PEMBANGUNAN APLIKASI PENYUSUNAN JADWAL KULIAH MENGGUNAKAN ALGORITMA SEMUT

Antonio Fernandez*, Eko Handoyo, ST, MT**, Maman Somantri, ST, MT**

Jurusan Teknik Elektro, Fakultas Teknik, Universitas Diponegoro
Jl. Prof. Sudharto, Tembalang, Semarang, Indonesia
e-mail: antonio.fernandez.888@gmail.com

Abstrak - Kegiatan perkuliahan umumnya dilakukan dengan tatap muka antara pengajar(dosen) dengan mahasiswa dalam waktu yang terbatas. Untuk mengatasi permasalahan itu, maka diperlukan efektifitas dalam penyusunan jadwal kuliah agar permasalahan kasus bentrok dapat diselesaikan. Kasus yang bisa dilihat sekarang ini adalah tentang bagaimana metode dalam pembuatan sebuah jadwal, metode masih berupa metode tradisional, yaitu penyusunan secara manual. Hal ini dapat menjadi permasalahan besar karena masih dimungkinkan adanya resiko permasalahan bentrok. Jadwal yang bentrok membuat mahasiswa kesulitan untuk memilih mata kuliah yang ingin mereka ambil dan permasalahan lain yang mungkin terjadi, hal ini dapat membuat berkurangnya waktu perkuliahan dalam satu semester. Semua permasalahan ini dapat diselesaikan dengan sebuah sistem yang dapat membuat jadwal secara otomatis. Tugas akhir ini akan membicarakan bagaimana membangun sebuah aplikasi penjadwalan otomatis mulai dari analisa kebutuhan hingga bagaimana sebuah permasalah penjadwalan dapat diselesaikan dengan menggunakan sebuah algoritma yang dinamakan algoritma semut. Algoritma ini adalah salah satu jenis meta-heuristic yang sudah terbukti dapat menyelesaikan banyak sekali permasalahan kombinatorial yang sulit. Algoritma ini meniru tingkah laku semut ketika mereka berada dalam sebuah koloni untuk mencari sebuah sumber makanan. Sistem yang ingin dibangun adalah sebuah aplikasi desktop dengan hasil akhir sebuah jadwal tanpa permasalahan bentrok lagi.

Kata kunci: Masalah penjadwalan, meta-heuristic, algoritma semut

1. PENDAHULUAN

1.1 Latar Belakang

Kegiatan perkuliahan umumnya dilakukan dengan tatap muka antara pengajar (dosen) dengan mahasiswa dalam waktu yang terbatas. Pertemuan ini biasanya dibuat dalam aturan Sistem Kredit Semester (SKS), yang tentunya waktunya juga dibatasi.

Untuk itu dibutuhkan suatu sistem pendukung perkuliahan untuk membuat jadwal secara otomatis sehingga bisa meningkatkan efisiensi kerja dari berbagai pihak. Aplikasi juga diharapkan bisa membantu mahasiswa yang ingin mengambil mata kuliah lebih dari jumlah SKS paket yang ditawarkan, agar tidak mengalami kesulitan untuk pemilihan mata kuliah yang ingin ditempuh pada semester tersebut.

Dalam Tugas Akhir ini dirancang sebuah aplikasi untuk mengatur jadwal kuliah secara otomatis agar menghasilkan keluaran berupa jadwal kuliah yang sudah optimal, jadwal ini masih memungkinkan untuk diubah secara manual sesuai dengan keinginan dari user. Untuk pembuatan aplikasi akan digunakan sebuah algoritma *metaheuristic* yaitu *Ant Colony* atau yang lebih dikenal dengan nama algoritma semut. Dengan menggunakan metode yang ada dalam algoritma semut maka dapat diperoleh hasil *metaheuristic* sesuai dengan yang diinginkan yang tidak mungkin diselesaikan oleh algoritma standard yang umum.

1.2 Tujuan

1. Tujuan Umum
   Tujuan dari Tugas Akhir ini adalah merancang dan membangun sebuah aplikasi yang dapat menyelesaikan masalah penyusunan jadwal kuliah secara otomatis agar diperoleh solusi yang optimal tanpa adanya bentrok jadwal lagi.
2. Tujuan Khusus
   Menganalisa optimalisasi program yang menggunakan algoritma semut ini dimulai dari penggunaan tipe data, waktu kalkulasi, dan efektivitas perulangan yang dilakukan. Selain itu akan dilakukan pula analisa mengenai penggunaan algoritma semut, mulai dari alasan penggunaan algoritma ini, hingga cara penyelesaian hingga optimasi yang dilakukan oleh algoritma semut.

1.3 Batasan Masalah

Agar tidak menyimpang dari pokok pembahasan, pada Tugas Akhir ini Penulis membuat batasan masalah pada hal-hal sebagai berikut.

1. Bahasa pemrograman yang akan digunakan adalah Microsoft Visual C#.
2. Perancangan menerapkan konsep *Object Oriented Programming* (OOP).
3. Pemodelan akan digunakan *Unified Modeling Language* (UML).
4. Menggunakan metode algoritma semut secara umum sebagai pencari solusi optimal.

* Mahasiswa Teknik Elektro Universitas Diponegoro
** Dosen Teknik Elektro Universitas Diponegoro

5. Tidak membahas teori *Object Oriented Programming* dan algoritma semut secara mendalam.

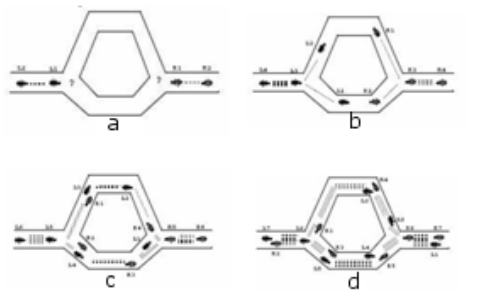
2. DASAR TEORI

2.1 Algoritma Semut

Algoritma ini terinspirasi dari tingkah laku dari semut dalam menemukan jalan dari koloninya ke titik makanan. Dalam dunia nyata, semut berkeliaran secara acak dan dalam proses pencarian makanan, mereka akan kembali ke dalam koloni mereka dengan meninggalkan jejak berupa cairan yang disekresikan lewat tubuhnya agar dapat dideteksi oleh semut yang lainnya. Jika semut lainnya menemukan jejak seperti ini, mereka tidak akan tetap berjalan secara acak lagi namun akan mengikuti jejak tersebut untuk kembali dan memberi tahu koloni bahwa mereka menemukan makanan.

2.2 Cara Kerja Algoritma Semut

1. Pada awalnya, semut berkeliling secara acak.
2. Ketika semut-semut menemukan jalur yang berbeda misalnya sampai pada persimpangan, mereka akan mulai menentukan arah jalan secara acak seperti Gambar 2.1.a.
3. Sebagian semut memilih berjalan ke atas dan sebagian lagi akan memilih berjalan ke bawah seperti Gambar 2.1.b.
4. Ketika menemukan makanan mereka kembali ke koloninya sambil memberikan tanda dengan jejak feromon.
5. Karena jalur yang ditempuh lewat jalur bawah lebih pendek, maka semut yang bawah akan tiba lebih dulu dengan asumsi kecepatan semua semut adalah sama seperti Gambar 2.1.c.
6. Feromon yang ditinggalkan oleh semut di jalur yang lebih pendek aromanya akan lebih kuat dibandingkan feromon di jalur yang lebih panjang seperti Gambar 2.1.d.
7. Semut-semut lain akan lebih tertarik mengikuti jalur bawah karena aroma feromon lebih kuat.



Gambar 2.1 Tingkah laku semut

2.3 Macam Metode Algoritma Semut

1. *Foraging*
   Menekankan pada metode untuk mencari jalur tercepat atau *shortest path* dalam menemukan solusi, metode ini menerapakan tingkah laku semut ketika mencari makan. Contoh: Routing jaringan.
2. *Division of labor*
   Mengimplementasikan tingkah laku semut dalam membagi pekerjaan ke dalam bagian-bagian tertentu dengan hak-hak tertentu. Contoh: Sistem adaptif.
3. *Cemetery organization and brood sorting*
   Menerapkan kerja sama ketika semut membangun makam atau ketika akan bertelur, mereka akan menyusun secara rapi dan teratur serta dilakukan secara bersama agar cepat selesai. Contoh: Pembagian graph.
4. *Cooperative transport*
   Menggabungkan banyak kemampuan dari semut untuk mendapatkan suatu sistem. Contoh: Implementasi robotika.

2.4 *Local search*

Sebuah algoritma *local search* dimulai dari sebuah kondisi kandidat dan secara iteratif bergerak ke solusi tetangganya. Hal ini dimungkinkan apabila relasi dengan tetangganya didefinisikan dalam lingkup pencarian. Sebagai sebuah contoh, permukaan vertex dari tetangga ke permukaan vertex hanya dibedakan oleh sebuah node.

Biasanya, setiap kandidat solusi memiliki lebih dari satu solusi tetangga, pilihan solusi mana yang akan diambil hanya berdasarkan informasi tentang solusi yang diperoleh oleh tetangganya sehingga dinamakan dengan *local search*. Akhir dari proses *local search* dapat dihentikan dengan batas waktu tertentu.

Algoritma *local search* diterapkan secara luas untuk banyak sekali permasalahan komputasi yang sulit, termasuk masalah dalam ilmu komputer (kecerdasan buatan), matematika, penelitian operasi, teknik, dan bioinformatika.

3. PERANCANGAN SISTEM

3.1 Spesifikasi Sistem

Aplikasi penyusunan jadwal otomatis yang dinamakan "*Ant Colony Timetable*" ini hanya akan dibagi ke dalam 2 mode pengguna yaitu mode *user* dan mode *administrator*.

Mode pertama (*user*) hanya akan diberikan kebebasan untuk membuka, menutup, menyimpan dokumen yang berisi konfigurasi untuk penjadwalan. Konfigurasi untuk penjadwalan disini berisi daftar nama kelas, subjek yang ditawarkan, dosen pengajar yang bersangkutan, dan ruangan yang nantinya akan ditempati untuk perkuliahan. *File* konfigurasi ini akan disimpan dengan format *.actx.

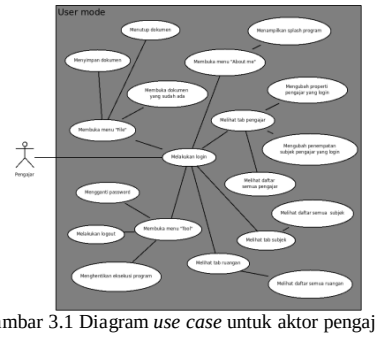
Aplikasi penjadwalan ini hanya dapat digunakan oleh *user* yang sudah terdaftar sebagai pengajar dari *file* konfigurasi yang bersangkutan, jika belum masuk maka *user* biasa tidak akan dapat melakukan *login*. *Default login username* adalah nama depan dari *user* yang sudah dimasukkan ke dalam *file* konfigurasi sedangkan *default password* untuk semua *user* yang sudah terdaftar adalah kosong, jadi *user* bisa melakukan *login* setelah terdaftar.

3.2 Diagram Perancangan

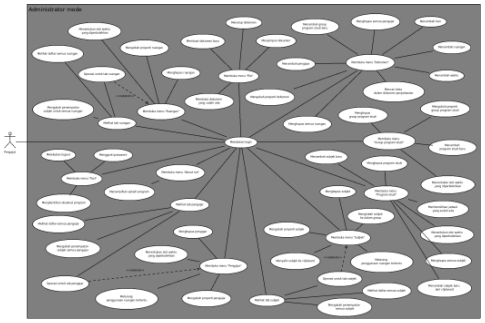
3.2.1 Diagram Fungsional

3.2.1.1 Diagram *Use Case*

Diagram *use case* adalah sebuah diagram yang digunakan untuk menunjukkan tampilah grafis dari fungsionalitas yang diberikan oleh sistem dilihat dari sisi aktor, tujuan aktor, dan hal yang berkaitan dengan *use case* yang ada.



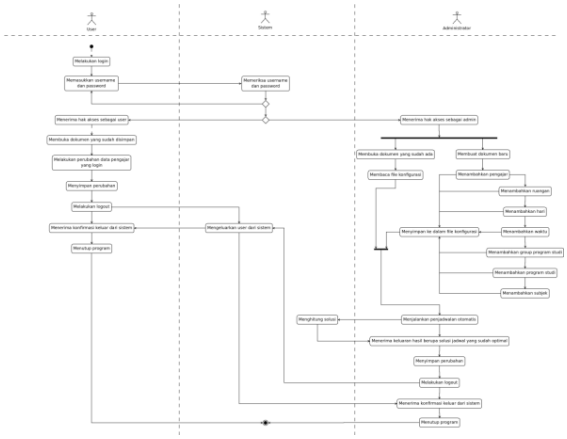
Gambar 3.1 Diagram *use case* untuk aktor pengajar



Gambar 3.2 Diagram *use case* untuk aktor *administrator*

3.2.1.2 Diagram Aktivitas

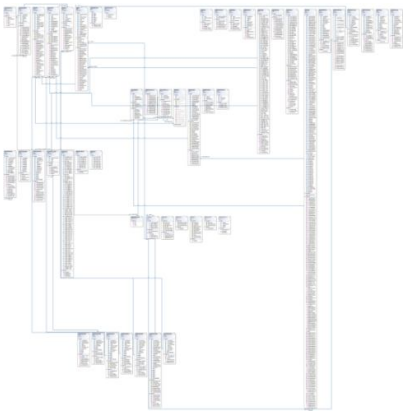
Diagram aktivitas adalah sebuah teknik penjelasan diagram yang bebas menunjukkan aliran aktivitas dan kegiatan langkah demi langkah. Diagram ini biasanya digunakan untuk menjelaskan aliran bisnis dan operasi dari sebuah komponen dari sistem**.**



Gambar 3.3 Diagram aktivitas proses kerja sistem aplikasi penjadwalan

3.2.2 Diagram Struktural

Diagram ini diwakilkan oleh diagram kelas yang merupakan sebuah tampilan statis struktur program. Diagram kelas menggambarkan struktur dengan memperlihatkan kelas dari sistem, baik atribut maupun relasi dengan kelas lain.

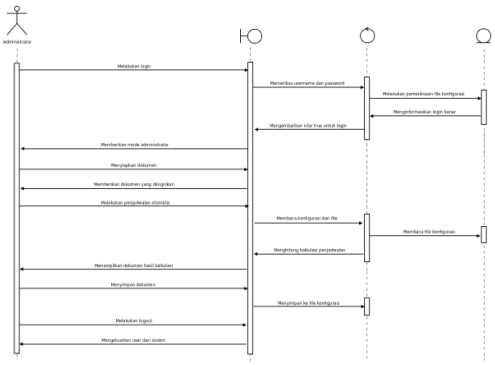


Gambar 3.4 Diagram kelas aplikasi penjadwalan lengkap

3.2.3 Diagram Tingkah Laku

3.2.3.1 Diagram Urutan

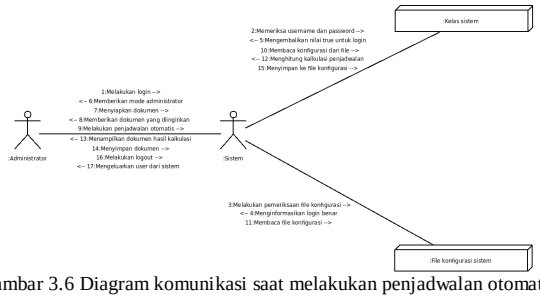
Diagram urutan adalah sebuah diagram interaksi di dalam UML yang menunjukkan proses suatu operasi dengan lainnya dengan fokus ke arah urutan dari proses mana yang mendahului maupun yang tertinggal.



Gambar 3.5 Diagram urutan aplikasi penjadwalan untuk melakukan penjadwalan otomatis

3.2.3.2 Diagram Komunikasi

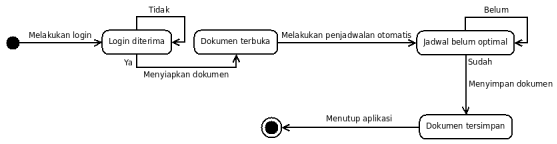
Diagram komunikasi memodelkan interaksi antara objek atau bagian dari kondisi pesan yang urut. Diagram ini merupakan kombinasi dari gabungan informasi diagram kelas, urutan, dan *use case* menjelaskan baik tingkah laku statis maupun dinamis sebuah sistem.



Gambar 3.6 Diagram komunikasi saat melakukan penjadwalan otomatis

3.2.3.3 Diagram *State Machine*

Diagram ini digunakan untuk menunjukkan tingkah laku dari sistem yang terdiri dari beberapa kondisi yang terbatas.



Gambar 3.7 Diagram *state machine* saat menjalankan proses penjadwalan otomatis

3.3 Rancangan Algoritma

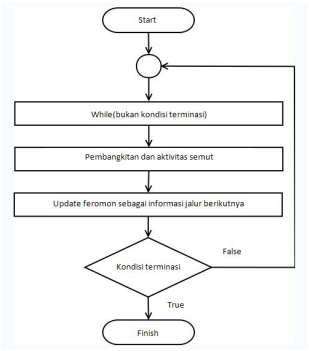
3.3.1 *Pseudocode* Algoritma Semut

```
procedure ACO_Meta_heuristic()
   while (termination_criterion_not_satisfied)
      schedule_activities
         ants_generation_and_activity();
         pheromone_evaporation();
         daemon_actions(); {optional}
      end schedule_activities
   end while
end procedure

procedure ants_generation_and_activity()
   while (available_resources)
      schedule_the_creation_of_a_new_ant();
      new_active_ant();
   end while
end procedure

procedure new_active_ant() {ant lifecycle}
   initialize_ant();
   M = update_ant_memory();
   while (current_state ≠ target_state)
      A = read_local_ant-routing_table();
      P = compute_transition_probabilities(A, M,
problem_constraints);
      next_state = apply_ant_decision_policy(P,
problem_constraints);
      move_to_next_state(next_state);
      if (online_step-by-step_pheromone_update)
         deposit_pheromone_on_the_visited_arc();
         update_ant-routing_table();
      end if
      M = update_internal_state();
   end while
   if (online_delayed_pheromone_update)
      evaluate_solution();
      deposit_pheromone_on_all_visited_arcs();
      update_ant-routing_table();
   end if
   die();
end procedure
```

3.3.2 *Flowchart* Algoritma Semut

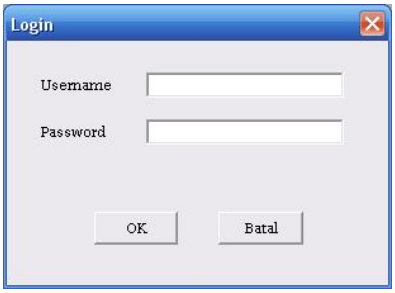


Gambar 3.8 Flowchart algoritma semut

4. IMPLEMENTASI DAN PENGUJIAN

4.1 Pengujian Unit

Pengujian ini dilakukan pada semua unit aplikasi yang berupa menu yang bisa diakses dan dilakukan dengan metode *black box* serta *white box*. Di bawah ini adalah salah satu pengujian unit pada menu login.



Gambar 4.1 Tampilan menu *login*

Gambar 4.2 Pesan selamat datang untuk *user* yang berhasil *login*

Gambar 4.3 Pesan salah untuk *user* yang gagal *login*

Senarai untuk menampilkan menu *login* adalah sebagai berikut:

```
BacaTulisXML readValidUsernamePassword =
new BacaTulisXML("datalogintabel.xml");
readValidUsernamePassword.getExtIDFromUser
name(this._txUsername.Text, out
this.txExtID, out this.txHashPassword);

readValidUsernamePassword.Conversion(this.
_txPassword.Text, out
this.getHashPassword);

if (this.txHashPassword ==
this.getHashPassword)
{
     UserSelectedEventArgs args = new
     UserSelectedEventArgs();

     args.Username = this._txUsername.Text;
     args.HashPassword =
     this.getHashPassword;
     onUserSelected(args);
     this.DialogResult = DialogResult.Yes;
}
else
{
     this._txUsername.Text = "";
     this._txPassword.Text = "";
     MessageBox.Show("Username atau
     Password yang Anda Masukkan Salah");
}
```

4.2 Pengujian Sistem

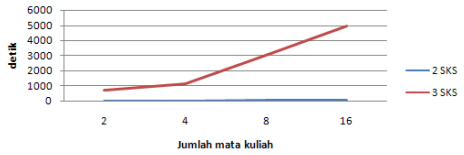
Pengujian ini dilakukan untuk keseluruhan sistem dari aplikasi yang dibangun ditinjau dari berbagai segi sebagai berikut:

1. Kebutuhan
   Dari segi kebutuhan sudah dapat dilihat pemenuhannya untuk menangani kasus penjadwalan. Adapun kebutuhan kasus penjadwalan adalah adanya subjek, pengajar, ruangan, hari, dan waktu.
2. Kegunaan
   Kegunaan dari sistem dari awal adalah untuk mendapatkan solusi penjadwalan optimal secara otomatis.
3. Keamanan
   Untuk faktor keamanan yang ingin dicapai adalah penggunaan aplikasi ini hanya bisa digunakan oleh pihak yang berkaitan dengan penjadwalan dalam suatu universitas melalui pembatasan hak akses *login*.
4. Performa
   Untuk performa sistem sendiri diinginkan penjadwalan otomatis dengan penemuan solusi yang cepat, namun setelah dianalisa lagi kebutuhan dalam penjadwalan prioritas waktu berbanding terbalik dengan pencapain solusi optimal.
5. Dokumentasi
   Untuk sebuah aplikasi yang baik diperlukan adanya sebuah dokumentasi agar mudah dalam penggunaan dalam hal ini akan digunakan *file* berformat *.pdf.

4.3 Pengujian Penerimaan

4.3.1 Pengujian *alpha*

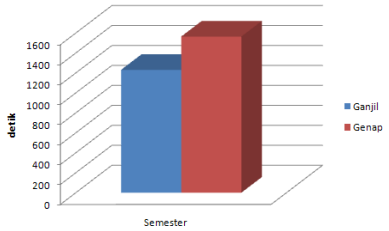
Pengujian ini dilakukan untuk membedakan efektivitas penggunaan dalam kasus penjadwalan 2SKS dan 3SKS. Berikut ini adalah grafik hasil percobaan dengan perbandingan banyaknya SKS dengan waktu.



Gambar 4.4 Grafik perbandingan pengujian terhadap waktu

4.3.2 Pengujian *beta*

Pengujian ini dilakukan untuk mata kuliah semester ganjil dan semester genap yang ada di Teknik Elektro Universitas Diponegoro dalam perbandingan dengan waktu.



Gambar 4.5 Grafik perbandingan penggunaan waktu kedua kasus

4.4 Pengujian Algoritma dan Struktur Data

4.4.1 Pengujian Algoritma

Untuk kasus penjadwalan termasuk ke dalam kasus kompleks *metaheuristic* yang tidak mungkin dapat ditangani oleh algoritma standard biasa. Penjadwalan memerlukan faktor *guessing* dalam penyelesaian masalahnya, dari tebakan itu lalu baru masuk ke algoritma untuk diolah dan diperoleh hasil optimal.

Ant Colony Optimization (ACO) atau sistem algoritma semut ini sudah banyak diterapkan pada banyak sekali kasus kombinatorial, algoritma semut ini juga memiliki keuntungan lebih dibandingkan Genetic Algorithm (GA) maupun Simulated Annealing (SA) dalam pendekatan untuk masalah yang sama yaitu ACO dapat menangani kasus perubahan dalam *graph* secara dinamis.

4.4.2 Pengujian Struktur Data

Algoritma semut seperti yang sudah dibahas di atas memerlukan sebuah kondisi awal ketika pembangkitan solusi hingga proses update feromon. Semua tipe data ini memerlukan range yang standard seperti pada program umumnya. Untuk itulah dipilih tipe data integer (di dalam .Net dikenal sebagai Int32) dengan ukuran 4 bytes dengan range -2,147,483,648 .. 2,147,483,647, untuk tipe angka desimal akan dipilih double sebagai tipe datanya dengan ukuran 8 bytes untuk masalah kepresisian data, -1.79769313486232e308 .. 1.79769313486232e308. Untuk menangani data nama dan karakter sudah ditangani oleh tipe data string yang default sudah diberikan sebagai tipe data *built-in* dalam C#, tipe data string menggunakan Unicode karakter dalam ASCII

Untuk penggunaan looping kebanyakan digunakan while dibandingkan penggunaan for maupun foreach. Karena dari beberapa sumber sudah dibuktiktan bahwa while memiliki proses kalkukasi perulangan yang lebih cepat dibandingkan for maupun foreach.

5. PENUTUP

5.1 Kesimpulan

1. Aplikasi Penjadwalan Mata Kuliah ini bertujuan untuk membangkitkan sebuah jadwal perkuliahan yang sudah terbebas dari masalah bentrok.
2. Aplikasi Penjadwalan Mata Kuliah ini berbasis pada proses acak *local search* dan dipercepat oleh algoritma semut.
3. Fungsi pengisian data hingga menjalankan penjadwalan hanya bisa dilakukan oleh *administrator* dengan keamanan menggunakan *password*.
4. *User* biasa hanya dapat melihat mata kuliah yang terdaftar untuk *user* itu sendiri dan melakukan perubahan hanya untuk porsinya.
5. Untuk mendapatkan solusi yang terbaik diperlukan iterasi dalam jumlah yang cukup banyak yang pastinya akan memakan waktu lebih lama karena mengingat parameter yang harus disusun cukup banyak.

5.2 Saran

1. Diharapkan aplikasi Penjadwalan Mata Kuliah ini nantinya bisa dikembangkan ke dalam aplikasi yang berbasiskan web.

2. Diharapkan metode algoritma semut yang digunakan bisa dikembangkan ke bidang lainnya misalnya untuk penyelesaian masalah *Travelling Salesman Problem* (TSP).
3. Diharapkan untuk pengembangan aplikasi selanjutnya bisa digunakan aplikasi database untuk penanganan kasus *metaheuristic* yang lebih kompleks.

DAFTAR PUSTAKA

[1] Abdullah, Salwani. *Heuristic Approaches for University Timetabling Problems*. Doctoral Dissertation. University of Nottingham, England. 2006.

[2] Blum, Christian. *Metaheuristics for Group Shop Scheduling*. Diplome Thesis. Universite Libre de Bruxelles, Belgium. 2002.

[3] Bonabeau, Eric, Marco Dorigo & Guy Theraulaz. *Swarm Intelligence From Natural to Artificial Systems*. New York: Oxford University Press. 1999.

[4] *C# Language Specification Version 3.0*. Microsoft Corporation. 2007.

[5] Dorigo, Marco, Gianni Di Caro & Luca M. Gambardella. *Ant Algorithms for Discrete Optimization*. Artificial Life Volume 5, Number 2. 1999.

[6] Fernandez, Antonio & Eko Handoyo. *Analyzing Evolutionary Algortihm Method to Optimize Time Table System (Case Study: University Scheduling Time Table)*. ICTS. Surabaya. 2008.

[7] Filev, Andrew, Tony Loton, Kevin McNeish, Ben Schoellmann, John Slater, Chaur G. Wu. *Professional UML with Visual Studio .NET: Unmasking Visio for Enterprise Architects*. Birmingham: Wrox Press Ltd. 2003.

[8] Ghoseiri, Keivan & Fahimeh Morshedsolouk. *ACS-TS: Train Scheduling Using Ant Colony System*. Journal of Applied Mathematics and Decision Sciences Volume 2006, Article ID 95060, Pages 1–28. 2006.

[9] Gunnerson, Eric. *A Programmer's Introduction to C#*. New York: Apress. 2000.

[10] Heinonen, J. & F. Pettersson. *Job-Shop Scheduling and Visibility Studies With A Hybrid ACO Algorithm*. Åbo Akademi University, Finland. 2007.

[11] Marte, Michael. *Models and Algorithms for School Timetabling – A Constraint-Programming Approach*. Doctoral Dissertation. Universitat Munchen, Germany. 2002.

[12] *MSDN Library Visual Studio 2008*. Microsoft Corporation. 2008.

[13] *Nash, Trey. Accelerated C# 2008*. New York: Apress. 2007.

[14] Nedjah, Nadia & Luiza de Macedo Mourelle. *Swarm Intelligent Systems*. Berlin: Springer. 2006.

[15] Pender, Tom. UML Bible. Indianapolis: Wiley Publishing, Inc. 2003.

[16] Petrovic, Sanja & Edmund Burke. *University Timetabling*. University of Nottingham. 2007.

[17] Socha, Krzysztof. *Metaheuristics for the Timetabling Problem*. Diplome Thesis. Universite of Libre de Bruxelles, Belgium. 2003.

[18] Sohier, Emmanuel. *Modelling a Complex Production Scheduling Problem Optimization Techniques*. Master Thesis. Blekinge Institute of Technology, Sweden. 2006.

[19] Willemen, Robertus J. *School Timetable Construction Algorithms and Complexity*. Universiteitsdrukkerij Technische Universiteit Eindhoven, Netherlands. 2002.

[20] Yang, Feng-Cheng & Yu-Hui Hung. *Ant Colony Optimization Method for Time Window Constrained Batching and Scheduling Problem*. APIEMS. Bali. 2008.

[21] --, *http://en.wikipedia.org/wiki/Ant_colony_optimization*. January 2009.

[22] --, *http://en.wikipedia.org/wiki/Local_search_(optimization)*. January 2009.

[23] --, *http://www.scholarpedia.org/article/Ant_colony_optimization. May 2008*.

[24] --, *iridia.ulb.ac.be/~mdorigo/ACO/ACO.html. February 2008*.

.

Biodata Penulis

Antonio Fernandez (L2F 005 515)

Dilahirkan di Tegal tanggal 26 Desember 1987. Saat ini sedang menyelesaikan studi S1 pada Fakultas Teknik Jurusan Teknik Elektro Universitas Diponegoro konsentrasi Informatika Komputer.

Menyetujui,

Pembimbing I	Pembimbing II
Eko Handoyo, ST, MT	Maman Somantri, ST, MT
NIP. 132 309 142	**NIP. 132 231 133**