SKRIPSI

PENGGUNAAN SPARSE MATRIKS PADA PROSES STOKASTIK DISCRITE TIME MARKOV CHAIN

THE USE OF SPARSE MATRIX IN THE STOCASTIC PROCESS OF DISCRITE TIME MARKOV CHAIN

FATHREZZA FIRMANULL ARIEF
15/383231/PA/16891

PROGRAM STUDI ILMU KOMPUTER
DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2020

SKRIPSI

PENGGUNAAN SPARSE MATRIKS PADA PROSES STOKASTIK DISCRITE TIME MARKOV CHAIN

THE USE OF SPARSE MATRIX IN THE STOCASTIC PROCESS OF DISCRITE TIME MARKOV CHAIN

Diajukan untuk memenuhi salah satu syarat memperoleh derajat
Sarjana Komputer

FATHREZZA FIRMANULL ARIEF
15/383231/PA/16891

PROGRAM STUDI ILMU KOMPUTER
DEPARTEMEN ILMU KOMPUTER DAN ELEKTRONIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS GADJAH MADA
YOGYAKARTA

2020

HALAMAN PENGESAHAN

SKRIPSI

PENGGUNAAN SPARSE MATRIKS PADA PROSES STOKASTIK DISCRITE TIME MARKOV CHAIN

Telah dipersiapkan dan disusun oleh

FATHREZZA FIRMANULL ARIEF
15/383231/PA/16891

Telah dipertahankan di depan Tim Penguji
Telah disetujui
pada tanggal November 2020

Susunan Tim Penguji

Dr.-Ing. Mhd. Reza M.I Pulungan, S.Si., M.Sc.	Suprapto, Drs., M.I.Kom.
Pembimbing Pertama	Ketua Penguji
Faizal Makhrus,S.Kom., M.Sc	Sri Mulyana, Drs., M.Kom.
Pembimbing Kedua	Anggota Penguji

PERNYATAAN

Dengan ini kami menyatakan bahwa dalam skripsi ini tidak terdapat karya yang pernah diajukan untuk memperoleh gelar Sarjana di suatu Perguruan Tinggi, dan sepanjang pengetahuan saya juga tidak terdapat karya atau pendapat yang ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka.

Yogyakarta, November 2020

Fathrezza Firmanull Arief

Halaman Motto dan Persembahan

Karya ini dipersembahkan untuk penulis sendiri,

Ayah, Ibu, dan Keluarga Besar tercinta,

Teman-Teman seperjuangan Ilmu Komputer UGM 2015,

dan segenap pembaca sekalian.

"Allah tidak membebani seseorang melainkan sesuai dengan kesanggupannya"
-Al-Baqarah : 286

"*The battlefield changes rapidly, in order to win you need to keep your cool and find flaws and opening*"
-Scoot (Langrisser II)

"Tidaklah Allah menciptakan/menjadikan sesuatu itu sia-sia melainkan ada hikmahnya"
-Al-Imran : 191

"*We must never quit, no matter how tough it gets. we have to keep going*"
-Lewis Hamilton

PRAKATA

Puji syukur kepada Allah SWT yang telah memberikan karunia, anugrah, rahmat dan petunjuk-Nya sehingga memudahkan penulis dalam menyelesaikan skripsi yang berjudul ”Deteksi Hoax Berita Bahasa Indonesia dengan Support Vector Machine Berbasis Particle Swarm Optimization”. Dalam pengerjaan tugas akhir ini, penulis banyak mendapatkan dukungan dari berbaagai pihak. Oleh karena itu, penulis mengucapkan terima kasih kepada:

1. Allah SWT yang selalu ada untuk memberikan jalan untuk segala halangan yang ada.
2. Keluarga besar penulis terutama Ibu dan Ayah yang memberikan dukungan moril maupun materil selama penulis menempuh jenjang pendidikan di Ilmu Komputer, Universitas Gadjah Mada.
3. Bapak M. Reza Pulungan, S.Si.,M.Sc.,Ph.D. selaku dosen pembimbing pertama Skripsi yang telah membimbing penulis dengan sabar dari proses pengerjaan Proposal hingga penyelesaian Skripsi.
4. Bapak Faizal Makhrus,S.Kom., M.Sc selaku dosen pembimbing Seminar dan pembimbing kedua Skripsi yang telah membimbing dalam proses Seminar dan Skripsi.
5. Bapak Dr. Azhari SN, MT selaku dosen pembimbing akademik yang telah sabar membantu penulis dalam pengisian KRS selama penulis menempuh jenjang pendidikan di Ilmu Komputer, Universitas Gadjah Mada.
6. Bapak dan Ibu dosen Ilmu Komputer yang telah memberikan ilmu dan pelajaran dengan sabar.
7. Teman-teman Ilmu Komputer UGM yang telah memberikan banyak wawasan dan kebahagiaan selama ini.
8. Wahid, Rochmad, dan Gusman yang sudah menemani dan menghibur penulis selama melakukan penyusunan Skripsi dalam rumah kos.
9. Teman-Teman DPM KM FMIPA UGM yang telah memberikan dukungan selama penulis ketika penulis menyusun skripsi

10. Ivan, Ihsan, Diko, dan Teta yang senantiasa mendukung dan membantu dalam memberikan motivasi dan semangat penulis dalam proses penulisan Skripsi.
11. Pihak-pihak lain yang tidak dapat disebutkan satu per satu.

Akhir kata penulis berharap agar skripsi ini dapat bermanfaat bagi perkembangan ilmu pengetahuan serta perkembangan Ilmu Komputer dan Teknologi Informasi.

Yogyakarta, November 2020

Penulis

DAFTAR ISI

HALAMAN PENGESAHAN **i**

HALAMAN PERNYATAAN **ii**

HALAMAN PERSEMBAHAN **iii**

HALAMAN MOTTO **iv**

PRAKATA **v**

DAFTAR ISI **vii**

DAFTAR TABEL **ix**

DAFTAR GAMBAR **x**

INTISARI **xi**

ABSTRACT **xii**

I PENDAHULUAN **1**

1.1 Latar Belakang Masalah 1
1.2 Rumusan Masalah 3
1.3 Batasan Masalah 3
1.4 Tujuan Penelitian 3
1.5 Manfaat Penelitian 3
1.6 Metodologi Penelitian 3
1.7 Sistematika Penulisan 4

II TINJAUAN PUSTAKA **6**

III LANDASAN TEORI **10**

3.1 Proses Stokastik dan Markov Chain 10
3.1.1 Proses Stokastik 10
3.1.2 Discrete Time Markov Chain 10
3.2 Peluang Transisi matriks 11

3.2.1 *Power Method* 12
3.3 *Stationary State Probability* 14
3.3.1 Konvergensi 15
3.4 Penyimpanan *Sparse Matrix* 18
3.4.1 Perkalian Matriks dalam Sparse 19

IV METODOLOGI PENELITIAN **21**
4.1 Deskripsi Umum Sistem 21
4.2 Kasus Uji 21
4.3 Rancangan Penyimpanan Sparse Matriks 22
4.4 Rancangan Proses Iterasi Stokastik 23
4.4.1 Mencari Konvergensi 24
4.5 Rancangan Pengujian 25

V IMPLEMENTASI **27**
5.1 Implementasi pembuatan matriks DTMC 27
5.2 Implementasi penyimpanan sparse matriks 28
5.2.1 Input File dan Penyimpanan Sparse 28
5.2.2 Implementasi pembuatan baris IA 29
5.3 Implementasi Power Method 30
5.4 Implementasi Konvergensi 31
5.4.1 Implementasi Konvergensi Absolut 31
5.4.2 Implementasi Konvergensi Relatif 32

VI HASIL DAN PEMBAHASAN **33**
6.1 Kasus Uji 33
6.2 Penggunaan Memori 34
6.3 Perbandingan runtime Berdasarkan Waktu Iterative 34
6.4 Perbandingan runtime *stationary-state probability* 37
6.4.1 Perbandingan dengan PRISM 39

VII KESIMPULAN DAN SARAN **41**
7.0.1 Kesimpulan 41
7.0.2 Saran 41

DAFTAR PUSTAKA **42**

DAFTAR TABEL

2.1 Ringkasan Tinjauan Pustaka 9

6.1 Kasus Uji 33
6.2 Hasil Konvergensi Absolut 38
6.3 Hasil Konvergensi Relatif 38
6.4 Hasil Konvergensi Absolut pada PRISM 39
6.5 Hasil Konvergensi Relatif pada PRISM 39

DAFTAR GAMBAR

4.1 Contoh isi dari file .tra 22
4.2 Alur Proses Stokastik 23

5.1 Visualisasi Model 27
5.2 Contoh coding pada salah satu model 28

6.1 Jumlah ρ tidak nol dalam masing-masing model 35
6.2 Grafik Total runtime pada sejumlah iterasi 36
6.3 Grafik rata-rata runtime setiap iterasi 37

INTISARI

PENGGUNAAN SPARSE MATRIKS PADA PROSES STOKASTIK DISCRITE TIME MARKOV CHAIN

Oleh

Fathrezza Firmanull Arief
15/383231/PA/16891

Perkembangan teknologi dalam tataran global pada saat ini telah berkembang pesat, dan semakin tinggi pula teknologi perlu lebih andal. Salah satu metode untuk mengecek keandalan sebuah teknologi adalah dengan *model checking*. Namun dengan perkembangan teknologi, maka model checking mengalami permasalahan *state-space explosion* dimana komponen kebutuhan berupa state meningkat secara exponensial. Salah satu metode untuk meningkatkan efisiensi dari state-space adalah dengan meningkatkan efisiensi penyimpannan seperti penyimpanan model sparse matriks. Banyak sekali metode untuk melakukan penyimpanan model sparse matriks, salah satu yang model penyimpanan tersebut merupakan Compressed-sparse row.

Pada penelitian ini, akan dilakukan eksperimen bagaimana cara kerja penyimpanan sparse matriks model Compressed-sparse row pada salah satu langkah pada stokastik *model checking* yaitu berupa transient-state dan *stationary-state* probability. Metode *stationary-state* yang akan digunakan merupakan metode dasar berupa *Power Method*. Data yang digunakan pada penelitian ini berupa DTMC dengan *state-space* yang sangat besar.

Dari hasil Penelitian, menunjukkan bahwa metode *Power method* mempunyai waktu kompleksitas linier dalam menjalankan model. Dalam space complexity, storage yang dibutuhkan merupakan konstan O(m+n) di mana m merupakan jumlah state dan n merupakan jumlah vektor transisi yang tidak nol pada setiap iterasi. Dilakukan juga test mengenasi konvergensi pada steady-state, diantaranya dalam penelitian ini dilakukan dua metode konvergensi yaitu Konvergensi absolut dan relatif. Diantara kedua konvergensi, relatif melakukan proses running time lebih cepat dibanding Konvergensi Absolut tetapi mempunyai kemungkinan untuk gagal mencari konvergensi.

Kata kunci : Discrete-time Markov Chain, Sparse matriks, Proses Stokastik, Model Checking, Power Method, stationary-state

ABSTRACT

THE USE OF SPARSE MATRIX IN THE STOCASTIC PROCESS OF DISCRITE TIME MARKOV CHAIN

By

Fathrezza Firmanull Arief
15/383231/PA/16891

The development of technology globally over the recent decade has grown rapidly and with it, more demand of the technology to be more reliable. One method to check the reliability of a technology is the use of model checking. However, with the development of technology it comes at cost which the model checking experienced an explosive state-space problem where the required component in the form of a state increased exponentially.One of the method to increase the efficiency of state-space is by storing the states using sparse matrix model. There are many methods for storing the sparse matrix model, one of which is the Compressed-sparse row.

In this study, an experiment will be carried out on how the sparse matrix storage of the Compressed-sparse row model works in one of the methods in the stochastic model examination, namely in the form of transient-state and stationary-state probability. The established method to be used for this experiment is the basic method in the form of the Power Method. The data used in this study is DTMC with a very large state space.

From the experiment results, it shows that the Power method has a linear time complexity in running the model. In space complexity, the required storage is a constant O (m + n) where m is the amount of states on DTMC and N is the non-zero number of transition vectors in each iteration. A test of convergence at steady-state was also carried out, in this study we include two convergence methods, namely absolute and relative convergence. Between the two convergences, the running time process is relatively faster than Absolute Convergence but has a chance to fail to get the convergence.

Keyword : Discrete-time Markov Chain, Sparse matrix, Stochastic Process, Model Checking, Power Method, stationary-state

BAB I

PENDAHULUAN

1.1 Latar Belakang Masalah

Perkembangan teknologi dalam tataran global pada saat ini telah berkembang dengan pesat, hampir semua aktivitas dan komunikasi antar individu sudah memanfaatkan kecanggihan teknologi. Perkembngan teknologi menjadi kebutuhan yang sangat penting dalam kehidupan manusia seperti penggunaan media televisi, telepon seluler, mobil pintar, dan teknologi lainnya. Teknologi tersebut sangat membantu semua kegiatan manusia menjadi lebih efektif dan efisien. Namun demikian perkembangan teknologi juga dimanfaatkan untuk berbagai bidang yang mempunyai tingkat risiko yang cukup tinggi, diantaranya adalah teknologi nuklir dan teknologi dirgantara. Jika terjadi kerusakan pada teknologi tersebut, maka akan menjadi ancaman kehidupan manusia. Ini membuat teknologi mempunyai dampak yang lebih tinggi dan semakin tergantung oleh bagaimana kualitas dari teknologi. Reliabilitas dari teknologi akan juga semakin diperlukan seiring tingginya ketergantungan terhadap teknologi. Untuk mendapatkan reliabilitas teknologi yang tinggi dan konsisten maka diperlukan sebuah sistem validasi dengan kualitas tinggi. (Zapreev, 2008).

Sistem validasi merupakan sebuah metode untuk membuktikan bahwa sebuah sistem yang akan digunakan mencapai hasil yang diinginkan (Terry Bahill dan Henderson, 2005). Sistem validasi dengan kualitas tinggi akan membuat reliabilitas dari teknologi semakin tinggi. Tetapi, sistem validasi dengan kualitas tinggi juga diperlukan tenaga kerja yang cukup tinggi. Keterbatasan tenaga manusia untuk melakukan sistem validasi dengan kualitas tinggi mengakibatkan harus menggunakan sistem validasi secara otomatis. Pada tahun 1990, muncul sebuah model verifikasi baru yaitu *model checking*. *Model checking* merupakan salah satu metode untuk memverifikasi sebuah struktur atau model memenuhi sebuah spesifikasi yang diinginkan (Alur et al., 1990). Lahirnya model checking membuat banyak peneliti ilmu komputer menghubungkan model checking dengan ilmu lain (Clarke, 2008). Dia juga menyebutkan bahwa salah satu permasalahan yang bisa diatasi dalam model checking adalah dalam *model checking probabilistic*.

Chatterjee dan Henzinger (2008) mengumpulkan banyak metode model probabilitas yang sudah pernah diteliti untuk digunakan dalam *model checking*. Salah

satu dari model probabilitas ini adalah sebuah model stokastik bernama *markov chain*. *Markov chain* merupakan sebuah proses probabilitas stokastik di mana keadaan atau *state* saat ini dan *state* yang akan mendatang independen dengan *state* yang dahulu (Stewart, 2009). Kumpulan dari *state* markov chain ini disebut dengan *state space* dan dihubungkan dengan sebuah probabilitas yang disebut dengan *transition probability*.

Tetapi Model Checking mempunyai permasalahan yang cukup besar, yaitu permasalahan *state-space explosion*. *State-space explosion* merupakan di mana jumlah kebutuhan komponen validasi dan komponen kebutuhan yang dianggap sebagai *state* meningkat secara exponensial. Salah satu penanggulangan dari *state-space explosion* bisa dilakukan dengan yaitu meningkatkan efisiensi dari penyimpanan *state-space*. Umumnya markov chain direpresentasikan dalam bentuk markov (Stewart, 2009), sehingga salah satu dari metode untuk meningkatkan efisiensi dari penyimpanan state space dalam markov chain adalah penyimpanan dengan model *sparse matrix*.

Sparse matrix merupakan sebuah matriks di mana jumlah elemen yang terdapat dalam matriks memiliki nilai nol atau tidak terisi jauh lebih banyak daripada elemen yang bernilai. Melakukan penyimpanan yang sangat efisien merupakan hal yang penting terutama jika data yang digunakan sangat besar. (Pissanetzky, 1984). Umumnya *probabilistic model checking* mempunyai struktur berbentuk *sparse matrix* (Clarke, 2008), sehingga metode ini bisa digunakan untuk meningkatkan efisiensi dari penyimpanan *state space*. Metode untuk melakukan penyimpanan *sparse matrix* sudah dilakukan dengan berbagai metode. Beberapa diantaranya ialah penggunaan *sparse matrix* sebagai preconditioner untuk sebuah proses iterasi matriks (Akhunov et al., 2013). Selain itu penggunaan GPU CUDA untuk melakukan *sparse matrix* pada LTL *model checking* juga sudah mulai dilakukan (Barnat et al., 2009).

Tetapi, penelitian untuk mencari bagaimana dampak sebuah *sparse matrix* pada proses stokastik *markov chain* sangat kurang. Ini akan menjadi masalah bagi orang yang ingin memasuki dunia *probabilistic model checking* karena *sparse matrix* merupakan hal yang sangat penting di era saat ini. Maka dari itu, pada penelitian ini akan menggunakan salah satu model checking, markov chain dalam waktu diskrit atau disebut Discrete-Time Markov Chain (DTMC), pada proses stokastik menggunakan metode penyimpanan *sparse matrix*.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah dipaparkan, disusun rumusan masalah yang berkaitan dengan topik penelitian ini adalah berapa banyak kompleksitas space dan kompleksitas waktu yang dibutuhkan untuk melakukan proses DTMC menggunakan sparse matriks.

1.3 Batasan Masalah

Batasan masalah yang dipakai pada skripsi ini adalah :

1. Proses stokastik yang dianalisis merupakan peluang transisi (transient-state) dan stationary-state DTMC.
2. Metode yang digunakan untuk probabilitas *stationary* adalah *power iteration*
3. Metode penyimpanan *sparse matrix* yang digunakan merupakan *Compressed sparse row-wise format*

1.4 Tujuan Penelitian

Tujuan dalam penelitian kali ini adalah melakukan analisis pada kompleksitas waktu dan space yang dibutuhkan untuk melakukan proses stokastik pada DTMC menggunakan sparse matriks. Selain itu, menunjukkan bagaimana implementasi sparse matriks dalam proses stokastik pada DTMC.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah memberikan informasi pembelajaran bagaimana kinerja sparse matrix pada proses stokastik DTMC. Selain itu, melihat bagaimana performa proses stokastik DTMC menggunakan *sparse matrix* pada tools model checking yang lain.

1.6 Metodologi Penelitian

1. Studi Literatur

Studi literatur dilakukan dengan memahami konsep DTMC, memahami tentang sparse matriks, serta memahami bagaimana proses stokastik terjadi dalam DTMC

2. Perancangan Penelitian

   Ditentukan proses penelitian yang dilakukan untuk algoritma yang akan digunakan dalam proses stokastik dan sparse matriks, variable yang diteliti, dan prosedur pengujian dalam proses stokastik.

3. Implementasi

   Setelah melakukan rancangan model, diimplementasikan algoritma yang telah dirancang dengan menggunakan parameter uji yang telah ditentukan. Implementasi dilakukan dengan menggunakan bahasa pemrograman python

4. Pengujian

   Dilakukan pengujian terhadap algoritma yang digunakan untuk melakukan proses stokastik DTMC dalam penyimpanan sparse matriks

5. Analisis hasil dan pembahasan

   Dilakukan analisis dari hasil dalam proses stokastik berupa waktu yang dipakai untuk melakukan penelitian. Selain itu, dilakukan pembahasan dari hasil tersebut, serta mencoba untuk dibandingkan dengan tools model checking yang sudah ada.

6. Penulisan Laporan

   Dilakukan penulisan laporan penelitian yang dilaksanakan beriringan dengan proses penelitian.

1.7 Sistematika Penulisan

Penulisan tugas akhir ini terdiri dari 7 bab dengan sistematika sebagai berikut

BAB I PENDAHULUAN

Berisi Latar Belakang dari dilakukannya penelitian, rumusan masalah penelitian, tujuan dan manfaat dari penelitian, serta batasan - batasan dalam penelitian.

BAB II TINJAUAN PUSTAKA
Berisi Uraian sistematis tentang informasi hasil penelitian dan menghubungkannya dengan masalah penelitian

BAB III LANDASAN TEORI
Berisi pengertian dan sifat-sifat yang diperlukan, serta berisi dasar teori yang digunakan untuk penelitian

BAB IV METODOLOGI PENELITIAN
Berisi setiap langkah eksperimen yang dilakukan dalam penelitian

BAB V IMPLEMENTASI
Implementasi sistem sesuai dengan rancangan dan metodologi berdasarkan tools pemrograman yang dipakai

BAB VI HASIL DAN PEMBAHASAN
Temuan ilmiah yang diperoleh sebagai data hasil penelitian

BAB VII KESIMPULAN DAN SARAN
Memuat secara singkat tentang hasil penelitian yang diperoleh, serta memberikan saran untuk menyampaikan masalah yang dimungkinkan untuk penelitian lebih lanjut

BAB II

TINJAUAN PUSTAKA

Beberapa penelitian terdahulu mengenai perhitungan stokastik *markov chain* dan *sparse matrix* sudah pernah dilakukan dilakukan. Zapreev (2008) melakukan penelitian dalam *Model Checking Markov chain* secara keseluruhan. Dalam penelitian tersebut berfokus pada membandingkan alat-alat *model checking* yang tersedia, terutama Markov Reward Model Checker (MRMC) dengan alat *model checking* lainnya. Penelitian tersebut juga menggunakan *sparse matrix* sebagai metode penyimpanan matrix. Penelitian ini berkesimpulan bahwa MRMC bisa bersaing dengan alat-alat *model checking* yang lainnya terutama pada ukuran *markov chain* yang lebih kecil.

Philippe et al. (1992) merupakan penelitian pertama dalam mencari perhitungan numerik pada proses stokastik *markov chain*. Dalam penelitiannya, dia mencoba untuk menyelesaikan permasalahan numerik *markov chain* dengan berbagai cara diantaranya *Power Method, Gauss-Seidel, symmetric successive overrelaxation method (SSOR)*, dan faktorisasi *incomplete LU* (ILU). Walaupun penelitian ini gagal dalam menemukan satu metode numerik yang paling efisien, penelitian ini berhasil dalam menemukan berbagai metode numerik untuk bisa digunakan dalam penelitian selanjutnya.

Terdapat berbagai penelitian penggunaan DTMC dalam permasalahan real. Penelitian menggunakan *markov chain* untuk keperluan *Google PageRank*. Penelitian oleh Langville dan Meyer (2006) menggunakan *stationary vector markov chain* untuk membantu algoritma vektor pada *Google PageRank*. Hasil dari Penelitian tersebut menunjukkan hasil yang cukup menjanjikan untuk bisa digunakan oleh *Google PageRank*. Metode yang digunakan dalam penilitan ini adalah metode *Iterative aggregation/disaggregation (IAD)* yang dipopulerkan oleh Stewart dan Wu (1992). Dalam penelitian Stewart, IAD sangat efisien dalam mencari stationary probability pada markov chain yang hampir terurai.

Aritonang et al., 2020 melakukan penelitian menggunakan DTMC dalam permasalahan belakangan ini yaitu permasalahan COVID-19. Penelitian ini melakukan prediksi mengenai jumlah pasien pada jangka lama di Indonesia menggunakan *steady-state* DTMC. Dalam penelitiannya, setiap state merepresentasikan range penambahan nilai pasian COVID-19 sebanyak 90 range pasien untuk menghindari terjadinya absorbing chain. Metode yang digunakan berupa metode direct dimana metode

ini melakukan perhitungan langsung menuju hasil *steady-state* tanpa melakukan iterasi. Penelitian lain mengenai prediksi menggunakan DTMC yaitu penelitian oleh Hasnaeni, 2017. Berbeda dengan penelitian milik Aritonang et al., 2020, penelitian ini menggunakan metode *Latent Markov Models* yang merupakan model lain dari *Hidden Markov Model*. Prediksi yang dilakukan berupa peluang konsumen untuk melakukan akuisisi rumah secara tunai maupun kredit dari tahun 2017, 2018, dan 2019. Kedua penelitian tidak menggunakan efisiensi penyimpanan *state-space* dikarenakan state yang digunakan tidak cukup banyak untuk diperlukan menggunakan efisiensi penyimpanan.

Terdapat berbagai penelitian untuk melakukan efisiensi penyimpanan dalam melakukan proses DTMC terutama *steady-state*. Sebelum Sparse matriks digunakan secara umum, metode untuk mengatasi permasalahan *state-space explosion* sudah dilakukan. Penelitian yang dilakukan Feinberg dan Chiu, 1987 menggunakan agregasi state untuk melakukan *steady-state* DTMC. Dalam penelitian ini, state-state yang terdapa dalam markov chain akan digabungkan menjadi sebuah superstate. Algoritma yang dibuat menurunkan kompleksitas dari $O(n^3$ floating-point menjadi $O(n^2$ floating-point. Walau penelitian tersebut belum membuktikan bagaimana cara kerja konvergensi secara keseluruhan, tetapi algoritma yang ditemukan bisa melakukan markov chain yang dense cukup baik.

Penelitian lain yang menggunakan sparse matriks dalam markov chain yaitu dalam penelitian oleh Barnat et al., 2009. Penelitian ini berfokus pada melakukan model checking LTL dalam DTMC dengan menggunakan *Compressed sparse row-wise format*. Selain itu, penelitian ini akan dilakukan dalam GPU NVIDIA yang memiliki teknologi CUDA. Algoritma yang digunakan untuk model checking LTL berupa Algoritma MAP. Algoritma tersebut digunakan karena sifat paralelnya dapat digunakan dalam teknologi GPU modern agar lebih efisien. Selain itu, Algoritma MAP bisa melakukan konvergensi secara *on-the-fly*. Hasil dari penelitian ini menunjukkan bahwa proses yang dilakukan dalam GPU CUDA bisa lebih cepat tetapi masih belum konsisten.

Duflot et al., 2006 melakukan sebuah analisis penelitian mengenai bluetooth pencarian perangkat. Dalam penelitiannya, peneliti menggunakan DTMC untuk merepresentasikan bluetooth sebagai analisis. Model penyimpanan yang digunakan berupa binary decision diagram (BDD). Penelitian ini menganalisis bagaimana kerja bluetooth versi 1.1 dan 1.2 dalam mencari perangkat lain menggunakan model DTMC dan melihat ekspektasi waktu terbaik dan terburuk. Penggunaan metode DTMC sa-

ngatlah bagus untuk menangani permasalahan state yang dibutuhkan dalam bluetooth yang mencapai lebih dari 5 $\times 10^{10}$ walau hasil yang dikeluarkan hanyalah numerik dan masih perlu penambahan asumsi probabilitas.

Penelitian oleh Akhunov et al. (2013) merupakan penelitian untuk mencari model *sparse matrix* yang efisien dalam perhitungan iteratif. Penelitian tersebut membandingkan tiga metode pola penyimpanan *sparse*, yaitu pola *Knuth's*, pola *Rheinboldt-Mesztenji's*, dan pola *Sparse Row Matriks*. Penelitiannya menunjukkan bahwa Sparse row Matriks merupakan metode penyimpanan sparse yang paling efisien. Penelitian tersebut juga menjelaskan bagaimana cara menggunakan *sparse matrix* dalam *pre-conditionaer* untuk perhitungan iteratif. pada penelitian ini, metode iterative yang digunakan adalah ILU.

Tabel 2.1: Ringkasan Tinjauan Pustaka

No	Peneliti	Penelitian	Metode
1	Zapreev, 2008	Model checking Markov Chains: Techniques and Tools	Penelitian Berfokus pada membandingkan semua alat yang pernah tersedia dalam memproses Model Checking dalam Probabilitas Matriks
2	Philippe et al., 1992	Numerical Method in Markov Chainn Modelling	Melakukan metode numerik dalam *Markov Chain* tanpa penggunaan Sparse matrix.
3	Stewart dan Wu, 1992	Numerical Experiments with Iteration and Aggregation for Markov Chain	Penilitan pertama yang menggunakan IAD.
4	Langville dan Meyer, 2006	Updating Markov Chain with An Eye on Google's PageRank	Metode yang digunakan untuk pendekatan stationary state markov chain menggunakan IAD.
5	Akhunov et al., 2013	Sparse Matrix Storage Formats and Acceleration of Iterative Solution of Linear Algebraic System with Dense Matrix	Metode iteratif yang digunakan untuk mengetes *Sparse Row-Matrix* menggulakan ILU.
6	Barnat et al., 2009	CUDA Accelerated LTL Model Checking	Penggunaan sparse matriks dalam GPU untuk model checking LTL.
7	Feinberg dan Chiu, 1987	A Method to Calculate Steady-State Distributions of Large Markov Chains by Aggregating States	Melakukan *steady-state* DTMC menggunakan agregasi state.
8	Aritonang et al., 2020	Analisis Pertambahan Pasien COVID-19 di Indonesia Menggunakan Metode Rantai Markov	Implementasi *steady-state* DTMC dalam prediksi pertambahan pasien COVID-19 menggunakan direct method.
9	Hasnaeni, 2017	Menilai dan Memprediksi Akuisisi Kepemilikan Rumah dari Produk Keuangan Berdasarkan Discrete Time Markov Chains Menggunakan Latent Markov Models	Prediksi akuisisi menggunakan Latent Markov Model berdasarkan DTMC.
10	Duflot et al., 2006	A Dormal Analysis of Bluetooth Device Discovery	Representasi dari pencarian perangkat bluetooth dalam bentuk DTMC menggunakan penyimpanan Binary Decision Diagram.

BAB III

LANDASAN TEORI

3.1 Proses Stokastik dan Markov Chain

3.1.1 Proses Stokastik

Sebelum mengetahui definisi dari Discrete Time Markov Chain, perlu diketahui pengertian dari proses stokastik. Proses stokastik merupakan sebuah proses dimana jalannya proses bergantung pada sebuah probabilitas acak. Proses stokastik dapat didefinisikan menjadi sebuah kumpulan dari variabel random $X(t), t \in T$, dimana $X(t)$ merupakan random variable dalam waktu t pada sebuah ruang probabilitas, dan T merupakan indeks waktu yang digunakan sebagai parameter. Apabila indeks waktu berupa waktu diskrit seperti $T = 0, 1, 2....$ maka proses stokastik tersebut merupakan proses berdasarkan waktu diskrit (Hasnaeni, 2017).

3.1.2 Discrete Time Markov Chain

Markov Chain merupakan sebuah proses stokastik yang memenuhi properti markov, dimana proses yang terjadi pada sekarang tidak tergantung pada proses yg berlalu. Sehingga apabila diketahui sebuah proses pada saat ini, maka tidak perlu proses yang sudah berlalu untuk bisa mengetahui prediksi pada proses selanjutnya (Myers et al., 2017). Maka sebuah markov chain dikatakan proses DTMC apabila proses stokastik waktu diskrit $\{X_t; t = 0, 1, 2, 3, \dots\}$ yang memenuhi properti markov. Secara umum, DTMC bisa ditulis dalam persamaan (3.1).

Anggap n merupakan bilangan asli dan i_n merupakan sebuah state state,
Untuk setiap $n \geq 0, Xn \in S$,
Untuk setiap $n \geq 0$ dan untuk setiap $i_0, ...i_{n1}, i_n \in S$, maka :

$$\begin{aligned} &Prob\{X_n = i_n | X_{n-1} = i_{n-1}, ..., X_0 = i_0\} \\ &\quad = Prob\{X_{n+1} = i_{n+1} | X_n = i_n\}. \end{aligned} \tag{3.1}$$

Dalam kondisi probabilitas Prob$\{X_{n+1} = i_{n+1} | X_n = i_n\}$ merupakan sebuah single-step transition probabilities atau disebut transition probabilites dari markov chain Stewart, 2009. Anggap $i_{n+1} = i$ dan $i_n = j$, maka persamaan 1 bisa ditulis dalam persamaan (3.2)

$$p_{ij}(n) = Prob\{X_{n+1} = i_j | X_n = i_i\}. \tag{3.2}$$

Matriks transisi DTMC dapat dibentuk dalam bentuk matriks. Untuk sebuah matriks transisi DTMC dibentuk dalam matriks $P(n)$, maka dibentuk dengan elemen matriks berisi $pij(n)$ dengan baris I dan kolom j. Sehingga matriks DTMC akan dibentuk dalam persamaan matriks (3.3)

$$P(n) = \begin{pmatrix} p_{00}(n) & p_{00}(n) & ... & p_{0j}(n) & ... \\ p_{10}(n) & p_{01}(n) & ... & p_{1j}(n) & ... \\ ... & ... & ... & ... & ... \\ p_{i0}(n) & p_{i1}(n) & ... & p_{ij}(n) & ... \\ ... & ... & ... & ... & ... \end{pmatrix} \tag{3.3}$$

Perlu diketahui bahwa setiap elemen dari matriks DTMC $P(n)$ dalam persamaan matriks (3.3) memenuhi properti berikut (Syuhada, 2012) :

1. $0 \leq p_{ij}(n) \leq 1$
2. $\sum_{semua\ j} p_{ij}(n) = 1$

3.2 Peluang Transisi matriks

Telah diketahui bahwa $P_{ij}(n)$ merupakan sebuah proses stokastik yang menghubungkan antara state (i) dengan state lainnya (j'). Proses berpindah dari sebuah state i pada waktu t menuju j disebut dengan peluang transisi matriks. Apabila state awal i) menuju sebuah state j membutuhkan lebih dari satu langkah, maka perlu dilakukan peluang transisi matriks secara berulang. Sehingga diperlukan metode untuk menghitung probabilitas pada state j secara keseluruhan. Anggap $\overrightarrow{\rho^o}$ merupakan representasi dari sebuah vektor probabilitas awal dari DTMC. Menurut Zapreev (2008), proses tersebut bisa ditulis dalam persamaan (3.3).

$$\overrightarrow{\rho^o(t)} = \overrightarrow{\rho^o(t-1)} \cdot P \tag{3.4}$$

$\overrightarrow{\rho^o(t)}$ merupakan kemungkinan sebuah vektor probabilitas DTMC setelah melakukan proses peluang transisi sebanyak waktu t. P dalam rumus (3.4) merupakan sebuah probabilitas stokastik DTMC yang sudah ditentukan.

3.2.1 *Power Method*

Power Method merupakan metode iteratif yang digunakan untuk mencari dominan *eigenvector* menggunakan *eigenvalue* yang dominan dalam sebuah matriks. Menurut Anton dan Rorres, 2013, *eigenvector* merupakan vektor tidak-nol yang apabila dikalikan dengan sebuah matriks, akan menghasilkan perkalian berupa eigenvector dengan sebuah skalar bernama *eigenvalue*. anggap λ merupakan skalar *eigenvalue* dan x merupakan *eigenvector*. Definisi ini dapat ditulis dalam persamaan (3.5)

$$Ax = \lambda x \tag{3.5}$$

di Dalam pencarian *eigenvector* dominan, digunakan sebuah perkiraan acak *eigenvector* dominan x_0 untuk melakukan inisiasi. *eigenvector* x_0 ini akan dilakukan perkalian dengan matriks M yang akan menghasilkan sebuah *eigenvector* x_1 dengan normalisasi menggunakan *eigenvalue* λ. Jika *eigenvector* x_0 mempunyai hasil yang sama dengan x_1, maka *eigenvector* tersebut merupakan *eigenvector* dominan. Jika hasil kedua *eigenvector* tidak sama, maka akan dilakukan perkalian *eigenvector* x_1 dengan matriks M dan akan terus berulang untuk *eigenvector* berikutnya sampai menemukan *eigenvector* yang hampir sama (Larson et al., 2004). Proses ini bisa ditampilkan dalam persamaan (3.6).

$$\begin{aligned} \lambda x_1 &= Ax_0 \\ \lambda x_2 = Ax_1 &= A(Ax_0) = A^2x_0 \\ &\vdots \\ \lambda x_k = Ax_{k-1} &= A(A^{k-1}x_0) = A^kx_0 \end{aligned} \tag{3.6}$$

Dari persamaan (3.6) menurut Stewart (2009) proses *power method* ini mempunyai kemiripan dengan persamaan (3.4) . Dalam pencarian *stationary state* menggunakan *power method*, anggap matriks M merupakan probabilitas antar *state* atau disebut peluang tranisi dengan *eigenvector* awal menggunakan probabilitas awal. Stokastik matriks mempunyai *eigenvalue* dominan berupa 1 atau modulus 1 dalam matriks periodik, sehingga tidak diperlukan proses normalisasi untuk menyamakan antar *probability state* (Stewart, 2009). Untuk menyesuaikan persamaan power method (3.6) dengan persamaan peluang transisi (3.4), maka diperlukan transpose pada matriks DTMC atau pada vektor ρ. sehingga rumus power method pada proses peluang

transisi menjadi seperti berikut.

$$\begin{aligned} p_t &= P^T \cdot p_{t-1} \\ &atau \\ p_t^T &= p_{t-1}^T \cdot P \end{aligned} \tag{3.7}$$

Contoh 1 Misalkan sebuah model DTMC dengan transisi probabilitas sebagai berikut

$$P = \begin{pmatrix} 0.1 & 0 & 0.9 & 0 \\ 0 & 0.3 & 0.4 & 0.3 \\ 0.5 & 0.45 & 0.05 & 0 \\ 0.75 & 0 & 0 & 0.25 \end{pmatrix} \tag{3.8}$$

Anggap sistem model dimulai dalam state 1, maka probablitas vektor awal ditulis dalam ρ sebagai berikut

$$\rho_{(0)}^T = \begin{pmatrix} 1 & 0 & 0 & 0 \end{pmatrix}$$

Menggunakan persamaan pada (3.7), maka akan dilakukan transisi pertama dengan melakukan perkalian di antara dua matriks. Transisi pertama akan menghasilkan sistem berada di state 1 dengan probabilitas 0.1 atau state 3 dengan probabilitas 0.9. matriks. Matriks transisi pada iterasi awal ditulis sebagai berikut.

$$\rho_{(1)}^T = \begin{pmatrix} 0.1 & 0 & 0.9 & 0 \end{pmatrix}$$

Hasil tersebut merupakan hasil dari persamaan $p_t^T = p_{t-1}^T P$. Hasil matriks transisi pada iterasi pertama akan digunakan pada iterasi kedua dan seterusnya. Maka pada iterasi kedua, proses secara detail sebagai berikut :

$$\rho_{(2)} = \begin{pmatrix} 0.1 & 0 & 0.9 & 0 \end{pmatrix} \begin{pmatrix} 0.1 & 0 & 0.9 & 0 \\ 0 & 0.3 & 0.4 & 0.3 \\ 0.5 & 0.45 & 0.05 & 0 \\ 0.75 & 0 & 0 & 0.25 \end{pmatrix} = \begin{pmatrix} 0.46 & 0.405 & 0.135 & 0 \end{pmatrix}$$

Perhitungan tersebut bisa dilakukan berulang-ulang sesuai dengan banyaknya iterasi yang diinginkan. Matriks transisi yang dihasilkan dalam iterasi akan digunakan dalam perhitungan iterasi selanjutnya. Sehingga apabila dilakukan pada iterasi ke-20 maka,

$$\rho_{(20)} = \begin{pmatrix} 0,2904 & 0.24 & 0.3716 & 0.0959 \end{pmatrix} P = \begin{pmatrix} 0.2868 & 0,2398 & 0.3767 & 0.0965 \end{pmatrix}$$

3.3 *Stationary State Probability*

Pada umumnya, sebuah proses peluang transisi bergantung dengan waktu untuk mencapai hal yang diinginkan. Tidak semua model stokastik mencari sebuah hasil probabilitas berdasarkan waktu. Sebagai gantinya, model stokastik bisa mencari hasil probabilitas sampai suatu state bernama *stationary state probability* atau terkadang disebut *steady-state*. *stationary state probability* merupakan kondisi di mana hasil probabilitas peluang transisi sama dengan probabilitas pada waktu sebelumnya (Stewart, 2009). Sehingga perhitungan *stationary state probability* bisa ditulis pada persamaan (3.9).

$$\vec{\rho} \cdot P = \vec{\rho}\,, \sum_{s \,\to\, S} P_s = 1 \tag{3.9}$$

Persamaan (3.9) juga bisa ditulis dalam persamaan *limit*. Anggap $\overrightarrow{\rho^{\circ,*}}$ merupakan vector probabilitas sehingga akan menghasilkan persamaan (3.10)

$$\overrightarrow{\rho^{\circ,*}} = \lim_{t \to \infty} \overrightarrow{\rho^{\circ}\,(t)} \tag{3.10}$$

Dalam mencari sebuah *stationary state* dalam proses DTMC menurut Stewart (2009) bisa dilakukan dengan dua metode, yaitu metode langsung dan metode iteratif. Metode langsung merupakan metode yang mencoba untuk langsung mencari hasil *stationary state*. Metode langsung harus dilakukan dengan hati-hati, karena metode langsung yang salah bisa mendapatkan margin error yang sangat besar.

Berbeda dengan metode iteratif, metode ini dimulai dengan mengambil sebuah perkiraan awal. Perkiraan awal ini akan dilakukan sebuah modifikasi atau proses sehingga perkiraan ini mendekati dengan *stationary state* yang diinginkan. Dengan proses iteratif yang cukup, diharapkan bisa menemukan *stationary state* yang diinginkan. Proses iteratif mempunyai kemungkinan untuk gagal mencari *stationary state* yang diinginkan, jika proses pendekatan atau perkiraan awal yang diambil salah. Walaupun waktu yang digunakan cukup lama, hasil yang didapatkan bisa lebih akurat dan margin error semakin kecil. Untuk itu, dalam mencari *stationary state* dibutuhkan sebuah metode untuk memberhentikan proses peluang matriks yaitu dengan metode konvergensi.

3.3.1 Konvergensi

Pada persamaan (3.9) diketahui bahwa untuk mencapai pada *stationary-state* diperlukan matriks probabilitas transisi harus sama dengan matriks probabilitas transisi sebelumnya. Dalam kasus real, sama seperti persamaan iterative yang lainnya, hal ini sangat susah terjadi. Untuk itu, diperlukan metode untuk memberhentikan proses iteratif tersebut. Salah satunya dengan menetapkan sebuah error yang ditentukan sebelum mencari perhitungan *stationary-state* dengan menggunakan konvergensi (Zapreev, 2008). Fungsi dari ditetapkan error adalah untuk menghindari iterasi yang tidak terbatas dengan membandingkan vektor transisi dengan jarak tertentu agar *steady-state* bisa didapatkan lebih awal. Menurut (Stewart, 2009) untuk mencari jumlah iterasi yang diperlukan sampai menemukan *error* yang diinginkan bisa dilakukan menggunakan persamaan (3.11)

$$k = \frac{log\ \varepsilon}{log\ \lambda} \tag{3.11}$$

Pada rumus tersebut, λ merupakan *eigenvalue* sekunder pada matriks DTMC. Tetapi pencarian *eigenvalue* pada mariks DTMC susah untuk dicari terutama untuk matriks dengan densitas matriks yang kecil, sehingga digunakan metode konvergensi lain. Tidak hanya itu, pencarian *eigenvalue* pada matriks DTMC terutama dengan state yang sangat banyak dinilai tidak efisien(Zapreev, 2008). (Stewart, 2009) mengajukan dua persamaan untuk melakukan konvergensi :

1. Konvergensi Absolut

$$\left\|\pi^k - \pi^{(k-m)}\right\| < \varepsilon \tag{3.12}$$

2. Konvergensi Relatif

$$max_i \left(\frac{\left|\pi_i^k - \pi_i^{(k-m)}\right|}{\left|\pi_i^k\right|} \right) < \varepsilon \tag{3.13}$$

Kedua persamaan tersebut lebih efektif daripada persamaan (3.11) karena persamaan tersebut tidak memerlukan perhitungan eigenvalue untuk mencari *stationary-state*. Selain itu, berbeda dengan persamaan (3.11) di mana konvergensi dilakukan sebelum melakukan perhitungan *stationary*, kedua persamaan ini dilakukan di tengah perhitungan *stationary* (*on-the-fly*). Perhitungan konvergensi seperti ini akan lebih

akurat dan lebih cepat untuk dijalankan sehingga metode ini lebih sering digunakan dibandingkan dengan persamaan (3.11).

Contoh 2 Menggunakan persamaan (3.8) sebagai contoh, apabila akan dicari besar epsilon pada iterasi 20, dan anggap m=1 maka persamaan adalah sebagai berikut :

1. Absolut :

   $error_{(20)} = \|\rho_{20} - \rho_{19}\|$

   $error_{(20)} = 0.00627455$

2. Relatif :

   $error_{(20)} = max_{[1,4]} \left(\frac{\left|\rho_i^{20} - \rho_i^{19}\right|}{\left|\rho_i^{20}\right|} \right)$

   $error_{(20)} = 0.0137244$

Contoh 3 Menggunakan persamaan (3.8) sebagai contoh, anggap $epsilon = 1^3$ pada konvergensi absolut dan $\epsilon = 1^{-1}$ pada konvergensi relatif. Anggap m = 1, maka apabila matrik transisi berjalan sampai menemukan konvergensi maka akan menghasilkan persamaan berikut

1. Absolut : Anggap transisi dimulai dari awal.

$$\rho_{(2)} = \begin{pmatrix} 0.1 & 0 & 0.9 & 0 \end{pmatrix} \times P = \begin{pmatrix} 0.46 & 0.405 & 0.135 & 0 \end{pmatrix}$$

cek konvergensi

$$error_{(2)} = \|\rho_2 - \rho_1\|$$

$$error_{(2)} = 0.937$$

. . . Dilakukan iterasi dari 2 sampai iterasi 125 . . .

$$\rho_{(124)} = \begin{pmatrix} 0,0262 & 0,0234 & 0,0354 & 0,0071 \end{pmatrix}$$

$$\rho_{125} = \rho_{(124)} \times P$$

$$\rho_{125} = \begin{pmatrix} 0,0025 & 0,0022 & 0,0347 & 0,0070 \end{pmatrix}$$

cek konvergensi

$$error_{(125)} = \|\rho_{125} - \rho_{124}\|$$
$$error_{(125)} = 1,1^{-3}$$

$$\rho_{(126)} = \rho_{(125)} \times P$$
$$\rho_{126} = \begin{pmatrix} 0,0025 & 0,0022 & 0,0347 & 0,0070 \end{pmatrix}$$
cek konvergensi
$$error_{(126)} = \|\rho_{126} - \rho_{125}\|$$
$$error_{(126)} = 98^{-4}$$

dikarenakan $error_{(126)} < \epsilon$, maka iterasi 126 merupakan *stationary-state* matriks DTMC menggunakan konvergensi absolut

2. Relatif :

$$\rho_{(15)} = \begin{pmatrix} 0,2353 & 0,2077 & 0,295 & 0,060 \end{pmatrix}$$
$$\rho_{(16)} = \rho_{(15)} \times P$$
$$\rho_{16} = \begin{pmatrix} 0.216 & 0,195 & 0,3096 & 0,062 \end{pmatrix}$$
cek konvergensi
$$error_{(16)} = max_{[1,4]} \left(\frac{\left|\rho_i^{16} - \rho_i^{15}\right|}{\left|\rho_i^{16}\right|} \right)$$
$$error_{(16)} = 0,0877$$

dikarenakan $error_{(16)} < \epsilon$, maka iterasi 16 merupakan *stationary-state* matriks DTMC menggunakan konvergensi relatif

Dari Contoh 3 s bisa dilihat bahwa perbandingan antara $\rho_{(m)}$ dengan $\rho_{(m-1)}$ pada *stationary-state* dengan sebelumnya tidak terlalu jauh, sehingga *stationary-state* terdeteksi pada iterasi 126 untuk Absolut dan iterasi 15 untuk Relatif. $\rho_{(126)}$ merupakan vektor matriks probabilitas *stationary-state* menggunakan Konvergensi Absolut dan $\rho_{(15)}$ merupakan vektor matriks probabilitas menggunakan Konvergensi Relatif.

3.4 Penyimpanan *Sparse Matrix*

DTMC yang sangat besar umumnya memiliki matriks yang sangat *sparse*, dimana jumlah elemen bernilai nol lebih banyak daripada elemen yang memiliki nilai bilangan yang tidak nol (Stewart, 2009). Oleh karena itu, perlu menggunakan metode penyimpanan *sparse matrix* agar bisa meningkatkan efisiensi dan efektivitas dari proses perhitungan stokastik pada DTMC. Matriks yang disimpan dalam bentuk *sparse* ini juga harus bisa melakukan operasi yang cukup penting dan mendasar seperti perkalian, pencarian determinan, ataupun transpose. Metode skema penyimpanan yang paling sederhana untuk menyimpan sparse matriks adalah menggunakan *compressed sparse-row* (Akhunov et al., 2013). Untuk mengetahui model penyimpanan *compressed sparse-row*, maka perlu diketahui penyimpanan sparse matriks berdasarkan array *sparse row-wise format*. Struktur dalam *sparse row-wise format* akan dibagi menjadi tiga : (1) array yang berisi elemen matriks yang bukan nol (AA), (2) array yang berisi posisi kolom dari elemen matriks tersebut (JC), dan (3) array yang berisi posisi baris pada elemen matriks tersebut (JR) (Saad, 2003).

Contoh 4 Diketahui matriks DTMC 5x5 yang akan disimpan dalam bentuk sparse matriks :

$$\begin{pmatrix} 0.4 & 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.8 \\ 0 & 0.8 & 0 & 0.1 & 0.1 \\ 0.6 & 0.2 & 0.1 & 0 & 0.1 \\ 0 & 0 & 0.1 & 0.9 & 0 \end{pmatrix} \tag{3.14}$$

AA	0.4 0.6 0.2 0.8 0.8 0.1 0.1 0.6 0.2 0.1 0.1 0.1 0.9
JC	1 3 3 5 2 4 5 1 2 3 5 3 4
JR	1 1 2 2 3 3 3 4 4 4 4 5 5

Contoh 4 diurutkan berdasarkan baris matriks, dimulai dari baris awal dan meningkat. *Sparse row-wise format* merupakan metode penyimpanan sparse matriks di mana semua elemen matriks yang tidak bernilai nol disimpan berdasarkan baris matriks, bersama dengan indeks kolom yang sesuai (Pissanetzky, 1984). Untuk mengubah metode *sparse row-wise format* menjadi *compressed sparse-row*, maka perlu dilakukan perubahan indeks menjadi seperti berikut

AA	0.4 0.6 0.2 0.8 0.8 0.1 0.1 0.6 0.2 0.1 0.1 0.1 0.9
JA	1 3 3 5 2 4 5 1 2 3 5 3 4
IA	1 3 5 8 12

Baris JR dalam contoh 4 diubah menjadi baris IA. Baris IA berisi penunjuk pada awal dari setiap baris AA dan JA, Sehingga isi dari IA(i) adalah posisi dalam array AA dan JA di mana baris i dimulai. Jika dibandingkan antara baris JR dan baris IA, baris IA merupakan posisi di mana baris JR berubah angka. Perubahan ini diperlukan agar perhitungan iterasi stokastik lebih efektif dan sederhana (Saad, 2003). Proses ini dapat ditulis dalam Algoritma 1.

Algorithm 1: Modifikasi JC menjadi IA

```
Result: IA()
JR = Array berisi baris pada sparse matriks;
trs = jumlah transisi dalam matriks;
A = IA array placeholder;
for z=0 to trs do
    if JR(z) > JR(z-1) then
        IA(g) = trs;
        A++;
    end
    if z = trs then
        IA(A) = z;
    end
end
```

3.4.1 Perkalian Matriks dalam Sparse

Dalam persamaan (3.3) dan (3.6), diketahui bahwa perkalian matriks yang akan dilakukan dalam Proses Stokastik merupakan perkalian antara Vektor dan Matriks.

White dan Sadayappan, 1997 menjelaskan algoritma dasar perkalian menggu-

nakan CSR adalah seperti pada algoritma 2.

Algorithm 2: Perkalian matriks dalam Power Method

Result: $\rho_{(2)}$
sta = states;
$\rho_{(1)}()$ = sumber matriks transisi;
$\rho(2)()$ = target matriks transisi;
for *q = 0 until sta* **do**
    temp = 0.0; **for** *r = IA(sta) to IA(sta+1)* **do**
        temp = temp + (AA(r) * $\rho_{(1)}$(r));
    end
    $\rho_{(2)}$(q) = temp;
end

Algoritma 2 merupakan kondisi apabila matriks transpose merupakan matriks P. Agar matriks transpose merupakan matriks vektor ρ, maka algoritma 2 berubah menjadi Algoritma 3.

Algorithm 3: Modifikasi Perkalian matriks dalam Power Method

Result: $\rho_{(2)}$
sta = states;
$\rho_{(1)}()$ = sumber matriks transisi;
$\rho_{(2)}()$ = target matriks transisi;
for *q = 0 until sta* **do**
    for *r = IA(sta) to IA(sta+1)* **do**
        $\rho_{(2)}$(JC(r)) += (AA(r) * $\rho_{(1)}$(r));
    end
end

BAB IV
METODOLOGI PENELITIAN

4.1 Deskripsi Umum Sistem

Dalam penelitian ini, akan dilakukan implementasi *sparse matrix* pada proses stokastik dalam matriks DTMC. Metode penyimpanan *sparse matrix* akan menggunakan model *compressed spase-row*. Proses stokastik yang akan digunakan untuk penelitian ini merupakan melakukan peluang transisi dalam waktu tertentu dan mencari *stationary state*. Dalam proses stokastik untuk mencari *stationary-state probability* akan menggunakan model iterasi berupa *power method*. Pada proses mencari *stationary state*, akan dilakukan dua metode untuk sebagai penghenti konvergensi, yaitu metode konvergensi absolut dan konvergensi relatif.

4.2 Kasus Uji

Kasus Uji yang dipakai dalam penelitian ini merupakan matriks DTMC dengan state yang besar. Matriks DTMC yang digunakan merupakan berbagai model yang terinspirasi dalam kasus nyata seperti Game Theory dan Networking. Matriks DTMC digunakan juga mempunyai state lebih dari satu juta untuk mempermudah melhat hasil data serta melihat apakah implementasi dapat dilakukan pada model yang sangat besar. Selain itu, matriks DTMC yang diuji memiliki densitas yang kecil atau *sparsity* yang besar.

Implementasi kasus uji matriks DTMC dibuat menggunakan PRISM model checker. PRISM model checker merupakan sebuah alat untuk pembuatan dan analisis pada sebuah sistem yang terdapat model random atau probabilitas (Kwiatkowska et al., 2011). Pembuatan data pada PRISM akan menghasilkan sebuah file data dengan format (.tra) yang yang berisi transisi matriks dari sebuah state menuju state lain dengan sebuah probabilitas tertentu. Isi dari file .tra dan detail bisa dilihat pada gambar 4.1.

Pada gambar 4.1, terdapat dua bagian, bagian pertama ada pada baris pertama dan bagian kedua terdapat pada baris kedua dan seterusnya. Pada bagian pertama, terdapat jumlah state dan jumlah transisi atau elemen matriks. Pada bagian kedua berisi state matriks asal (baris dalam matriks), matriks tujuan (kolom pada matriks), dan probabilitas transisi (isi elemen matriks).

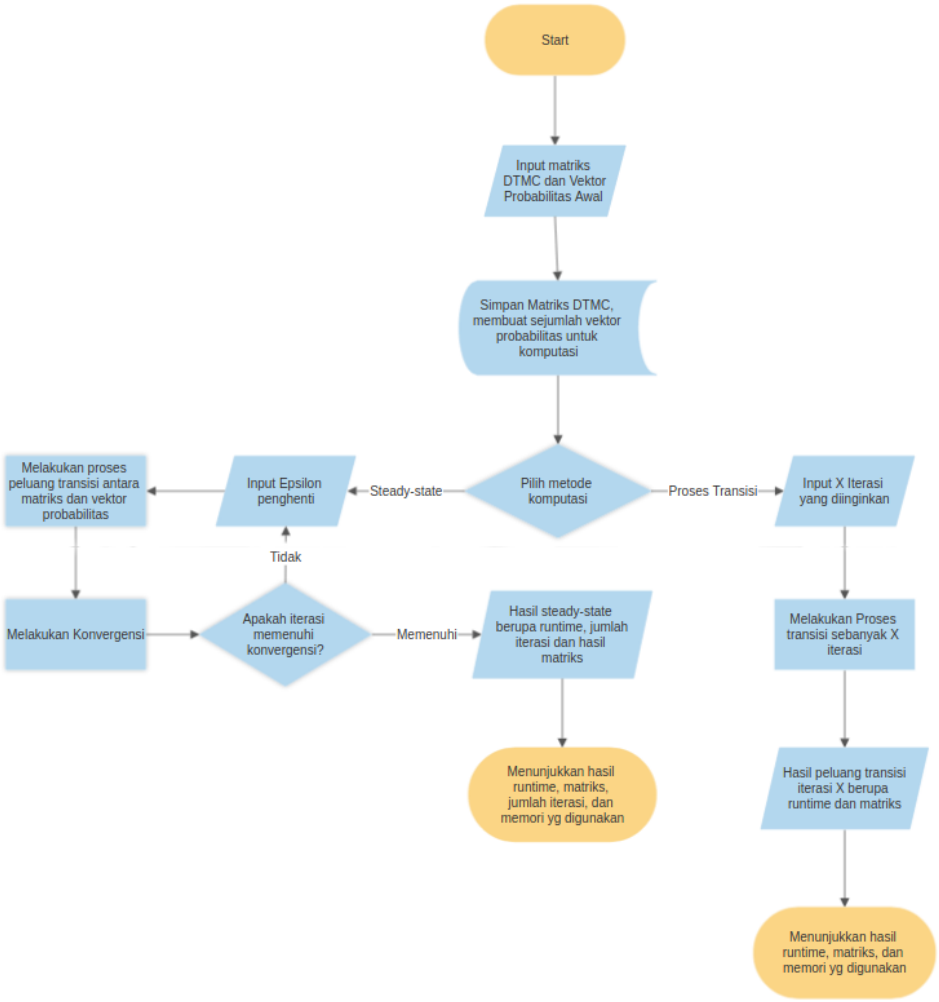
```
#jumlah state  #jumlah transisi/elemen matriks
1400           2693
#state asal/baris      #state tujuan/kolom   #probabilitas transisi/isi matriks
0                      1                     0.333333
0                      2                     0.333333
0                      3                     0.333333
1                      4                     1
2                      5                     0.5
2                      6                     0.5
3                      1                     0.25
3                      8                     0.25
3                      9                     0.25
3                      10                    0.25
4                      11                    0.333333
4                      12                    0.333333
....
....
1399 13 0.5
1399 1393 0.5
```

Gambar 4.1: Contoh isi dari file .tra

4.3 Rancangan Penyimpanan Sparse Matriks

Implementasi untuk penyimpanan sparse matriks dilakukan dalam C++. Matriks yang digunakan untuk proses stokastik disimpan terlebih dahulu dalam bentuk sparse matriks. Pada gambar 4.1, matriks DMTC yang terdapat dalam bentuk file .tra sudah dibuat dalam sparse, sehingga hanya perlu mempertahankan *sparsity* yang telat dibuat menuju implementasi program. File .tra ini dimasukkan kedalam implementasi sparse matriks menggunakan library <fstream>. Bentuk penyimpanan sparse matriks yang digunakan merupakan dalam model Compressed-sparse row atau juga disebut Yale format. Dalam penyimpanan model yale format, dibuat 3 baris satu dimensi yang diantaranya merupakan baris berisi elemen matriks yang bukan nol atau matriks transisi (AA) , lokasi kolom dari elemen matriks atau state asal dalam DTMC (JC), dan lokasi kolom dari elemen matriks atau state tujuan dalam DTMC (JR).

Setelah matriks disimpan dalam bentuk sparse matriks, sparse matriks dilakukan modifikasi untuk mempermudah perhitungan perkalian matriks pada model iterasi yang dilakukan pada Power method. Modifikasi yang dilakukan pada baris (JR) diubah menjadi baris (IA) yang berisi baris dimulai berdasarkan baris (JR). Algoritma pembuatan baris IA terdapat pada algoritma 1.



Gambar 4.2: Alur Proses Stokastik

4.4 Rancangan Proses Iterasi Stokastik

Setelah matriks disimpan dalam bentuk sparse, maka proses selanjutnya adalah perhitungan stokastik. Semua implementasi perhitungan stokastik dilakukan dalam C++. Karena perhitungan per kalian matriks tidak dilakukan dalam dua dimensi baris, melainkan satu dimensi baris dalam bentuk sparse, maka algoritme perhitungan matriks perlu disesuaikan. Pada penelitian kali ini, proses stokastik dalam peluang transisi dan *steady-state probability* menggunakan metode Power Method.

Vektor matriks transisi awal akan ditentukan melalui file .lab yang di-*export* dari aplikasi PRISM. Dalam file .lab berisi state mana yang merupakan state awal

dan state yang akan menghasilkan deadlock. Perkalian matriks yang terjadi dalam power method menggunakan sparse matriks dilakukan dengan perkalian antara vektor transisi dengan baris AA. Algoritma 3 dijadikan algoritma perkalian sparse matriks dalam penelitian ini. Hasil dari proses stokastik berupa sebuah vektor ρ di mana vektor tersebut akan digunakan untuk melakukan perhitungan Power Method pada iterasi selanjutnya, sebagai hasil akhir, atau sebagai dasar perhitungan konvergensi iterasi. Apabila digunakan untuk iterasi selanjutnya, maka matriks transisi yang baru akan menggantikan vektor ρ sebelumnya untuk digunakan pada iterasi selanjutnya.

4.4.1 Mencari Konvergensi

Apabila proses stokastik merupakan proses stationary-state probability, maka perlu dilakukan sebuah konvergensi setelah perhitungan power method sebagai penghenti perhitungan iterasi dalam proses stokastik. Terdapat dua metode konvergensi yang dilakukan dalam penelitian ini, diantaranya konvergensi absolut dan konvergensi relatif. Konvergensi absolut dilakukan dengan mencari aboslut value dari selisih antara vektor matriks transisi hasil power method yang dilakukan pada algoritma 3 pada iterasi N dengan matriks transisi pada iterasi N-1. Hasil dari konvergensi ini berupa sebuah besar error yang dijadikan dasar sebagai penghenti iterasi pada *stationary-state* probability. Algoritma konvergensi absolut bisa dilihat pada algoritma 4.

Algorithm 4: Algoritma Konvergensi Absolut

Result: error
ele = elemen matriks;
temp = temporary memory; **for** *n = 0 until max ele* **do**
    abs(n) = $\rho_{(2)}$(n) - $\rho_{(1)}$(n);
    temp = temp + (abs(n) * abs(n));
end
error = sqrt(temp);

Berberda dengan Konvergensi absolut, di mana digunakan vektor matriks transisi sebagai pembanding, dalam konvergensi relatif perbandingan dilakukan setiap elemen matriks transisi. Akan diambil hasil maksimal hasil error pada setiap elemen konvergensi relatif matriks transisi. Algoritma konvergensi relatif bisa dilihat dalam

algoritma 5.

Algorithm 5: Algoritma Konvergensi Relatif

Result: error
ele = elemen matriks;
rel() = relative array; **for** *n = 0 until max ele* **do**
  rel(n) = ($\rho_{(2)}$(n) - $\rho_{(1)}$(n)) / $\rho_{(2)}$(n);
  if *rel(n) < 0* **then**
    rel(n) = rel(n) * -1
  end
  ;
  if *rel(n) > rel(n)* **then**
    error = rel(n);
  end
end

Apabila error pada setiap iterasi sudah ditemukan, maka hasil error tersebut dibandingkan dengan ϵ yang sudah ditentukan. Apabila error lebih kecil dibanding ϵ, maka iterasi tersebut merupakan iterasi *stationary-state*

.

4.5 Rancangan Pengujian

Beberapa perhatian dalam evaluasi dan pengujian pada proses stokastik adalah sebagai berikut :

1. *Runtime* pada sejumlah iterasi dimulai dari 5 iterasi hingga 10000 iterasi serta rata-rata waktu per iterasi.
2. *Runtime* mencari stationary probability menggunakan absolut convergence
3. *Runtime* mencari stationary probability menggunakan relatif convergence
4. Perbandingan *Runtime* antar Konvergensi
5. Kompleksitas waktu dan kompleksitas memori dalam Implementasi

Pengujian Runtime akan dibagi menjadi dua, yaitu pada saat menyiapkan sparse matriks mulai dari membaca file sampai modifikasi baris (JR) menjadi baris (IA) dan saat perhitungan stokastik. Untuk pencarian *stationary-state* probability untuk

kedua konvergensi, akan digunakan perbandingan secara berurutan (matriks transisi pada saat m dan m-1). Pada Konvergensi, menggunakan ϵ yang sama dengan (Zapreev, 2008), ϵ yang digunakan untuk konvergensi absolut dan relatif adalah masing-masing 1^{-6} dan $1^{-}-1$.

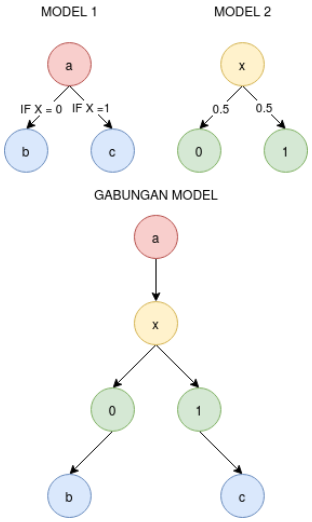
BAB V

IMPLEMENTASI

Pada bab ini akan menjelaskan semua implementasi dari algoritma yang sudah dijelaskan pada bab sebelumnya. Implementasi pembuatan matriks akan dilakukan dengan menggunakan Model checker tools PRISM dan untuk implementasi pada penyimpanan matriks serta perhitungan stokastik akan menggunakan C++.

5.1 Implementasi pembuatan matriks DTMC

Implementasi untuk pembuatan model matriks DTMC akan menggunakan pseudo-model. Pseudo-model merupakan sebuah model dimana dalam model probabilitas membutuhkan model lain untuk melakukan sebuah transisi (Kwiatkowska et al., 2004). Sehingga untuk melakukan suatu transisi diperlukan model lain untuk memenuhi kondisi transisi matriks tersebut. Visualisasi model ini bisa dilihat pada gambar 5.1.

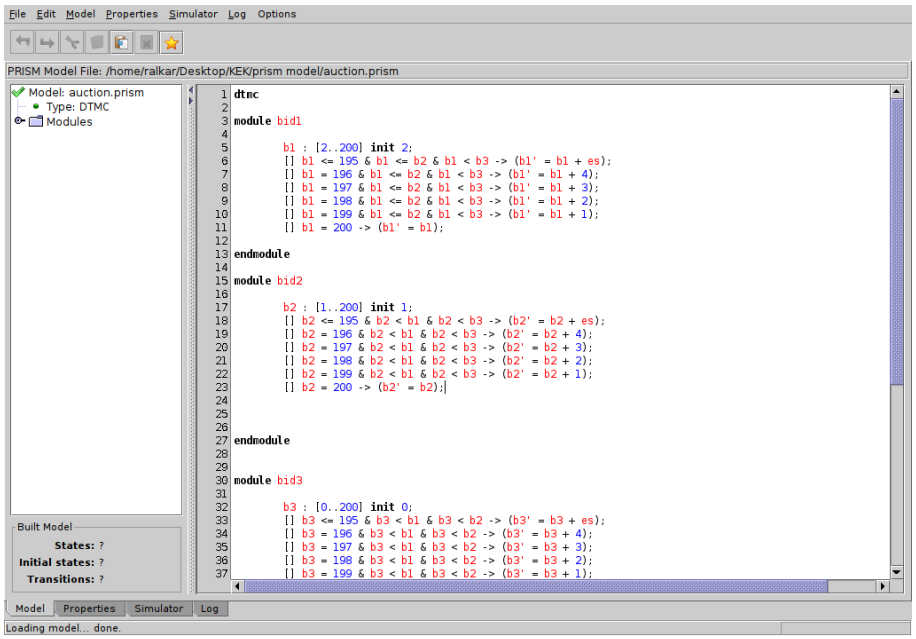


Gambar 5.1: Visualisasi Model

Dalam gambar 5.1 menunjukkan terdapat 2 model dengan state a dan state s. Dalam model 1, state a melakukan probabilitas menuju state b dan state c berdasarkan state x yang terdapat dalam model 2. Apabila menggunakan pseudocode dalam PRISM, maka kedua model tersebut akan digabung dan menjadi gabungan model yang terdapat dalam gambar 5.1. Contoh implementasi coding pada model pseudoco-

de bisa dilihat dalam gambar 5.2. Penjelasan penuh implementasi dari pseudo-model dalam PRISM terdapat dalam paper milik

Kwiatkowska et al., 2004



Gambar 5.2: Contoh coding pada salah satu model

5.2 Implementasi penyimpanan sparse matriks

5.2.1 Input File dan Penyimpanan Sparse

Digunakan library C++ <fstream> untuk melakukan input file pada program. Sparse matriks yang dimasukkan menggunakan ifstream akan langsung dimasukkan dalam bentuk array. Sparse matriks yang akan disimpan dibagi menjadi tiga array, yaitu array JR, array JC, dan array AA. Untuk program input sparse matriks bisa dilihat dalam program V.1,

```
1  int* JR;
2  int* JC;
3  double* AA;
4  int elem;
5  int states;
6
7  ifstream myfile ("matrix.tra");
```

```
8   myfile.is_open();
9   myfile >> state >> elem;
10  JC = (int*) malloc(elem*sizeof(int));
11  JR = (int*) malloc(elem*sizeof(int));
12  AA = (double*) malloc(elem*sizeof(double));
13  for (int z = 0; z < elem; z++){
14  myfile >> JR[z];
15  myfile >> JC[z];
16  myfile >> AA[z];
17  }
```

Program V.1: Input sparse matrkiks

Implementasi penyimpanan dalam array akan menggunakan pointer malloc() pada library C <stdlib.h>. Penggunaan array malloc() pada implementasi C++ tidak berbeda dengan penggunaan deklarasi array[]. Perbedaan utama di antara kedua deklarasi adalah pada malloc() menggunakan penyimpanan dinamis. Penyimpanan dinamis pada malloc() ini akan mengembalikan pointer ke sejumlah n *byte* ruang memori. Pada baris 10, 11, dan 1 setiap array akan disimpan berdasarkan jumlah transisi matriks DTMC dikalikan byte dari variable pemanggilan (int dan double). Hal ini dibutuhkan untuk menghindari batasan memori yang digunakan pada deklarasi array[a], karena elemen dan state yang digunakan akan cukup besar.

5.2.2 Implementasi pembuatan baris IA

Akan dibuat baris baru IA untuk membuat perhitungan power method lebih cepat dan efisien. Pembuatan baris baru akan menggunakan perulangan berdasarkan array JR. Setelah IA dibentuk, maka baris JR akan dikosongkan karena sudah tidak dibutuhkan.

```
1   int* IA;
2   IA = (int*) malloc(elem*sizeof(int));
3   int g = 1;
4     for (int z=0; z<elem; z++){
5       if(JR[z] > JR[z-1]){
6         IA[g] = z;
7         g++;
```

```
8       }
9       if(z==elem){
10        IA[g] = z+1; g++;}
```

Program V.2: Modifikasi Baris

Implementasi pada pembuatan baris IA ini dilakukan dengan melakukan loop pada baris JR. Apabila terdeteksi JR(i) lebih besar dari sebelumnya, maka lokasi elemen pada array tersebut akan dimasukkan pada baris IA. Kegunaan pemanggilan g = 1 pada baris 3 adalah untuk menghindari peningkatan kompleksitas loop atau pengulangan pada saat terdeteksi peningkatan angka pada baris JR.

5.3 Implementasi Power Method

Implementasi Power Method akan didasarkan pada Algoritma 2, sehingga implementasi Power Method bisa dilihat dalam Program V.3.

```
1 double* rho1;
2 double* rho2;
3 rho1 = (double*) malloc(states*sizeof(double));
4 rho2 = (double*) malloc(states*sizeof(double));
5 int iterasi = 10000;
6 rho2[0] = 1;
7   for(int a=0;a<iterasi;a++){
8     for(int p=0; q<sta;q++){rho1[p] = rho2[p]; rho2[p] = 0;}
9       for(int q=0; q<sta;q++){
10        for(int r = IA[q]; r < IA[q+1]; r++){
11          if(rho1[q]!=0){
12          rho2[JC[r]] += (AA[r] * rho1[q]);
13  } } } }
```

Program V.3: Power Method

Transisi awal untuk memulai stokastik matriks dipanggil pada $\rho_{(2)}$ karena akan pada baris 9 isi dari $\rho_{(2)}$ dipindahkan ke $\rho_{(1)}$. Fungsi loop pada baris 9 merupakan pemindahan pada hasil matriks sebelumnya yaitu $\rho_{(2)}$, menuju $\rho_{(1)}$ dimana digunakan untuk perkalian power method. Fungsi ini tidak bisa disatukan pada perulangan baris 11

karena pada pengulangan dari baris 10 sampai bars 12 membutuhkan seluruh state $\rho_{(1)}$ terpenuhi.

5.4 Implementasi Konvergensi

Berbeda dengan program V.3, pada konvergensi penghenti loop akan menggunakan error yang dihitung melalui konvergensi, dibandingkan menggunakan jumlah iterasi. Konvergensi akan dijalankan tepat setelah melakukan perhitungan power method.

5.4.1 Implementasi Konvergensi Absolut

```
1  double error;
2  double epsilon = 10e-6;
3  double* abs;
4  abs = (double*) malloc(states*sizeof(double));
5  double temp;
6  do{
7    powermethod();
8    error = 0;
9    for(int c = 0; c<states; c++){
10     abs[c] = rho2[c]-rho1[c];
11     temp += (abs[c]*abs[c]);
12   }
13   error = sqrt(temp);
14   temp = 0;
15 }while(error > epsilon);
```

Program V.4: Konvergensi Absolut

Konvergensi dimulai dengan mengurangi semua elemen antara $\rho_{(2)}$ dan $\rho_{(1)}$. Hasil dari pengurangan ini akan dikuadratkan dan disimpan dalam sebuah penyimpanan sementara. Hasil kuadrat ini akan dijumlahkan dengan hasil perhitungan absolut pada elemen lain. Setelah loop dari baris 9 sampai baris 12 sudah dilakukan, maka hasil pada penyimpanan sementara yang dihasilkan pada loop akan diakarkan pangkat dua. Hasil dari akar tersebut merupakan hasil error pada konvergensi absolut.

5.4.2 Implementasi Konvergensi Relatif

```
1  double error;
2  double epsilon = 10e-1;
3  double* rel;
4  rel = (double*) malloc(states*sizeof(double));
5  do{
6    powermethod();
7    error = 0;
8    for(int b = 0; b<sta; b++){
9      if(rho2[b] != 0) {rel[b] = abs(rho2[b] - rho1[b]) / abs(rho2[b])};
10     if(rel[b] > error) error = rel[b];
11   }
12 } while(error > epsilon);
```

Program V.5: Konvergensi Relatif

Berbeda dengan implementasi pada konvergensi aboslut, hasil error pada perhitungan konvergensi sudah ditemukan di dalam loop. Pada baris ke-7, dilakukan reset pada error agar pada program baris ke 10 tidak membandingkan dengan error pada iterasi sebelumnya. Dalam baris 9, dilakukan pengecekan apabila $\rho_{(2)}$ tidak sama dengan nol agar hasil tidak kembali infinite pada saat melakukan pembagian.

BAB VI

HASIL DAN PEMBAHASAN

Setelah dilakukan pengujian menggunakan implementasi yang telah dibuat, hasil yang diukur adalah runtime dari masing-masing algoritma. Runtime yang akan diuji berupa runtime persiapan sparse matriks, runtime per iterasi, dan runtime total. Semua runtime diukur dalam satuan detik (*seconds*).

6.1 Kasus Uji

Pada penelitian ini terdiri dari 5 model matriks DTMC. Kasus uji bisa dilihat pada Tabel 6.1

Tabel 6.1: Kasus Uji

Matriks	State	Elemen	Densitas	Persiapan matriks (s)
NAND	32.934.572	52.080.692	$4.8014 \times 10^{8}\%$	40,231
BRP	1.327.112	1.769.475	$1.0046 \times 10^{6}\%$	1,16954
Crowds	10.633.591	18.261.191	$1.6149 \times 10^{7}\%$	27,137
Contract	66.060.286	67.108.861	$1.5377 \times 10^{8}\%$	41,0166
Rubber	2.500.000	29.989.905	$4.7983 \times 10^{6}\%$	29,05

Berikut ini merupakan penjelasan pada setiap model :

1. NAND Multiplexing (NAND) : Model yang dipopulerkan oleh Norman et al., 2005 ini merupakan model melakukan proses *multiplexing* pada jaringan komputer menggunakan gerbang NAND.

2. Bounded Retransmission Protocol (BRP) : Sebuah varian dari alternating bit protocol, di mana protokol mengirimkan file dalam sejumlah potongon hanya dengan transmisi ulang dalam jumlah terbatas. Model ini dipopulerkan oleh D'Argenio et al., 2001

3. Crowds Protocol (Crowds) : Sistem di mana sebuah grup mengirim pesan satu sama lain tanpa mengetahui pengirim pesan tersebut dan tanpa mengetahui pesan tersebut rusak atau tidak. Model ini dipolulerkan oleh Shmatikov, 2004

4. Contract Signing Protocol (Contract) : Model di mana sebanyak N orang melakukan persetujuan kontrak secara bersama tanpa mempercayai satu sama lain

dan tanpa pihak ketiga. Model ini dipopulerkan oleh Norman dan Shmatikov, 2006

5. Rubber Banding (Rubber) : Sistem yang dipakai agar sebuah permainan lebih adil dengan memberikan *bonus* kepada pemain yang sedang kalah atau merugikan pemain yang sedang menang. Model kali ini akan memakai Rubber Banding pada sebuah balapan otomotif.

6.2 Penggunaan Memori

Untuk melihat penggunaan memori pada setiap proses maka akan dilihat implementasi pada bab 5. Akan dibagi menjadi dua bagian, di antaranya proses pembuatan IA dan proses iterasi Power Method.

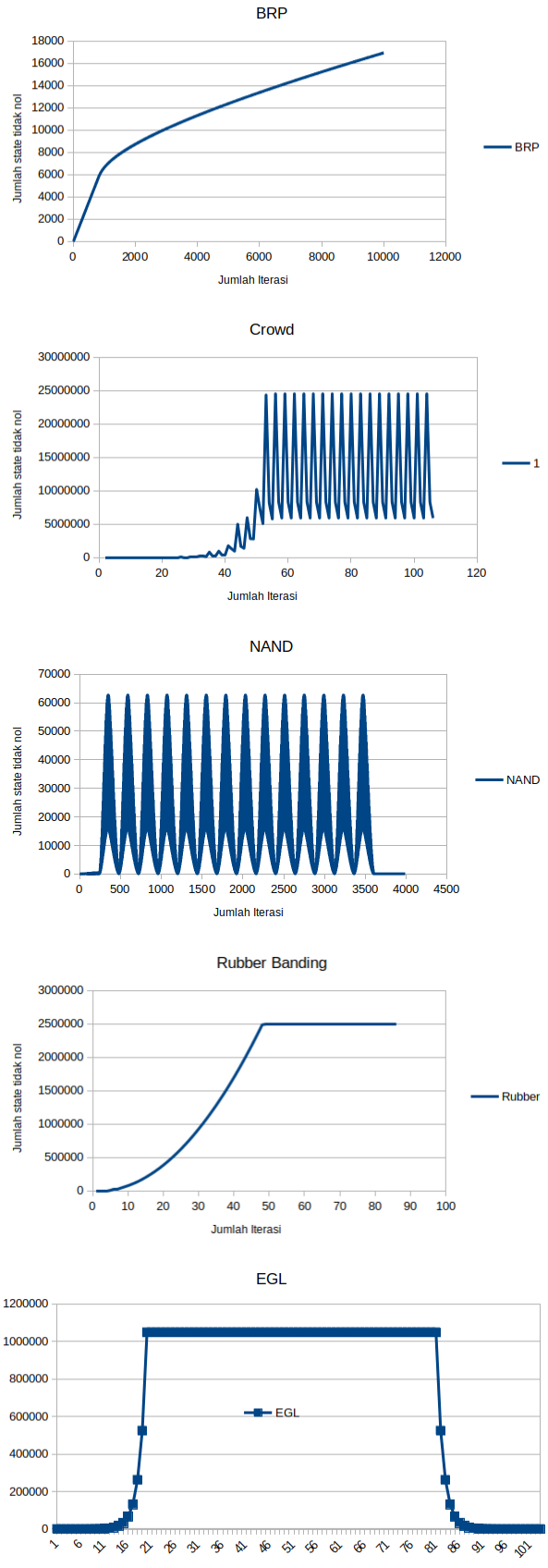
Pada proses pembuatan IA dalam program V.2 hanya menggunakan dua array memory, di mana array tersebut merupakan array JR dan array IA. Kedua array tersebut memiliki panjang sebanyak state dari matriks DTMC, sehingga memori yang digunakan berupa jumlah state matriks ditambah dengan jumlah elemen matriks.

Pada proses perhitungan *Power Method* dalam program V3, terdapat 5 array yang digunakan dalam implementasi, di antaranya array $\rho_{(2)}$, $\rho_{(1)}$, IA, AA, dan array JC. Pada baris 11 terdapat pengambilan keputusan saat $\rho_{(1)}$ tidak sama nol. Sehingga tidak semua memori dalam array JC dan AA digunakan, maka memori akan berubah sesuai beberapa banyak angka tidak nol dalam array $\rho_{(1)}$. Untuk melihat detail jumlah array $\rho_{(1)}$ tidak nol, berikut merupakan detail dalam gambar 6.1

Dari Gambar 6.1, Bisa dilihat bahwa pada setiap model matriks memiliki kebutuhan space dan pola perubahan space yang berbeda. Dari kelima model, hanya 2 model memiliki jumlah state yang tidak nol pada ρ mengalami kenaikan secara konstan. Sehingga, kompleksitas memori yang digunakan untuk melakukan proses stokastik merupakan $O(N + M)$ di mana M berdasarkan pada model matriks DTMC yang diuji, dan N merupakan panjang array $\rho_{(2)}$, $\rho_{(1)}$, serta IA, yaitu 3 $\times$ states. Dalam proses konvergensi absolut, maka N berubah menjadi 4 $\times$ jumlah state dan dalam proses konvergensi relatif, maka faktor M akan bertambah satu

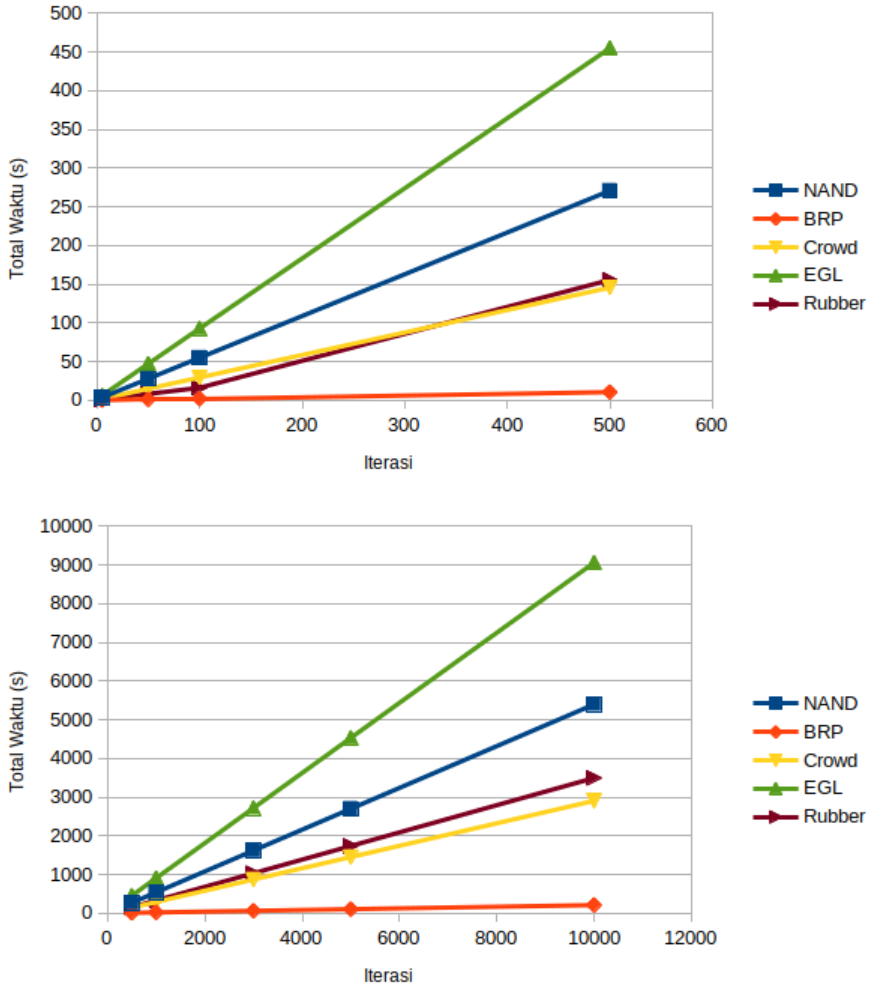
6.3 Perbandingan runtime Berdasarkan Waktu Iterative

Dilakukan perbandingan performa Power Method pada waktu iterasi tertentu pada setiap matriks DTMC. Pengambilan runtime dimulai dari pembuatan array IA



Gambar 6.1: Jumlah ρ tidak nol dalam masing-masing model

sampai iterasi 10000. Hasil dari runtime dapat dilihat pada Gambar Grafik 6.2.

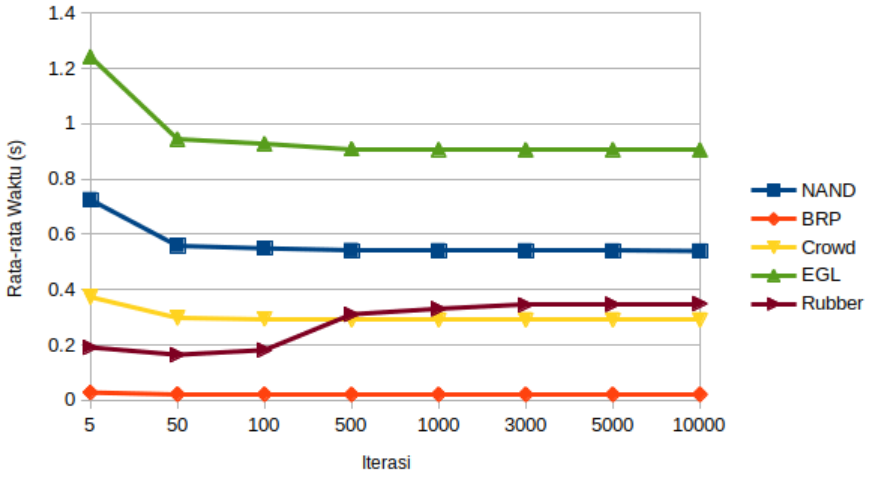


Gambar 6.2: Grafik Total runtime pada sejumlah iterasi

Pada Grafik 6.2 menunjukkan setiap matriks mempunyai runtime yang berbeda. EGL memiliki runtime terlama dikarenakan EGL memiliki jumlah state dan jumlah transisi terbanyak di antara kasus uji. Pada model Crowds dan Rubber, waktu yang dibutuhkan pada kedua model untuk melakukan 10000 iterasi tidak terlalu jauh walaupun perbandingan state di antara kedua matriks berbanding jauh. Hal ini disebabkan karena elemen matriks atau probabilitas matriks pada Rubber lebih banyak jika dibandingkan dengan Crowds, sehingga model Rubber bisa melakukan proses transisi lebih lama dibanding Crowds walaupun dengan state lebih sedikit.

Pada kelima model matriks, runtime pada transisi berjalan dengan kompleksitas waktu linear. untuk melihat lebih detail rata-rata runtime pada setiap iterasi bisa

dilihat pada gambar 6.3



Gambar 6.3: Grafik rata-rata runtime setiap iterasi

Pada kasus rata-rata runtime, runtime pada setiap model matriks merupakan linear. Sehingga kompleksitas waktu untuk melakukan proses iterasi transisi matriks merupakan O(kN). Dalam waktu kompleksitas O(kN) sangat susah untuk menentukan faktor k karena terdapat banyak faktor dalam menjalankan implementasi program, seperti jumlah state dan jumlah elemen matriks.

6.4 Perbandingan runtime *stationary-state probability*

Dicari *stationary-state* pada setiap matriks dalam konvergensi absolut dengan epsilon sebesar 1×10^{-6} dan pada konvergensi relatif dengan epsilon sebesar 1×10^{-1}. Ditampilkan jumlah iterasi dan runtime yang dibutuhkan untuk mencapai *stationary-state*, rata-rata runtime per iterasi, dan probabilitas terakhir. Hasil dari proses *steady-state probability* menggunakan konvergensi absolut dapat dilihat dalam tabel 6.2 dan konvergensi relatif pada tabel 6.3.

Pada pencarian *stationary-state* menggunakan konvergensi absolut, model NAND dan BRP tidak mencapai target epsilon, sehingga hanya mencapai maksimal batasan iterasi yang digunakan. Model Crowd mendapatkan hasil *stationary-state* pada 221 tierasi. Model EGL mencapai *stationary-state* pada saat iterasi 103 dengan hasil error terakhir berupa nol. Hal ini bisa terjadi apabila $\rho_{(103)}$ dan $\rho_{(102)}$ mempunyai nilai probabilitas matriks yang sama persis. Dalam gambar 6.1, dapat dilihat bahwa proses EGL pada iterasi lebih dari 103 hanya mempunyai 1 state. Hal tersebut menjadi salah

Tabel 6.2: Hasil Konvergensi Absolut

Matriks	Iterasi	Rata-rata (s)	Error terakhir
NAND	10000	0.806	0.207
BRP	10000	0.03	2.29×10^{1}
Crowd	221	0.4753	8.21×10^{7}
EGL	103	1.44	0
Rubber	8361	0.391	9.996×10^{7}

satu faktor pada model EGL mendapatkan *steady-state* pada iterasi ke 103. Terakhir merupakan model Rubber di mana hasil *stationary-state* tertemukan pada iterasi yang cukup dekat dengan batasan.

Tabel 6.3: Hasil Konvergensi Relatif

Matriks	Iterasi	Rata-rata (s)	Error terakhir	Keterangan
NAND	10000	0.736	1	INF
BRP	10000	0.028	1	INF
Crowd	10000	0.503	1	INF
EGL	103	1.364	0	-
Rubber	971	0.322	0.097	-

Hasil pada *steady-state probability* menggunakan konvergensi relatif berbeda dengan konvergensi absolut. Pada program V.5, apabila tidak terdapat pengambilan keputusan "rho2[b] != 0", maka akan mempunyai kemungkinan untuk menghasilkan probabilitas tak terhingga (INF) seperti yang terjadi dalam beberapa model pada kolom keterangan.

Apabila terdapat pengambilan keputusan tersebut, NAND, BRP, dan Crowd menghasilkan error 1 pada setiap iterasi dan masih belum bisa menemukan hasil konvergensi dalam iterasi 10000. Pada model EGL, hasil dari iterasi dan probabilitas error terakhir sama seperti dalam konvergensi absolut. Pada model rubber *stationary-state* dapat ditemukan dalam 971 iterasi dengan probabilitas mendekati epsilon.

Dari kedua perbandingan Konvergensi, ditunjukkan bahwa runtime empat dari lima proses Konvergensi Relatif lebih cepat dibanding dengan konvergensi absolut. Tetapi hasil konvergensi mempunyai kemungkinan untuk menghasilkan probabilitas INF.

6.4.1 Perbandingan dengan PRISM

Sampai saat ini, terdapat empat tools model checking yang bisa melakukan proses probabilitas dan hanya dua diantaranya bisa melakukan proses transisi dan s*teady-state probability* dalam DTMC, diantaranya MRMC dan PRISM. Dari kedua tools, MRMC hanya memiliki metode gauss-seidel dan gauss-jacobi untuk *steady-state*. Sehingga pada bagian ini akan melihat bagaimana PRISM melakukan power method menggunakan sparse matriks. Hasil dari *steady-state probability* menggunakan tool model checking PRISM pada tabel 6.4 untuk konvergensi absolut dan tabel 6.5 untuk konvergensi relatif.

Tabel 6.4: Hasil Konvergensi Absolut pada PRISM

Matriks	Iterasi	Rata-rata (s)	Error terakhir
NAND	3602	0.5001	0.113
BRP	10000	0.0224	0.386
Crowd	221	0.301	8×10^{-6}
EGL	103	0.909	0
Rubber	9956	0.1121	9×10^{-6}

Tabel 6.5: Hasil Konvergensi Relatif pada PRISM

Matriks	Iterasi	Rata-rata (s)	Error terakhir	Keterangan
NAND	3602	0.5	1	INF
BRP	10000	0.0217	1	INF
Crowd	10000	0.647	1	INF
EGL	103	0.884	0	
Rubber	971	0.108	0.0991	

Dari tabel 6.4 dan 6.5, terdapat beberapa perbedaan dengan tabel 6.2 dan 6.3. Selain Proses *steady-state* menggunakan konvergensi relatif pada model Crowd, PRISM melakukan proses *steady-state* lebih cepat. Selain itu, pada model NAND kedua metode konvergensi berhenti pada iterasi 3602. Hal ini dikarenakan pada PRISM memiliki proses reachability menggunakan Breadth-First Search (BFS) untuk menghindari iterasi tak terbatas lebih awal. Untuk detail mengenai proses reachability menggunakan BFS bisa dilihat dalam Wijs, 2016. Model Rubber menemukan *steady-state probability* pada iterasi berbeda dengan tabel 6.2 dan 6.3, tetapi memiliki hasil error terakhir yang sama. Perbedaan terakhir, pada PRISM saat melakukan proses

steady-state probability menggunakan konvergensi absolut pada model Rubber mendapatkan iterasi lebih lama. Meskipun dengan perbandingan diatas, kedua tools yang dibuat mempunyai hasil yang serupa.

BAB VII

KESIMPULAN DAN SARAN

7.0.1 Kesimpulan

Kesimpulan yang didapat pada penelitian ini adalah:

1. Kompleksitas ruang yang dibutuhkan untuk sparse matriks dalam proses stokastik matriks berupa $O(m + n)$, di mana n merupakan jumlah state dan m merupakan jumlah vektor ρ yang tidak nol pada setiap iterasi.

2. Kompleksitas waktu yang dijalankan dalam proses stokastik matriks menggunakan Power Method yaitu linear $O(n)$ di mana n merupakan gabungan antara jumlah state dan jumlah elemen matriks dalam model DTMC

3. Dari pengujian yang dilakukan, Konvergensi relatif dapat melakukan runtime konvergensi lebih cepat dibandingkan konvergensi absolut, tetapi mempunyai kemungkinan akan menghasilkan probabilitas tak terhingga.

7.0.2 Saran

Beberapa saran dalam penelitian ini:

1. Membandingkan dengan metode penyimpanan lain yang bisa digunakan dalam proses stokastik seperti gabungan antara multi-terminal binary decision diagrams (MTBDD) dengan sparse matriks

2. Membandingkan waktu kompleksitas dengan metode lain untuk proses stokastik seperti Gauss-seidel dan Jacobi

DAFTAR PUSTAKA

Akhunov, R., Kuksenko, S., Salov, V., dan Gazizov, T. (2013), Sparse matrix storage formats and acceleration of iterative solution of linear algebraic systems with dense matrices, *Journal of Mathematical Sciences* 191.1, pp. 10–18.

Alur, R., Courcoubetis, C., dan Dill, D. (1990), Model-checking for real-time systems, *[1990] Proceedings. Fifth Annual IEEE Symposium on Logic in Computer Science*, IEEE, pp. 414–425.

Anton, H. dan Rorres, C., 2013, *Elementary linear algebra: applications version*, John Wiley & Sons.

Aritonang, K., Tan, A., Ricardo, C., Surjadi, D., Fransiscus, H., Pratiwi, L., Nainggolan, M., Sudharma, S., dan Herawati, Y. (2020), Analisis Pertambahan Pasien COVID-19 di Indonesia Menggunakan Metode Rantai Markov, *Jurnal Rekayasa Sistem Industri* 9.2, pp. 69–76.

Barnat, J., Brim, L., Ceška, M., dan Lamr, T. (2009), CUDA accelerated LTL model checking, *2009 15th International Conference on Parallel and Distributed Systems*, IEEE, pp. 34–41.

Chatterjee, K. dan Henzinger, T. A. (2008), Value iteration, *25 Years of Model Checking*, Springer, pp. 107–138.

Clarke, E. M. (2008), The birth of model checking, *25 Years of Model Checking*, Springer, pp. 1–26.

D'Argenio, P., Jeannet, B., Jensen, H., dan Larsen, K. (2001), Reachability analysis of probabilistic systems by successive refinements, *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modelling and Verification (PAPM/PROBMIV'01)*, ed. by L. de Alfaro dan S. Gilmore, vol. 2165, LNCS, Springer, pp. 39–56.

Duflot, M., Kwiatkowska, M., Norman, G., dan Parker, D. (2006), A formal analysis of Bluetooth device discovery, *International journal on software tools for technology transfer* 8.6, pp. 621–632.

Feinberg, B. N. dan Chiu, S. S. (1987), A method to calculate steady-state distributions of large Markov chains by aggregating states, *Operations Research* 35.2, pp. 282–290.

Hasnaeni, H. (2017), Menilai dan Memprediksi Akuisisi Kepemilikan Rumah dari Produk Keuangan Berdasarkan Discrete Time Markov Chains Menggunakan Latent Markov Models (Studi Kasus: PT Ghanitun Hasanun), PhD thesis, Universitas Islam Negeri Alauddin Makassar.

Kwiatkowska, M., Norman, G., dan Parker, D. (2011), PRISM 4.0: Verification of Probabilistic Real-time Systems, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ed. by G. Gopalakrishnan dan S. Qadeer, vol. 6806, LNCS, Springer, pp. 585–591.

Kwiatkowska, M., Norman, G., dan Parker, D. (2004), PRISM 2.0: A tool for probabilistic model checking, *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.* IEEE, pp. 322–323.

Langville, A. N. dan Meyer, C. D. (2006), Updating Markov chains with an eye on Google's PageRank, *SIAM journal on matrix analysis and applications* 27.4, pp. 968–987.

Larson, R, Edwards, B., dan Falvo, D. (2004), *Elementary Linear Algebra.*

Myers, D. S., Wallin, L., dan Wikström, P. (2017), An introduction to Markov chains and their applications within finance,

Norman, G., Parker, D., Kwiatkowska, M., dan Shukla, S. (2005), Evaluating the Reliability of NAND Multiplexing with PRISM, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24.10, pp. 1629–1637.

Norman, G. dan Shmatikov, V. (2006), Analysis of Probabilistic Contract Signing, *Journal of Computer Security* 14.6, pp. 561–589.

Philippe, B., Saad, Y., dan Stewart, W. J. (1992), Numerical methods in Markov chain modeling, *Operations research* 40.6, pp. 1156–1179.

Pissanetzky, S., 1984, *Sparse Matrix Technology-electronic edition*, Academic Press.

Saad, Y., 2003, *Iterative methods for sparse linear systems*, SIAM.

Shmatikov, V. (2004), Probabilistic Model Checking of an Anonymity System, *Journal of Computer Security* 12.3/4, pp. 355–377.

Stewart, W. J., 2009, *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*, Princeton university press.

Stewart, W. J. dan Wu, W. (1992), Numerical experiments with iteration and aggregation for Markov chains, *ORSA Journal on Computing* 4.3, pp. 336–350.

Syuhada, K. I. (2012), *Bab 4 : Matriks Stokastik dan Rantai Markov Lecture Notes*.

Terry Bahill, A dan Henderson, S. J. (2005), Requirements development, verification, and validation exhibited in famous failures, *Systems engineering* 8.1, pp. 1–14.

White, J. B. dan Sadayappan, P. (1997), On improving the performance of sparse matrix-vector multiplication, *Proceedings Fourth International Conference on High-Performance Computing*, IEEE, pp. 66–71.

Wijs, A. (2016), BFS-based model checking of linear-time properties with an application on GPUs, *International Conference on Computer Aided Verification*, Springer, pp. 472–493.

Zapreev, I. S., 2008, *Model checking Markov chains: techniques and tools.* University of Twente, Enschede, Netherlands.