gambar 2.3 [2].

Algoritma *Online Heuristic Miner*	
Input : S event stream; $max_{Q_A}, max_{Q_C}, max_{Q_R}$ maximum memory sizes of queues Q_A, Q_C, and Q_R; fw_A, fw_R model policy;	
Langkah	**Penjelasan Langkah**
forever do	
$e \leftarrow observe(S)$	Mengobservasi sebuah *event* yang terdiri atas *case, aktivitas*, dan waktu ekseskusi aktivitas

if $\nexists(a,w) \in Q_A\, s.t.\, a = a_i$ then
if $size\ (Q_A) = max_{Q_A}$ then
$removeLast\ (Q_A)$
end
$w \leftarrow 0$
	Apabila muncul aktivitas baru dimana *queue* Q_A sudah penuh, maka entri terakhir pada Q_A dihapus.

else $w \leftarrow get(Q_A, a_i)$ end	Mengambil jumlah aktivitas a_i dan menghapus (a_i, w).
$insert(Q_A, (a_i, w))$ $Q_A \leftarrow fw_A(Q_A)$	Memasukkan (a_i, w) di awal *queue* Q_A dan mempebarui bobot jumlah aktivitas pada Q_A.
if $\exists(c,a) \in Q_C\, s.t.\, c = c_i$ then $a \leftarrow get(Q_c, c_i)$	Apabila *case* dan aktivitas terdapat pada *queue* Q_C, maka diambil aktivitas dari *case* tersebut dan dihapus (c_i, a).
if $\nexists(a_s, a_f, u) \in Q_R\, s.t. (a_s = a) \wedge (a_f = a_i)$ then	Apabila aktivitas awal a_s dan aktivitas selanjutnya a_f tidak terdapat pada *queue* Q_R dimana

if $size\ (Q_R) = max_{Q_R}$ then $removeLast\ (Q_R)$ end $u \leftarrow 0$	Q_R sudah penuh, maka entri terakhir pada Q_R dihapus.
else $u \leftarrow get(Q_R, a, a_i)$ end	Mengambil jumlah relasi $a \rightarrow a_i$ and menghapus *(a, a_i, u)*.
$insert(Q_R, (a, a_i, u))$ $Q_R \leftarrow fw_R(Q_R)$	Memasukkan (a, a_i, u) di awal *queue* Q_R dan mempebarui bobot jumlah relasi pada Q_R.
else if $size\ (Q_C) = max_{Q_C}$ then $removeLast\ (Q_C)$ end	Apabila *queue* Q_C penuh, maka menghapus entri terakhir Q_C.
$insert(Q_C, (c_i, a_i))$	Memasukkan (c, a_i) di awal *queue* Q_C
if *model* then *Heuristic Miner*(Q_A , Q_R) end end	Membentuk model dilakukan dengan algoritma *Heuristic Miner* yang melibatkan *queue* Q_A dan Q_R.

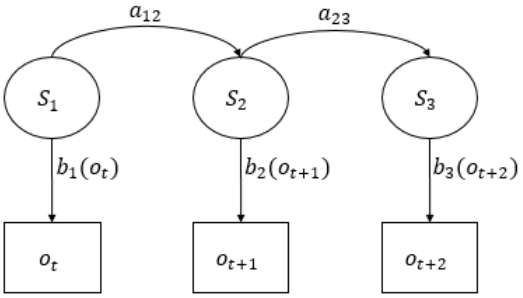
Gambar 2.3 Algoritma *Online Heuristic Miner*

2.5 Model Markov Tersembunyi

Model Markov Tersembunyi adalah kombinasi dari dua proses dimana salah satu prosesnya memiliki variabel tersembunyi yang disebut sebagai *state* dan penetapan variabel tersebut berdasarkan proses lain yang memproduksi rangkaian variabel random atau rangkaian *observer* [6]. Contoh dari Model Markov Tersembunyi dapat dilihat pada Gambar 2.4.

Model matematika dari Model Markov Tersembunyi adalah HMM=(S,V, π,A,B) [11] dimana S=$\{S_1, \dots, S_N\}$ merupakan kumpulan *state* tersembunyi, V=$\{v_1, \dots, v_M\}$ merupakan kumpulan *observer*, π merupakan vektor probabilitas *state* sebagai *state* awal,

A merupakan matrik probabilitas transisi *state*, dan B merupakan matrik probabilitas *observer*.



Gambar 2.4 Contoh Model Markov Tersembunyi

Probabilitas dari *state* yang diidentifikasi sebagai *state* awal ditampung pada vektor π. Rumus π dijabarkan pada Persamaan 2.1.

$$\pi = \{\pi_i\} = p\{q_1 = S_i\}, \quad 1 \le i \le N \tag{2.1}$$

Keterangan:

π_i : probabilitas yang menunjukkan bahwa model berada di *state i* saat waktu 0 atau saat awal proses.

$p\{q_1 = S_i\}$: himpunan nilai probabilitas *state i* (S_i) sebagai *state* pada waktu=0 (q_1).

N : jumlah *state* pada Model Markov Tersembunyi.

Matrik probabilitas transisi *state* dengan *state* lainnya dilambangkan dengan A. Rumus pembentukan matrik A dijabarkan pada Persamaan 2.2.

$$\mathrm{A} = \{a_{ij}\} = p\{q_{t+1} = S_j \mid q_t = S_i\}, \quad 1 \le i, j \le N \tag{2.2}$$

Keterangan pada Persamaan 2.2:

a_{ij} : probabilitas transisi *state i* ke *state j*.

$p\{q_{t+1} = S_j \mid q_t = S_i\}$: himpunan peluang kemunculan *state j* (S_j) sebagai *state* pada waktu *t+1* (q_{t+1}) dengan syarat *state i* (S_i) sebagai *state* pada waktu=t (q_t) telah terjadi.

N : jumlah *state* pada Model Markov Tersembunyi.

Sedangkan, matrik probabilitas *observer* terhadap *state* dilambangkan dengan B. Rumus pembentukan matrik B dijabarkan pada Persamaan 2.3.

$$B = \{b_i(k)\} = p\{o_t = v_k \mid q_t = S_i\}, \quad 1 \le i \le N, 1 \le k \le M \tag{2.3}$$

Keterangan:

$b_i(k)$: probabilitas *observer k* yang bergantung pada *state i*.

$p\{o_t = v_k \mid q_t = S_i\}$: himpunan peluang kemunculan *observer k* (v_k) sebagai *observer* pada waktu *t* (o_t) dengan syarat *state i* (S_i) sebagai *state* pada waktu *t* (q_t) telah terjadi.

M : jumlah *observer* pada Model Markov Tersembunyi.

N : jumlah *state* pada Model Markov Tersembunyi.

2.6 Metode *Forward*

Metode *Forward* adalah metode yang digunakan untuk menghitung probabilitas dari suatu rangkaian *observer* jika diketahui Model Markov Tersembunyi [6]. Persamaan 2.4 menunjukkan metode *Forward* secara matematis.

$$\alpha(t,i) = P(O_1,...,O_t, q_t = S_i) \quad (2.4)$$

Keterangan pada Persamaan 2.4:

$\alpha(t,i)$: nilai dari metode *Forward* yang berhubungan dengan *state i* pada waktu *t*.

$P(O_1,...,O_t, q_t = S_i)$: peluang rangkaian *observer* $O_1, \dots, O_t$ dan *state i* (S_i) pada waktu *t*.

Metode *Forward* terdiri atas tiga langkah yaitu langkah inisialisasi, langkah induksi dan langkah terminasi. Langkah inisialisasi dijelaskan pada Persamaan 2.5, langkah induksi dijelaskan pada Persamaan 2.6 dan langkah terminasi dijelaskan pada Persamaan 2.7.

$$\alpha(1,i) = \pi_i b_i(O_1) \quad (2.5)$$

$$\alpha(t+1,i) = \sum_{j=1}^{N} \alpha(t,j) a_{ji} b_i(O_{t+1}) \quad (2.6)$$

$$P(0) = \sum_{i=1}^{N} \alpha(T,i) \quad (2.7)$$

Keterangan:

$\alpha(1,i)$: nilai dari metode F*orward* pada waktu awal yang berhubungan dengan *state i*.

$\alpha(t+1,i)$: nilai dari metode *Forward* pada waktu selanjutnya (*t+1*) yang berhubungan dengan *state i*.

$\alpha(t,j)$: nilai dari metode *Forward* pada waktu sebelumnya yang berhubungan dengan *state j*.

$\alpha(T,i)$: nilai akhir dari metode *Forward* yang berhubungan dengan *state i*.

π_i : nilai probabilitas *state i* sebagai *state* awal pada Model Markov Tersembunyi.

a_{ji} : probabilitas transisi *state j* ke *state i*.

$b_i(O_{t+1})$: probabilitas *observer* pada waktu *t+1* terhadap *state i*.
N : jumlah dari *state* pada Model Markov Tersembunyi.
O_1 : *observer* awal pada rangkaian *observer*.
$P(0)$: peluang seluruh rangkaian *observer* terhadap Model Markov Tersembunyi menggunakan metode *Forward*.

2.7 Metode *Backward*

Metode *Backward* memiliki fungsi yang sama dengan metode *Forward* akan tetapi perhitungannya secara terbalik yaitu dari akhir ke awal [6]. Persamaan 2.8 menunjukkan metode *Backward* secara matematis.

$$\beta(t,i) = P(O_{t+1},...,O_T \mid q_t = S_i), \quad 1 \le t \le T\text{-}1 \tag{2.8}$$

Keterangan:
$\beta(t,i)$: nilai dari metode *Backward* yang berhubungan dengan *state i* pada waktu *t*.
$P(O_{t+1},...,O_T \mid q_t = S_i)$: peluang dari rangkaian *observer* $O_{t+1}, \dots, O_T$ diberikan *state i* pada waktu *t*.
T : waktu dari *observer* terakhir pada rangkaian *observer*.

Metode *Backward* terdiri atas dua langkah yaitu langkah inisialisasi dan langkah induksi. Langkah inisialisasi dijelaskan pada Persamaan 2.9 sedangkan langkah induksi dijelaskan pada Persamaan 2.10.

$$\beta(T,i) = 1 \tag{2.9}$$

$$\beta(t-1,i) = \sum_{j=1}^{N} a_{ij} b_j(O_t) \beta(t,j) \tag{2.10}$$

Keterangan pada Persamaan 2.9 dan 2.10:

$\beta(t,i)$: nilai dari metode *Backward* pada waktu akhir yang berhubungan dengan *state i*.

$\beta(t-1,i)$: nilai dari metode *backward* pada waktu sebelumnya (*t-1*) yang berhubungan dengan *state i*.

$\beta(t,j)$: nilai dari metode *backward* pada waktu setelahnya yang berhubungan dengan *state j*.

a_{ij} : nilai probabilitas transisi *state i* ke *state j*.

$b_j(O_t)$: nilai probabilitas *observer* pada waktu *t* terhadap *state j*.

N : jumlah *state* pada Model Markov Tersembunyi.

2.8 Metode Baum-Welch

Metode Baum-Welch adalah metode yang digunakan untuk menentukan Model Markov Tersembunyi yang memiliki probabilitas maksimal untuk memproduksi rangkaian *observer* yang telah diketahui [1]. Metode Baum-Welch memanfaatkan metode *Forward* (α) dan metode *Backward* (β) dalam perhitungannya. Persamaan 2.11 sampai dengan Persamaan 2.15 menunjukkan metode Baum-Welch secara matematis [11].

$$\xi_t(i,j) = \frac{\alpha(t,i)a_{ij}\beta(t+1,j)b_j(o_{t+1})}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha(t,i)a_{ij}\beta(t+1,j)b_j(o_{t+1})} \tag{2.11}$$

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i,j), \quad 1 \le i \le N, 1 \le t \le M \tag{2.12}$$

$$\overline{\pi}_i = \gamma_1(i), \quad 1 \le i \le N \tag{2.13}$$

$$\overline{a_{ij}} = \frac{\sum_{t=1}^{T} \xi_t(i,j)}{\sum_{t=1}^{T} \gamma_t(i)}, \quad 1 \le i, j \le N \tag{2.14}$$

$$\overline{b_i(k)} = \frac{\sum_{\substack{t=1 \\ o_t = O_k}}^{T} \gamma_t(i)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{2.15}$$

Keterangan pada Persamaan 2.11 sampai 2.15:

$\alpha(t,i)$	:	nilai *Forward* yang berhubungan dengan *state i* pada waktu *t*.
$\beta(t+1,j)$	:	nilai *Backward* yang berhubungan dengan *state j* pada waktu *t+1*.
$\xi_t(i,j)$	:	peluang proses pada saat waktu *t* berada pada *state i* dan pada saat waktu *t+1* berada pada *state j*.
$\gamma_1(i)$	:	peluang proses berada pada *state i* pada waktu awal.
$\gamma_t(i)$	:	peluang proses berada pada *state i* pada waktu *t*.
$\overline{\pi_i}$	:	hasil metode Baum-Welch yang merupakan himpunan probabilitas *state i* sebagai *state* awal Model Markov Tersembunyi.
$\overline{a_{ij}}$	:	hasil metode Baum-Welch yang merupakan himpunan probabilitas transisi *state i* ke *state j*.
a_{ij}	:	nilai probabilitas transisi *state i* ke *state j*.
$\overline{b_i(k)}$	:	hasil metode Baum-Welch yang merupakan himpunan probabilitas *observer k* yang bergantung pada *state i*.
$b_j(o_{t+1})$	:	nilai probabilitas *observer* yang berjalan pada waktu *t+1* terhadap *state j*.
M	:	jumlah *observer* Model Markov Tersembunyi.
N	:	jumlah *state* pada Model Markov Tersembunyi.

T : waktu *observer* terakhir pada rangkaian *observer*.

2.9 Metode Viterbi

Metode Viterbi digunakan untuk mencari rangkaian *state* yang memiliki probabilitas tertinggi $j^* = \{j_1^*, j_2^*, ..., j_t^*\}$ jika diketahui rangkaian *observer* $O = \{o_1, o_2, ..., o_t\}$ dan Model Markov Tersembunyi [11]. Persamaan 2.16 menunjukkan metode Viterbi secara matematis.

$$\delta_t(i) = \max_{q_1, q_2, ..., q_{t-1}} p\{q_1, q_2, ..., q_{t-1}, q_t = S_i, o_1, o_2, ..., o_{t-1} \mid \lambda\} \qquad (2.16)$$

Keterangan:

$\delta_t(i)$: nilai metode Viterbi untuk *state i* pada waktu *t*.

λ : Model Markov Tersembunyi.

o_{t-1} : *observer* yang berjalan pada waktu *t-1*.

q_t : nilai paling maksimal dari probabilitas *state* terhadap rangkaian *observer*.

S_i : *state i.*

Metode Viterbi dibagi ke dalam dua langkah utama yaitu langkah inisialisasi dan langkah induksi. Langkah inisialisasi dapat dilihat pada Persamaan 2.17 dan langkah induksi dapat dilihat pada Persamaan 2.18.

$$\delta_1(j) = \pi_j b_j(o_1), \quad 1 \le j \le N \qquad (2.17)$$

$$\delta_{t+1}(j) = b_j(o_{t+1}) \left[\max_{1 \le i \le N} \delta_t(i) a_{ij} \right] \qquad (2.18)$$

Keterangan:

$\delta_1(j)$: nilai metode Viterbi untuk *state j* pada waktu awal.

$\delta_{t+1}(j)$: nilai metode Viterbi untuk *state j* pada waktu *t+1*.

$\delta_t(i)$: nilai metode Viterbi untuk *state i* pada waktu *t*.

a_{ij} : probabilitas transisi *state i* ke *state j*.
$b_j(o_{t+1})$: nilai probabilitas *observer* pada waktu *t+1* yang bergantung pada *state j*.
N : jumlah *state* pada Model Markov Tersembunyi.

Dengan mengacu pada Persamaan 2.17 dan 2.18, rangkaian *state* probabilitas paling optimal $j^* = \{j_1^*, j_2^*, ..., j_t^*\}$ dapat ditemukan dengan menggunakan Persamaan 2.19.

$$j_t^* = \arg \max_{1 \le j \le N} \delta_t(j) \tag{2.19}$$

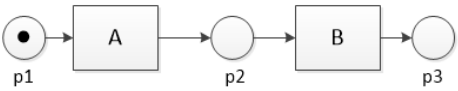
Keterangan:
$\delta_t(j)$: nilai Viterbi untuk *state j* pada waktu *t*.
j_t^* : *state* pada waktu *t* yang paling optimal.
N : jumlah *state* pada Model Markov Tersembunyi.

2.10 *Petri net*

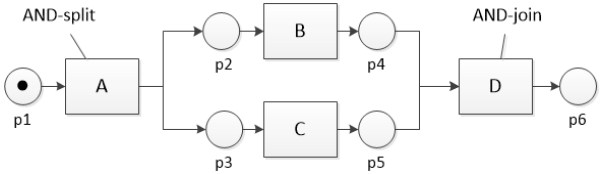
Petri net adalah salah satu bentuk model proses bisnis sebagai hasil dari *process discovery*. *Petri net* adalah grafik dua arah dengan dua yang terdiri atas dua tipe simpul, yaitu *place* dan transisi [12]. Setiap simpul dihubungkan oleh panah dimana simpul dengan tipe sama tidak boleh berhubungan. *Place* digambarkan dalam bentuk lingkaran sedangkan transisi digambarkan dalam bentuk persegi. Selain itu, *Petri net* juga memiliki *token* yang digunakan untuk mempresentasikan mengenai jalannya pengeksekusian aktivitas. Token digambarkan dalam bentuk bulatan hitam.

Tipe penggambaran relasi pada *Petri net* dibagi menjadi tiga, yaitu sekuensial, paralel dan kondisional [12]. Relasi paralel digambarkan dalam bentuk relasi AND, sedangkan relasi kondisional digambarkan dalam bentuk relasi *exclusive OR* atau disebut dengan relasi XOR [12]. Model perelasian pada *Petri net* akan digunakan sebagai acuan dalam pembuatan Model Markov

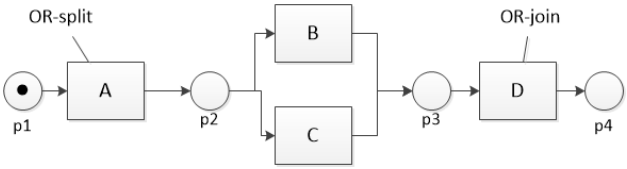
Tersembunyi. Bentuk dari ketiga tipe tersebut pada Gambar 2.5, 2.6 dan 2.7.



Gambar 2.5 Tipe sekuensial



Gambar 2.6 Tipe paralel



Gambar 2.7 Tipe kondisional

2.11 Pendekatan *Periodic Reset*

Pendekatan *periodic reset* adalah pendekatan yang digunakan untuk mengoleksi *event* pada *streaming event log* ke dalam memori sehingga dapat dianalisa oleh algoritma *process discovery* [13]. Cara pendekatan *periodic reset* adalah menyimpan *event* hingga mencapai titik maksimum. Pada *periodic reset*, *event* akan dihapus apabila sudah mencapai titik maksimum [13]. Titik maksimum dapat diartikan sebagai titik dimana tempat untuk menampung data telah penuh atau waktu yang telah diatur.

Kelebihan dan kekurangan dari pendekatan ini akan dipaparkan dalam Tabel 2.1 [13].

Tabel 2.1 Kelebihan dan Kekurangan *Periodic Reset*

Kelebihan	Kekurangan
Menghemat pemakaian memori.	Tidak ada data masa lampau yang dapat mempengaruhi akurat hasil dari *process discovery*.
Mempermudah algoritma *process discovery* untuk menganalisa *streaming event log.*	

Melihat kelebihan dari *periodic reset*, maka pendekatan ini juga digunakan dalam mengelola *streaming event log* pada Tugas Akhir ini. Cara yang dilakukan untuk mengatasi kekurangan *periodic reset* dengan menyimpan Model Markov Tersembunyi yang menampung probabilitas kemunculan data masa lampau.

2.12 *Confusion Matrix*

Confusion Matrix adalah matrik yang digunakan untuk mengukur keberhasilan dari suatu prediksi. *Confusion matrix* dibagi menjadi matrix 2x2 yang terdiri atas inputan sebagai berikut [1]:

- *tp* : jumlah dari *true positive* berupa data yang benar-benar diklasifikasikan positif.
- *fn* : jumlah dari *false negative* berupa data yang diprediksi negatif akan tetapi seharusnya diklasifikasikan positif.
- *fp* : jumlah dari *false positive* berupa data yang diprediksi positif akan tetapi seharusnya diklasifikasikan negatif.
- *tn* : jumlah dari *true negative* berupa data yang benar-benar diklasifikasikan negatif.

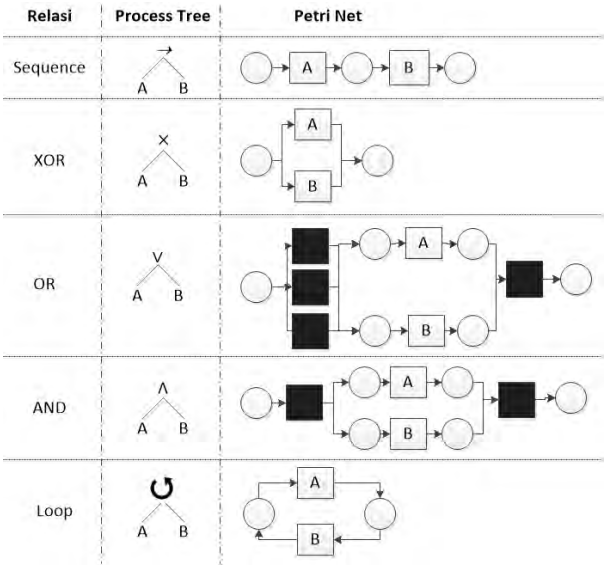
Pada kasus *process discovery*, kelas prediksi adalah *trace* yang terbentuk dari model proses hasil *process discovery* sedangkan kelas sebenarnya adalah *trace* di *event log* [1]. Nilai *true* atau *positif* (+) terjadi apabila terdapat *trace*, sebaliknya nilai *false* atau *negative* (-) terjadi apabila tidak terdapat *trace*. *Confusion matrix* untuk *process discovery* dapat dilihat pada Gambar 2.8.

		Kelas Prediksi (model proses)	
		+	-
Kelas sebenarnya (*event log*)	+	*tp*	*fn*
	-	*fp*	*tn*

Gambar 2.8 *Confusion Matrix*

2.13 *Process Tree*

Process Tree adalah model umum yang digunakan untuk mendukung kualitas dari model proses. Hal ini dikarenakan setiap proses dari *process tree* tidak mengandung anomali [14]. Anomali adalah data yang tidak tepat yang mempengaruhi hasil. Hasil ini dapat diartikan sebagai model proses pada Tugas Akhir ini.



Gambar 2.9 Notasi *Process Tree*

Suatu *process tree* memiliki *operator nodes* dan *leaf node*. *Operator node* melambangkan relasi antar anak cabang [14].

Operator node terdiri dari relasi *sequence* (→), relasi AND (∧), relasi OR (∨), relasi XOR (×) dan *loop* (↻) [14]. *Leaf node* pada *process tree* merupakan aktivitas pada model proses [14]. Urutan pengeksekusian *node* pada *process tree* dimulai dari cabang paling kiri ke cabang paling kanan serta cabang paling bawah ke cabang paling atas [14]. Gambar 2.9 pada halaman sebelumnya menunjukkan contoh *process tree* yang mendeskripsikan model proses *Petri net*.

2.14 Evaluasi *Fitness* pada Model Proses Bisnis

Fitness digunakan untuk mengukur kesesuaian antara *event log* dan model proses. Nilai *fitness* adalah berkisar antara 0-1, dimana sebuah model proses memiliki *fitness* baik atau bernilai 1 apabila semua *trace* di *event log* dapat diwakili oleh model proses. Sebaliknya, jika banyak *trace* di *event log* tidak dapat diwakili oleh model proses dari awal sampai akhir, maka model proses tersebut memiliki *fitness* yang buruk atau bernilai mendekati 0 [15].

Kualitas *fitness* (Q_f) dapat dihitung dengan merata-rata hasil dari dua rumus: *parsing measure* (PM) dan *continuous parsing measure* (CPM) [15]. *Parsing measure* (PM) didefinisikan pada Persamaan 2.20 [15]. Sedangkan, rumus untuk menghitung *continuous parsing measure* (CPM) dengan rincian yang lebih tinggi (yaitu mengidentifikasi aktivitas daripada *trace*) didefiniskan pada Persamaan 2.21 [15]. Rumus akhir *fitness* (Q_f) dapat dilihat pada Persamaan 2.22.

$$\mathrm{PM} = \frac{c}{t} \tag{2.20}$$

$$\mathrm{CPM} = \frac{1}{2}\frac{(e-m)}{e} + \frac{1}{2}\frac{(e-r)}{e} \tag{2.21}$$

$$Q_f = \frac{CPM+PM}{2} \tag{2.22}$$

Keterangan pada Persamaan 2.20 sampai 2.22:

c : jumlah *trace* dari *event log* yang ditampilkan pada model proses.

t : total *trace* dari *event log*.

m : jumlah aktivitas pada *event log* yang tidak diimplementasikan di model proses.

r : jumlah aktivitas yang ada pada *trace* di *event log* setelah aktivitas akhir yang digambarkan di model proses.

e : total aktivitas pada *event log*.

2.15 Evaluasi Presisi pada Model Proses Bisnis

Presisi merupakan salah satu aspek dalam mengevaluasi model proses sebagai keputusan akhir penentuan algoritma yang lebih baik.

Presisi menyatakan bahwa suatu model proses tidak seharusnya menunjukkan proses yang cenderung berbeda dengan proses yang terlihat pada *event log* [1]. Presisi berkaitan dengan notasi *overfitting* dalam konteks *data mining* [1]. Suatu model proses dikatakan *overfitting* apabila model proses tersebut sangat spesifik dan berpatokan penuh pada contoh proses di *event log* [1]. Nilai presisi berkisar antara 0-1 dimana semakin nilai presisi suatu model proses mendekati 1, maka semakin besar kecenderungan model tersebut dalam notasi *overfitting* [1].

Rumus presisi (Q_p) menggunakan *confusion matrix* didefinisikan pada Persamaan 2.23 [1].

$$Q_p = \frac{tp}{p'} \tag{2.23}$$

Keterangan:

tp : jumlah dari *true positive* (*trace* pada *event log* digambarkan di model proses).

p' : jumlah dari *true positive* dan *false negative* (seluruh *trace* yang digambarkan di model proses).

2.16 Evaluasi Generalisasi pada Model Proses Bisnis

Generalisasi menyatakan bahwa suatu model proses seharusnya menunjukkan generalisasi dari contoh proses yang terlihat pada *event log* [1]. Generalisasi berkaitan dengan notasi *underfitting* dalam konteks *data mining* [1]. Suatu model proses dikatakan *underfitting* apabila model proses tersebut juga menunjukkan proses yang cenderung berbeda dengan proses yang terlihat pada *event log* [1]. Nilai generalisasi berkisar antara 0-1 dimana semakin nilai generalisasi suatu model proses mendekati 1, maka semakin besar kecenderungan model tersebut dalam notasi *underfitting*. Perhitungan generalisasi suatu model proses memperhatikan frekuensi dari kemunculan *node* pada *process tree* berdasarkan *event log* [14].

Rumus generalisasi (Q_g) didefinisikan pada Persamaan 2.24 [14].

$$Q_g = 1 - \frac{\sum_{1}^{nt} \left(\sqrt{\#executions} \right)^{-1}}{\#nt} \tag{2.24}$$

Keterangan:

$\#nt$	:	jumlah *operator node* pada *process tree*.
nt	:	*operator node* yang diimplementasikan di *event log*.
$\#executions$	:	jumlah *operator node* yang diimplementasikan di *trace* pada *event log*.

Hasil maksimal dari perhitungan kualitas generalisasi terjadi apabila jumlah *trace* besar yaitu minimal adalah 100. Oleh karena itu, apabila jumlah *trace* kurang dari 100, maka dilakukan Persamaan 2.25 sebelum menjalankan rumus generalisasi.

$$\#executions = \frac{\#executions}{\mathrm{n}_{\mathrm{trace}}} \times 100 \tag{2.25}$$

Keterangan pada Persamaan 2.25:

$\#executions$: jumlah *operator node* yang diimplementasikan pada *trace* di *event log*.

n_{trace} : jumlah *trace* pada *event log*.

2.17 Evaluasi *Simplicity* pada Model Proses Bisnis

Simplicity menyatakan bahwa suatu model proses seharusnya dibuat sesederhana mungkin tanpa menghilangkan realisasi proses yang ditangkap dari *event log* [1].

Perhitungan *simplicity* suatu model proses dengan cara membandingkan ukuran *process tree* dari suatu model proses dengan aktivitas pada *event log* [14]. Nilai *similarity* berkisar antara 0-1. Apabila aktivitas yang muncul pada *process tree* hanya sekali dan semua aktivitas pada *event log* tergambar pada *process tree*, maka model tersebut memiliki *similarity* yang tinggi atau mendekati nilai 1 [14].

Rumus *simplicity* (Q_S) didefinisikan pada Persamaan 2.26 [14].

$$Q_s = 1 - \frac{\#da + \#ma}{\#npt + \#eventclass} \tag{2.26}$$

Keterangan:

$\#da$: jumlah macam aktivitas pada *event log* yang ditampilkan redundan pada *process tree*.

$\#ma$: jumlah macam aktivitas pada *event log* yang tidak ditampilkan pada *process tree* dan jumlah *leaf node* yang tidak terdapat dalam *event log*.

$\#npt$: jumlah *leaf node* pada *process tree*.

$\#eventclass$: jumlah macam aktivitas pada *event log*.

[Halaman ini sengaja dikosongkan]

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 proses model dari suatu *event log*.

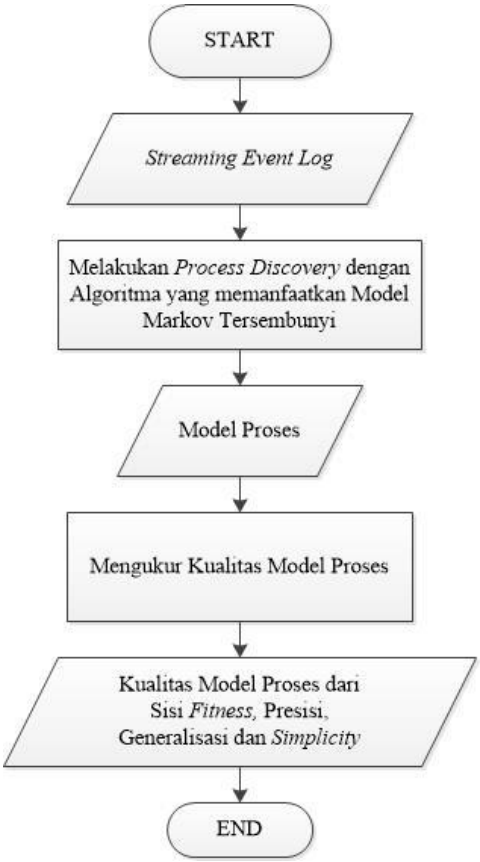
3.1 Cakupan Permasalahan

Permasalahan utama yang diangkat pada pembuatan Tugas Akhir ini adalah menemukan proses model yang tepat (*process discovery*) dari *streaming event log* serta memperbaiki *event log* yang terpotong pada saat *process discovery*. Pada Tugas Akhir ini, algoritma *process discovery* yang digunakan adalah algoritma yang diusulkan dengan memanfaatkan Model Markov Tersembunyi.

Permasalahan pertama yang dipecahkan dalam Tugas Akhir ini adalah menggambarkan model proses yang bisa beradaptasi dengan perkembangan proses pada pertengahan *streaming event log*. Penggambaran model proses tersebut dengan memanfaatkan Model Markov Tersembunyi. Model proses yang diharapkan adalah model proses yang memiliki relasi yang tepat antar aktivitas. Penentuan relasi menggunakan metode pada algoritma yang diusulkan dengan memanfaatkan probabilitas transisi *state* (A) pada Model Markov Tersembunyi.

Permasalahan kedua yang dipecahkan pada Tugas Akhir ini adalah memperbaiki *event log* yang terpotong dengan menambahkan *event log* prediksi untuk melengkapi *event log* yang terpotong tersebut. Cara melengkapi *event log* ini dengan menggunakan metode Viterbi dan metode *Backward*.

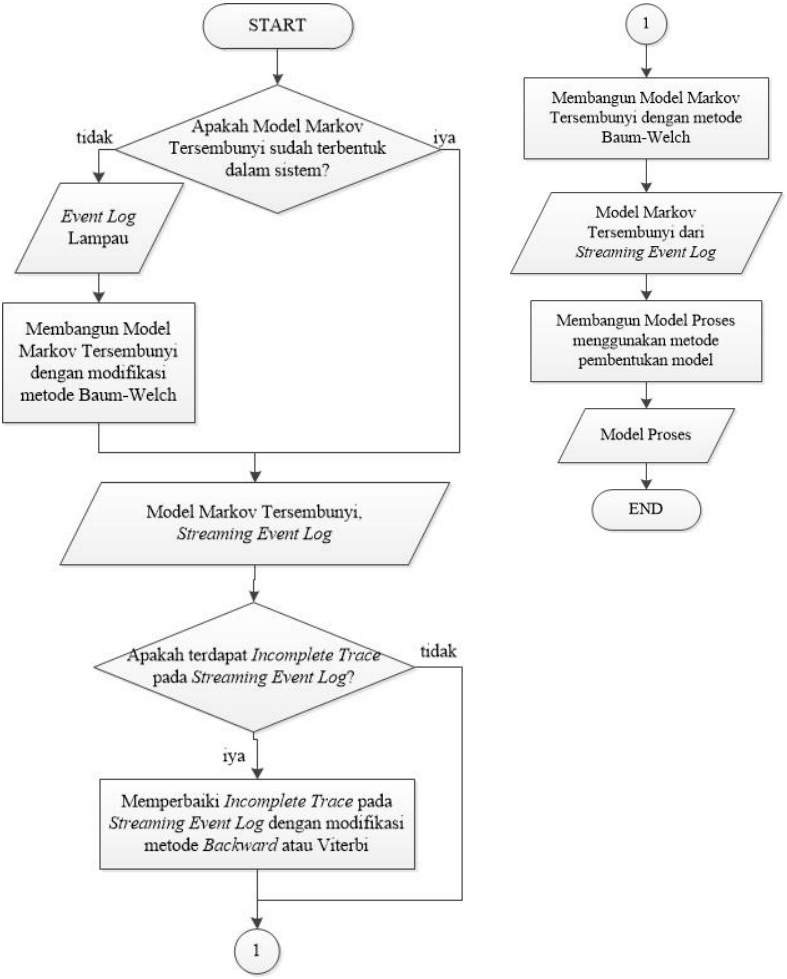
Permasalahan ketiga yang dipecahkan pada Tugas Akhir ini adalah pengukuran kualitas model proses dari sisi *fitness,* presisi, generalisasi dan *simplicity* pada algoritma yang diusulkan. Pengukuran dilakukan sebagai tolak ukur kinerja dari algoritma tesebut.



Gambar 3.1 Alur Proses Pengerjaan Tugas Akhir *Process Discovery* secara Umum

Gambar 3.1 adalah gambaran umum alur proses pengerjaan Tugas Akhir. Proses yang dilakukan adalah membentuk model proses menggunakan algoritma yang diusulkan dan mengukur kualitas model proses dari 4 sisi yaitu *fitness*, presisi, generalisasi dan *simplicity*. Algoritma yang diusulkan membutuhkan masukan berupa *streaming event log* dan Model Markov Tersembunyi. Apabila Model Markov Tersembunyi belum terdapat dalam sistem,

maka akan dibentuk menggunakan data *event log* lampau. Alur dari algoritma yang diusulkan dapat dilihat pada Gambar 3.2.



Gambar 3.2 Alur Proses Algoritma yang memanfaatkan Model Markov Tersembunyi

3.2 Membangun Model Markov Tersembunyi (HMM) berdasarkan *Event Log* Lampau

Model Markov Tersembunyi berdasarkan *event log* lampau digunakan sebagai masukan dalam pembentukan Model Markov Tersembunyi berdasarkan *streaming event log*. Model Markov Tersembunyi tersebut didasarkan pada model Petri net. Dikarenakan setiap transisi di Petri net yang mengandung aktivitas selalu bergantung dengan *place* di Petri net, maka *state* pada Model Markov Tersembunyi adalah *place* sedangkan *observer* pada Model Markov Tersembunyi adalah rangkaian aktivitas. Perbedaan Petri net dengan Model Markov Tersembunyi adalah *place* pada Model Markov Tersembunyi memiliki lebih dari satu aktivitas pada relasi *paralel*, sedangkan Petri net hanya menggambarkan kondisi tersebut pada relasi *conditional*.

Penentuan aktivitas yang memiliki relasi paralel atau tidak mengacu pada metode *Process discovery based on Activity Lifespan* [16]. Metode *Process discovery based on Activity Lifespan* mengemukakan bahwa suatu aktivitas memiliki relasi paralel dengan aktivitas lain apabila waktu mulai aktivitas tersebut sebelum waktu selesai aktivitas lain. Oleh karena itu, *event log* lampau yang digunakan sebagai acuan pertama menggunakan waktu ganda (waktu mulai dan waktu selesai suatu aktivitas). Penggambaran Model Markov Tersembunyi berdasarkan *event log* lampau maupun *streaming event log* dapat dilihat pada Gambar 3.3. Pengaplikasian metode Baum-Welch mengalami modifikasi dalam inisialisasi *state* dan *observer* yang dapat dilihat pada baris kode Gambar 3.4 yang bercetak tebal.

Gambar 3.3 Gambaran Model Markov Tersembunyi

Modifikasi Metode Baum-Welch	**Metode Baum-Welch**
Masukan : *event log* lampau	**Masukan** : *event log* lampau
Keluaran : Model Markov Tersembunyi dari *event log* lampau	**Keluaran** : Model Markov Tersembunyi dari *event log* lampau
1. Pengelompokkan aktivitas sebagai *observer* dimana aktivitas yang mempunyai relasi paralel dengan aktivitas lain dimasukkan dalam kelompok yang sama. **2. Jumlah *state* adalah jumlah kelompok *observer* dimana *observer* dalam satu kelompok bergantung pada *state* yang sama.** **3.** Pembangunan Model Markov Tersembunyi menggunakan metode Baum-Welch yang dipaparkan pada subbab 2.8.	1. Pembangunan Model Markov Tersembunyi menggunakan metode Baum-Welch yang dipaparkan pada subbab 2.8.

Gambar 3.4 *Pseudo-code* Modifikasi Metode Baum-Welch dan Metode Baum-Welch

3.3 Memperbaiki Incomplete Trace

Pada bagian ini dijelaskan mengenai langkah memperbaiki *incomplete trace* sebelum dilakukan *process discovery* dengan algoritma yang diusulkan pada Tugas Akhir ini. *Incomplete trace* dapat disebabkan oleh 2 hal, yaitu: (1) *trace* yang terpotong di awal; (2) *trace* yang terpotong di akhir. Perbaikan yang dilakukan melibatkan metode *Backward* dan metode Viterbi.

Perbaikan *incomplete trace* dilakukan dengan memodifikasi metode *Backward* dan metode Viterbi berupa penambahan *code*. *Code* tambahan adalah baris kode yang dicetak tebal pada Gambar 3.5. Penambahan *code* pada baris 2 sampai baris 4 dan baris 10 sampai 12 digunakan untuk menanggulangi apabila aktivitas pada *incomplete trace* belum termasuk dalam Model Markov Tersembunyi. Baris 2 sampai baris 4 menyatakan apabila aktivitas

pertama tidak ada pada Model Markov Tersembunyi, maka aktivitas tersebut dihilangkan dan aktivitas setelahnya menjadi aktivitas pertama.

Kemudian, penambahan *code* pada baris 10 sampai 12 menyatakan apabila aktivitas terakhir tidak ada pada Model Markov Tersembunyi, maka aktivitas tersebut dihilangkan dan aktivitas sebelumnya menjadi aktivitas terakhir. Sedangkan, penambahan *code* pada baris 7 dan baris 15 digunakan untuk memperbaiki *incomplete trace* dengan menambahkan aktivitas untuk melengkapi *incomplete trace* tersebut. Penambahan *code* ini dilakukan karena metode Viterbi dan metode *Backward* hanya menghasilkan *state*, sedangkan yang dibutuhkan untuk perbaikan *incomplete trace* adalah *observer* yang bergantung pada *state* tersebut. Baris 17 digunakan untuk menyimpan rangkaian aktivitas hasil perbaikan *incomplete trace*.

Perbaikan *Incomplete Trace* (Modifikasi metode Viterbi dan *Backward*)	**Metode Viterbi dan *Backward***
Masukan : A_L, B_L, π_L, *seq_stream*	**Masukan** : A_L, B_L, π_L, *seq_stream*
Keluaran : *seq_repair*	**Keluaran** : $stateX_0, stateX_T$

//kondisi pertama
1. $X_0 \leftarrow seq\_stream[0]$
2. **while** $\sum_{i=1}^{N} \boldsymbol{b_i(X_0) = 0}$ **do**
3. $\boldsymbol{remove\ X_0}$
4. $\boldsymbol{X_0 \leftarrow seq\_stream[0]}$
5. while $\pi_{stateX_0} = 0$ do
6. $S_S \leftarrow \underset{j=0,..,N}{\text{argmax}}\ Backward(j, stateX_0)$
7. $\boldsymbol{addFront(seq\_stream, O_{S_S})}$
8. $stateX_0 \leftarrow S_S$
 	
//kondisi kedua
9. $X_T \leftarrow seq\_stream[-1]$
10. while $\sum_{i=1}^{N} \boldsymbol{b_i(X_T) = 0}$ **do**
11. $\boldsymbol{remove\ X_T}$
12. $\boldsymbol{X_T \leftarrow seq\_stream[-1]}$
	1. $X_0 \leftarrow seq\_stream[0]$
	2. while $\pi_{stateX_0} = 0$ do
	3. $S_S \leftarrow \underset{j=0,..,N}{\text{argmax}}\ Backward(j, stateX_0)$
	4. $stateX_0 \leftarrow S_S$
	5. $X_T \leftarrow seq\_stream[-1]$
	6. while $\sum_{j=1}^{N} a_{stateX_T j} > 0$ do
	7. $S_S \leftarrow Viterbi(stateX_T)$
	8. $stateX_T \leftarrow S_S$

13. while $\sum_{j=1}^{N} a_{stateX_T j} > 0$ do **14.** $S_S \leftarrow Viterbi(stateX_T)$ **15.** $\boldsymbol{addLast(seq\_stream, O_{S_S})}$ **16.** $stateX_T \leftarrow S_S$ **17.** ***seq_repair ← seq_stream***	

Gambar 3.5 *Pseudo-code* Perbaikan *Incomplete Trace* dengan *Pseudo-code* metode Viterbi dan *Backward*

Keterangan:

π_L	: vektor probabilitas *state* sebagai *state* awal dari *event log* lampau.
$a_{stateX_T j}$	: nilai probabilitas transisi dari *state* yang memiliki *observer* X_T ke *state* j.
$addFront(seq\_stream, O_{S_S})$	: *observer* dari *state S* ditambahkan pada bagian pertama dari *incomplete trace*.
$addLast(seq\_stream, O_{S_S})$	: *observer* dari *state S* ditambahkan pada bagian akhir dari *incomplete trace*.
A_L	: matrik probabilitas transisi *state* dari *event log* lampau.
$b_i(X_0)$	: nilai probabilitas *observer* X_0 terhadap *state i*.
B_L	: nilai probabilitas *observer* X_0 terhadap *state i*.
Backward(j,$stateX_0$*)*	: nilai metode *Backward* dari *state j* dimana *stateX₀* merupakan *state* yang terjadi setelah *state j*.
N	: jumlah *state* pada matrik A_L.
seq_repair	: hasil perbaikan *incomplete trace*.
$seq\_stream[0]$	: aktivitas pertama pada *incomplete trace*.
$seq\_stream[-1]$	: aktivitas terakhir pada *incomplete trace*.
π_{stateX_0}	: nilai probabilitas *state* yang memiliki *observer* X_0 sebagai *state* awal.

$Viterbi(stateX_T)$	: hasil dari metode Viterbi berupa *state* dimana $stateX_T$ merupakan state yang terjadi sebelum hasil metode Viterbi.
X_0	: nama aktivitas awal pada *trace* di *event log*.
X_T	: nama aktivitas akhir pada *trace* di *event log*.

3.4 Membangun Model Markov Tersembunyi (HMM) berdasarkan Streaming Event Log

Model Markov Tersembunyi digunakan sebagai masukan untuk algoritma yang diusulkan pada Tugas Akhir ini serta memperbarui Model Markov Tersembunyi dari *event log* lampau (A_L) untuk *streaming event log* pada waktu proses selanjutnya. Pembangunan Model Markov Tersembunyi berdasarkan *streaming event log* memiliki langkah yang sama dengan pembangunan Model Markov Tersembunyi berdasarkan *event log* lampau (dapat dilihat pada Gambar 3.4). Karena keseluruhan *streaming event log* tidak tersimpan dalam sistem serta adanya perkembangan proses yang menghasilkan tambahan aktivitas baru, maka terdapat tambahan metode berupa penggabungan Model Markov Tersembunyi yang baru dibentuk dengan Model Markov Tersembunyi dari *event log* lampau (A_L). *Pseudo-code* dari Pembangunan Model Markov Tersembuyi berdasarkan *streaming event log* dapat dilihat pada Gambar 3.6.

Baris kode 1 menyatakan program akan berjalan apabila terdapat *trace* pada *streaming event log*. Baris kode 3 sampai baris kode 6 digunakan untuk memperbarui matrik probabilitas transisi *state* (A_L) apabila ukuran matrik pada Model Markov Tersembunyi dari *event log* lampau dan dari *streaming event log* berbeda. Perbedaan ini bisa disebabkan oleh penambahan aktivitas di tengah-tengah proses. Sedangkan baris kode 7 sampai baris kode 11 digunakan untuk memperbarui matrik probabilitas *observer* (B_L) dan vektor *state* sebagai *state* awal (π_L). Perbaharuan Model Markov Tersembunyi tersebut digunakan sebagai Model Markov

Tersembunyi dari *event log* lampau untuk *process discovery* berikutnya. Pernyataan kalimat sebelumnya menjelaskan baris kode 12 sampai baris kode 14.

Pembangunan Model Markov Tersembunyi (HMM) berdasarkan *Streaming Event Log*
Masukan : A_L, B_L, π_L, *seq_repair*
Keluaran : Model Markov Tersembunyi dari *Streaming Event Log* (A_S, B_S, π_S) dan Model Markov Tersembunyi dari *Event Log* Lampau (A_L, B_L, π_L)
1. while *seq_repair* ≠ null do **2.** //Pembentukan Model Markov Tersembunyi (A_S, B_S, π_S) sesuai Gambar 3.3 **3.** if length(A_s)>length(A_L) then **4.** $A \leftarrow A_s + A_L$ **5.** else **6.** $A \leftarrow A_L + A_s$ **7.** if length(B_s)>length(B_L) then **8.** $B \leftarrow B_s + B_L$ **9.** else **10.** $B \leftarrow B_L + B_s$ **11.** $\pi = \pi_S + \pi_L$ **12.** A_L=A **13.** B_L=B **14.** $\pi_L = \pi$

Gambar 3.6 *Pseudo-code* Pembangunan Model Markov Tersembunyi (HMM) berdasarkan *Streaming event log*

Keterangan:

π_S : vektor probabilitas *state* sebagai *state* awal dari *streaming event log*.

π_L : vektor probabilitas *state* sebagai *state* awal dari *event log* lampau.

A_L : matrik probabilitas transisi *state* dari *event log* lampau.

A_S : matrik probabilitas transisi *state* dari *streaming event log*.

B_L : matrik probabilitas *observer* dari *event log* lampau.
B_S : matrik probabilitas *observer* dari *streaming event log*.
seq_repair : rangkaian *observer* pada *streaming event log* yang mengalami perbaikan apabila terdapat *incomplete trace.*

3.5 Metode Pembentukan Model Proses

Pada bagian ini dijelaskan tentang Metode Pembentukan Model Proses. Metode Pembentukan Model Proses dibagi kedalam dua tahapan yaitu penentuan relasi XOR, OR, AND dan *sequence* dan pembentukan relasi aktivitas pada model proses.

3.5.1 Penentuan Relasi XOR, OR, AND dan Sequence

Penentuan relasi XOR, OR, AND dan *sequence* dilakukan dengan memanfaatkan probabilitas transisi *state* (A_S) pada Model Markov Tersembunyi. Apabila suatu *state* memiliki satu *observer* dan memiliki probabilitas transisi dengan *state* lain dimana jumlah *state* lain adalah lebih dari satu, maka relasi *state* tersebut adalah XOR. Sedangkan, apabila suatu *state* memiliki lebih dari satu *observer*, maka relasi *state* tersebut adalah OR atau AND. Selain dari kedua kondisi tersebut, maka relasi *state* tesebut adalah *sequence*.

Nilai yang digunakan untuk menentukan relasi OR atau AND ditampung dalam RM(S_i). Batasan penentuan antara kedua relasi tersebut adalah avgPP. AvgPP adalah nilai rata-rata dari probabilitas transisi *state* yang bernilai lebih dari 0. Jika nilai RM(S_i) lebih atau sama dengan nilai avgPP, maka relasi *state* tersebut adalah OR dan sebaliknya untuk relasi AND. Langkah penentuan relasi dapat dilihat pada Gambar 3.7.

$$\mathrm{RM}(i \rightarrow j) = \frac{a_{ii}}{a_{ij}} \times \frac{1}{n_{O_i}} \tag{3.1}$$

$$\text{avgPP}=\frac{\sum_{i=1}^{n_A}\sum_{j=1}^{n_A} a_{ij}}{n_{a_{ij}}} \quad (3.2)$$

Keterangan pada Persamaan 3.1 dan 3.2:

a_{ii} : nilai probabilitas transisi *state i* ke *state* itu sendiri.

a_{ij} : nilai probabilitas transisi *state i* ke *state j* (a_{ij}) dengan syarat nilai a_{ij} lebih dari 0.

n_A : jumlah *state* pada matrik probabilitas transisi *state* dari *streaming event log* (A_S).

n_{O_i} : jumlah *observer* yang bergantung pada *state i*.

$n_{a_{ij}}$: jumlah banyaknya a_{ij} yang bernilai lebih dari 0.

Pseudo-code Penentuan Relasi XOR, OR, AND, dan *Sequence* digunakan untuk menentukan relasi antar aktivitas. Baris kode 1 dan 2 menyatakan inisialisasi dari aktivitas sebelum dan sesudah suatu relasi. Baris kode 3 dijalankan apabila memenuhi syarat bahwa *observer* aktivitas sebelum atau aktivitas sesudah harus satu. Sedangkan baris kode 4 dan 5 merupakan syarat untuk pembentukan relasi XOR. Baris kode 6 dan 7 merupakan syarat untuk pembentukan relasi *sequence*. Sedangkan baris kode 9 sampai 12 merupakan syarat untuk pembentukan relasi OR maupun AND dengan melibatkan Persamaan 3.1 dan Persamaan 3.2.

Penentuan relasi XOR, OR, AND, dan *Sequence*
Masukan : A_S, B_S,S_{act_before}, S_{act_after}
Keluaran : $relation$
1. i = S_{act_before} **2.** j = S_{act_after} **3.** if n_{O_i}=1 OR n_{O_j}=1 then **4.** if (n_{O_i}=1 AND $n_{j\rightarrow}$>1) OR (n_{O_j}=1 AND $n_{i\rightarrow}$>1) then **5.** $relation = relasi\ XOR$ **6.** else

```
7.      graph[act_before → act_after] = relasi sequence
8.  else
      //menghitung RM(i → j) dan avgPP sesuai Persamaan 3.1 dan
      Persamaan 3.2
9.    if RM(i → j) < avgPP then
10.     relation = relasi OR
11.   else
12.     relation = relasi AND
```

Gambar 3.7 *Pseudo-code* Penentuan Relasi XOR, OR, AND, dan *Sequence*

Keterangan:

A_S	: matrik probabilitas transisi *state* dari *streaming event log*.
B_S	: matrik probabilitas *observer* dari *streaming event log*.
graph[act_before → act_after]	: penampung relasi antara aktivitas awal (act_before) dengan aktivitas akhir (act_after).
n_{O_i}	: jumlah *observer* yang bergantung pada *state* i.
$n_{j\rightarrow}$	: jumlah *state* yang bertransisi dengan *state* j.
relation	: variable untuk menyimpan relasi antar aktivitas.
S_{act_before}	: *state* yang memiliki *observer* sebagai aktivitas awal pada relasi.
S_{act_after}	: *state* yang memiliki *observer* sebagai aktivitas akhir pada relasi.

3.5.2 Penentuan Relasi Aktivitas

Pseudo-code penentuan relasi aktivitas digunakan untuk menentukan rangkaian model dari relasi aktivitas. *Pseudo-code* penentuan relasi aktivitas dijelaskan di Gambar 3.8.

Baris kode 1 dan 2 menyatakan perulangan sebanyak *state* pada Model Markov Tersembunyi dari *streaming event log*. Baris kode 3 dan 4 adalah syarat untuk menambahkan aktivitas yang

diindikasi memiliki relasi. Syarat tersebut adalah terdapat probabilitas transisi *state* tersebut ke *state* lainnya dan *state* tersebut tidak memiliki probabilitas transisi ke dirinya sendiri. Baris kode 7 sampai 11 digunakan untuk menentukan relasi *Split* sedangkan baris kode 12 sampai 20 digunakan untuk menentukan relasi *Join.* Baris kode 8 dan 9 serta baris kode 13 dan 14 digunakan untuk menjelaskan syarat pembentukan relasi *sequence*. Baris kode 10 dan 11 serta baris kode 19 dan 20 digunakan untuk menentukan relasi menggunakan *pseudo-code* penentuan relasi XOR, OR, AND, dan *sequence*. Baris kode 15 sampai 18 digunakan untuk menentukan relasi khusus, yaitu aktivitas yang terkena dampak relasi XOR yang redundan.

Penentuan Relasi Aktivitas
Masukan : A_S, B_S,graph
Keluaran : $graph$
1. for S_i in [0,...,n_{A_S}] do
2. for S_j in [0,...,n_{A_S}] do
3. if $a_{S_i S_j} > 0$ then
4. if $(S_i \neq S_j) \vee ((S_i = S_j) \wedge n_{O_{S_i}} = 1)$ then
5. graph[act_before, act_after]= [O_{S_i} , O_{S_j}]
6. for act_before in graph do
7. if (n_{act_after}> 1):
8. if act_before=act_after then
9. graph[act_before → act_after] = $relasi\ sequence$
10. else
11. graph[act_before → act_after] = relasi ditentukan sesuai Gambar 3.7 + " *Split*"
12. if (n_{act_before}> 1):
13. if act_before=act_after then
14. graph[act_before → act_after] = $relasi\ sequence$
15. else
16. if graph[act_before → act_after] = $relasi\ XOR\ Split$ then
17. relation_before = graph[act_before → act_after]
18. graph[act_before → act_after] = relasi_before + relasi ditentukan sesuai Gambar 3.7 + " *Join*"

19.	else
20.	graph[act_before → act_after] = relasi ditentukan sesuai Gambar 3.7 + " *Join*"

Gambar 3.8 *Pseudo-code* Penentuan Relasi Aktivitas

Keterangan:

$a_{S_i S_j}$	:	nilai probabilitas transisi *state* S_i ke *state* S_j.
A_S	:	matrik probabilitas transisi *state* dari *streaming event log*.
B_S	:	matrik probabilitas *observer* dari *streaming event log*.
graph[act_before, act_after]	:	penampung aktivitas awal dan aktivitas akhir untuk penentuan relasi.
graph[act_before →act_before]	:	penampung relasi antara aktivitas awal menuju aktivitas akhir.
n_{act_after}	:	jumlah aktivitas akhir untuk pembentukan suatu relasi.
n_{act_before}	:	jumlah aktivitas awal untuk pembentukan suatu relasi.
n_{A_S}	:	jumlah *state* pada A_S.
$n_{O_{S_i}}$	:	jumlah *observer* yang bergantung pada *state* S_i.
O_{S_i}	:	kumpulan *observer* yang bergantung pada *state* S_i.

3.6 Contoh Sederhana Process Discovery menggunakan Algoritma yang memanfaatkan Model Markov Tersembunyi

Pada bagian ini dijelaskan contoh penerapan *process discovery* menggunakan algoritma yang diusulkan, dimulai dari pembangunan Model Markov Tersembunyi dari *event log* lampau sampai dengan pembentukan model proses menggunakan peraturan dan rumus yang diusulkan.

3.6.1 Contoh Pembangunan Model Markov Tersembunyi dari Event Log Lampau

Tabel 3.1 menunjukkan *event log* lampau yang akan digunakan sebagai inisialisasi awal penerapan algoritma yang diusulkan serta perbaikan *incomplete trace*. Untuk *event log* lampau terdiri dari 10 *case*.

Tabel 3.1 *Event Log* Lampau

Case ID	Activity	Start Stamp	End Stamp	Case ID	Activity	Start Stamp	End Stamp
PP1	A	3/8/16 10:32	3/8/16 13:42	PP6	A	3/10/16 22:42	3/11/16 1:52
PP1	C	3/8/2016 13:42	3/8/2016 16:52	PP6	C	3/11/16 1:52	3/11/16 5:02
PP1	E	3/8/2016 16:52	3/8/2016 20:02	PP6	E	3/11/16 5:02	3/11/16 8:12
PP2	A	3/8/2016 20:02	3/8/2016 23:12	PP7	A	3/11/16 8:12	3/11/16 11:22
PP2	B	3/8/2016 23:12	3/9/2016 2:22	PP7	C	3/11/16 11:22	3/11/16 14:32
PP2	D	3/9/2016 2:22	3/9/2016 5:32	PP7	E	3/11/16 14:32	3/11/16 17:42
PP2	E	3/9/2016 5:32	3/9/2016 8:42	PP8	A	3/11/16 17:42	3/11/16 20:52
PP3	A	3/9/2016 8:42	3/9/2016 11:52	PP8	C	3/11/16 20:52	3/12/16 0:02
PP3	D	3/9/2016 11:52	3/9/2016 15:02	PP8	E	3/12/16 0:02	3/12/16 3:12
PP3	B	3/9/2016 15:02	3/9/2016 18:12	PP9	A	3/12/16 3:12	3/12/16 6:22
PP3	E	3/9/2016 18:12	3/9/2016 21:22	PP9	C	3/12/16 6:22	3/12/16 9:32
PP4	A	3/9/2016 21:22	3/10/2016 0:32	PP9	E	3/12/16 9:32	3/12/16 12:42
PP4	D	3/10/16 0:32	3/10/16 3:42	PP10	A	3/12/16 12:42	3/12/16 15:52
PP4	B	3/10/16 3:42	3/10/16 6:52	PP10	B	3/12/16 15:52	3/12/16 19:02
PP4	E	3/10/16 6:52	3/10/16 10:02	PP10	D	3/12/16 19:02	3/12/16 22:12
PP5	A	3/10/16 10:02	3/10/16 13:12	PP10	E	3/12/16 22:12	3/13/16 1:22
PP5	D	3/10/16 13:12	3/10/16 16:22				
PP5	B	3/10/16 16:22	3/10/16 19:32				
PP5	E	3/10/16 19:32	3/10/16 22:42				

Hasil dari penerapan metode Baum-Welch dipaparkan pada Tabel 3.2 sampai 3.4.

Tabel 3.2 Vektor Probabilitas *State* Awal Model Markov Tersembunyi dari *Event Log* Lampau

π_L	
State	Probabilitas
P1	1

Tabel 3.3 Matrik Probabilitas Transisi *State* Model Markov Tersembuyi dari *Event Log* Lampau

A_L					
State	*State*	Probabilitas	*State*	*State*	Probabilitas

awal	akhir		awal	akhir	
P1	P2	0.6	P4	P5	1
P1	P4	0.4	P5	P3	1
P2	P3	1			

Tabel 3.4 Matrik Probabilitas *Observer* Nama Aktivitas terhadap *State* Model Markov Tersembunyi dari *Event Log* Lampau

B_L					
State	*Observer*	Probabilitas	*State*	*Observer*	Probabilitas
P1	A	1	P4	B	1
P2	C	1	P5	D	1
P3	E	1			

3.6.2 Contoh Perbaikan Incomplete Trace

Tabel 3.5 menunjukkan *trace* pada *streaming event log* yang terekam. Dengan mengikuti langkah perbaikan *incomplete trace* pada Gambar 3.5, *case* PP15 memenuhi kondisi kedua sehingga dilakukan perbaikan menggunakan metode Viterbi. Tabel 3.6 menunjukkan langkah-langkah perbaikan *incomplete trace*.

Tabel 3.5 *Trace* dari *Streaming event log*

Case ID	Activity	Start Stamp	End Stamp
PP11	A	3/9/2016 11:52	3/9/2016 15:02
PP11	B	3/9/2016 15:02	3/9/2016 18:12
PP11	D	3/9/2016 18:12	3/9/2016 21:22
PP11	E	3/9/2016 21:22	3/10/2016 0:32
PP12	A	3/10/2016 0:32	3/10/2016 3:42
PP12	B	3/10/2016 3:42	3/10/2016 6:52
PP12	D	3/10/2016 6:52	3/10/2016 10:02
PP12	E	3/10/2016 10:02	3/10/2016 13:12

Case ID	Activity	Start Stamp	End Stamp
PP13	A	3/10/2016 13:12	3/10/2016 16:22
PP13	C	3/10/2016 16:22	3/10/2016 19:32
PP13	E	3/10/2016 19:32	3/10/2016 22:42
PP14	A	3/10/2016 22:42	3/11/2016 1:52
PP14	C	3/11/2016 1:52	3/11/2016 5:02
PP14	E	3/11/2016 5:02	3/11/2016 8:12
PP15	A	3/11/2016 8:12	3/11/2016 11:22

Tabel 3.6 Langkah Perbaikan *Incomplete Trace* PP15

		Viterbi (1)	Viterbi (2)
State	P1	0	0
	P2	**0.6**	0
	P3	0	**0.6**
	P4	0.4	0
	P5	0	0
Hasil *State*		P2	P3
Nama Aktivitas		**C**	**E**

Contoh penjelasan rinci mengenai hasil Viterbi (1) pada *state* P3 serta hasil Viterbi (2) pada *state* P5 dimana A merupakan *observer* dari *state* P1:

v1_P1 = (P(P1)*P(P1|P1)*P(O|P1))
= $(1 * a_{P1P1} * b_{P1}(A))$
= (1*0*1)
= 0

v1_P2 =(P(P1)*P(P2|P1)*P(O|P2))
= $(1 * a_{P1P2} * b_{P2}(C))$
= $(1 * 0.6 * 1)$
= 0.6

v2_P3 = (v1_P2*P(P3|P2)*P(O|P3))
= $(0.6 * a_{P2P3} * b_{P3}(E))$
=$(0.6 * 1 * 1)$
= 0.6

Hasil maksimal dengan menggunakan metode Viterbi adalah P2 dan P3. Karena *observer* nama aktivitas dari P2 adalah C sedangkan P3 adalah E, maka *trace* perbaikan PP15 adalah = [A,C,E].

3.6.3 Contoh Pembentukan Model Markov Tersembunyi dari Streaming Event Log yang Diperbaiki

Bagian ini akan memaparkan pembentukan Model Markov Tersembunyi berdasarkan Model Markov Tersembunyi dari *event log* lampau dan *incomplete trace* yang diperbaiki pada *streaming event log*. Pada contoh kasus berikut, data diambil dari hasil sub subbab 3.5.1 dan 3.5.2. Tabel 3.7 sampai Tabel 3.9 merupakan hasil Model Markov Model dari *streaming event log* yang diperbaiki.

Tabel 3.7 Vektor Probabilitas *State* Awal Model Markov Tersembunyi dari *Streaming event log*

π_S	
State	Probabilitas
P1	1

Tabel 3.8 Matrik Probabilitas Transisi *State* Model Markov Tersembunyi dari *Streaming event log*

A_S					
State awal	*State* akhir	Probabilitas	*State* awal	*State* akhir	Probabilitas
P1	P2	0.6	P4	P5	1
P1	P4	0.4	P5	P3	1
P2	P3	1			

Tabel 3.9 Matrik Probabilitas *Observer* Nama Aktivitas terhadap *State* Model Markov Tersembunyi dari *Streaming event log*

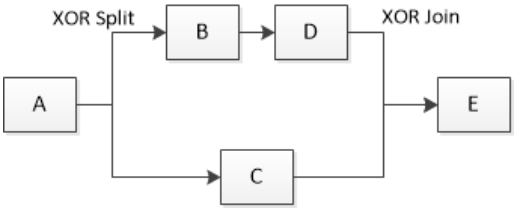
B_S					
State	*Observer*	Probabilitas	*State*	*Observer*	Probabilitas
P1	A	1	P4	B	1
P2	C	1	P5	D	1
P3	E	1			

3.6.4 Contoh Pembangunan Relasi Model Proses

Bagian ini akan menjelaskan contoh pencarian relasi dengan mengikuti peraturan dan rumus yang dijabarkan pada *pseudo-code* Gambar 3.7 dan Gambar 3.8. Nilai avgPP yang digunakan *threshold* yang akan digunakan sesuai dengan Persamaan 3.2 adalah 0.87. Hasil relasi dapat dilihat pada Tabel 3.10 dan model proses yang terbentuk dapat dilihat pada Gambar 3.9.

Tabel 3.10 Tabel Relasi Aktivitas

Hasil Relasi Aktivitas		
Aktivitas awal	Aktivitas akhir	Relasi
A	C,B	XOR Split
B	D	Sequence
C,D	E	XOR Join



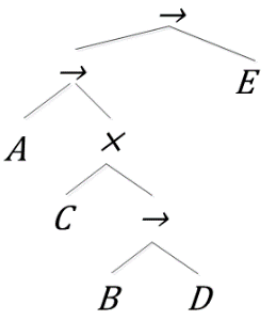
Gambar 3.9 Model Proses

3.7 Perhitungan Kualitas Model Proses

Berdasarkan pengertian dari kualitas *fitness*, presisi, generalisasi dan *simplicity* yang telah dijelaskan pada subbab 2.13 sampai subbab 2.16, kita dapat mengevaluasi kualitas dari proses model yang ditemukan berdasarkan algoritma yang diusulkan. Kualitas diukur berdasarkan data *streaming event log* yaitu PP11 sampai PP15.

Kualitas *fitness* dan presisi diukur dengan menggunakan rumus yang dijelaskan pada subbab 2.13 dan 2.14. Untuk perhitungan presisi, *trace* yang digambarkan di model proses adalah ABDE dan ACE dimana kedua *trace* tersebut adalah keseluruhan *trace* yang dapat digambarkan dari model proses. Perhitungan nilai *fitness* dan presisi dengan menggunakan algoritma yang diusulkan dapat dilihat pada Tabel 3.12.

Sedangkan untuk mengukur kualitas generalisasi dan *simplicity*, model proses harus diubah dalam bentuk *process tree* yang dijelaskan pada subbab 2.12. *Process tree* dari model proses dapat dilihat pada Gambar 3.10. Rincian perhitungan kualitas generalisasi dapat dilihat pada Tabel 3.11 dan perhitungan akhir kualitas generalisasi serta *simplicity* dapat dilihat pada Tabel 3.12.



Gambar 3.10 *Process Tree* dari Model Proses Gambar 3.9

Tabel 3.11 Tabel Rincian Perhitungan Awal Kualitas Generalisasi

No	*nt*	*#executions*	n_{trace}	$\sum_{1}^{nt}\left(\sqrt{\#executions}\right)^{-1}$
1	→(B,D)	2	5	0.16
2	x(C, →)	5		0.1
3	→(A, x)	5		0.1
4	→(→,E)	5		0.1

Keterangan pada Tabel 3.11:

nt : *operator node* yang diimplementasikan di *event log*.

n_{trace} : jumlah *trace* pada *event log*.

#executions : jumlah *operator node* yang diimplementasikan di *trace* pada *event log*.

Tabel 3.12 Matrik Kualitas Proses Model dari Algoritma yang memanfaatkan Model Markov Tersembunyi

Kualitas		Rincian Perhitungan	Hasil Akhir
Fitness (Q_f)	PM	$=\frac{5}{5}$	1
	CPM	$=\frac{1}{2}\frac{(15-0)}{15}+\frac{1}{2}\frac{(15-0)}{15}$	
Presisi (Q_p)		$=\frac{2}{2}$	1
Generalisasi (Q_g)		$=1-\frac{0.46}{4}$	0.89
Simplicity (Q_s)		$=1-\frac{0+0}{5+5}$	1

Dengan memperhatikan Tabel 3.12, didapatkan nilai *fitness*, presisi, *simplicity* model proses adalah 1 dan nilai generalisasi model proses mendekati 1. Hal ini dapat disimpulkan bahwa model proses untuk contoh *streaming event log* dari algoritma yang diusulkan memiliki kualitas yang baik dari semua sisi.

[Halaman ini sengaja dikosongkan]

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 deskripsi umum sistem, kebutuhan fungsional sistem, dan kasus penggunaan sistem.

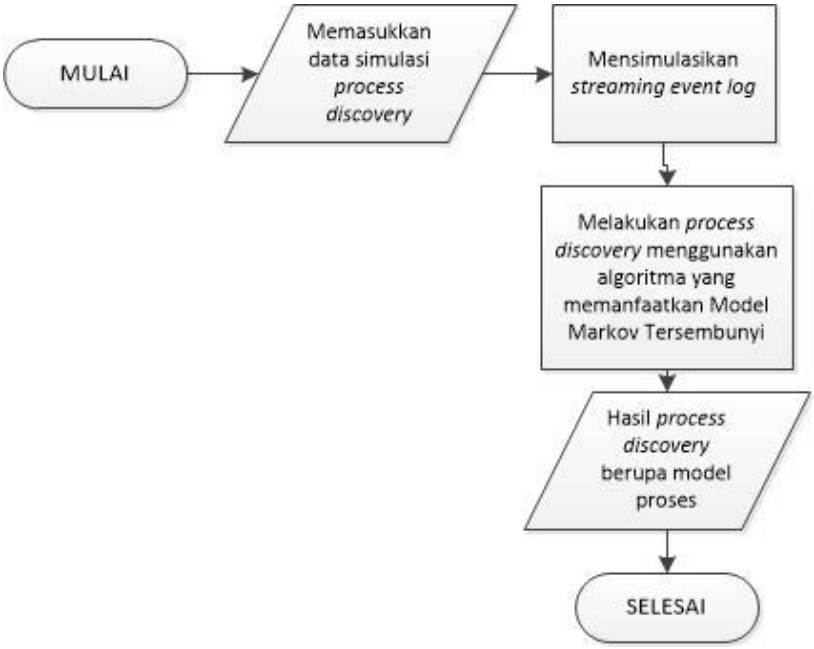
4.1.1 Deskripsi Umum Sistem

Perangkat lunak yang akan dibangun dapat menghasilkan model dari proses bisnis menggunakan algoritma yang diusulkan. Terdapat dua data yang dibutuhkan untuk memodelkan proses bisnis, yaitu *event log* lampau dan *streaming event log*. *Event log lampau* adalah *event log* yang sudah tersimpan sebelumnya. Dalam Tugas Akhir ini, *event log* lampau disimpan dalam bentuk Excel (.xlsx). Gambar 4.1 menjelaskan mengenai alur proses sistem.

Ketika sistem dijalankan, maka akan muncul halaman utama untuk memilih file berformat Excel yang berisi data *event log* lampau dan *streaming event log*. *Streaming event log* dilakukan dengan bantuan *library threading* pada Python. Setelah pengguna memilih file, maka sistem akan memproses *event log* lampau serta menjalankan *streaming event log*.

Pemrosesan model proses didasarkan pada *periodic reset* yang dipaparkan pada subbab 2.10. Terdapat 3 waktu pemrosesan model proses yaitu setiap 5 detik, setiap 10 detik dan setiap 15 detik

dimana waktu eksekusi setiap aktivitas adalah 0,5 detik. Pemrosesan model proses menggunakan algoritma yang memanfaatkan Model Markov Tersembunyi (dipaparkan pada bab III). Proses model yang merupakan hasil *process discovery* menggunakan algoritma yang memanfaatkan Model Markov Tersembunyi akan ditampilkan pada sistem secara langsung.



Gambar 4.1 Alur Proses Sistem

4.1.2 Kebutuhan Fungsional Sistem

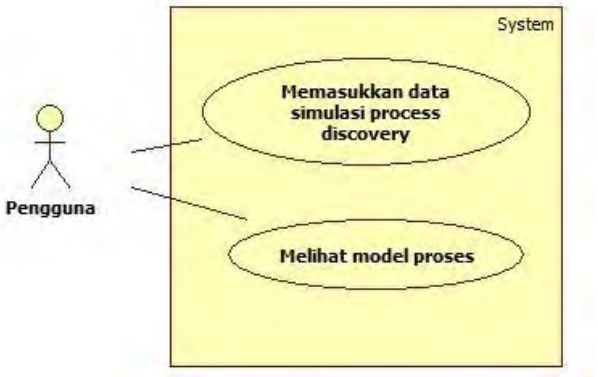
Kebutuhan fungsional berisi kebutuhan utama yang harus dipenuhi oleh sistem agar dapat bekerja dengan baik. Kebutuhan fungsional mendefinisikan layanan yang harus disediakan oleh sistem, bagaimana reaksi terhadap masukan, dan hal-hal yang harus dilakukan sistem pada situasi khusus. Daftar kebutuhan fungsional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

Tabel 4.1 Daftar Kebutuhan Fungsional Perangkat Lunak

Kode Kebutuhan	Kebutuhan Fungsional	Deskripsi
F-01	Memasukkan data simulasi *process discovery*	Pengguna dapat memasukkan *event log* dalam bentuk Excel (.xlsx) yang berisi *event log* lampau dan *streaming event log* sebagai data simulasi untuk *process discovery*.
F-02	Melihat model proses	Pengguna dapat melihat model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi sesuai rentang waktu pemrosesan model proses.

4.1.3 Kasus Penggunaan Sistem

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.



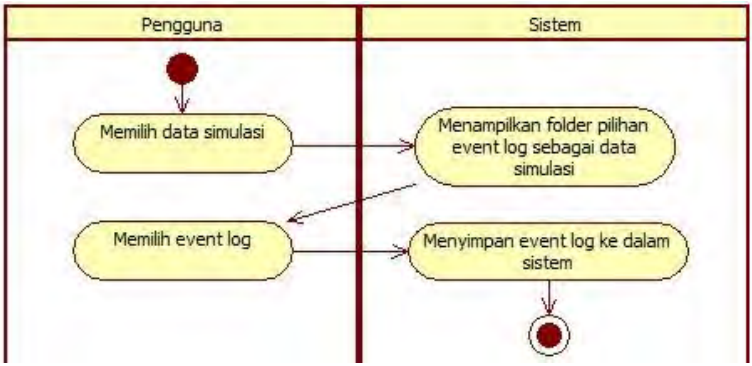
Gambar 4.2 Diagram Kasus Penggunaan

4.1.3.1 Deskripsi Kasus Pengunaan F-01

Kasus Pengunaan F-01 merupakan kasus dimana pengguna memilih *file event log* yang akan diolah oleh sistem dalam mensimulasikan *process discovery*. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.2 dan diagram aktivitas dari kasus pengunaan ini bisa dilihat pada Gambar 4.3.

Tabel 4.2 Spesifikasi Kasus Penggunaan F-01

Nama	Memasukkan data simulasi *process discovery*.
Kode	F-01.
Deskripsi	Pengguna memilih *file event log* yang akan diolah oleh sistem.
Aktor	Pengguna.
Kondisi Awal	*File event log* belum tersimpan.
Kondisi Akhir	Sistem menyimpan *file event log*.
Aliran	
Kejadian Normal	1. Pengguna memilih data simulasi. 2. Sistem menampilkan folder data. 3. Pengguna memilih salah satu *file event log* yang digunakan sebagai data simulasi . 4. Sistem menyimpan *file event log* pilihan ke dalam sistem.



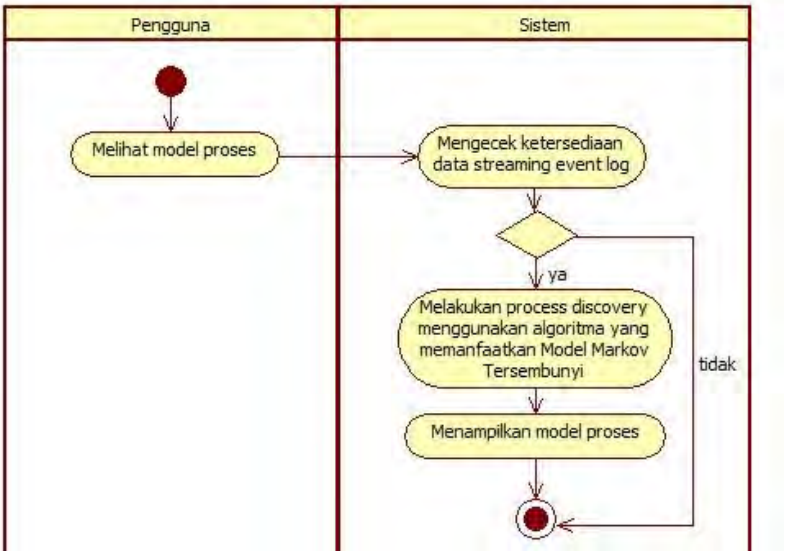
Gambar 4.3 Diagram Aktivitas Kasus Pengunaan F-01

4.1.3.2 Deskripsi Kasus Pengunaan F-02

Kasus Pengunaan F-02 merupakan kasus dimana pengguna melihat model proses hasil dar algoritma yang memanfaatkan Model Markov Tersembunyi. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 4.3 dan diagram aktivitas dari kasus pengunaan ini bisa dilihat pada Gambar 4.4.

Tabel 4.3 Spesifikasi Kasus Penggunaan F-02

Nama	Melihat model proses.
Kode	F-02.
Deskripsi	Pengguna melihat model proses.
Aktor	Pengguna.
Kondisi Awal	Model proses belum terbentuk.
Kondisi Akhir	Sistem menampilkan model proses.
Alur Kejadian	
Kejadian Normal	1. Pengguna melihat model proses. 2. Sistem melakukan *process discovery* pada *streaming event log* menggunakan algoritma yang memanfaatkan Model Markov Tersembunyi. 3. Sistem menampilkan model proses.
Kejadian Alternatif (Tidak ada data *streaming event log*)	1. Sistem tidak memproses dan model proses dari data *event log* lampau yang ditampilkan.



Gambar 4.4 Diagram Aktivitas Kasus Pengunaan F-02

4.2 Perancangan Sistem

Penjelasan tahap perancangan perangkat lunak yaitu perancangan proses analisis sehingga menghasilkan *output* model proses bisnis dari *streaming event log*. Perancangan sistem mengacu pada kebutuhan yang dipaparkan dalam kasus penggunaan F-01 dan F-02.

4.2.1 Antarmuka Memasukkan Data Simulasi

Antarmuka ini digunakan untuk memenuhi kebutuhan pengunaan F-01. Pada antarmuka ini, pengguna akan memilih *event log* dengan menekan tombol "Data" dan sistem akan memproses *event log* tersebut apabila pengguna telah menekan tombol "Proses". Gambar antarmuka dapat dilihat pada Gambar 4.5 dan spesifikasi elemen dapat dilihat pada Tabel 4.4.

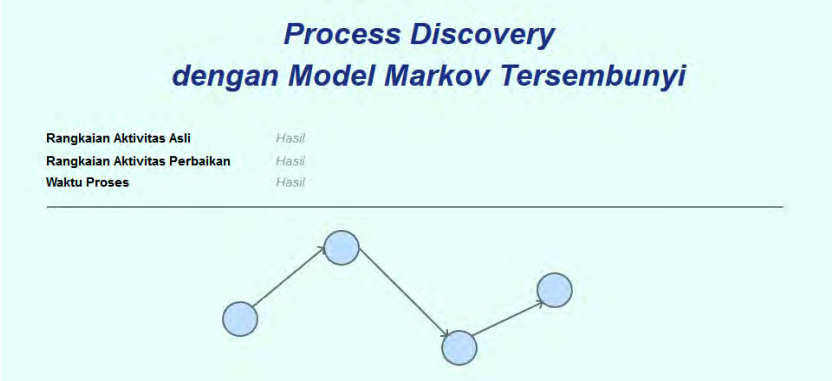
Gambar 4.5 Antarmuka Awal Menjalankan Simulasi *Process discovery*

Tabel 4.4 Spesifikasi Elemen pada Antarmuka Awal Menjalankan Simulasi *Process discovery*

Nama Objek	Jenis Objek	Tampilan Objek	Fungsionalitas
L_Judul	Label	***Process discovery dengan Model Markov Tersembunyi***	Menampilkan judul dari sistem.
L_Subjudul	Label	**Data Simulasi**	Menampilkan identitas dari inputan dari pengguna.
B_*Event*	Button	Data	Menampilkan pilihan *event log* yang akan dijadikan data simulasi.
LV_*Event*	Line View		Menampilkan nama dari *event log* yang dipilih pengguna.
B_Proses	Button	Proses	Menyimpan dan memproses *event log*.

4.2.2 Antarmuka Memasukkan Data Simulasi

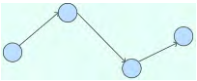
Antarmuka ini digunakan untuk memenuhi kebutuhan pengunaan F-02. Pada antarmuka akhir, proses model akan terbentuk dan ditampilkan setiap waktu yang ditentukan. Gambar antarmuka dapat dilihat pada Gambar 4.6 dan spesifikasi elemen dapat dilihat pada Tabel 4.5.



Gambar 4.6 Antarmuka Akhir Menjalankan Simulasi *Process discovery*

Tabel 4.5 Spesifikasi Elemen pada Antarmuka Akhir Menjalankan Simulasi *Process discovery*

Nama Objek	Jenis Objek	Tampilan Objek	Fungsionalitas
L_Judul	Label	***Process discovery dengan Model Markov Tersembunyi***	Menampilkan judul dari sistem.
B_*Event*Log	Button	Membentuk Model Proses	Mengambil *streaming event log* dan memproses menjadi model

			proses.
L_Asli	Label	**Rangkaian Aktivitas Asli**	Menunjukkan rangkaian aktivitas *streaming event log*.
L_Perbaikan	Label	**Rangkaian Aktivitas Perbaikan**	Menunjukkan rangkaian aktivitas *streaming event log* yang sudah diperbaiki apabila *incomplete trace*.
L_Waktu	Label	**Waktu Proses**	Menunjukkan waktu eksekusi *process discovery*.
TV_Asli	Text View	*Hasil*	Menampilkan hasil sesuai label disampingnya.
G_ModelProses	Graph		Menampilkan model proses.

[Halaman ini sengaja dikosongkan]

BAB V
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bahasa pemrograman yang digunakan adalah bahasa pemrograman Python.

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 : Toshiba.
- Seri : Satellite.
- Prosesor : AMD A8-5545M APU @ 1.70 GHz.
- RAM : 4 GB.

5.1.2 Perangkat Lunak

Perangkat lunak yang mendukung fungsionalitas sistem adalah:

1. Sistem Operasi Windows
   Sistem operasi yang digunakan adalah Microsoft Windows 8.1 Single Languange 64-bit.
2. Python-2.7.11
   Kakas bantu yang digunakan dalam mengembangkan sistem ini adalah python-2.7.11 dengan bahasa pemrograman Python.

3. Django

   Kakas bantu lain yang digunakan dalam mengembangkan sistem ini adalah Django. Django adalah *web framework* Python yang didesain untuk membuat web dinamis.

5.2 Implementasi Proses

Pada subbab ini akan dibahas tentang fungsi yang berjalan pada sistem. Proses yang terjadi dimulai dari pembentukan data *streaming event log* dan membentuk model proses dari *streaming event log* tersebut.

5.2.1 Implementasi Data *Event Log* Lampau

Data *Event Log* Lampau diambil dari *file* Excel yang diunggah oleh pengguna. Penjelasan data *event log* lampau dipaparkan pada subbab 6.2. Setiap *event* pada *event log* terdiri dari Case ID, nama aktivitas, waktu eksekusi awal (*start stamp*) dan waktu eksekusi akhir (*end stamp*). Data *event log* yang diunggah oleh pengguna akan disimpan di sistem dalam bentuk *list* yang berisi nama aktivitas sebanyak Case ID. Untuk memudahkan pengimplementasian algoritma yang memanfaatkan Model Markov Tersembunyi, maka nama aktivitas diubah ke dalam angka sesuai urutan huruf dimana aktivitas A disimbolkan dengan angka 0, begitu seterusnya. *Pseudo-code* implementasi data *event log* lampau dipaparkan pada Gambar 5.1.

Baris kode 1 digunakan untuk menyimpan semua *event* pada data *event log* lampau ke dalam *list stream*. Baris kode 3 digunakan untuk mengubah nama aktivitas pada *event* di *list stream* ke dalam bentuk angka sesuai dengan urutan alphabet, dimana aktivitas A diubah menjadi angka 0 dan begitu seterusnya. Baris kode 4 sampai 6 digunakan untuk menyimpan aktivitas pada *list seq_lampau* untuk aktivitas pertama atau aktivitas yang masih memiliki CaseID yang sama dengan aktivitas sebelumnya. *append*() adalah fungsi untuk menambah *list seq_lampau* dan *addLast()* adalah fungsi untuk menambahkan aktivitas pada *list seq_lampau*. Sedangkan baris kode 7 dan 8 digunakan untuk

menyimpan aktivitas yang memiliki CaseID yang berbeda dengan aktivitas sebelumnya.

Implementasi Data *Event Log* Lampau
Masukan : $file$ EventLogLampau
Keluaran : *list seq_lampau*
1. *stream* ← *event* in EventLogLampau **2.** for *nStream* in [0,…,n_{stream}] do **3.** *act* ← changetoInt(*stream*[*nStream*][‘*activity*’]) **4.** if *stream*[*nStream*][‘CaseID’] = *stream*[*nStream-1*] [‘CaseID’] OR *nStream* = 0 then **5.** *append*(*seq_lampau*) **6.** *addLast*(*seq_lampau*[-1],*act*) **7.** else **8.** *addLast*(*seq_lampau*[-1],*act*)

Gambar 5.1 *Pseudo-code* Implementasi Data *Event Log* Lampau

Keterangan:

EventLogLampau : data *event log* lampau dalam bentuk Excel.

n_{stream} : jumlah baris *event* pada data *event log* lampau.

5.2.2 Implementasi Data *Streaming Event Log*

Data *Streaming Event Log* diambil dari *file* Excel yang diunggah oleh pengguna. Cara pembentukan data *streaming event log* adalah sistem akan menyimpan *event* selanjutnya setelah waktu eksekusi dari *event* sebelumnya selesai. Bentuk data *streaming event log* di dalam sistem sama dengan bentuk data *event log* lampau. *Pseudo-code* implementasi data *streaming event log* dipaparkan pada Gambar 5.2.

Baris kode 1 digunakan untuk menyimpan semua *event* pada data *streaming event log* ke dalam *list stream*. Baris kode 3 digunakan untuk mengubah nama aktivitas pada *event* di *list stream* ke dalam bentuk angka sesuai dengan urutan alphabet, dimana aktivitas A diubah menjadi angka 0 dan begitu seterusnya. Baris kode 4 sampai 6 digunakan untuk menyimpan aktivitas pada *list seq_stream* untuk aktivitas pertama atau aktivitas yang

masih memiliki CaseID yang sama dengan aktivitas sebelumnya. *append*() adalah fungsi untuk menambah *list seq_stream* dan *addLast()* adalah fungsi untuk menambahkan aktivitas pada *list seq_stream*. Sedangkan baris kode 7 dan 8 digunakan untuk menyimpan aktivitas yang memiliki CaseID yang berbeda dengan aktivitas sebelumnya. Baris kode 9 digunakan untuk menghentikan proses penyimpanan aktivitas ke dalam sistem sesuai dengan waktu eksekusi aktivitas.

Implementasi Data *Streaming Event Log*
Masukan : $file$ StreamingEventLog, t_stop
Keluaran : *list seq_stream*
1. *stream* ← *event* in StreamingEventLog **2.** for *nStream* in [0,…,n_{stream}] do **3.** *act* ← changetoInt(*stream*[*nStream*][‘*activity*’]) **4.** if *stream*[*nStream*][‘CaseID’] = *stream*[*nStream-1*] [‘CaseID’] OR *nStream* = 0 then **5.** *append*(*seq_stream*) **6.** *addLast*(*seq_stream*[-1],*act*) **7.** else **8.** *addLast*(*seq_stream*[-1],*act*) **9.** *time.sleep(*t_stop)

Gambar 5.2 *Pseudo-code* Implementasi Data *Streaming Event Log*

Keterangan:

StreamingEventLog : data *streaming event log* dalam bentuk Excel.

n_{stream} : jumlah baris *event* pada data *streaming event log*.

t_stop : waktu eksekusi aktivitas setiap *event*.

5.2.3 Implementasi Algoritma yang memanfaatkan Model Markov Tersembunyi

Sub subbab ini akan menjelaskan mengenai implementasi dari algoritma yang digunakan untuk *process discovery* pada *streaming event log*. Pengimplementasian dari algoritma yang memanfaatkan Model Markov Tersembunyi sesuai dengan

pseudo-code modifikasi Baum-Welch sampai dengan *pseudo-code* penentuan relasi aktivitas.

5.2.3.1 Implementasi Membangun Model Markov Tersembunyi dari *Event Log* Lampau

Implementasi disesuaikan dengan *pseudo-code* Modifikasi Baum-Welch pada subbab 3.2. Implementasi ditunjukkan pada Gambar 5.3.

Baris kode 1 sampai dengan baris kode 8 mengimplementasikan baris kode 1 pada *pseudo-code* Modifikasi Baum-Welch. Apabila aktivitas pada *list stream* yang memiliki waktu eksekusi awal sebelum waktu eksekusi akhir aktivitas sebelumnya, maka aktivitas tersebut akan dikelompokkan dengan *list observe* aktivitas sebelumnya. Selain itu, aktivitas akan dimasukkan dalam *list observe* tersendiri. Baris kode 9 sampai dengan baris kode 15 digunakan untuk membuat inisialisasi awal sebelum pembentukan Model Markov Tersembunyi menggunakan metode Baum-Welch. Baris kode 12 dan 13 menunjukkan bahwa jumlah *state* sama dengan jumlah kelompok *observer* yang ditunjukkan oleh variabel $n_{observe}$. Baris kode 16 mengimplementasikan baris kode 3 pada *pseudo-code* Modifikasi Metode Baum-Welch.

Implementasi Modifikasi Metode Baum-Welch
Masukan : $list$ *seq_lampau, stream*
Keluaran : Model Markov Tersembunyi (A_L, B_L, π_L)
1. for *nStream* in [0,…,n_{stream}] do **2.** *act* ← changetoInt(*stream*[*nStream*][‘*activity*’]) **3.** if *act* not in *observe* then **4.** if (*stream*[*nStream*][‘*start_stamp*’] < *stream*[*nStream-1*][‘*end_stamp*’]) then **5.** *addLast*(*observe*[-1][-1],*act*) **6.** else **7.** *append(observe)* **8.** *addLast*(*observe*[-1][-1],*act*) **9.** for *i* in [0,...,$n_{observe}$] do **10.** for *j* in [0,...,$n_{observe[i]}$] do

```
11.      B_L[i][j] ← 1
12.  for i in [0,...,n_observe] do
13.      for j in [0,...,n_observe] do
14.      A_L[i][j] ← 1
15.  π_L[0][observe[0][0]] ← 1
16.  A_L, B_L, π_L=Baum-Welch(A_L, B_L, π_L, seq_lampau)
```

Gambar 5.3 *Pseudo-code* Implementasi Modifikasi Baum-Welch

Keterangan:

observe	:	*list* data kelompok aktivitas sebagai *observer*.
$n_{observe}$	:	jumlah *list observe*.
seq_lampau	:	*list* penyimpanan keluaran *pseudo-code* Implementasi Data *Event Log* Lampau berupa rangkaian aktivitas.
stream	:	*list event* dari Data *Event Log* Lampau.
$n_{observe[i]}$	:	jumlah *list observe* di kelompok *observer* ke-i.

5.2.3.2 Implementasi Memperbaiki *Incomplete Trace*

Implementasi disesuaikan dengan *pseudo-code* Perbaikan *Incomplete Trace* pada subbab 3.3. Implementasi ditunjukkan pada Gambar 5.4. Penjelasan lengkap mengenai implementasi Perbaikan *Incomplete Trace* dapat dilihat pada subbab 3.3. Baris kode 1 sampai dengan baris kode 8 digunakan untuk memperbaiki *incomplete trace* apabila terpotong di bagian awal dan baris kode 9 sampai dengan baris kode 16 digunakan untuk memperbaiki *incomplete trace* apabila terpotong di bagian akhir. Hasil perbaikan akan disimpan pada *list seq_repair* seperti yang dipaparkan pada baris kode 17.

Implementasi Perbaikan *Incomplete Trace*
Masukan : A_L, B_L, π_L, *seq_stream*
Keluaran : *seq_repair*
//kondisi pertama **1.** $X_0 \leftarrow seq\_stream[0]$ **2.** while $\sum_{i=1}^{N} b_i(X_0) = 0$ do **3.** $remove\ X_0$

4. $X_0 \leftarrow seq\_stream[0]$
5. while $\pi_{stateX_0} = 0$ do
6. $S_S \leftarrow \underset{j=0,..,N}{\text{argmax}}\, Backward(j, stateX_0)$
7. $addFront(seq\_stream, O_{S_S})$
8. $stateX_0 \leftarrow S_S$

//kondisi kedua
9. $X_T \leftarrow seq\_stream[-1]$
10. while $\sum_{i=1}^{N} b_i(X_T) = 0$ do
11. $remove\ X_T$
12. $X_T \leftarrow seq\_stream[-1]$
13. while $\sum_{j=1}^{N} a_{stateX_T j} > 0$ do
14. $S_S \leftarrow Viterbi(stateX_T)$
15. $addLast(seq\_stream, O_{S_S})$
16. $stateX_T j \leftarrow S_S$
17. *seq_repair* ← *seq_stream*

Gambar 5.4 *Pseudo-code* Implementasi Perbaikan *Incomplete Trace*

5.2.3.3 Implementasi Membangun Model Markov Tersembunyi dari *Streaming Event Log*

Implementasi disesuaikan dengan *pseudo-code* Modifikasi Baum-Welch pada subbab 3.4. Implementasi ditunjukkan pada Gambar 5.5. Penjelasan lengkap mengenai implementasi Membangun Model Markov Tersembunyi dari *Streaming Event Log* dapat dilihat pada subbab 3.4. Baris kode 2 digunakan untuk membangun Model Markov Tersembunyi menggunakan metode Baum-Welch.

Implementasi Pembangunan Model Markov Tersembunyi (HMM) berdasarkan *Streaming Event Log*
Masukan : A_L, B_L, π_L, *seq_repair*
Keluaran : Model Markov Tersembunyi dari *Streaming Event Log* (A_S, B_S, π_S) dan Model Markov Tersembunyi dari *Event Log* Lampau (A_L, B_L, π_L)
1. while *seq_repair* ≠ null do

2.	A_s, B_s, π_s=Baum-Welch($A_L, B_L, \pi_L, seq\_repair$)
3.	if length(A_s)>length(A_L) then
4.	$A \leftarrow A_s + A_L$
5.	else
6.	$A \leftarrow A_L + A_s$
7.	if length(B_s)>length(B_L) then
8.	$B \leftarrow B_s + B_L$
9.	else
10.	$B \leftarrow B_L + B_s$
11.	$\pi = \pi_S + \pi_L$
12.	A_L=A
13.	B_L=B
14.	$\pi_L = \pi$

Gambar 5.5 *Pseudo-code* Implementasi Pembangunan Model Markov Tersembunyi (HMM) berdasarkan *Streaming event log*

5.2.3.4 Implementasi Membentuk Model Proses

Implementasi disesuaikan dengan *pseudo-code* Metode Pembentukan Model Proses pada subbab 3.5. Implementasi ditunjukkan pada Gambar 5.6 dan Gambar 5.7. Penjelasan lengkap mengenai Implementasi Penentuan Relasi Aktivitas pada Model Proses dapat dilihat pada sub subbab 3.5.2 dimana variabel *i* mengimplementasikan *state i* (S_i) dan variable *j* mengimplementasikan *state j* (S_j). Pada baris kode 11, 19 dan 22 di *pseudo-code* Implementasi Penentuan Relasi Aktivitas pada Model Proses, fungsi relasi() adalah fungsi yang mengakses *pseudo-code* Implementasi Penentuan relasi XOR, OR, AND dan *Sequence*.

Implementasi Penentuan Relasi Aktivitas pada Model Proses	
Masukan : A_S, B_S,graph	
Keluaran : $graph$	
1.	for i in [0,...,n_{A_S}] do
2.	for j in [0,...,n_{A_S}] do
3.	if $a_{ij} > 0$ then
4.	if $(i \neq j) \vee ((i = j) \wedge n_{O_i} = 1)$ then

```
5.              graph[act_before, act_after]= [O_i , O_j]
6.   for act_before in graph do
7.    if (n_act_after > 1):
8.      if act_before=act_after then
9.        graph[act_before → act_after] = relasi sequence
10.     else
11.       relation = relasi(A_S, B_S,S_act_before, S_act_after)
12.       graph[act_before → act_after] = relation + " Split"
13.   if (n_act_before > 1):
14.     if act_before=act_after then
15.       graph[act_before → act_after] = relasi sequence
16.     else
17.       if  graph[act_before → act_after] = relasi XOR Split then
18.         relation_before = graph[act_before → act_after]
19.         relation = relasi(A_S, B_S,S_act_before, S_act_after)
20.         graph[act_before → act_after] = [relasi_before, relation + "
            Join"]
21.   else
22.     relation = relasi(A_S, B_S,S_act_before, S_act_after)
23.     graph[act_before → act_after] = relation + " Join"
```

Gambar 5.6 *Pseudo-code* Implementasi Penentuan Relasi Aktivitas pada Model Proses

Setiap baris *code* pada Gambar 5.7 mengimplementasika *pseudo-code* Penentuan Relasi XOR, OR, AND, dan *Sequence* yang dapat dilihat pada sub subbab 3.5.1 dimana baris *code* 9 dan 10 mengimplementasikan Persamaan 3.1 dan 3.2.

Hasil keluaran dari *pseudo-code* Implementasi Penentuan Relasi XOR, OR, AND, dan *Sequence* adalah relasi aktivitas yang digunakan pada *pseudo-code* Implementasi Penentuan Relasi Aktivitas pada Model Proses.

Implementasi Penentuan relasi XOR, OR, AND, dan *Sequence*
Masukan : A_S, B_S,S_{act_before}, S_{act_after}
Keluaran : $relation$
1. i = S_{act_before} **2.** j = S_{act_after}

```
3.  if n_BS[i]=1 OR n_BS[j]=1 then
4.    if (n_BS[i]=1 AND n_AS[j]>1) OR (n_BS[j]=1 AND n_AS[i]>1) then
5.      relation = relasi XOR
6.    else
7.      graph[act_before → act_after] = relasi sequence
8.  else
9.      RM(i → j) ← A_S[i][i]/A_S[i][j] × 1/n_BS[i]
10.     avgPP←∑_{i=0}^{n_AS} ∑_{j=0}^{n_AS} A_S[i][j]/n_AS[i][j]
11.   if RM(i → j) < avgPP then
12.     relation ← relasi OR
13.   else
14.     relation ← relasi AND
```

Gambar 5.7 *Pseudo-code* Implementasi Penentuan Relasi XOR, OR, AND, dan *Sequence*

Keterangan:

$n_{B_S[i]}$: jumlah *observer* yang bergantung pada *state i* yang ditunjukkan dengan nilai yang di atas 0 pada matrik B_S bagian i.

$n_{A_S[i]}$: jumlah *state* yang bertransisi dengan *state i* yang ditunjukkan dengan nilai yang di atas 0 pada matrik A_S bagian i.

5.3 Implementasi Antarmuka

Pada subbab ini akan dibahas mengenai implementasii antarmuka sistem berdasarkan rancangan antarmuka yang dibahas pada bab IV.

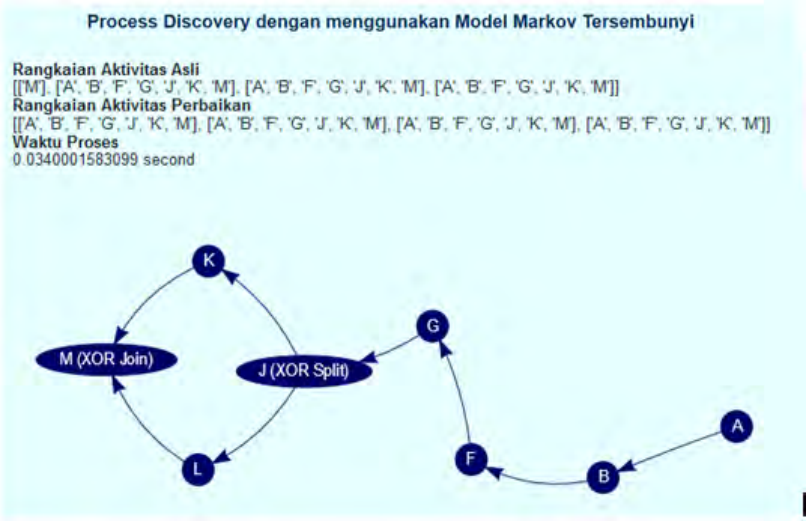
5.3.1 Implementasi Antarmuka Memasukkan Data Simulasi *Process Discovery*

Halaman ini merupakan halaman awal saat pengguna pertama kali mengakses sistem. Halaman ini terdiri dari judul serta tombol yang digunakan untuk mengunggah data *event log* lampau dan *streaming event log*. Halaman antarmuka dapat dilihat pada Gambar 5.8.

Gambar 5.8 Antarmuka Memasukkan Data Simulasi *Process Discovery*

5.3.2 Implementasi Antarmuka Melihat Model Proses

Halaman ini merupakan halaman untuk menampilkan model proses yang merupakan hasil algoritma yang memanfaatkan Model Markov Tersembunyi. Halaman ini terdiri dari tampilan model proses, *incomplete trace* yang telah diperbaiki serta waktu eksekusi algoritma. Halaman antarmuka dapat dilihat pada Gambar 5.9.



Gambar 5.9 Antarmuka Melihat Model Proses

[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 : AMD A8-5545M APU @ 1.70GHz.
   b. Memori (RAM) : 4 GB.
   c. Tipe sistem : 64-bit sistem operasi.
2. Perangkat lunak
   a. Sistem operasi : Windows 8.1 Single Languange.
   b. Kakas bantu : Python-2.7.11.

6.2 Data Studi Kasus

Data studi kasus menggunakan data penanganan ulasan jurnal. Data studi kasus ini didapatkan dari website *http://www.processmining.org/* yang mengalami modifikasi. Modifikasi dilakukan dengan mengubah data asli (berbentuk .xml) menjadi bentuk Excel serta memberi tambahan aktivitas yang dijelaskan pada sub subbab 6.2.2. Modifikasi ini dilakukan karena sistem hanya mampu menerima data Excel serta memunculkan kondisi pertambahan aktivitas di tengah-tengah proses.

Penanganan ulasan jurnal adalah proses penyeleksian diterima tidaknya suatu jurnal. Penyeleksian ini didasarkan pada keputusan dari juri yang ditunjuk. Hasil akhirnya adalah jurnal

tersebut diterima atau jurnal tersebut ditolak. Contoh bentuk asli dari data penanganan jurnal dapat dilihat pada Gambar 6.1. Gambar 6.1 menunjukkan aktivitas *inviting reviewers* pada *case* pertama dengan waktu mulai adalah 01-01-2006 00:00 dan waktu selesai 06-01-2006 00:00.

```
?xml version="1.0" encoding="UTF-8" ?>
<!-- MXML version 1.0 -->
<Source program="CPN Tools simulation"/>
<Process id="DEFAULT" description="Simulated process">
  <ProcessInstance id="1" description="Simulated process
  instance">
    <AuditTrailEntry>
    <WorkflowModelElement>inviting
    reviewers</WorkflowModelElement>
    <EventType >start</EventType>
    <Timestamp>2006-01-01T00:00:00.000+01:00</Timestamp>
    </AuditTrailEntry>
    <AuditTrailEntry>
    <WorkflowModelElement>inviting
    reviewers</WorkflowModelElement>
    <EventType >complete</EventType>
    <Timestamp>2006-01-06T00:00:00.000+01:00</Timestamp>
    </AuditTrailEntry>
  </ProcessInstance>
```

Gambar 6.1 Model Data Asli Penanganan Jurnal

6.2.1 Data *Event Log* Lampau

Data *event log* lampau merupakan 20 *case* dari keseluruhan *case* pada *event log* penanganan ulasan jurnal. *Event log* yang digunakan mengandung informasi *Case ID*, aktivitas, waku mulai pemrosesan aktivitas (*start time*), dan waktu selesai pemrosesan aktivitas (*end time*). Rincian dari aktivitas yang muncul pada data *event log* lampau dipaparkan pada Tabel 6.1 dan potongan *event log* lampau akan dipaparkan pada Tabel 6.2. Model proses yang dari *event log* lampau dapat dilihat pada Gambar 6.2.

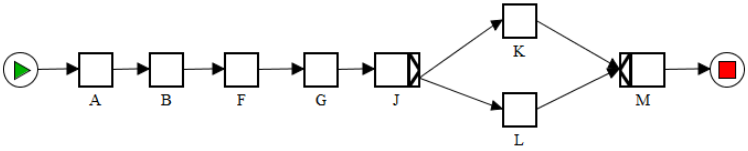
Tabel 6.1 Rincian Aktivitas *Event Log* Lampau

Nama Aktivitas	Kode Aktivitas di *Event log*
Inviting Reviewers	A
Getting Reviews from Judges	B
Collecting Reviews	F
Checking Reviews from Judges	G
Deciding Final Result of Journal based on Reviews	J
Determining The Final Result as Accepted Journal	K
Determining The Final Result as Rejected Journal	L
Annoucing the Final Result	M

Tabel 6.2 Potongan *Event Log* Lampau

Case ID	Activity	Start Stamp	End Stamp
PP1	A	3/8/2016 10:32	3/8/2016 10:35
PP1	B	3/8/2016 10:35	3/8/2016 10:38
PP1	F	3/8/2016 10:38	3/8/2016 10:41
PP1	G	3/8/2016 10:41	3/8/2016 10:44
PP1	J	3/8/2016 10:44	3/8/2016 10:47
PP1	K	3/8/2016 10:47	3/8/2016 10:50
PP1	M	3/8/2016 10:50	3/8/2016 10:53
PP2	A	3/8/2016 10:53	3/8/2016 10:56
PP2	B	3/8/2016 10:56	3/8/2016 10:59
PP2	F	3/8/2016 10:59	3/8/2016 11:02
PP2	G	3/8/2016 11:02	3/8/2016 11:05
PP2	J	3/8/2016 11:05	3/8/2016 11:08
PP2	K	3/8/2016 11:08	3/8/2016 11:11
PP2	M	3/8/2016 11:11	3/8/2016 11:14

PP18	A	3/8/2016 16:29	3/8/2016 16:32
PP18	B	3/8/2016 16:32	3/8/2016 16:35
PP18	F	3/8/2016 16:35	3/8/2016 16:38
PP18	G	3/8/2016 16:38	3/8/2016 16:41
PP18	J	3/8/2016 16:41	3/8/2016 16:44
PP18	L	3/8/2016 16:44	3/8/2016 16:47
PP18	M	3/8/2016 16:47	3/8/2016 16:50
PP19	A	3/8/2016 16:50	3/8/2016 16:53
PP19	B	3/8/2016 16:53	3/8/2016 16:56
PP19	F	3/8/2016 16:56	3/8/2016 16:59
PP19	G	3/8/2016 16:59	3/8/2016 17:02
PP19	J	3/8/2016 17:02	3/8/2016 17:05
PP19	L	3/8/2016 17:05	3/8/2016 17:08
PP19	M	3/8/2016 17:08	3/8/2016 17:11



Gambar 6.2 Model Data *Event log* Lampau

Gambar 6.2 menunjukkan adanya relasi XOR antara aktivitas J dengan K,L serta aktivitas K,L dengan M. Dapat diartikan bahwa K dan L adalah aktivitas pilihan dimana suatu proses hanya boleh menjalankan salah satu dari aktivitas tersebut.

6.2.2 Data *Streaming Event Log*

Data *streaming event log* yang digunakan merupakan *case* selain yang dipakai pada data *event log* lampau dari keseluruhan *case* pada *event log* penanganan ulasan jurnal. Data *streaming event log* yang masih statis akan dijalankan dengan bantuan *threading* pada bahasa pemrograman *python* sehingga membentuk *streaming event log* sebenarnya. Untuk simulasi pada Tugas Akhir ini, setiap lama waktu pengeksekusian untuk setiap aktivitas adalah 0.5 detik.

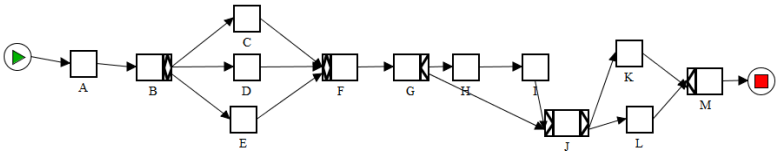
Pengembangan proses terjadi pada data *streaming event log*. Pengembangan proses pertama adalah adanya kebutuhan untuk menambah juri (*inviting additional judges*) dan

mendapatkan keputusan juri tambahan (*getting reviews from additional judges)* setelah pengecekan keputusan juri tetap (*checking reviews from judges*). Hal ini dikarenakan terdapat kemungkinan bahwa keputusan juri tetap tidak cukup sebagai pertimbangan keputusan status jurnal. Oleh karena itu dibutuhkan tambahan juri sebagai tambahan pertimbangan.

Pengembangan proses kedua adalah aktivitas mendapatkan keputusan juri (*getting reviews from judges)*. Juri yang dicantumkan pada *event log* ini adalah tiga juri yaitu juri A, juri B dan juri C. Setiap juri tidak diharuskan untuk mengulas semua jurnal yang ada, oleh karena itu setiap jurnal bisa diulas oleh keseluruhan juri atau beberapa juri saja. Menanggapi kebutuhan tersebut, maka setelah aktivitas *getting reviews from judges* terdapat 3 aktivitas tambahan yang digunakan untuk menjabarkan spesifikasi juri sebagai pemberi ulasan pada jurnal. Rincian dari aktivitas tambahan yang muncul pada data *streaming event log* dipaparkan pada Tabel 6.3 dan model proses yang menggabungkan tambahan aktivitas tersebut dapat dilihat pada Gambar 6.3.

Tabel 6.3 Rincian Aktivitas *Event log* Lampau

Nama Aktivitas	Kode Aktivitas di *Event log*
Getting Reviews from Judges A	C
Getting Reviews from Judges B	D
Getting Reviews from Judges C	E
Inviting Additional Judges	H
Getting Reviews from Additional Judges	I



Gambar 6.3 Model Proses Akhir dari *Streaming Event Log*

Dengan memperhatikan Gambar 6.3, dapat terlihat tambahan relasi pilihan yaitu relasi OR antara aktivitas C, D, E serta relasi XOR antara aktivitas H dan J. Relasi OR diperuntukkan untuk memenuhi kebutuhan pengembangan proses kedua sedangkan relasi XOR diperuntukkan untuk memenuhi kebutuhan pengembangan proses pertama.

6.3 Pengujian Fungsionalitas

Pengujian pada sistem ini mengacu pada pengujian *Blackbox* untuk menguji apakah fungsionalitas sistem telah berjalan sebagaimana mestinya. Pengujian mengacu pada setiap fitur yang telah diimplementasikan. Berdasarkan perancangan antarmuka di bab IV, maka pengujian fitur dibagi menjadi dua yaitu fitur awal menjalankan simulasi *process discovery* dan fitur akhir menjalankan simulasi *process discovery*.

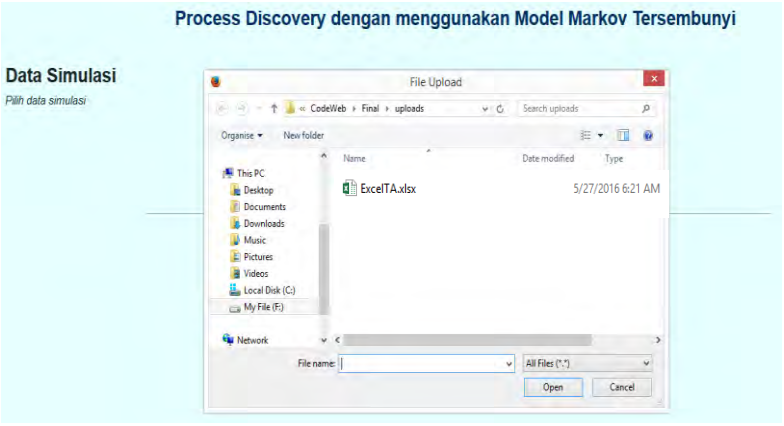
6.3.1 Pengujian Fitur Memasukkan Data Simulasi *Process Discovery*

Pengujian fitur ini adalah memasukkan *file* simulasi berupa *event log* sebagai masukan untuk menjalankan *process discovery*. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.4. Sedangkan tampilan antarmuka yang menunjukkan keberhasilan pengujian dilihat pada Gambar 6.4 dan Gambar 6.5.

Tabel 6.4 Pengujian Fitur Memasukkan Data Simulasi *Process Discovery*

Nama	Pengujian Memasukkan Data Simulasi *Process Discovery*.
Kode	UF-01
Referensi Kasus Penggunaan	F-01.
Tujuan Pengujian	Menguji fitur memasukkan data simulasi *process discovery*.
Skenario	Penguna memilih data simulasi berupa *event log* berformat Excel dan sistem akan menyimpan data simulasi tersebut.

Kondisi awal	Sistem menampilkan halaman utama sistem.
Data Uji	Data uji menggunakan *file event log* penanganan ulasan jurnal yang berada di direktori komputer.
Langkah Pengujian	1. Pengguna memilih data simulasi dan sistem menampilkan direktori komputer. 2. Pengguna memilih *file* data simulasi.
Hasil Yang Diharapkan	Sistem mampu mengenali *file* dengan benar.
Hasil Yang Didapatkan	*File* dikenali dengan baik dan disimpan dalam sistem.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Tampilan nama data simulasi.



Gambar 6.4 Sistem Menampilkan Jendela Pilihan Data Simulasi

Gambar 6.5 Sistem Menampilkan Nama Data Simulasi Pilihan Pengguna

Gambar 6.4 dan Gambar 6.5 menunjukkan bahwa sistem berhasil dalam pengujian F-01, terbukti dengan nama dari *file* pilihan pada Gambar 6.4 sama dengan nama *file* yang ditampilkan pada Gambar 6.5.

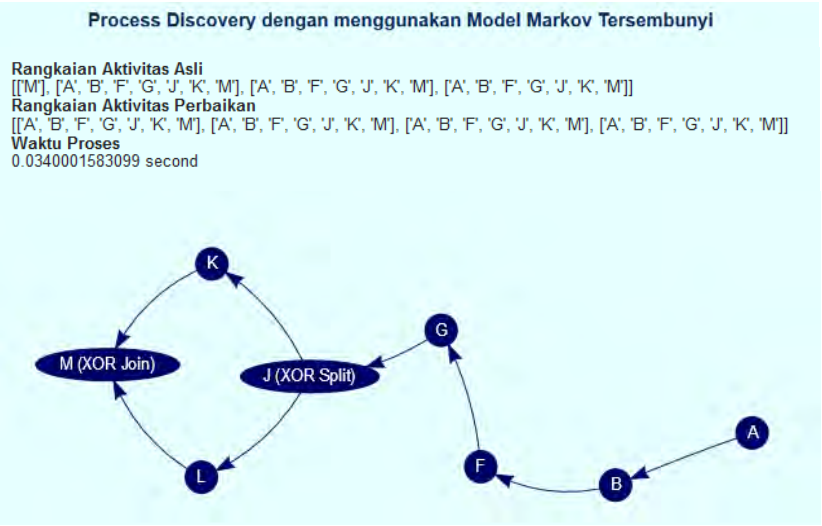
6.3.2 Pengujian Fitur Melihat Model Proses

Pengujian fitur ini adalah menampilkan model proses setiap waktu pemrosesan model proes yang ditentukan. Waktu yang digunakan dalam Tugas Akhir ini adalah 5 detik, 10 detik dan 15 detik. Rincian pengujian fitur ini dapat dilihat pada Tabel 6.5. Sedangkan tampilan antarmuka yang menunjukkan keberhasilan pengujian dilihat pada Gambar 6.6.

Tabel 6.5 Pengujian Fitur Melihat Model Proses

Nama	Pengujian Fitur Melihat Model Proses.
Kode	UF-02
Referensi Kasus Penggunaan	F-01.
Tujuan Pengujian	Menguji fitur menampilkan model proses yang merupakan hasil dari *process discovery*.
Skenario	Sistem menampilkan model proses per waktu yang ditentukan.

Kondisi awal	Sistem menampilkan halaman utama sistem dan sistem menyimpan data simulasi pilihan pengguna.
Data Uji	Data uji menggunakan file *event log* penanganan ulasan jurnal yang berada di direktori komputer.
Langkah Pengujian	1. Pengguna memproses data simulasi. 2. Sistem akan menampilkan model proses sesuai waktu yang ditentukan.
Hasil Yang Diharapkan	Mampu memperbaiki *trace* yang terpotong dan menghasilkan model proses yang tepat.
Hasil Yang Didapatkan	*Trace* yang sudah diperbaiki dan model proses sesuai waktu yang ditentukan.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Model Proses terbentuk.



Gambar 6.6 Sistem Menampilkan Model Proses

6.4 Pengujian Kebenaran Model Proses dan Kualitas Model Proses

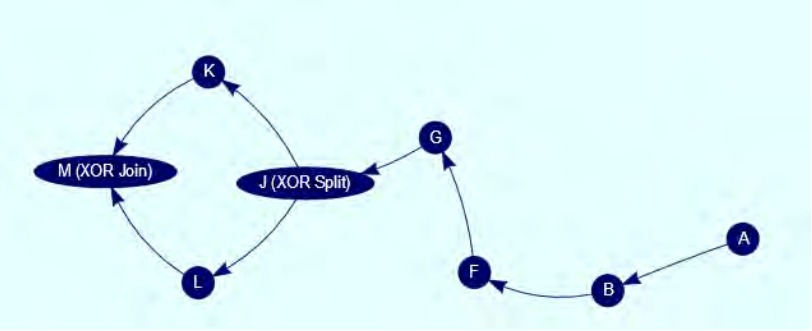
Subbab ini akan memaparkan pengujian kebenaran dan kualitas model proses yang dihasilkan oleh algoritma yang menggunakan Model Markov Tersembunyi dengan algoritma pembanding, *Online Heuristic Miner*. Pengujian ini menggunakan data studi kasus seperti contoh yang dilampirkan pada Lampiran A yang dipecah menjadi tiga studi kasus. Ketiga studi kasus tersebut yaitu: (1) waktu pemrosesan model proses setiap 5 detik, (2) waktu pemrosesan model proses setiap 10 detik, (3) waktu pemrosesan model proses setiap 15 detik.

6.4.1 Pengujian Kebenaran Model Proses

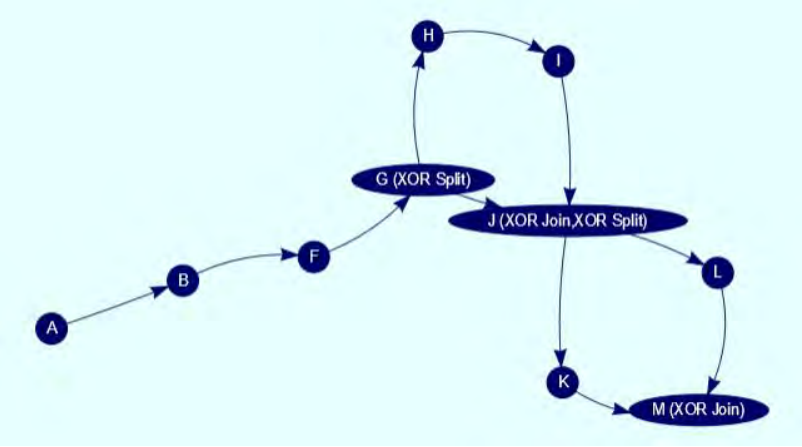
Pengujian Kebenaran Model Proses ini dilakukan dengan cara membandingkan hasil model proses dari algoritma yang menggunakan Model Markov Tersembunyi dan dari algoritma *Online Heuristic Miner* dengan model proses sebenarnya yang dipaparkan pada subbab 6.2. Prosentase kebenaran model proses dihitung dengan cara membagi model proses yang sesuai dengan model proses sebenarnya dengan keseluruhan model proses yang terbentuk. Gambar 6.7 sampai 6.16 menunjukkan model proses yang terbentuk dari kedua algoritma berdasarkan ketiga studi kasus.

Process discovery dari Algoritma yang memanfaatkan Model Markov Tersembunyi menghasilkan empat macam model proses. Gambar 6.7 menunjukkan model proses dengan 2 macam proses yang tergambar, yaitu A→B→F→G→J→K→M dan A→B→ F→G→J→L→M. Kemudian, Gambar 6.8 menunjukkan model proses dengan 4 macam proses yang tergambar, yaitu proses pada Gambar 6.7 dengan tambahan proses baru, yaitu A→B→F→G→H→I→J→K→M dan A→B→F→G→H→I→ J→L→M. Sedangkan, Gambar 6.9 menunjukkan model proses dimana 8 macam proses yang tergambar, yaitu A→B→C→D→F→G→J→K→M, A→B→C→D→F→G→J→ L→M, A→B→C→D→F→G→H→I→J→K→M, A→B→C→

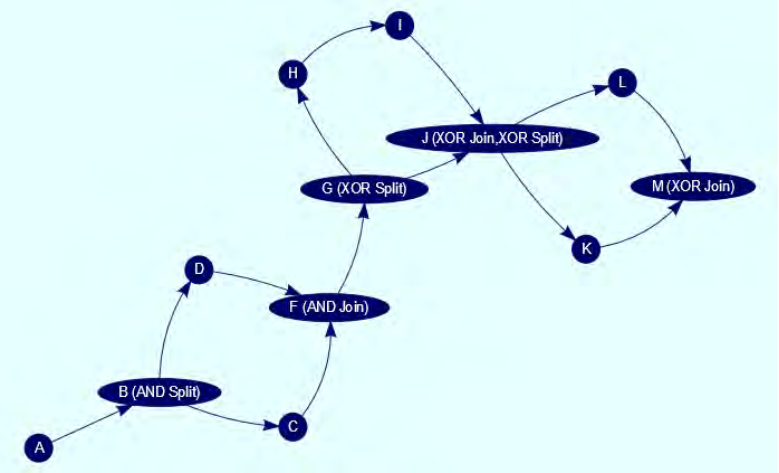
D→F→G→H→I→J→L→M, A→B→D→C→F→G→J→K→M , A→B→D→C→F→G→J→L→M, A→B→D→C→F→G→H →I→J→K→M, dan A→B→D→C→F→G→H→I→J →L→M. Gambar 6.10 menunjukkan model proses dimana proses yang tergambar sama seperti Gambar 6.9 akan tetapi terdapat 6 kemungkinan relasi aktivitas setelah aktivitas B dan sebelum aktivitas F yaitu C, D, E, C→D, C→E, D→E.



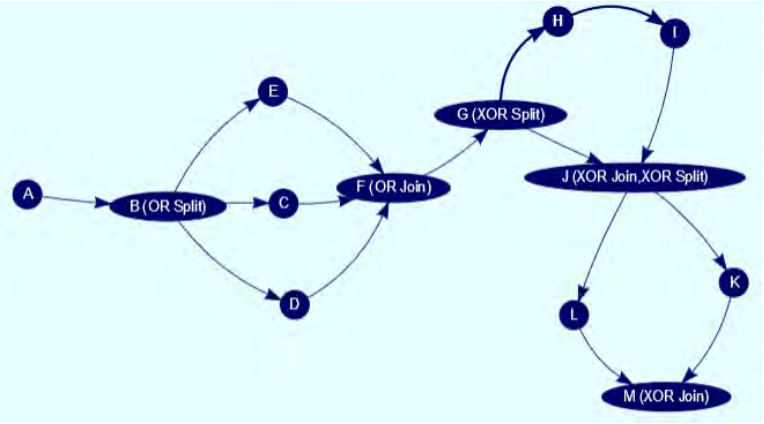
Gambar 6.7 Hasil *Process Discovery* dari Algoritma yang memanfaatkan Model Markov Tersembunyi (1)



Gambar 6.8 Hasil *Process Discovery* dari Algoritma yang memanfaatkan Model Markov Tersembunyi (2)

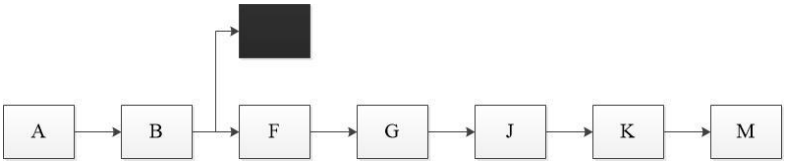


Gambar 6.9 Hasil *Process Discovery* dari Algoritma yang memanfaatkan Model Markov Tersembunyi (3)



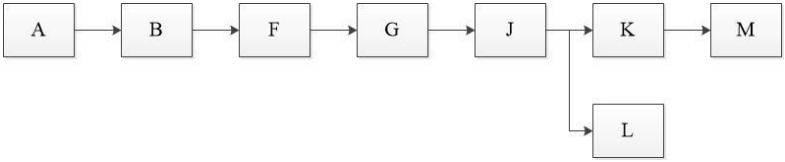
Gambar 6.10 Hasil *Process Discovery* dari Algoritma yang memanfaatkan Model Markov Tersembunyi (4)

Sedangkan, *process discovery* dari Algoritma *Online Heuristic Miner* menghasilkan enam macam model proses.



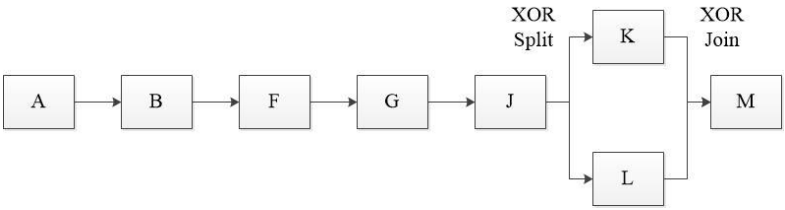
Gambar 6.11 Hasil *Process Discovery* dari Algoritma *Online Heuristic Miner* (1)

Gambar 6.11 menunjukkan model proses dimana terdapat 2 proses yang tergambar, yaitu A→B→F→G→J→K→M dan A→B. Kotak hitam pada Gambar 6.11 menunjukkan akhir suatu proses.



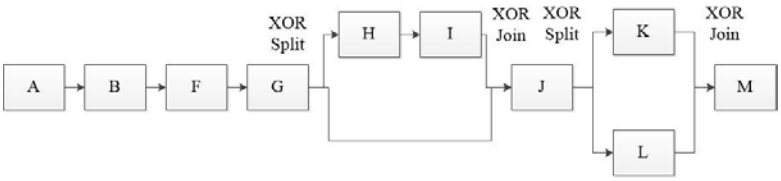
Gambar 6.12 Hasil *Process Discovery* dari Algoritma *Online Heuristic Miner* (2)

Gambar 6.12 menunjukkan model proses dimana terdapat 2 proses yang tergambar, yaitu A→B→F→G→H→I→J→ K→M dan A→ B→F→G→H→I→J→L.



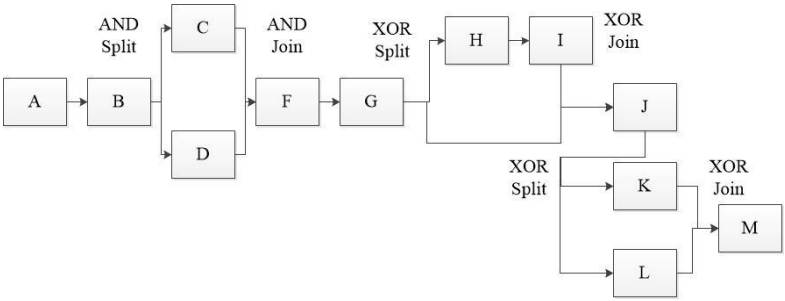
Gambar 6.13 Hasil *Process Discovery* dari Algoritma *Online Heuristic Miner* (3)

Gambar 6.13 menunjukkan model proses dimana terdapat 2 proses yang tergambar, yaitu A→B→F→G→J→K→M dan A→B→F→ G→J→L→M.



Gambar 6.14 Hasil *Process Discovery* dari Algoritma *Online Heuristic Miner* (4)

Gambar 6.14 menunjukkan model proses dimana terdapat 4 proses yang tergambar. Proses tersebut adalah proses pada Gambar 6.13 dengan tambahan proses baru, yaitu A→B→F→ G→H→I→J→K→M dan A→B→F→G→H→I→J →L→M.

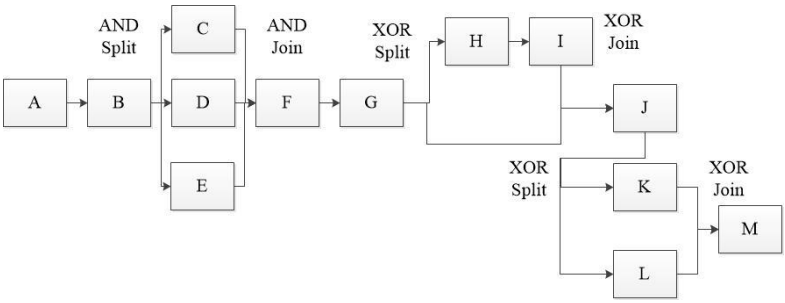


Gambar 6.15 Hasil *Process Discovery* dari Algoritma *Online Heuristic Miner* (5)

Gambar 6.15 menunjukkan model proses dimana proses yang tergambar adalah proses pada Gambar 6.14 dengan 2 variasi alur proses aktivitas setelah aktivitas B dan sebelum aktivitas F. Variasi alur tersebut adalah C→D dan D→C.

Kemudian, Gambar 6.16 menunjukkan model proses dimana proses yang tergambar sama seperti Gambar 6.15 dengan 6 kemungkinan alur proses aktivitas setelah aktivitas B dan

sebelum aktivitas F. Kemungkinan alur proses tersebut yaitu C→D→E, C→E→D, D→C→E, D→E→C, E→C→D, dan E→D→C.



Gambar 6.16 Hasil *Process Discovery* dari Algoritma *Online Heuristic Miner* (6)

Hasil model proses yang sesuai dengan model sebenarnya adalah Gambar 6.7, 6.8, 6.10, 6.13 dan 6.14. Sehingga, dapat disimpulkan bahwa kebenaran model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi adalah 75% sedangkan kebenaran model proses dari algoritma *Online Heuristic Miner* adalah 33%.

6.4.2 Pengujian Kualitas Model Proses

Pada sub subbab ini akan dibahas perhitungan kualitas dari hasil data studi kasus menggunakan algoritma yang memanfaatkan Model Markov Tersembunyi dengan algoritma pembanding, yaitu *Online Heuristic Miner*. Kualitas hasil akan diukur dari empat sisi yaitu *fitness*, presisi, generalisasi dan *simplicity*. Contoh rincian perhitungan kualitas model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi dengan model proses dari algoritma *Online Heuristic Miner* dipaparkan pada bagian Lampiran B. Tabel 6.6 menunjukkan hasil kualitas model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi sedangkan Tabel 6.7 menunjukkan hasil kualitas model proses dari algoritma *Online Heuristic Miner*.

Berdasarkan hasil dari Tabel 6.6 dan 6.7, dapat disimpulkan bahwa kualitas sisi *fitness,* presisi, generalisasi dan *simplicity* model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi lebih baik dibandingkan kualitas model proses dari algoritma *Online Heuristic Miner*. Hal ini dikarenakan nilai kualitas model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi lebih tinggi untuk semua studi kasus dibandingkan dengan dari algoritma *Online Heuristic Miner*.

Kualitas *fitness* dan presisi model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi lebih baik apabila rentang waktu pemrosesan model lebih lama, sedangkan kualitas generalisasi dan *simplicity* model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi lebih baik apabila rentang waktu pemrosesan model lebih singkat. Hal ini dibuktikan dengan nilai *fitness* dan presisi paling tinggi pada studi kasus 3 sedangkan nilai generalisasi dan *simplicity* paling tinggi pada studi kasus 1.

Tabel 6.6 Hasil Akhir Kualitas Model Proses dari Algoritma yang memanfaatkan Model Markov Tersembunyi

SK	Total Q_f	Total Q_p	Total Q_g	Total Q_s
1	0,982	0,448	0,887	0,947
2	0,983	0,435	0,862	0,94
3	0,985	0,553	0,861	0,937

Tabel 6.7 Hasil Akhir Kualitas Model Proses dari Algoritma *Online Heuristic Miner*

SK	Total Q_f	Total Q_p	Total Q_g	Total Q_s
1	0,912	0,406	0,88	0,927
2	0,885	0,375	0,851	0,925
3	0,839	0,461	0,854	0,92

Keterangan Tabel 6.6 dan Tabel 6.7:

Q_f : nilai dari kualitas sisi *fitness*.
Q_p : nilai dari kualitas sisi presisi.
Q_g : nilai dari kualitas sisi generalisasi.
Q_s : nilai dari kualitas sisi *simplicity*.
SK 1 : studi kasus pengujian berdasarkan waktu pemrosesan setiap 5 detik.
SK 2 : studi kasus pengujian berdasarkan waktu pemrosesan setiap 10 detik.
SK 3 : studi kasus pengujian berdasarkan waktu pemrosesan setiap 15 detik.

[Halaman ini sengaja dikosongkan]

LAMPIRAN A

Rincian Beberapa Proses *Process Discovery* menggunakan Algoritma yang Diusulkan pada Studi Kasus 1 (waktu pemrosesan setiap 5 detik)

Waktu streaming: 20:50:17.647000 - 20:50:22.648000
Rangkaian aktivitas asli : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M']]
Waktu eksekusi algoritma : 0,0309998989105 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G'], 'relation': 'XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 20:50:22.648000 - 20:50:27.675000
Rangkaian aktivitas asli : [['F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M']]
Waktu eksekusi algoritma : 0,141999959946 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G'], 'relation': 'XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 20:51:50.645000 - 20:51:55.726000
Rangkaian aktivitas asli : [['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,0859999656677 second

Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 20:51:55.729000 - 20:52:00.817000
Rangkaian aktivitas asli : [['G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'H', 'I']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,0920000076294 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 20:52:47.336000 - 20:52:52.475000
Rangkaian aktivitas asli : [['H', 'I', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'H']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,111000061035 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'AND Split', 'output': ['C', 'D']}
C {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
D {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['C', 'D'], 'relation': 'AND Join', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}

H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 20:53:12.858000 - 20:53:18.174000
Rangkaian aktivitas asli : [['B', 'C', 'E', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'C', 'E', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'J', 'K', 'M']]
Waktu eksekusi algoritma : 0,184000015259 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'OR Split', 'output': ['C', 'D', 'E']}
C {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
D {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
E {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['C', 'D', 'E'], 'relation': 'OR Join', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 20:54:28.564000 - 20:54:33.648000
Rangkaian aktivitas asli : [['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,0649998188019 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'OR Split', 'output': ['C', 'D', 'E']}
C {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
D {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
E {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['C', 'D', 'E'], 'relation': 'OR Join', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}

J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Rincian Beberapa Proses *Process discovery* menggunakan Algoritma yang Diusulkan pada Studi Kasus 2 (waktu pemrosesan setiap 10 detik)

Waktu streaming: 21:01:33.263000 - 21:01:43.264000
Rangkaian aktivitas asli : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M']]
Waktu eksekusi algoritma : 0,0350000858307 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G'], 'relation': 'XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:02:43.480000 - 21:02:53.574000
Rangkaian aktivitas asli : [['B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,166999816895 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G'], 'relation': 'XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:03:13.847000 - 21:03:23.947000
Rangkaian aktivitas asli : [['F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'H', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,149999856949 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:03:43.981000 - 21:03:54.096000
Rangkaian aktivitas asli : [['I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,138999938965 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:05:04.185000 - 21:04:14.185000
Rangkaian aktivitas asli : [['C', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'D', 'E', 'F', 'G', 'J']]

Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'D', 'C', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'D', 'E', 'F', 'G', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,210999965668 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'OR Split', 'output': ['C', 'D', 'E']}
C {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
D {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
E {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['C', 'D', 'E'], 'relation': 'OR Join', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:05:54.210000 - 21:06:04.278000
Rangkaian aktivitas asli : [['G', 'J', 'K', 'M'], ['A', 'B', 'E', 'C', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'D', 'C', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'E', 'C', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,150000095367 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'OR Split', 'output': ['C', 'D', 'E']}
C {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
D {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
E {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['C', 'D', 'E'], 'relation': 'OR Join', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Rincian Beberapa Proses *Process Discovery* menggunakan Algoritma yang Diusulkan pada Studi Kasus 3 (waktu pemrosesan setiap 15 detik)

Waktu streaming: 21:57:50.844000 - 21:58:05.845000
Rangkaian aktivitas asli : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M']]
Waktu eksekusi algoritma : 0,0620000362396 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G'], 'relation': 'XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:58:20.854000 - 21:58:35.859000
Rangkaian aktivitas asli : [['J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'K', 'M']]
Waktu eksekusi algoritma : 0,134000062943 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G'], 'relation': 'XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:59:20.875000 - 21:59:35.878000
Rangkaian aktivitas asli : [['J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B'

, 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,194000005722 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 21:59:35.881000 - 21:59:50.889000
Rangkaian aktivitas asli : [['L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'L', 'M'], ['A', 'B', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'F', 'G', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,164999961853 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['B'], 'relation': 'sequence', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 22:00:35.901000 - 22:00:50.903000
Rangkaian aktivitas asli : [['A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'C', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'C', 'E', 'D', 'F', 'G', 'H', 'I', 'J']]

Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'C', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'M'], ['A', 'B', 'C', 'E', 'D', 'F', 'G', 'H', 'I', 'J', 'K', 'M']]
Waktu eksekusi algoritma : 0,186000108719 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'OR Split', 'output': ['C', 'D', 'E']}
C {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
D {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
E {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['C', 'D', 'E'], 'relation': 'OR Join', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

Waktu streaming: 22:02:20.991000 - 22:02:35.992000
Rangkaian aktivitas asli : [['F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M']]
Rangkaian aktivitas setelah diperbaiki : [['A', 'B', 'D', 'C', 'F', 'G', 'J', 'K', 'M'], ['A', 'B', 'C', 'D', 'F', 'G', 'J', 'L', 'M']]
Waktu eksekusi algoritma : 0,115999937057 second
Graph sekarang:
A {'input': [], 'relation': 'sequence', 'output': ['B']}
B {'input': ['A'], 'relation': 'OR Split', 'output': ['C', 'D', 'E']}
C {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
D {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
E {'input': ['B'], 'relation': 'sequence', 'output': ['F']}
F {'input': ['C', 'D', 'E'], 'relation': 'OR Join', 'output': ['G']}
G {'input': ['F'], 'relation': 'XOR Split', 'output': ['J', 'H']}
H {'input': ['G'], 'relation': 'sequence', 'output': ['I']}
I {'input': ['H'], 'relation': 'sequence', 'output': ['J']}
J {'input': ['G', 'I'], 'relation': 'XOR Join,XOR Split', 'output': ['K', 'L']}
K {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
L {'input': ['J'], 'relation': 'sequence', 'output': ['M']}
M {'input': ['K', 'L'], 'relation': 'XOR Join', 'output': []}

[Halaman ini sengaja dikosongkan]

LAMPIRAN B

Rincian Perhitungan Kualitas *Fitness* pada Studi Kasus 1 dengan Algoritma yang memanfaatkan Model Markov Tersembunyi (waktu pemrosesan setiap 5 detik)

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:50:17.647000 - 20:50:22.648000	2	2	1	0	0	14	1	1
20:50:22.648000 - 20:50:27.675000	2	2	1	0	0	14	1	1
20:50:27.676000 - 20:50:32.705000	3	3	1	0	0	21	1	1
20:50:32.706000 - 20:50:37.728000	3	3	1	0	0	21	1	1
20:50:37.729000 - 20:50:42.825000	2	2	1	0	0	14	1	1
20:50:45.826000 - 20:50:50.899000	3	3	1	0	0	21	1	1
20:50:51.000000 - 20:50:55.033000	3	3	1	0	0	21	1	1
20:50:55.034000 - 20:51:00.194000	3	3	1	0	0	21	1	1
20:51:00.195000 - 20:51:05.275000	2	2	1	0	0	14	1	1
20:51:05.276000 - 20:51:10.025000	3	3	1	0	0	21	1	1
20:51:10.026000 - 20:51:15.044000	3	3	1	0	0	21	1	1

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:51:15.045000 - 20:51:20.324000	2	2	1	0	0	14	1	1
20:51:20.325000 - 20:51:25.083000	2	2	1	0	0	14	1	1
20:51:25.084000 - 20:51:29.923000	2	2	1	0	0	14	1	1
20:51:29.924000 - 20:51:34.996000	2	2	1	0	0	14	1	1
20:51:34.997000 - 20:51:40.199000	2	2	1	0	0	14	1	1
20:51:40.200000 - 20:51:45.299000	3	3	1	0	0	21	1	1
20:51:45.301000 - 20:51:50.644000	2	2	1	0	0	14	1	1
20:51:50.645000 - 20:51:55.726000	2	2	1	0	0	16	1	1
20:52:00.818000 - 20:52:05.925000	2	2	1	0	0	18	1	1
20:52:05.926000 - 20:52:11.224000	3	3	1	0	0	27	1	1
20:52:11.226000 - 20:52:16.031000	2	2	1	0	0	14	1	1
20:52:16.032000 - 20:52:21.375000	2	2	1	0	0	18	1	1
20:52:21.376000 - 20:52:26.777000	3	3	1	0	0	27	1	1
20:52:26.778000 - 20:52:32.036000	2	2	1	0	0	18	1	1
20:52:32.037000 - 20:52:37.361000	2	2	1	0	0	18	1	1
20:52:37.362000 - 20:52:42.045000	3	3	1	0	0	27	1	1
20:52:42.052000 - 20:52:47.335000	2	2	1	0	0	18	1	1

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:52:47.336000 - 20:52:52.475000	2	2	1	0	0	18	1	1
20:52:52.476000 - 20:53:02.481000	2	2	1	0	0	20	1	1
20:53:02.483000 - 20:53:07.803000	2	2	1	0	0	21	1	1
20:53:07.804000 - 20:53:12.857000	1	3	0,3	0	2	21	0,952	0,643
20:53:12.858000 - 20:53:18.174000	2	2	1	0	0	20	1	1
20:53:23.395000 - 20:53:28.074000	3	3	1	0	0	21	1	1
20:53:02.483000 - 20:53:07.803001	2	2	1	0	0	18	1	1
20:53:28.075000 - 20:53:33.229000	2	2	1	0	0	19	1	1
20:53:33.231000 - 20:53:38.063000	2	2	1	0	0	20	1	1
20:53:38.064000 - 20:53:43.309000	2	2	1	0	0	20	1	1
20:53:43.312000 - 20:53:48.549000	3	3	1	0	0	30	1	1
20:53:48.550000 - 20:53:53.348000	2	2	1	0	0	20	1	1
20:53:53.351000 - 20:53:58.086000	2	2	1	0	0	20	1	1
20:53:58.087000 - 20:54:03.097000	3	3	1	0	0	31	1	1
20:54:03.099000 - 20:54:08.412000	1	2	0,5	1	0	21	0,976	0,738
20:54:08.413000 - 20:54:13.356000	2	2	1	0	0	20	1	1
20:54:13.357000 - 20:54:18.097000	1	2	0,5	1	0	21	0,976	0,738

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:54:18.099000 - 20:54:23.103000	2	2	1	0	0	18	1	1
20:54:23.105000 - 20:54:28.562000	2	2	1	0	0	20	1	1
20:54:28.564000 - 20:54:33.648000	1	1	1	0	0	9	1	1

Rincian Perhitungan Kualitas *Fitness* pada Studi Kasus 1 dengan Algoritma *Online Heuristic Miner* (waktu pemrosesan setiap 5 detik)

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:50:17.647000 - 20:50:22.648000	2	1	0,5	0	5	14	0,821	0,661
20:50:22.648000 - 20:50:27.675000	2	1	0,5	0	1	14	0,964	0,732
20:50:27.676000 - 20:50:32.705000	3	3	1	0	0	21	1	1
20:50:32.706000 - 20:50:37.728000	3	3	1	0	0	21	1	1
20:50:37.729000 - 20:50:42.825000	2	2	1	0	0	14	1	1
20:50:45.826000 - 20:50:50.899000	3	3	1	0	0	21	1	1
20:50:51.000000 - 20:50:55.033000	3	3	1	0	0	21	1	1
20:50:55.034000 - 20:51:00.194000	3	3	1	0	0	21	1	1
20:51:00.195000 - 20:51:05.275000	2	2	1	0	0	14	1	1

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:51:05.276000 - 20:51:10.025000	3	3	1	0	0	21	1	1
20:51:10.026000 - 20:51:15.044000	3	3	1	0	0	21	1	1
20:51:15.045000 - 20:51:20.324000	2	2	1	0	0	14	1	1
20:51:20.325000 - 20:51:25.083000	2	2	1	0	0	14	1	1
20:51:25.084000 - 20:51:29.923000	2	2	1	0	0	14	1	1
20:51:29.924000 - 20:51:34.996000	2	2	1	0	0	14	1	1
20:51:34.997000 - 20:51:40.199000	2	2	1	0	0	14	1	1
20:51:40.200000 - 20:51:45.299000	3	3	1	0	0	21	1	1
20:51:45.301000 - 20:51:50.644000	2	2	1	0	0	14	1	1
20:51:50.645000 - 20:51:55.726000	2	2	1	0	0	16	1	1
20:52:00.818000 - 20:52:05.925000	2	2	1	0	0	18	1	1
20:52:05.926000 - 20:52:11.224000	3	3	1	0	0	27	1	1
20:52:11.226000 - 20:52:16.031000	2	2	1	0	0	14	1	1
20:52:16.032000 - 20:52:21.375000	2	2	1	0	0	18	1	1
20:52:21.376000 - 20:52:26.777000	3	3	1	0	0	27	1	1
20:52:26.778000 - 20:52:32.036000	2	2	1	0	0	18	1	1
20:52:32.037000 - 20:52:37.361000	2	2	1	0	0	18	1	1

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:52:37.362000 - 20:52:42.045000	3	3	1	0	0	27	1	1
20:52:42.052000 - 20:52:47.335000	2	2	1	0	0	18	1	1
20:52:47.336000 - 20:52:52.475000	2	2	1	0	0	18	1	1
20:52:52.476000 - 20:53:02.481000	2	1	0,5	0	2	20	0,95	0,725
20:53:02.483000 - 20:53:07.803000	2	1	0,5	0	1	21	0,976	0,738
20:53:07.804000 - 20:53:12.857000	3	2	0,67	0	1	21	0,976	0,821
20:53:12.858000 - 20:53:18.174000	2	1	0,5	0	1	20	0,975	0,738
20:53:23.395000 - 20:53:28.074000	3	2	0,667	0	1	21	0,976	0,821
20:53:02.483000 - 20:53:07.803001	2	2	1	0	0	18	1	1
20:53:28.075000 - 20:53:33.229000	2	2	1	0	0	19	1	1
20:53:33.231000 - 20:53:38.063000	2	2	1	0	0	20	1	1
20:53:38.064000 - 20:53:43.309000	2	1	0,5	0	1	20	0,975	0,738
20:53:43.312000 - 20:53:48.549000	3	2	0,667	0	1	30	0,983	0,825
20:53:48.550000 - 20:53:53.348000	2	1	0,5	0	1	20	0,975	0,738
20:53:53.351000 - 20:53:58.086000	2	2	1	0	0	20	1	1
20:53:58.087000 - 20:54:03.097000	3	2	0,667	0	1	31	0,984	0,825
20:54:03.099000 - 20:54:08.412000	2	1	0,5	1	1	21	0,952	0,726

Waktu Proses	c	t	PM	r	m	e	CPM	Q_f (Nilai *fitness)*
20:54:08.413000 - 20:54:13.356000	2	1	0,5	0	1	20	0,975	0,738
20:54:13.357000 - 20:54:18.097000	2	1	0,5	1	1	21	0,952	0,726
20:54:18.099000 - 20:54:23.103000	2	1	0,5	0	1	18	0,972	0,736
20:54:23.105000 - 20:54:28.562000	2	2	1	0	0	20	1	1
20:54:28.564000 - 20:54:33.648000	1	0	0	0	1	9	0,944	0,472

Keterangan:

c : jumlah *trace* dari *event log* yang ditampilkan pada model proses.
t : total *trace* dari *event log*.
m : jumlah aktivitas pada *event log* yang tidak diimplementasikan di model proses.
r : jumlah aktivitas yang ada pada *trace* di *event* log setelah aktivitas akhir yang digambarkan di model proses.
e : total aktivitas pada *event log*.
CPM : *Continuous Parsing Measure*.
PM : *Parsing Measure*.

Rincian Perhitungan Kualitas Presisi pada Studi Kasus 1 dengan Algoritma yang memanfaatkan Model Markov Tersembunyi
(waktu pemrosesan setiap 5 detik)

Waktu Proses	*tp*	*p'*	Q_p (Nilai presisi)
20:50:17.647000 - 20:50:22.648000	1	2	0,5
20:50:22.648000 - 20:50:27.675000	1	2	0,5
20:50:27.676000 - 20:50:32.705000	1	2	0,5
20:50:32.706000 - 20:50:37.728000	1	2	0,5
20:50:37.729000 - 20:50:42.825000	1	2	0,5
20:50:45.826000 - 20:50:50.899000	1	2	0,5
20:50:51.000000 - 20:50:55.033000	2	2	1
20:50:55.034000 - 20:51:00.194000	2	2	1
20:51:00.195000 - 20:51:05.275000	2	2	1
20:51:05.276000 - 20:51:10.025000	2	2	1
20:51:10.026000 - 20:51:15.044000	2	2	1
20:51:15.045000 - 20:51:20.324000	1	2	0,5

Waktu Proses	*tp*	*p'*	Q_p (Nilai presisi*)*
20:51:20.325000 - 20:51:25.083000	1	2	0,5
20:51:25.084000 - 20:51:29.923000	2	2	1
20:51:29.924000 - 20:51:34.996000	1	2	0,5
20:51:34.997000 - 20:51:40.199000	2	2	1
20:51:40.200000 - 20:51:45.299000	1	2	0,5
20:51:45.301000 - 20:51:50.644000	2	4	0,5
20:51:50.645000 - 20:51:55.726000	2	4	0,5
20:51:55.729000 - 20:52:00.817000	2	4	0,5
20:52:00.818000 - 20:52:05.925000	2	4	0,5
20:52:05.926000 - 20:52:11.224000	2	4	0,5
20:52:11.226000 - 20:52:16.031000	2	4	0,5
20:52:16.032000 - 20:52:21.375000	1	4	0,25
20:52:21.376000 - 20:52:26.777000	2	4	0,5
20:52:26.778000 - 20:52:32.036000	2	4	0,5
20:52:32.037000 - 20:52:37.361000	2	4	0,5
20:52:37.362000 - 20:52:42.045000	2	4	0,5

Waktu Proses	*tp*	*p'*	Q_p (Nilai presisi)
20:52:42.052000 - 20:52:47.335000	1	4	0,25
20:52:47.336000 - 20:52:52.475000	1	8	0,125
20:52:52.476000 - 20:53:02.481000	2	12	0,167
20:53:02.483000 - 20:53:07.803000	2	12	0,167
20:53:07.804000 - 20:53:12.857000	2	12	0,167
20:53:12.858000 - 20:53:18.174000	2	12	0,167
20:53:18.176000 - 20:53:23.394000	2	12	0,167
20:53:23.395000 - 20:53:28.074000	2	12	0,167
20:53:28.075000 - 20:53:33.229000	2	12	0,167
20:53:33.231000 - 20:53:38.063000	2	12	0,167
20:53:38.064000 - 20:53:43.309000	2	12	0,167
20:53:43.312000 - 20:53:48.549000	3	12	0,25
20:53:48.550000 - 20:53:53.348000	2	12	0,167
20:53:53.351000 - 20:53:58.086000	2	12	0,167
20:53:58.087000 - 20:54:03.097000	3	12	0,25
20:54:03.099000 - 20:54:08.412000	2	12	0,167

Waktu Proses	*tp*	*p'*	Q_p (Nilai presisi)
20:54:08.413000 - 20:54:13.356000	3	12	0,25
20:54:13.357000 - 20:54:18.097000	2	12	0,167
20:54:18.099000 - 20:54:23.103000	2	12	0,167
20:54:23.105000 - 20:54:28.562000	2	12	0,167
20:54:28.564000 - 20:54:33.648000	1	12	0,083

Rincian Perhitungan Kualitas Presisi pada Studi Kasus 1 dengan Algoritma *Online Heuristic Miner* (waktu pemrosesan setiap 5 detik)

Waktu Proses	*tp*	*p'*	Q_p (Nilai presisi)
20:50:17.647000 - 20:50:22.648000	1	2	0,5
20:50:22.648000 - 20:50:27.675000	1	2	0,5
20:50:27.676000 - 20:50:32.705000	2	2	1
20:50:32.706000 - 20:50:37.728000	1	2	0,5
20:50:37.729000 - 20:50:42.825000	1	2	0,5
20:50:45.826000 - 20:50:50.899000	1	2	0,5

Waktu Proses	***tp***	***p'***	Q_p **(Nilai presisi)**
20:50:51.000000 - 20:50:55.033000	2	2	1
20:50:55.034000 - 20:51:00.194000	2	2	1
20:51:00.195000 - 20:51:05.275000	2	2	1
20:51:05.276000 - 20:51:10.025000	2	2	1
20:51:10.026000 - 20:51:15.044000	2	2	1
20:51:15.045000 - 20:51:20.324000	1	2	0,5
20:51:20.325000 - 20:51:25.083000	1	2	0,5
20:51:25.084000 - 20:51:29.923000	2	2	1
20:51:29.924000 - 20:51:34.996000	1	2	0,5
20:51:34.997000 - 20:51:40.199000	2	2	1
20:51:40.200000 - 20:51:45.299000	1	2	0,5
20:51:45.301000 - 20:51:50.644000	2	4	0,5
20:51:50.645000 - 20:51:55.726000	2	4	0,5
20:51:55.729000 - 20:52:00.817000	2	4	0,5
20:52:00.818000 - 20:52:05.925000	2	4	0,5
20:52:05.926000 - 20:52:11.224000	2	4	0,5

Waktu Proses	*tp*	*p'*	Q_p (Nilai presisi)
20:52:11.226000 - 20:52:16.031000	2	4	0,5
20:52:16.032000 - 20:52:21.375000	1	4	0,25
20:52:21.376000 - 20:52:26.777000	2	4	0,5
20:52:26.778000 - 20:52:32.036000	2	4	0,5
20:52:32.037000 - 20:52:37.361000	2	4	0,5
20:52:37.362000 - 20:52:42.045000	2	4	0,5
20:52:42.052000 - 20:52:47.335000	1	4	0,25
20:52:47.336000 - 20:52:52.475000	1	8	0,125
20:52:52.476000 - 20:53:02.481000	1	12	0,083
20:53:02.483000 - 20:53:07.803000	1	12	0,083
20:53:07.804000 - 20:53:12.857000	1	12	0,083
20:53:12.858000 - 20:53:18.174000	1	12	0,083
20:53:18.176000 - 20:53:23.394000	1	12	0,083
20:53:23.395000 - 20:53:28.074000	1	12	0,083
20:53:28.075000 - 20:53:33.229000	1	12	0,083
20:53:33.231000 - 20:53:38.063000	1	12	0,083

Waktu Proses	*tp*	*p'*	Q_p (Nilai presisi)
20:53:38.064000 - 20:53:43.309000	1	12	0,083
20:53:43.312000 - 20:53:48.549000	2	12	0,167
20:53:48.550000 - 20:53:53.348000	1	12	0,083
20:53:53.351000 - 20:53:58.086000	1	12	0,083
20:53:58.087000 - 20:54:03.097000	2	12	0,167
20:54:03.099000 - 20:54:08.412000	1	12	0,083
20:54:08.413000 - 20:54:13.356000	2	12	0,167
20:54:13.357000 - 20:54:18.097000	1	12	0,083
20:54:18.099000 - 20:54:23.103000	1	12	0,083
20:54:23.105000 - 20:54:28.562000	1	12	0,083
20:54:28.564000 - 20:54:33.648000	0	12	0

Keterangan:

tp : jumlah dari *true positive* (*trace* pada *event log* digambarkan di model proses).

p' : jumlah dari *true positive* dan *false negative* (seluruh *trace* yang digambarkan di model proses).

Rincian Perhitungan Kualitas Generalisasi pada Studi Kasus 1 dengan Algoritma yang memanfaatkan Model Markov Tersembunyi
(waktu pemrosesan setiap 5 detik)

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:50:17.647000 - 20:50:22.648000	0,7	7	0,9
20:50:22.648000 - 20:50:27.675000	0,7	7	0,9
20:50:27.676000 - 20:50:32.705000	0,7	7	0,9
20:50:32.706000 - 20:50:37.728000	0,7	7	0,9
20:50:37.729000 - 20:50:42.825000	0,7	7	0,9
20:50:45.826000 - 20:50:50.899000	0,7	7	0,9
20:50:51.000000 - 20:50:55.033000	0,7	7	0,9
20:50:55.034000 - 20:51:00.194000	0,7	7	0,9
20:51:00.195000 - 20:51:05.275000	0,7	7	0,9
20:51:05.276000 - 20:51:10.025000	0,7	7	0,9
20:51:10.026000 - 20:51:15.044000	0,7	7	0,9
20:51:15.045000 - 20:51:20.324000	0,7	7	0,9
20:51:20.325000 - 20:51:25.083000	0,7	7	0,9
20:51:25.084000 - 20:51:29.923000	0,7	7	0,9

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:51:29.924000 - 20:51:34.996000	0,7	7	0,9
20:51:34.997000 - 20:51:40.199000	0,7	7	0,9
20:51:40.200000 - 20:51:45.299000	0,7	7	0,9
20:51:45.301000 - 20:51:50.644000	0,7	7	0,9
20:51:50.645000 - 20:51:55.726000	1,08	10	0,892
20:51:55.729000 - 20:52:00.817000	1	10	0,9
20:52:00.818000 - 20:52:05.925000	1	10	0,9
20:52:05.926000 - 20:52:11.224000	1	10	0,9
20:52:11.226000 - 20:52:16.031000	1	10	0,9
20:52:16.032000 - 20:52:21.375000	1	10	0,9
20:52:21.376000 - 20:52:26.777000	1	10	0,9
20:52:26.778000 - 20:52:32.036000	1	10	0,9
20:52:32.037000 - 20:52:37.361000	1	10	0,9
20:52:37.362000 - 20:52:42.045000	1	10	0,9
20:52:42.052000 - 20:52:47.335000	1	10	0,9
20:52:47.336000 - 20:52:52.475000	1,49	12	0,876

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:52:52.476000 - 20:53:02.481000	1,49	12	0,876
20:53:02.483000 - 20:53:07.803000	1,76	12	0,854
20:53:07.804000 - 20:53:12.857000	1,76	12	0,854
20:53:12.858000 - 20:53:18.174000	1,2	12	0,9
20:53:18.176000 - 20:53:23.394000	1,28	12	0,893
20:53:23.395000 - 20:53:28.074000	1,4	12	0,883
20:53:28.075000 - 20:53:33.229000	1,57	12	0,869
20:53:33.231000 - 20:53:38.063000	1,28	12	0,893
20:53:38.064000 - 20:53:43.309000	1,28	12	0,893
20:53:43.312000 - 20:53:48.549000	1,4	12	0,883
20:53:48.550000 - 20:53:53.348000	1,28	12	0,893
20:53:53.351000 - 20:53:58.086000	1,28	12	0,893
20:53:58.087000 - 20:54:03.097000	1,25	12	0,896
20:54:03.099000 - 20:54:08.412000	1,57	12	0,869
20:54:08.413000 - 20:54:13.356000	1,25	12	0,896
20:54:13.357000 - 20:54:18.097000	1,57	12	0,869

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:54:18.099000 - 20:54:23.103000	3	12	0,75
20:54:23.105000 - 20:54:28.562000	1,28	12	0,893
20:54:28.564000 - 20:54:33.648000	3	12	0,75

Rincian Perhitungan Kualitas Generalisasi pada Studi Kasus 1 dengan Algoritma *Online Heuristic Miner*
(waktu pemrosesan setiap 5 detik)

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:50:17.647000 - 20:50:22.648000	0,783	7	0,888
20:50:22.648000 - 20:50:27.675000	0,783	7	0,888
20:50:27.676000 - 20:50:32.705000	0,7	7	0,9
20:50:32.706000 - 20:50:37.728000	0,7	7	0,9
20:50:37.729000 - 20:50:42.825000	0,7	7	0,9
20:50:45.826000 - 20:50:50.899000	0,7	7	0,9
20:50:51.000000 - 20:50:55.033000	0,7	7	0,9
20:50:55.034000 - 20:51:00.194000	0,7	7	0,9

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:51:00.195000 - 20:51:05.275000	0,7	7	0,9
20:51:05.276000 - 20:51:10.025000	0,7	7	0,9
20:51:10.026000 - 20:51:15.044000	0,7	7	0,9
20:51:15.045000 - 20:51:20.324000	0,7	7	0,9
20:51:20.325000 - 20:51:25.083000	0,7	7	0,9
20:51:25.084000 - 20:51:29.923000	0,7	7	0,9
20:51:29.924000 - 20:51:34.996000	0,7	7	0,9
20:51:34.997000 - 20:51:40.199000	0,7	7	0,9
20:51:40.200000 - 20:51:45.299000	0,7	7	0,9
20:51:45.301000 - 20:51:50.644000	0,7	7	0,9
20:51:50.645000 - 20:51:55.726000	1,08	10	0,892
20:51:55.729000 - 20:52:00.817000	1	10	0,9
20:52:00.818000 - 20:52:05.925000	1	10	0,9
20:52:05.926000 - 20:52:11.224000	1	10	0,9
20:52:11.226000 - 20:52:16.031000	1	10	0,9
20:52:16.032000 - 20:52:21.375000	1	10	0,9
20:52:21.376000 - 20:52:26.777000	1	10	0,9

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:52:26.778000 - 20:52:32.036000	1	10	0,9
20:52:32.037000 - 20:52:37.361000	1	10	0,9
20:52:37.362000 - 20:52:42.045000	1	10	0,9
20:52:42.052000 - 20:52:47.335000	1	10	0,9
20:52:47.336000 - 20:52:52.475000	1,49	12	0,876
20:52:52.476000 - 20:53:02.481000	1,49	12	0,876
20:53:02.483000 - 20:53:07.803000	1,44	12	0,88
20:53:07.804000 - 20:53:12.857000	1,44	12	0,88
20:53:12.858000 - 20:53:18.174000	1,57	12	0,869
20:53:18.176000 - 20:53:23.394000	1,57	12	0,869
20:53:23.395000 - 20:53:28.074000	1,76	12	0,854
20:53:28.075000 - 20:53:33.229000	1,57	12	0,869
20:53:33.231000 - 20:53:38.063000	1,57	12	0,869
20:53:38.064000 - 20:53:43.309000	1,57	12	0,869
20:53:43.312000 - 20:53:48.549000	1,76	12	0,854
20:53:48.550000 - 20:53:53.348000	1,57	12	0,869

Waktu Proses	total akar kuadrat eksekusi	#nt	Q_g (Nilai generalisasi)
20:53:53.351000 - 20:53:58.086000	1,57	12	0,869
20:53:58.087000 - 20:54:03.097000	1,76	12	0,854
20:54:03.099000 - 20:54:08.412000	1,57	12	0,869
20:54:08.413000 - 20:54:13.356000	1,76	12	0,854
20:54:13.357000 - 20:54:18.097000	1,57	12	0,869
20:54:18.099000 - 20:54:23.103000	3,29	12	0,726
20:54:23.105000 - 20:54:28.562000	1,57	12	0,869
20:54:28.564000 - 20:54:33.648000	3,29	12	0,726

Keterangan:

total akar kuadrat eksekusi : hasil dari rumus generalisasi bagian $\sum_{1}^{nt}\left(\sqrt{\#executions}\right)^{-1}$.

$Q_g = 1 - \frac{1}{\#nt}$

#nt : jumlah *operator node* pada *process tree*.

Rincian Perhitungan Kualitas *Simplicity* pada Studi Kasus 1 dengan Algoritma yang memanfaatkan Model Markov Tersembunyi (waktu pemrosesan setiap 5 detik)

Waktu Proses	*#da*	*#ma*	*#npt*	*#eventclass*	Q_s (Nilai *simplicity)*
20:50:17.647000 - 20:50:22.648000	0	1	8	7	0,933
20:50:22.648000 - 20:50:27.675000	0	1	8	7	0,933
20:50:27.676000 - 20:50:32.705000	0	1	8	7	0,933
20:50:32.706000 - 20:50:37.728000	0	1	8	7	0,933
20:50:37.729000 - 20:50:42.825000	0	1	8	7	0,933
20:50:45.826000 - 20:50:50.899000	0	1	8	7	0,933
20:50:51.000000 - 20:50:55.033000	0	0	8	8	1
20:50:55.034000 - 20:51:00.194000	0	0	8	8	1
20:51:00.195000 - 20:51:05.275000	0	0	8	8	1
20:51:05.276000 - 20:51:10.025000	0	0	8	8	1
20:51:10.026000 - 20:51:15.044000	0	0	8	8	1
20:51:15.045000 - 20:51:20.324000	0	1	8	7	0,933
20:51:20.325000 - 20:51:25.083000	0	1	8	7	0,933

Waktu Proses	*#da*	*#ma*	*#npt*	*#eventclass*	Q_s (Nilai *simplicity)*
20:51:25.084000 - 20:51:29.923000	0	0	8	8	1
20:51:29.924000 - 20:51:34.996000	0	0	8	8	1
20:51:34.997000 - 20:51:40.199000	0	0	8	8	1
20:51:40.200000 - 20:51:45.299000	0	1	8	7	0,933
20:51:45.301000 - 20:51:50.644000	0	1	8	7	0,933
20:51:50.645000 - 20:51:55.726000	1	1	11	9	0,9
20:51:55.729000 - 20:52:00.817000	1	0	11	10	0,952
20:52:00.818000 - 20:52:05.925000	1	0	11	10	0,952
20:52:05.926000 - 20:52:11.224000	1	0	11	10	0,952
20:52:11.226000 - 20:52:16.031000	1	0	11	10	0,952
20:52:16.032000 - 20:52:21.375000	1	1	11	9	0,9
20:52:21.376000 - 20:52:26.777000	1	0	11	10	0,952
20:52:26.778000 - 20:52:32.036000	1	0	11	10	0,952
20:52:32.037000 - 20:52:37.361000	1	1	11	9	0,9
20:52:37.362000 - 20:52:42.045000	1	0	11	10	0,952
20:52:42.052000 - 20:52:47.335000	1	1	11	9	0,9

Waktu Proses	*#da*	*#ma*	*#npt*	*#eventclass*	Q_s (Nilai *simplicity)*
20:52:47.336000 - 20:52:52.475000	1	0	13	12	0,96
20:52:52.476000 - 20:53:02.481000	1	1	14	12	0,923
20:53:02.483000 - 20:53:07.803000	1	0	14	13	0,963
20:53:07.804000 - 20:53:12.857000	1	0	14	13	0,963
20:53:12.858000 - 20:53:18.174000	1	0	14	13	0,963
20:53:18.176000 - 20:53:23.394000	1	0	14	13	0,963
20:53:23.395000 - 20:53:28.074000	1	0	14	13	0,963
20:53:28.075000 - 20:53:33.229000	1	0	14	13	0,963
20:53:33.231000 - 20:53:38.063000	1	0	14	13	0,963
20:53:38.064000 - 20:53:43.309000	1	1	14	12	0,923
20:53:43.312000 - 20:53:48.549000	1	1	14	12	0,923
20:53:48.550000 - 20:53:53.348000	1	0	14	13	0,963
20:53:53.351000 - 20:53:58.086000	1	1	14	12	0,923
20:53:58.087000 - 20:54:03.097000	1	0	14	13	0,963
20:54:03.099000 - 20:54:08.412000	1	1	14	12	0,923
20:54:08.413000 - 20:54:13.356000	1	0	14	13	0,963

Waktu Proses	*#da*	*#ma*	*#npt*	*#eventclass*	Q_s (Nilai *simplicity)*
20:54:13.357000 - 20:54:18.097000	1	1	14	12	0,923
20:54:18.099000 - 20:54:23.103000	1	2	14	11	0,88
20:54:23.105000 - 20:54:28.562000	1	1	14	12	0,923
20:54:28.564000 - 20:54:33.648000	1	4	14	9	0,783

Rincian Perhitungan Kualitas *Simplicity* pada Studi Kasus 1 dengan Algoritma *Online Heuristic Miner*
(waktu pemrosesan setiap 5 detik)

Waktu Proses	*#da*	*#ma*	*#npt*	*#eventclass*	Q_s (Nilai *simplicity)*
20:50:17.647000 - 20:50:22.648000	0	5	7	7	0,643
20:50:22.648000 - 20:50:27.675000	0	1	7	7	0,929
20:50:27.676000 - 20:50:32.705000	0	0	8	7	1
20:50:32.706000 - 20:50:37.728000	0	1	8	7	0,933
20:50:37.729000 - 20:50:42.825000	0	1	8	7	0,933
20:50:45.826000 - 20:50:50.899000	0	1	8	7	0,933
20:50:51.000000 - 20:50:55.033000	0	0	8	8	1

Waktu Proses	***#da***	***#ma***	***#npt***	***#eventclass***	Q_s **(Nilai *simplicity)***
20:50:55.034000 - 20:51:00.194000	0	0	8	8	1
20:51:00.195000 - 20:51:05.275000	0	0	8	8	1
20:51:05.276000 - 20:51:10.025000	0	0	8	8	1
20:51:10.026000 - 20:51:15.044000	0	0	8	8	1
20:51:15.045000 - 20:51:20.324000	0	1	8	7	0,933
20:51:20.325000 - 20:51:25.083000	0	1	8	7	0,933
20:51:25.084000 - 20:51:29.923000	0	0	8	8	1
20:51:29.924000 - 20:51:34.996000	0	0	8	8	1
20:51:34.997000 - 20:51:40.199000	0	0	8	8	1
20:51:40.200000 - 20:51:45.299000	0	1	8	7	0,933
20:51:45.301000 - 20:51:50.644000	0	1	8	7	0,933
20:51:50.645000 - 20:51:55.726000	1	1	11	9	0,9
20:51:55.729000 - 20:52:00.817000	1	0	11	10	0,952
20:52:00.818000 - 20:52:05.925000	1	0	11	10	0,952
20:52:05.926000 - 20:52:11.224000	1	0	11	10	0,952
20:52:11.226000 - 20:52:16.031000	1	0	11	10	0,952

Waktu Proses	*#da*	*#ma*	*#npt*	*#eventclass*	Q_s (Nilai *simplicity)*
20:52:16.032000 - 20:52:21.375000	1	1	11	9	0,9
20:52:21.376000 - 20:52:26.777000	1	0	11	10	0,952
20:52:26.778000 - 20:52:32.036000	1	0	11	10	0,952
20:52:32.037000 - 20:52:37.361000	1	1	11	9	0,9
20:52:37.362000 - 20:52:42.045000	1	0	11	10	0,952
20:52:42.052000 - 20:52:47.335000	1	1	11	9	0,9
20:52:47.336000 - 20:52:52.475000	1	0	13	12	0,96
20:52:52.476000 - 20:53:02.481000	1	1	13	12	0,92
20:53:02.483000 - 20:53:07.803000	1	0	13	13	0,962
20:53:07.804000 - 20:53:12.857000	1	0	13	13	0,962
20:53:12.858000 - 20:53:18.174000	1	1	13	13	0,923
20:53:18.176000 - 20:53:23.394000	1	1	13	13	0,923
20:53:23.395000 - 20:53:28.074000	1	1	13	13	0,923
20:53:28.075000 - 20:53:33.229000	1	1	13	13	0,923
20:53:33.231000 - 20:53:38.063000	1	1	13	13	0,923
20:53:38.064000 - 20:53:43.309000	1	2	13	12	0,88

Waktu Proses	*#da*	*#ma*	*#npt*	*#eventclass*	Q_s (Nilai *simplicity)*
20:53:43.312000 - 20:53:48.549000	1	2	13	12	0,88
20:53:48.550000 - 20:53:53.348000	1	1	13	13	0,923
20:53:53.351000 - 20:53:58.086000	1	2	13	12	0,88
20:53:58.087000 - 20:54:03.097000	1	1	13	13	0,923
20:54:03.099000 - 20:54:08.412000	1	2	13	12	0,88
20:54:08.413000 - 20:54:13.356000	1	1	13	13	0,923
20:54:13.357000 - 20:54:18.097000	1	2	13	12	0,88
20:54:18.099000 - 20:54:23.103000	1	3	13	11	0,833
20:54:23.105000 - 20:54:28.562000	1	2	13	12	0,88
20:54:28.564000 - 20:54:33.648000	1	4	13	9	0,773

Keterangan:

#da : jumlah macam aktivitas pada *event log* yang ditampilkan redundan pada *process tree*.

#ma : jumlah macam aktivitas pada *event log* yang tidak ditampilkan pada *process tree* dan jumlah *leaf node* yang tidak terdapat dalam *event log*.

#npt : jumlah *leaf node* pada *process tree*.

#eventclass : jumlah macam aktivitas pada *event log*.

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. Cara algoritma yang memanfaatkan Model Markov Tersembunyi untuk memperbaiki *incomplete trace* dengan memodifikasi metode Viterbi dan *Backward* pada Model Markov Tersembunyi. Pemodifikasian dilakukan dengan menambahkan kode untuk menanggulangi aktivitas yang tidak terdapat dalam Model Markov Tersembunyi serta mengubah hasil keluaran metode menjadi *observer* dikarenakan aktivitas merupakan *observer* pada Model Markov Tersembunyi.
2. Cara algoritma yang memanfaatkan Model Markov Tersembunyi dalam menggali proses (*process discovery*) dari *streaming event log* dengan membentuk Model Markov Tersembunyi dari *event log* lampau menggunakan modifikasi metode Baum-Welch, memperbaiki *incomplete trace* dengan menggunakan metode Viterbi dan *Backward*, membentuk Model Markov Tersembunyi dari *streaming event log* yang mengalami perbaikan *incomplete trace* dengan metode Baum-Welch serta membentuk model proses dengan metode pembentukan model proses. Langkah rinci dari algoritma yang memanfaatkan Model Markov Tersembunyi dijabarkan pada bab III.
3. Hasil evaluasi menunjukkan bahwa model proses dari algoritma yang memanfaatkan Model Markov Tersembuyi mampu menghasilkan nilai *fitness*, generalisasi dan *simplicity* yang lebih baik dibandingkan model proses dari

algoritma *Online Heuristic Miner* . Hasil evaluasi kualitas model proses dari algoritma yang memanfaatkan Model Markov Tersembunyi memiliki nilai *fitness* dan presisi yang lebih baik apabila rentang waktu untuk pembentukan proses semakin besar dan kualitas generalisasi dan *simplicity* yang lebih baik apabila rentang waktu untuk pembentukan proses semakin singkat.

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 sebagai berikut:

1. Fitur memasukkan *event log* lampau dapat dikembangkan dengan format lain selain format Excel.
2. Nilai presisi lebih ditingkatkan.
3. Mampu menanggulangi permasalahan *invisible task* pada saat pemodelan.

DAFTAR PUSTAKA

[1] W. M. P. van der Aalst, Process Mining - Discovery, Conformance and Enhancement of Business Processes, Schleiden: Springer, 2010.

[2] A. S. Burattin dan W. van der Aalst, "Heuristic Miners for Streaming Event Data," *ArXiv CoRR,* 2012.

[3] K. Audhkhasi, O. Osoba dan B. Kosko, "Noisy hidden Markov models for speech recognition," dalam *Neural Networks (IJCNN), The 2013 International Joint Conference*, 2013.

[4] A. C. Testa, J. Hane, S. Ellwood dan R. P. Oliver, "Coding Quarry: highly accurate hidden Markov model gene prediction in fungal genomes using RNA-seq transcripts," *BMC genomics,* vol. 16, no. 1, p. 1, 2015.

[5] M. R. Hassan, K. Ramamohanarao, J. Rahman dan M. M. Hossain, "A HMM-based adptive fuzzy inference system for stock market forecasting," *Neurocomputing,* vol. 104, pp. 10-25, 2013.

[6] A. Tenyakov, "Estimation of Hidden Markov Models and Their Applications in Finance," Electronic Thesis and Dissertation Repository, 2014.

[7] L. Wen, J. Wang, W. van der Aalst, B. Huang dan J. Sun, "Mining process models with prime invisible tasks," *Data & Knowledfe Engineering,* vol. 69, pp. 999-1021, 2010.

[8] A. Weijters, W. van der Aalst dan A. de Medeiros, "Process mining with the Heuristics Miner-algorithm," Eindhoven University of Technology, Netherlands, 2006.

[9] B. Prawira, "Pixelbali," 12 Agustus 2014. [Online]. Available: http://pixelbali.com/informasi-teknologi/critical-path-method.html. [Diakses 5 May 2016].

[10] D. A. Loreto, "Sampling and Abstract Interpretation for

Trackling Noise in Process Discovery," Universitat Politecnica De Catalunya, Barcelona, 2012.

[11] P. Dymarski, Hidden Markov Model, Theory and Application, India: InTech, 2011.

[12] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *Journal of circuits, systems, and computers,* vol. 8, pp. 21-66, 1998.

[13] P. Ceravolo, B. Russo dan R. Accorsi, Data-Driven Process Discovery and Analysis: 4th International Symposium, Milan: Springer, 2015.

[14] C. A. M. Buijs, B. F. van Dongen dan W. M. P. van der Aalst, "On The Role of Fitness, Precision, Generalization and Simplicity in Process Discovery," dalam *Lecture Notes in Computer Science*, Berlin, Springer-Verlag, 2012, pp. 305-322.

[15] De Cnudde, S. J. Claes dan G. Poels, "Improving the quality of the Heuristics Miner in ProM 6.2.," *Expert Systems with Applications,* vol. 41, pp. 7678-7690, 2014.

[16] R. A. Sutrisnowati, H. Bae, L. Dongha dan K. Minsoo, "Process Model Discovery based on Activity Lifespan," dalam *International Conference on Technology Innovation and Industrial Management*, Seoul, 2014.

DAFTAR ISTILAH

Case
Suatu kasus tertentu 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.

Case ID
Nomor identitas dari kasus tertentu pada *event log*.

Eksekusi
Pengolahan instruksi program dimana terdiri dari dua proses utama, yaitu pembacaan instruksi dan pelaksanaan intruksi.

Event
Aktivitas.

Event log
Suatu kumpulan eksekusi proses berdasarkan data aktivitas proses bisnis yang disimpan dalam bentuk tertentu.

Event log Lampau
Event log yang digunakan sebagai masukan dari Model Markov Tersembunyi awal yang menjadi acuan dalam pembentukan Model Markov Tersembunyi dari *streaming event log*.

Iterasi
Proses yang digunakan secara berulang-ulang.

Incomplete trace
Rangkaian aktivitas yang tidak komplit dikarenakan terpotong pada aktivitas awal atau terpotong pada aktivitas akhir.

Observer
Bagian dari Model Markov Tersembunyi sebagai masukan dalam menentukan model.

Overfitting
Kategori model proses yang berarti model proses tersebut sangat spesifik dan berpatokan penuh pada contoh proses di *event log*.

Pengujian Blackbox
Pengujian perangkat lunak dengan cara menguji fungsionalitas dari perangkat lunak tersebut.

Pseudo-code
Kode yang digunakan untuk menulis sebuah algoritma dengan cara bebas yang tidak terikat dengan bahasa pemrograman tertentu.
Process discovery
Salah satu tugas dari *process mining* yang bertujuan untuk menghasil model proses dengan cara menggali informasi dari data pada *event log*.
Running
Pemanggilan program.
SOP
Data sesungguhnya yang digunakan sebagai pengukur keberhasilan model proses dari suatu algoritma.
Surjektif
Pemetaan anggota himpunan yang memenuhi syarat jika dan hanya jika untuk anggota kodomain terdapat paling tidak satu relasi dengan anggota dalam domain.
State
Bagian dari Model Markov Tersembunyi sebagai yang probabilitasnya dapat diketahui melalui rangkaian *observer*.
Threading
Metode pemrograman untuk menjalankan beberapa pekerjaan secara bersamaan.
Threshold
Ambang batas.
Trace
Rangkaian dari aktivitas.
Underfitting
Kategori model proses yang berarti model tersebut juga menunjukkan proses yang cenderung berbeda dengan proses yang terlihat pada *event log*.

INDEX

S

T

U

V

X

BIODATA PENULIS

Kelly Rossa Sungkono lahir di Surabaya, Jawa Timur pada tanggal 9 Juni 1994. Pendidikan yang telah ditempuh adalah SDN Lawang V (2000-2006), SMPN 16 Malang (2006-2009), SMAN 7 Tangerang Selatan (2009-2012) dan S1 Teknik Informatika ITS (2012-2016).

Selama kuliah, penulis pernah mengemban amanah sebagai asisten dosen pada mata kuliah Basis Data dan staf magang LPTSI bagian dokumentasi yang menangani proyek Sistem Informasi Trisakti serta Sistem Informasi BPFK Surabaya. Selain itu, penulis juga aktif berorganisasi menjadi staf Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS 2014/2015 dan anggota klub saman TC 2012/2015. Rumpun mata kuliah (RMK) yang diambil oleh penulis adalah Manajemen Informasi serta memiliki ketertarikan di bidang *Process Mining*, Audit Sistem, Rekayasa Pengetahuan serta Basis Data. Alat komunikasi yang disediakan oleh penulis adalah surel:kelsungkono@gmail.com.

[Halaman ini sengaja dikosongkan]