SKRIPSI

MODULARISASI ALGORITMA SHORTEST PATH PADA PERANGKAT LUNAK KIRI MENGGUNAKAN STRATEGY PATTERN

Muhammad Aldi Rivandi

NPM: 6182001029

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2025

UNDERGRADUATE THESIS

MODULARIZATION OF THE SHORTEST PATH ALGORITHM ON KIRI SOFTWARE USING STRATEGY PATTERN

Muhammad Aldi Rivandi

NPM: 6182001029

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY

2025

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

DAFTAR ISI

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Perangkat lunak KIRI adalah perangkat lunak berbasis web yang dirancang untuk membantu pengguna menemukan rute perjalanan menggunakan angkutan kota (angkot). Pada perangkat lunak KIRI, pengguna dapat memasukkan titik awal perjalanan dan titik tujuan. KIRI kemudian akan mencarikan berbagai alternatif rute angkot yang bisa digunakan untuk mencapai tujuan tersebut.

Arsitektur aplikasi KIRI terbagi menjadi dua bagian utama. Bagian frontend, yang dinamakan Tirtayasa, dibangun menggunakan bahasa pemrograman PHP dan mengandalkan basis data MySQL untuk menyimpan serta mengelola data. Selain itu, Tirtayasa juga menggunakan framework CodeIgniter 3. Saat menerima permintaan pencarian, Tirtayasa meneruskannya ke bagian backend, yaitu NewMenjangan. Hasil dari NewMenjangan kemudian diformat agar dapat dibaca dengan baik oleh pengguna. Bagian ini diimplementasikan dalam bahasa pemrograman Java dan berperan penting dalam perhitungan rute optimal.

NewMenjangan merupakan program daemon yang berjalan secara otomatis saat server dinyalakan dan terus beroperasi hingga server dimatikan. Daemon sendiri adalah program komputer yang berjalan di latar belakang dan tidak berinteraksi langsung dengan pengguna[1]. Pada saat eksekusi, NewMenjangan terhubung ke basis data MySQL untuk mengambil data rute angkot yang tersimpan dalam format LineString. LineString adalah tipe data geometri yang digunakan dalam basis data, untuk merepresentasikan garis atau rangkaian segmen garis[2]. Setiap titik pada LineString merepresentasikan lokasi potensial untuk penumpang naik atau turun. Dari data tersebut, NewMenjangan membangun *weighted graph* dalam memori (RAM) dalam bentuk *adjacency list* dan melakukan prakomputasi. Setiap titik pada LineString menjadi *node*, dan antara titik ke-i dan titik ke-(i+1) dihubungkan dengan *edge*. Jika ada dua titik dari rute angkot berbeda yang berdekatan (jarak di bawah konstanta tertentu), maka dibuatkan juga *edge*, yang menunjukkan kemungkinan seseorang dapat turun dari suatu angkot dan naik ke angkot lainnya untuk meneruskan perjalanan.

Saat NewMenjangan menerima permintaan pencarian dari titik A ke titik B, kedua titik tersebut dijadikan *node* sementara, dan dibuatkan *edge* sementara ke *node-node* yang sudah ada sebelumnya, jika jaraknya di bawah konstanta tertentu. Pencarian jarak terdekat pada graf tersebut dilakukan menggunakan algoritma Dijkstra versi teroptimasi (*priority queue* dengan struktur data *heap*). Proses ini dapat dilakukan secara paralel dengan aman (*thread-safe*) tanpa mengubah graf utama.

Pada saat ini algoritma yang digunakan KIRI masih terikat dengan algoritma Dijkstra. Oleh

[1] https://www.ibm.com/docs/en/aix/7.1?topic=processes-

[2] https://dev.mysql.com/doc/refman/8.4/en/gis-linestring-property-functions.html

karena itu, pada tugas akhir ini akan diimplementasikan algoritma lainnya, yaitu algoritma A-star dan Floyd Warshall sebagai *concrete strategy*. Selain itu, akan dilakukan juga penerapan arsitektur kelas *strategy pattern* sehingga aplikasi KIRI akan menjadi lebih fleksibel dalam pemilihan algoritma *shortest path* yang akan digunakan dan juga memudahkan apabila akan dilakukan perubahan atau perbaikan pada suatu algoritma yang digunakan.

1.2 Rumusan Masalah

1. Bagaimana cara melakukan perubahan kode pada NewMenjangan untuk menerapkan *strategy pattern*?
2. Bagaimana mengimplementasikan algoritma A-star dan Floyd Warshall sebagai *concrete strategy*?

1.3 Tujuan

1. Melakukan perubahan arsitektur kelas dengan menerapkan *strategy pattern*.
2. Melakukan implementasi algoritma A-star dan Floyd Warshall.

1.4 Batasan Masalah

...

1.5 Metodologi

Metodologi yang akan digunakan dalam pembuatan tugas akhir ini adalah sebagai berikut:

1. Melakukan eksplorasi fungsi-fungsi dan cara kerja perangkat lunak KIRI.
2. Mempelajari modul-modul yang terdapat pada Tirtayasa dan NewMenjangan.
3. Mempelajari bahasa pemrograman PHP dan framework CodeIgniter 3.
4. Melakukan studi literatur mengenai penerapan arsitektur kelas *strategy pattern*.
5. Mempelajari cara kerja algoritma Dijkstra, A-star, dan Floyd Warshall.
6. Mengubah implementasi algoritma Dijkstra yang sudah ada ke dalam *strategy pattern*.
7. Mengimplementasikan algoritma A-star dan Floyd Warshall.
8. Melakukan pengujian dan eksperimen.
9. Menulis dokumen tugas akhir.

1.6 Sistematika Pembahasan

Tugas akhir ini akan disusun menjadi beberapa bab sebagai berikut:

- **Bab 1:** Pendahuluan
  Bab ini berisi latar belakang, rumusan masalah,tujuan, batasan masalah, metodologi, dan sistematika pembahasan.
- **Bab 2:** Landasan Teori ...
- **Bab 3:** Analisis ...

BAB 2

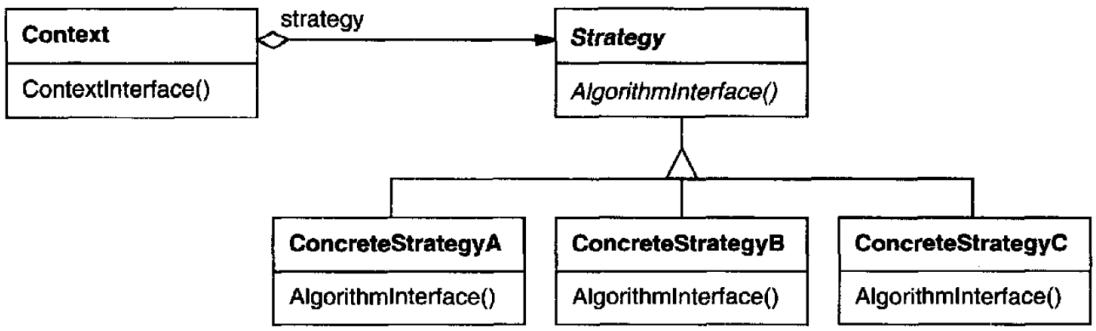
LANDASAN TEORI

2.1 Strategy Pattern

Strategy Pattern adalah sebuah design pattern yang memungkinkan objek untuk memilih saat runtime. Ini mendefinisikan sekumpulan algoritma, merangkum masing-masing algoritma, dan membuatnya dapat dipertukarkan. Hal ini memungkinkan klien untuk memilih algoritma berdasarkan kebutuhan mereka tanpa mengubah kode yang menggunakan algoritma tersebut. Ide utama penerapan Strategy Pattern adalah merancang antarmuka fleksibel yang dapat bekerja dengan algoritma berbeda tanpa perlu mengubah kode ketika algoritma baru diperkenalkan.

Strategy Pattern cocok digunakan dalam skenario-skenario berikut:

1. Ketika ada beberapa kelas terkait yang hanya berbeda dalam perilakunya. Strategy Pattern memungkinkan kelas dikonfigurasikan dengan salah satu dari banyak kemungkinan perilaku.
2. Ketika ada kebutuhan untuk varian algoritma yang berbeda.
3. Ketika suatu algoritma melibatkan data yang harus tetap tersembunyi dari klien. Strategy Pattern memastikan bahwa struktur data yang kompleks dan spesifik algoritma tidak terekspos.
4. Ketika sebuah kelas mendefinisikan banyak perilaku, dan perilaku ini diimplementasikan melalui beberapa pernyataan kondisional. Dalam kasus seperti ini, Strategy Pattern menyederhanakan kode dengan memindahkan cabang kondisional terkait ke dalam kelas Strategi masing-masing



Gambar 2.1: Struktur Strategy Pattern

2.2 MySQL

MySQL adalah *open source relational database management system* (RDBMS) yang digunakan untuk menyimpan dan mengelola data yang dikembangkan oleh Oracle. MySQL adalah salah satu sistem manajemen basis data paling populer di dunia. SQL adalah singkatan dari *Structured Query Language* yang merupakan bahasa pemrograman yang digunakan untuk mengambil, memperbarui, menghapus, dan memanipulasi data dalam database relasional. Sebagai database relasional, MySQL menyimpan data dalam tabel baris dan kolom yang disusun dalam skema. Skema mendefinisikan bagaimana data diatur dan disimpan serta menjelaskan hubungan antara berbagai tabel.
Manfaat utama MySQL meliputi hal berikut:

- ***Ease Of use***. Pengembang dapat menginstal MySQL dengan mudah, dan database mudah dikelola.
- ***Reliability***. MySQL adalah salah satu database yang paling matang dan banyak digunakan. Teknologi ini telah diuji dalam berbagai skenario selama hampir 30 tahun, termasuk oleh banyak perusahaan terbesar di dunia.
- ***Scalability***. MySQL berskala untuk memenuhi permintaan aplikasi yang paling banyak diakses. Arsitektur replikasi asli MySQL memungkinkan organisasi, termasuk Facebook, Netflix, dan Uber, meningkatkan aplikasi untuk mendukung puluhan juta pengguna atau lebih.
- ***Performance***. MySQL adalah sistem database tanpa administrasi yang terbukti berkinerja tinggi dan hadir dalam berbagai edisi untuk memenuhi hampir semua permintaan.
- ***High availability***. MySQL menghadirkan serangkaian teknologi replikasi asli dan terintegrasi penuh untuk ketersediaan tinggi.
- ***Security***. Keamanan data mencakup perlindungan data dan kepatuhan terhadap peraturan industri dan pemerintah.
- ***Flexibility***. Penyimpanan Dokumen MySQL memberi pengguna fleksibilitas maksimum dalam mengembangkan aplikasi database tradisional bebas skema SQL dan NoSQL.

2.2.1 LineString

Di MySQL, LineString adalah tipe data spasial yang digunakan untuk merepresentasikan geometri linier, seperti jalur atau garis, dalam ruang dua dimensi. MySQL menyediakan berbagai fungsi spasial untuk bekerja dengan geometri LINESTRING, termasuk mengukur panjangnya, menentukan perpotongannya, dan banyak lagi. Fitur-fitur ini sangat berguna dalam aplikasi Sistem Informasi Geografis (GIS) dimana data spasial seperti jalan atau sungai perlu disimpan dan dianalisis.
Format untuk mendefinisikan LINESTRING adalah sebagai berikut:

```
LINESTRING(x1 y1, x2 y2, x3 y3, ...)
```

2.3 Algoritma Dijkstra

[1] Algoritma Dijkstra menyelesaikan masalah *single-source shortest path*, yaitu menemukan jalur terpendek dari satu titik asal ke semua titik lainnya dalam sebuah graf berarah dengan bobot tepi non-negatif. Proses ini dimulai dengan menginisialisasi perkiraan jarak terpendek dari titik asal s ke semua titik lain. Jarak titik asal sendiri diatur ke nol, sedangkan titik lainnya diatur sebagai tak hingga. Algoritma ini menggunakan sebuah struktur *min-priority queue* (antrean prioritas minimum) yang menyimpan titik-titik dengan prioritas sesuai dengan perkiraan jarak terpendek mereka dari titik asal.

Selama eksekusi, algoritma Dijkstra akan secara bertahap memindahkan titik dengan estimasi jarak terpendek dari antrean ke dalam satu set S, yang menampung titik-titik dengan jarak terpendek yang sudah final. Untuk setiap titik u yang baru dipindahkan ke dalam set S, algoritma akan memeriksa setiap tetangganya v dan memperbarui perkiraan jarak terpendek ke v jika melalui u memberikan jarak yang lebih pendek. Proses ini dikenal sebagai "relaksasi" tepi, yaitu memperbarui perkiraan jarak dan menunjuk u sebagai pendahulu v bila ditemukan jalur yang lebih optimal.

Proses algoritma berlanjut hingga semua titik di graf telah diproses, sehingga jarak terpendek dari titik asal s ke setiap titik yang dapat dijangkau sudah final. Kompleksitas waktu dari algoritma ini bergantung pada implementasi antrean prioritas yang digunakan, dengan menggunakan *Fibonacci heap*, algoritma Dijkstra dapat mencapai kompleksitas $O(VlogV + E)$, yang efisien untuk graf yang jarang (sparse). Algoritma ini sangat bermanfaat dalam berbagai aplikasi yang melibatkan pencarian jalur terpendek, seperti sistem navigasi dan perutean jaringan.

2.4 Algoritma Floyd-Warshall

[1] Algoritma Floyd-Warshall menyelesaikan masalah jalur terpendek untuk semua pasangan titik dalam graf berarah dengan menggunakan pendekatan pemrograman dinamis. Algoritma ini sangat berguna untuk graf yang memiliki bobot sisi negatif, selama tidak terdapat siklus dengan bobot negatif dalam graf tersebut. Pendekatan ini menghitung jalur terpendek antara semua pasangan titik dengan menggunakan tabel bobot antar titik dan mengulanginya secara bertahap untuk mencapai solusi optimal.

Langkah pertama dalam algoritma ini adalah mempersiapkan matriks bobot jalur terpendek yang akan terus diperbarui. Algoritma memulai dengan menganggap setiap titik memiliki jalur langsung ke dirinya sendiri dengan bobot nol, sementara bobot antar titik lain mengikuti nilai bobot sisi pada graf. Secara rekursif, algoritma memperbarui jalur terpendek dengan menambahkan titik perantara secara bertahap, yaitu jika titik k menjadi perantara dari titik i ke j, maka bobot jalur terpendek d_{ij} akan diperbarui menjadi minimum dari d_{ij} atau $d_{ik} + d_{kj}$. Proses ini mengoptimalkan semua jalur antara pasangan titik dengan menambahkan satu titik perantara setiap kali iterasi dilakukan. Algoritma Floyd-Warshall memiliki kompleksitas waktu $O(V^3)$ karena terdiri dari tiga lapisan perulangan untuk semua titik dalam graf, dengan V sebagai jumlah titik. Meskipun kompleksitasnya tinggi, algoritma ini cukup praktis untuk graf ukuran sedang dan memiliki struktur yang sederhana sehingga dapat diimplementasikan secara efisien. Selain itu, hasil algoritma ini dapat digunakan untuk mendeteksi siklus dengan bobot negatif dalam graf, jika ada nilai negatif pada diagonal utama dari matriks akhir, maka graf tersebut memiliki siklus negatif.

Algoritma ini juga memungkinkan pencarian jalur terpendek melalui matriks pendahulu yang mencatat titik sebelumnya pada jalur terpendek untuk setiap pasangan titik. Dengan matriks ini, jalur terpendek antara titik manapun dapat direkonstruksi secara efisien.

2.5 Algoritma A*

[2] Algoritma A* adalah metode pencarian yang meminimalkan estimasi total biaya solusi dengan menggabungkan dua fungsi, yaitu $g(n)$ dan $h(n)$ Fungsi $g(n)$ menghitung biaya aktual dari titik awal hingga simpul n, sedangkan $h(n)$ memperkirakan biaya tersisa dari n ke tujuan. Kombinasi ini menghasilkan $f(n) = g(n) + h(n)$, yang memberikan perkiraan total biaya solusi jika rute melalui simpul n. Algoritma ini biasanya dipilih karena dapat mencapai solusi yang optimal dan lengkap, terutama jika fungsi heuristik $h(n)$ memenuhi kriteria tertentu.

Kondisi utama yang diperlukan agar A* memberikan solusi optimal adalah heuristik $h(n)$ yang bersifat *admissible*, yaitu tidak pernah melebih-lebihkan biaya ke tujuan, dan *consistent* atau *monotonic*, di mana nilai h tidak menurun di sepanjang jalur. Dengan adanya heuristik yang memenuhi syarat ini, A* dapat menghindari eksplorasi simpul-simpul yang tidak relevan, mengurangi waktu dan memori yang dibutuhkan.

Terdapat kendala utama dari algoritma A*, yaitu penggunaan memori yang besar karena algoritma ini perlu menyimpan semua simpul yang telah dihasilkan. Meskipun waktu komputasi dapat diatasi dengan baik, kebutuhan memori yang tinggi sering kali menjadi tantangan. Untuk mengatasi hal ini, terdapat varian A* seperti *Iterative-Deepening A** (IDA*) yang mengurangi kebutuhan memori tanpa mengorbankan optimalitas solusi, dengan biaya eksekusi yang sedikit lebih tinggi.

DAFTAR REFERENSI

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., dan Stein, C. (2009) *Introduction to Algorithms*, 3rd edition. The MIT Press, Cambridge, MA.

[2] Russell, S. dan Norvig, P. (2009) *Artificial Intelligence: A Modern Approach*, 3rd edition. Prentice Hall, Upper Saddle River, NJ.

LAMPIRAN A

KODE PROGRAM

Kode A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

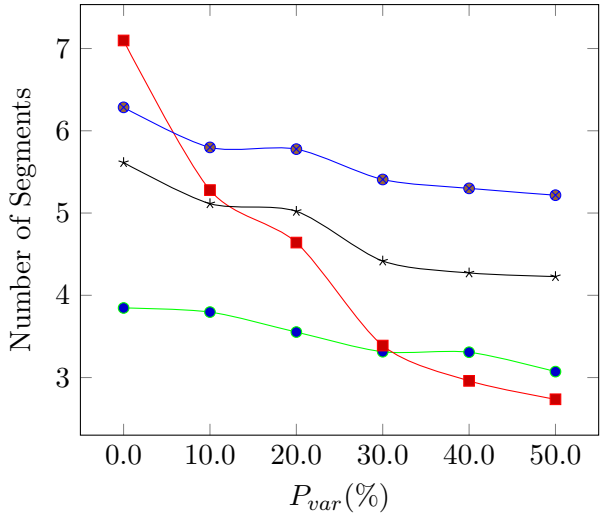
Kode A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

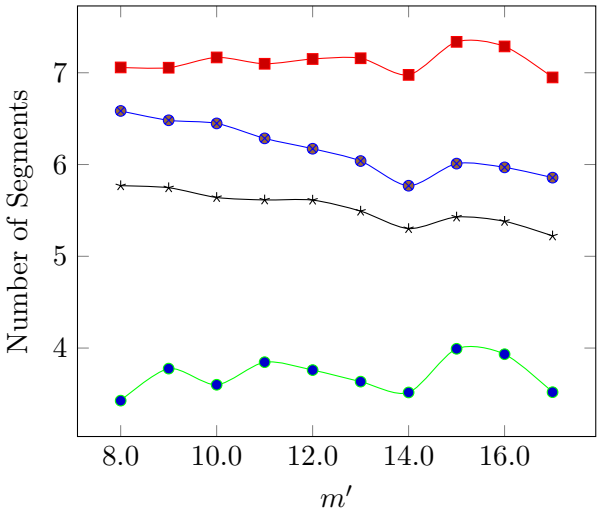
LAMPIRAN B

HASIL EKSPERIMEN

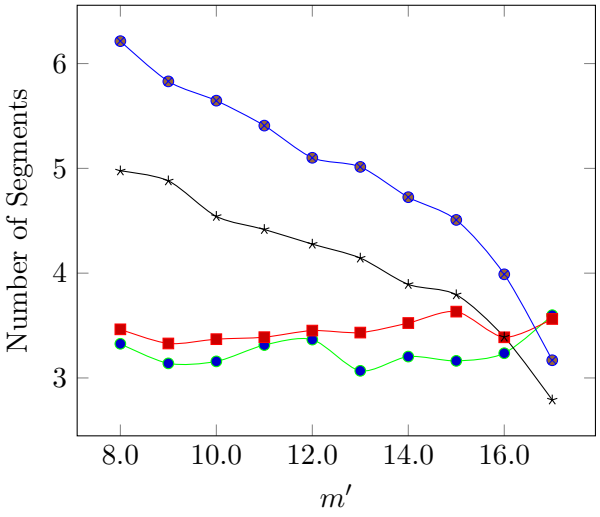
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



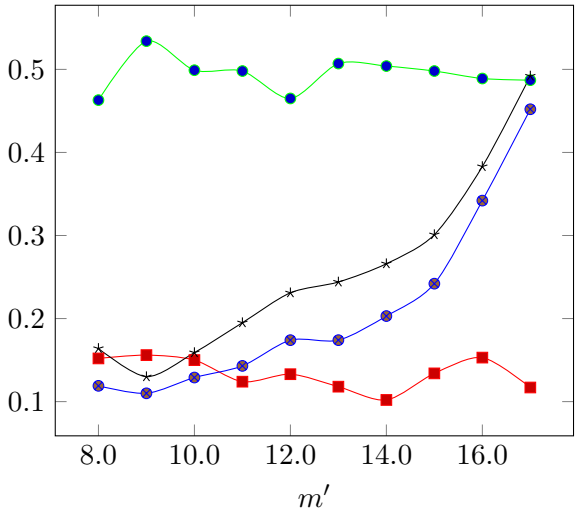
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4