ALGORITMA HPCMA

REKAYASA ALGORITMA MEMETIKA UNTUK PENGENALAN SIDIK JARI

DR. PRIATI, S.KOM., M.KOM

BPSDM KUMHAM PRESS

ALGORITMA HPCMA

REKAYASA ALGORITMA MEMETIKA UNTUK PENGENALAN SIDIK JARI

ALGORITMA HPCMA

REKAYASA ALGORITMA MEMETIKA UNTUK
PENGENALAN SIDIK JARI

Penulis:

Dr. Priati, S.Kom., M.Kom

Editor: Sopi Ahyar

Desain Sampul dan Tata Letak: Sopi Ahyar

ISBN:

Penerbit: BPSDM KUMHAM Press

Redaksi: Jl. Raya Gandul no. 4 Kecamatan Cinere Kota Depok

Tel +62217540123

Email: humas.bpsdmkumham@gmail.com

Cetakan pertama,

SAMBUTAN

KEPALA BADAN PENGEMBANGAN SUMBER DAYA MANUSIA KEMENTERIAN HUKUM DAN HAM R.I

Assalamualaikum wr.wb.

Salam sejahtera untuk kita semua.
Pembaca yang saya hormati,

Buku ini merupakan bentuk ringkas dari disertasi yang telah ibu Priati pertahankan dalam sidang promosi Doktor Ilmu Komputer Universitas Bina Nusantara Jakarta. Dalam buku ini ibu Priati ini merekayasa algoritma memetika menjadi *high performance computing memetik algoritm (HPCMA)*. HPCMA dapat mengidentifikasi data gambar sidik jari dengan lebih cepat menggunakan perangkat yang minimalis.

Penelitian ini sangat berpotensi untuk diadopsi secara luas pada aplikasi sipil maupun komersial, pada Kementerian Hukum dan HAM sendiri, penelitian ini dapat dimanfaatkan dalam system passport dan kontrol perbatasan.

Semoga dengan lahirnya algoritma baru yaitu algoritma memetika komputasi kinerja tinggi (HPCMA) untuk pengenalan sidik jari yang dibangun oleh ibu Priati dapat memberikan khazanah baru dalam bidang ilmu pengetahuan dan teknologi.

Terima kasih

Wassalamualaikum wr.wb.

Depok, Oktober 2022

Dr. Asep Kurnia

DIREKTUR POLITEKNIK IMIGRASI

Riset adalah salah satu upaya penerapan ilmu pengetahuan sebagai solusi yang selaras dengan kehidupan manusia. Penelitian yang baik hendaknya didokumentasikan dengan baik pula agar dapat dikembangkan dikemudian hari, baik oleh peneliti sendiri maupun oleh peneliti lain.

Penelitian yang telah ibu Priati lakukan yang kemudian disajikan sebagai penelitian Disertasi serta berhasil dipertahankan didepan dewan penguji saat sidang promosi Doktor Ilmu Komputer Universitas Bina Nusantara Jakarta ini selayaknya dapat dibukukan untuk semakin memperkaya khazanah ilmu pengetahuan terutama di bidang ilmu komputer, lebih spesifik lagi pada bidang rekayasa algoritma.

Algoritma yang direkayasa oleh ibu Priati dapat dijadikan pedoman untuk perancangan sistem mutakhir yang dapat mengidentifikasi sidik jari secara cepat dengan sumber daya komputasi sederhana yang banyak terdapat dipasaran. Harapannya, kedepan teknologi ini akan semakin banyak diadopsi baik secara massal baik oleh instansi pemerintahan maupun oleh swasta.

Buku Algoritma HPCMA ini disusun dengan cermat, disadur dari naskah disertasi ibu Priati dengan penuh kehati-hatian. Kami menyadari bahwa masih banyak yang perlu ditingkatkan untuk menjawab tantangan kemajuan teknologi komputasi dimasa mendatang. Oleh karena itu, kami menyambut dengan baik segala masukan dan koreksi yang disampaikan untuk kesempurnaan saduran yang dilakukan.

Depok, Oktober 2022

Direktur Politeknik Imigrasi

Wisnu Widayat, S.H.,M.Si

KATA PENGANTAR

Puji dan syukur kepada Allah SWT, Tuhan semesta alam, berkat rahmat dan karunia-Nya buku ALGORITMA HPCMA dapat terselesaikan dengan baik.

Buku ini adalah bentuk ringkas dari disertasi yang telah penulis pertahankan dalam sidang promosi Doktor Ilmu Komputer Universitas Bina Nusantara Jakarta pada bulan Mei 2022.

Penulis menyadari bahwa dalam penyusunan buku ini masih terdapat banyak kekurangan yang perlu ditambahkan. Oleh karena itu penulis mengharapkan kritik dan saran yang membangun sehingga dimasa mendatang buku ini akan lebih berkembang.

Untuk korespondensi silahkan hubungi penulis via email di tie.assiroj@gmail.com

Penulis

DAFTAR ISI

BAB I
PENDAHULUAN

Identifikasi personal menjadi salah satu masalah yang penting dalam masyarakat terkait dengan kendali akses, kriminalitas dan identifikasi forensik, perbankan maupun sistem komputer. Fitur *biometric* yang dapat digunakan untuk identifikasi diantaranya adalah iris mata, suara, DNA dan sidik jari. Berdasarkan penelitian terkait, sidik jari merupakan fitur *biometric* yang paling banyak digunakan karena keunikan, universal dan stabilitasnya. Sidik jari banyak digunakan sebagai fitur keamanan untuk pengenalan *forensic*, akses gedung, otentikasi ATM atau pembayaran.

Identifikasi sidik jari secara otomatis telah menjadi topik penelitian yang menarik dalam dua dekade terakhir, sidik jari banyak digunakan karena keunikan, ukuran dan ke-khas-annya.

Alat pengenalan sidik jari sangat mudah didapatkan sekarang ini, atas dasar tersebut banyak perusahaan dan lembaga yang menggunakannya, seiring dengan bertambahnya jumlah orang yang harus diidentifikasi secara personal.

Pengenalan sidik jari dapat dikelompokkan dalam dua bentuk masalah yang berbeda yaitu verifikasi dan identifikasi. Verifikasi adalah membandingkan satu sidik jari dengan satu sidik jari yang lain. Sedangkan identifikasi adalah mencocokkan satu sidik jari *input* dengan data sidik jari yang ada didalam *database.* Dengan demikian, identifikasi dapat diartikan sebagai perluasan dari verifikasi yang dilakukan dengan membandingkan satu sidik jari ke banyak sidik jari, dan perlu diketahui bahwa identifikasi pada dasarnya lebih kompleks dari verifikasi.

Sidik jari dianggap sebagai salah satu metode verifikasi pribadi yang dapat diandalkan karena keunikan dan ke-permanen-annya. Karena dua hal tersebut juga, system verifikasi sidik jari otomatis sedang diteliti secara ektensive oleh berbagai komunitas pengenalan pola dan pengenalan gambar *(graph matching)*. *Graph matching* adalah proses menemukan kecocokan dua tepi gambar dengan beberapa batasan untuk memastikan bahwa substruktur serupa dalam satu gambar dipetakan ke substruktur serupa digambar lainnya. Pada aplikasi *graph matching*, operasi yang dilakukan adalah membandingkan dua objek atau

melakukan perbandingan antara objek dan model. Kemajuan penelitian yang signifikan telah didapatkan namun sistem verifikasi sidik jari yang handal masih merupakan masalah yang sulit.

Masalah bertambah seiring dengan banyaknya *dataset* sidik jari, yang mengakibatkan semakin besarnya waktu yang diperlukan untuk proses identifikasi. Namun ada cara untuk dapat mengatasi kompleksitas tersebut yaitu klasifikasi, dengan melakukan ekstraksi fitur terhadap gambar sidik jari kemudian di klasifikasikan sesuai dengan kelas-kelas yang telah dibuat sebelumnya. Kemudian cara mengatasi kompleksitas yang lain adalah dengan peng-indeks-an. Peng-indeks-an yang dilakukan adalah dengan membuat nilai indeks untuk setiap identitas pada *database* sidik jari, dengan demikian sidik jari dengan nilai indeks kecocokan yang lebih tinggi dapat di petakan dengan seksama satu dengan yang lain. Selain kedua cara tersebut, dapat juga dilakukan optimasi algoritma evolusi dengan menambahkan metode pencarian lokal, hal ini dilakukan untuk mengurangi waktu pencarian dengan hanya melakukan pencarian pada daerah yang telah ditentukan. Algoritma

evolusi yang ditambahkan dengan fitur pencarian local sering disebut sebagai Algoritma Memetika.

Algoritma memetika merupakan peningkatan dari algoritma evolusi dengan proses pencarian lokal yang terpisah. Algoritma memetika termasuk algoritma yang sederhana dengan performa yang dapat diandalkan serta dapat memberikan solusi yang akurat untuk mengatasi permasalahan di dunia nyata. Ada tantangan baru seiring dengan berkembangnya *dataset* yang semakin membesar yang diantaranya adalah pada proses peng-klasteran analisis teks, simulasi DNA molekuler, seleksi fitur dan peramalan, serta penanganan optimasi berskala besar seperti simulasi kompleks, data mining, kimia kuantum, analisis spektroskopi, analisis geofisika, penemuan obat, serta studi genom.

Pada mulanya, para peneliti berusaha menggunakan algoritma evolusi sebagai solusi untuk klasifikasi, pengenalan pola dan pengolahan gambar. Kemudian digunakanlah algoritma genetika yang dapat mendeteksi dan menyesuaikan setiap rotasi gambar dan mengembalikannya ke bentuk aslinya atau posisi awal. Dengan algoritma genetika para peneliti dapat mengambil sudut rotasi dengan akurasi yang hampir

100%. Algoritma genetika dapat diterapkan pada skala besar dan data berdimensi tinggi seperti pengenalan wajah, pengenalan sidik jari dan pengenalan iris.

Algoritma memetika telah terbukti sangat kompetitif dalam optimasi skala besar. Dengan menggunakan metode pencarian lokal kinerja algoritma dapat ditingkatkan secara cepat. Kecepatan kinerja ini sangat berguna pada optimasi skala besar. Algoritma Memetika (MA) telah sukses digunakan untuk sistem pengenalan wajah. Kedua algoritma ini digunakan untuk mengurangi dimensi waktu dan meningkatkan pengenalan. Hasil penelitian menunjukkan algoritma memetika mengungguli sistem pengenalan dengan metode konvensional. Hasil penelitian juga menunjukkan Algoritma Memetika lebih baik dari Algoritma Genetika. Untuk mempersingkat waktu proses perhitungan dan meningkatkan kualitas hasil yang diperoleh algoritma memetika dapat diparalelkan. Berbagai algoritma optimasi yang diusulkan biasanya terkendala pada waktu penyelesaian yang lambat atau memori komputer yang terbatas pada saat proses komputasi. Solusi untuk masalah tersebut adalah dengan menggunakan algoritma memetika. Namun masalah lain muncul karena

umumnya terkendala pada prasyarat komputasi yang tinggi sehingga untuk mengatasinya dibutuhkan sistem komputasi berkinerja tinggi atau *High Performance Computing (HPC)*.

Pada sistem *High Performance Computing (HPC)* atau komputasi berkinerja tinggi, untuk mendapatkan waktu proses yang efisien, ada dua cara yang biasa dilakukan yaitu dengan membuat prosesor yang cepat dan dengan menghitungnya secara paralel dengan multi-prosesor. Jika cara pertama dilakukan, perusahaan elektronik pembuat prosesor harus mengurangi bagian-bagian elektronik yang dapat menjadikan produknya lebih kecil dan sinyal yang dihasilkan menjadi lebih pendek. Teknologi semikonduktor saat ini menggunakan teknik litografi yang sudah hampir mencapai batasnya. Teknik litografi adalah teknik mencetak pada permukaan yang licin seperti pada pembuatan *chip*. *Chip* prosesor terbaru dibuat dengan teknologi fabrikasi 45nm dan jika dikurangi lagi kemungkinan kesalahan proses manufaktur menjadi lebih tinggi, selain itu kehandalannya akan berkurang. Oleh karena itu, peluang untuk meningkatkan kecepatan komputasi yang masih memungkinkan adalah dengan cara komputasi paralel.

Sheng dkk menggunakan algoritma memetika untuk mencocokkan sidik jari melalui fitur *minutiae*, dan metodenya cenderung lambat dan tidak cocok untuk sistem *real-time*. Berbeda dengan penelitian Sheng dkk, penelitian ini menggunakan algoritma HPCMA dengan mengkonversi data gambar sidik jari ke bentuk *string array* kemudian dikonversi lagi ke bentuk kode biner. HPCMA digunakan untuk menyempurnakan penelitian Sheng, dkk. Metode konversi dari gambar ke string array kemudian dari string array ke kode biner dilakukan karena pada dasarnya pada lingkungan HPC sangat cocok untuk komputasi numerik dengan sejumlah perhitungan yang kompleks dan dieksekusi dengan cepat.

Buku ini membahas mengenai identifikasi sidik jari dengan sistem *High Performance Computing* (HPC) menggunakan algoritma memetika, algoritma HPCMA dapat diandalkan untuk memproses data sidik jari yang besar dalam waktu yang singkat atau cepat.

BAB II
KOMPUTASI BERKINERJA TINGGI

Komputasi berkinerja tinggi atau *High Performance Computation (HPC)* terkait erat dalam banyak aspek sebagai penerus "superkomputer". Sepanjang sejarah komputasi, istilah "superkomputer" telah diterapkan pada banyak sistem komputer. Di akhir tahun tujuh puluhan dan awal tahun delapan puluhan, ada sedikit keraguan tentang apa itu superkomputer. Superkomputer ditandai oleh eksklusivitas dan harga yang tinggi yang hanya dimiliki oleh beberapa institusi militer dan pemerintah (terutama Amerika Serikat), laboratorium, perusahaan kedirgantaraan dan minyak serta beberapa pusat superkomputer untuk pengguna universitas – bisa membeli komputer super ini, dan beberapa peneliti mempunyai akses kefasilitas *supercomputer*.

Untuk mendefinisikan istilah *supercomputer*, biasanya digunakan istilah yang lebih netral yaitu "komputasi kinerja tinggi". Namun, berbeda orang dapat berbeda pula definisi tentang *HPC* ini. *HPC* adalah bidang penelitian interdisipliner yang berkaitan dengan semua aspek solusi skala besar terhadap masalah ilmiah yang

membutuhkan sumber daya komputasi yang besar. Beberapa pengertian komputasi berkinerja tinggi sebagai berikut:

1) Komputasi berkinerja tinggi adalah sistem komputasi parallel yang terdiri dari banyak prosesor dan mempunyai banyak FPGA (*Field Programmable Gate Array*).
2) Komputasi berkinerja tinggi, atau *supercomputer* adalah bidang komputasi yang mengeksploitasi paralisme secara masif melalui teknologi arsitektur perangkat keras yang canggih untuk memecahkan masalah aplikasi yang besar.
3) Komputasi berkinerja tinggi adalah sistem komputer yang terdiri dari banyak *node* dimana untuk setiap *node* terdiri dari beberapa prosesor dengan kemampuan *multi-thread* yang saling terhubung melalui jaringan berkecepatan tinggi.
4) Komputasi kinerja tinggi adalah alat yang mendukung kegiatan sains modern seperti melakukan perhitungan yang rumit dengan waktu yang ringkas atau cepat dengan menggunakan struktur komputasi besar yang memadai.

Menurut penulis komputasi berkinerja tinggi adalah sistem komputer yang terdiri dari banyak prosesor dengan kemampuan *multi-thread* yang saling terhubung untuk memecahkan aplikasi yang besar. Dengan memanfaatan fitur *multi-thread* pada prosesor, pemrosesan data menjadi lebih cepat jika dibandingkan dengan komputasi pada umumnya dan penelitian ini menggunakan prosesor dengan fitur *multi-thread*.

Topik dalam *HPC* meliputi: aplikasi intensif secara numerik, algoritma, matematika numerik, analisis dan optimasi kinerja, model dan teknik pemrograman, kompiler dan alat (preprosesor, vektorisasi otomatis dan paralelisasi), arsitektur vektor dan paralel serta visualisasi. Bahkan *HPC* telah berhasil digunakan dibanyak masalah pengenalan pola.

Beberapa teknik komputasi yang dikenal saat ini adalah *Computer Clusters*, *Supercomputer*, *Grid Computing* dan *Cloud Computing*. Selanjutnya teknik-teknik komputasi tersebut disederhanakan dengan istilah teknologi *system* komputasi berkinerja tinggi. Untuk mewujudkan komputasi paralel, diperlukan dukungan perangkat keras yang menyediakan banyak prosesor, dan juga sistem operasi untuk membagi beban

komputasi ke seluruh prosesor tersebut. Sistem ini ternyata tidak mudah, sehingga pada awalnya, komputasi paralel hanya bisa dinikmati oleh sistem yang mahal dan besar seperti komputer super. Untunglah dengan berkembangnya zaman, komputasi paralel mulai bisa ditemukan pada komputer biasa.

Komputasi berkinerja tinggi menjadi sangat diperlukan oleh kalangan perusahaan, peneliti ilmiah dan lembaga pemerintah untuk menghasilkan inovasi terhadap layanan dan produk. Salah satu tujuan dari komputasi paralel adalah kecepatan atau *speed up*. Keuntungan yang didapatkan dengan menggunakan HPC diantaranya adalah kecepatan proses, harga yang murah, serta pengembangan yang fleksibel. Dalam bidang penelitian ilmiah yang menggunakan simulasi seperti cuaca, dinamika fluida, astronomi, HPC berperan penting dalam membantu penyeselaian model simulasi dengan lebih cepat. Dunia industry menjadi semakin cepat dalam mengembangkan produknya. Pada bidang analisis data HPC dapat digunakan untuk menangani pengolahan data yang besar dengan komputasi yang kompleks, sedangkan pada bidang kecerdasan buatan HPC berperan

mempercepat proses training model pada dataset yang besar.

Kontribusi *HPC* sangat signifikan terhadap kemajuan ilmiah, peningkatan daya saing sektor industri dan keamanan nasional. Oleh karena itu, banyak negara mengembangkan *system* komputasi kinerja tinggi untuk menghasilkan system yang paling canggih yang dapat mereka terapkan secara luas pada lembaga akademik, *industry* dan pemerintahan agar secara mandiri mereka dapat mengatasi tantangan seperti kesehatan, keselamatan publik, prakiraan cuaca, perubahan iklim, dan perlindungan lingkungan. Mereka meyakini *system computer* yang canggih dapat meningkatkan daya saing dan memberikan keunggulan komparatif bagi negara mereka karena dengan menggunakan *system computer* yang canggih mereka dapat menjadi yang paling cepat dan terdepan dalam riset dan inovasi.

Komputasi kinerja tinggi menggunakan *supercomputer* dengan teknik paralelisasi sudah dilakukan secara masif. Teknik ini digunakan untuk mengatasi tantangan komputasi yang kompleks dengan cara pemodelan, simulasi dan analisis data secara cepat dan efektif.

Kecepatan (S) waktu proses algoritma adalah perbandingan antara waktu proses secara sekuensial (*Tseq*) dengan waktu proses secara parallel (*Tpar*), seperti dirumuskan berikut ini.

$$S = \frac{Tseq}{Tpar}$$

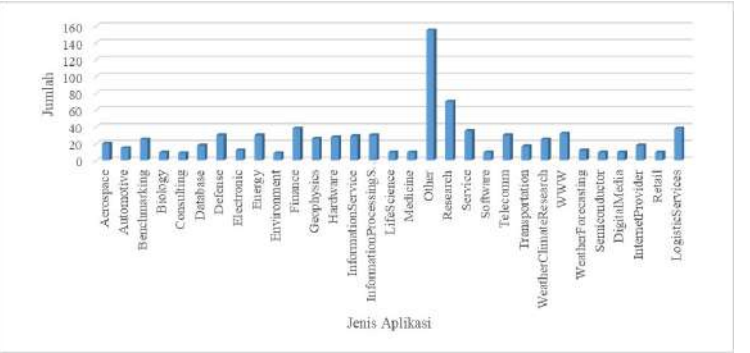
Sistem HPC pada dasarnya merepresentasikan jaringan CPU yang masing-masing berisi banyak inti (*multi-core*) dan memori untuk menjalankan berbagai perangkat lunak aplikasi.

Supercomputer dapat digambarkan seperti ribuan komputer desktop yang bekerja secara bersama-sama beriringan untuk memproses dan menyelesaikan miliaran atau bahkan triliunan bit data setiap detik. Beberapa *supercomputer* melakukan beragam tugas seperti pemodelan dan simulasi data analitik, juga tugas-tugas yang lebih spesifik seperti untuk layanan komputasi awan seperti video dan *music streaming*.

Penggunaan *HPC* hampir menyentuh segala aspek kehidupan sehari-hari seperti *energy*, transportasi, komunikasi, kedokteran, infrastruktur, keuangan, manajemen bisnis dan lain sebagainya. *HPC* sangat cocok digunakan untuk komputasi numerik yang membutuhkan sejumlah perhitungan yang kompleks

agar dapat dieksekusi dengan cepat termasuk diantaranya dibidang fisika, geografi, keamanan nasional, biologi, teknik, pemodelan iklim, ruang angkasa dan *energy*. Juga untuk memodelkan *system* yang kompleks seperti pola cuaca, dinamika sel, pergerakan udara maupun pesawat ruang angkasa karena *HPC* dapat membantu mengungkap interaksi dan proses yang mengatur setiap komponen. *HPC* menyentuh *detail* setiap segi pengembangan produk otomotif dan luar angkasa, eksplorasi minyak dan gas, penemuan obat-obatan, prediksi cuaca dan pemodelan iklim, pemodelan keuangan yang kompleks, desain dan optimisasi produk, animasi *3-D*, dan analisis bisnis.

HPC juga berkontribusi dalam peningkatan keamanan nasional seperti kriptografi, pemrosesan sinyal serta desain dan pengujian senjata, terutama senjata nuklir. Gambar 2.1 berikut adalah gambaran area pengembangan *supercomputer* di dunia dari tahun 1993-2015. Sumbu *x* pada gambar menunjukkan area aplikasi yang dikembangkan sedangkan sumbu *y* menunjukkan jumlahnya.



Gambar 2.1 Area Aplikasi HPC

BAB III
BIOMETRIK

Biometrik adalah karakteristik khusus pada manusia yang merupakan ciri unik setiap individu dan dapat dijadikan acuan identifikasi dan verifikasi. Ciri biometrik dapat dikategorikan dalam dua jenis yaitu *physical* (contohnya sidik jari, iris mata, retina mata, dan muka) dan ciri tingkahlaku (contohnya suara, tanda tangan, cara berjalan, geometri tangan, tulisan tangan, *electro cardio graph* (ECG)).

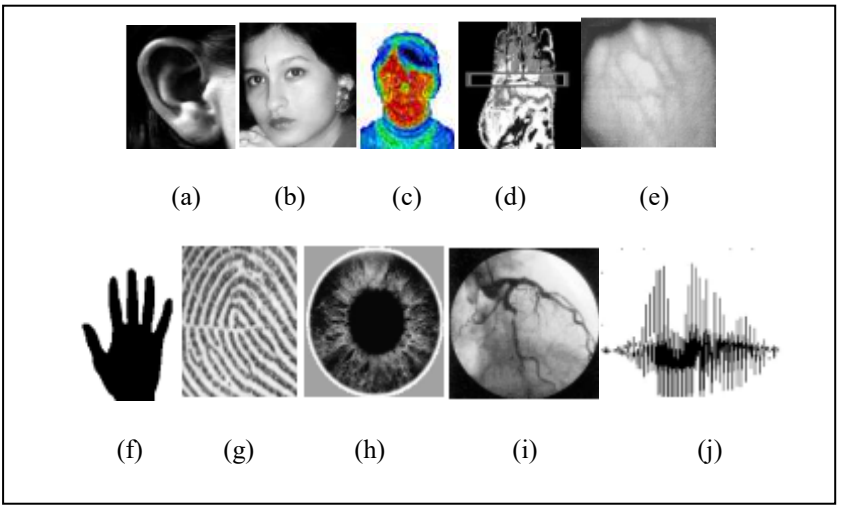
Sidik jari merupakan salah satu fitur biometrik yang paling handal untuk digunakan karena keunikannya, disamping itu juga karena tidak mungkin terjadi kesamaan sidik jari antara dua orang yang berbeda. Fitur biometrik ini dapat digunakan untuk mengenali seseorang secara otomatis dengan melakukan pengenalan pola dan menentukan keaslian karakteristik fisiologisnya.

Sistem biometrik dikelompokan dalam dua bentuk yang berbeda yaitu verifikasi dan identifikasi. Verifikasi adalah membandingkan karakteristik biometrik inputan yang ditangkap oleh alat dengan data yang sudah disimpan dalam *system*. Artinya melakukan

perbandingan satu ke satu. Sistem verifikasi dapat menolak atau menerima data inputan. Sedangkan *system* identifikasi dapat mengenali seseorang dengan cara mencari kecocokan dari seluruh data yang tersimpan, melakukan perbandingan satu ke banyak. Dalam sistem identifikasi, sistem menetapkan identitas subjek (atau gagal jika subjek tidak terdaftar dalam *database system*) tanpa subjek harus mengklaim identitas. Fitur fisiologi manusia dapat digunakan sebagai cara identifikasi *biometric* jika memenuhi persyaratan berikut:

1) Universalitas, artinya setiap orang harus memiliki fitur biometrik.
2) Khas, artinya setiap orang mempunyai ciri fitur biometric yang berbeda.
3) Permanen, artinya tetap atau *invariant*. Keadaan setiap ciri fitur biometric harus tetap selama periode waktu tertentu.

Beberapa contoh fitur biometrik yang dapat diidentifikasi terlihat pada Gambar 3.1 berikut:



Gambar 3.1 Fitur Biometrik (Maltoni et al., 2009)

a) Telinga, b) Wajah, c) Termo wajah, d) Termo tangan, e) Vena tangan, f) Geometri tangan, g) Sidik jari, h) Iris, i) Retina, j) Suara

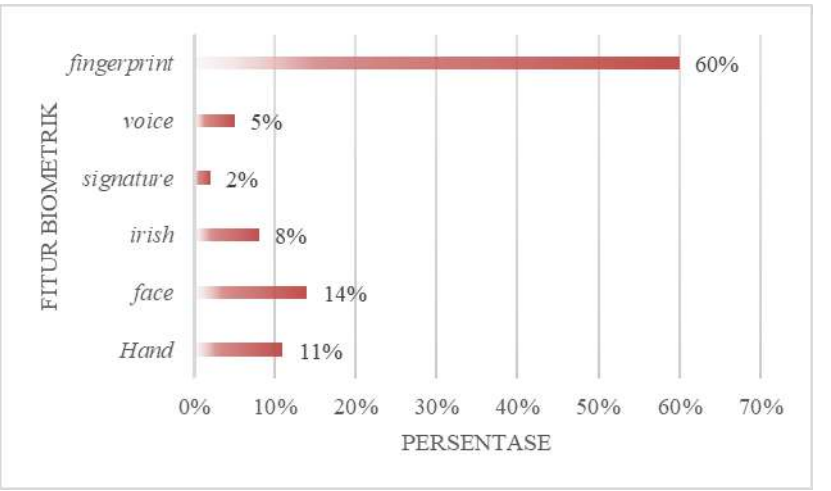
Beberapa hal yang harus dipertimbangkan dalam sistem identifikasi biometrik adalah:

1) Kinerja. Mengacu pada akurasi, kecepatan, ketahanan, maupun faktor lingkungan yang mempengaruhi kecepatan dan ketepatan proses identifikasi.

2) Penerimaan atau akseptabilitas. Merujuk pada sejauh mana orang bersedia menerima pengenal *biometric* tertentu dalam kehidupan sehari-hari.

Sistem pengenalan biometrik harus memiliki akurasi dan kecepatan yang baik dengan menggunakan sumber daya yang wajar, tidak berbahaya, diterima oleh masyarakat dan cukup handal untuk mencegah tindak kejahatan.

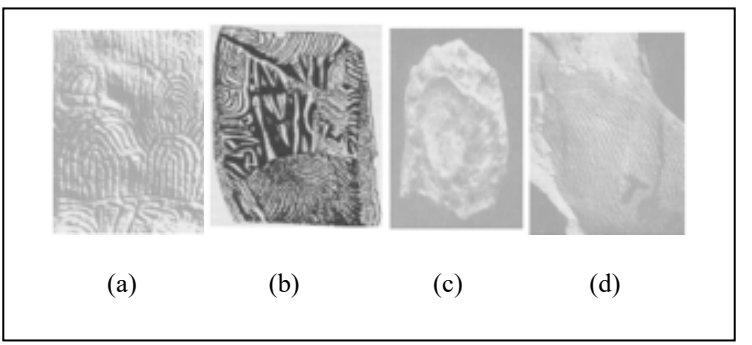
Pengenalan fitur biometrik sidik jari adalah salah satu yang paling handal. Penggunaannya sudah meluas sampai pada perangkat bergerak seperti telepon pintar saat ini. Pada tahun 2012, pendapatan industri yang berkaitan dengan penggunaan fitur *biometric* ini mencapai 60% dari total semua fitur biometrik. Pengenalan wajah ada diurutan kedua dengan 14% seperti terlihat pada Gambar 2.3. Sumbu *x* pada gambar adalah persentase penggunaan fitur *biometric* sedangkana sumbu y adalah jenis *biometric* yang digunakan.



Gambar 3.2 Pendapatan Produk dengan Fitur Biometrik

Sidik Jari adalah pola punggungan dan lembah grafis diujung jari manusia. Sidik jari manusia banyak ditemukan pada berbagai benda bersejarah seperti terlihat pada Gambar 3.3. Temuan ini membuktikan bahwa manusia pada zaman dahulu sudah menyadari dan memberikan perhatian yang khusus terhadap individualitas sidik jari meskipun mereka belum memiliki dasar ilmiah. Sejarah tentang sidik jari dimulai pada tahun 1684, seorang ahli morfologi dari Inggris, Nehemiah Grew, menerbitkan tulisan ilmiah tentang penelitiannya seputar punggungan, alur dan struktur pori pada sidik jari. Tahun 1788, Mayer melakukan

deskripsi terperinci tentang bentuk sidik jari. Thomas Bewik pada tahun 1809 mulai menggunakan sidik jari sebagai ciri khasnya, hal ini diyakini sebagai tonggak dalam studi ilmiah pengenalan sidik jari.



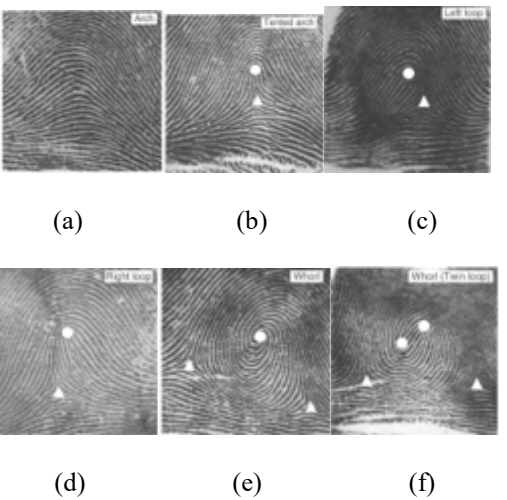
Gambar 3.3 Sidik Jari pada Benda Bersejarah

Pada tahun 1823, Purkinje membuat klasifikasi sidik jari pertama. Dia mengklasifikasi sidik jari ke dalam sembilan kategori berdasarkan konfigurasi punggungannya dan pada tahun 1880 Henry Fauld dan Herschel mempraktekkan pengenalan sidik jari secara ilmiah. Penemuan ini merupakan dasar dari pengenalan sidik jari modern. Pada akhir abad ke sembilan belas, Sir Francis Galton meneliti lebih lanjut tentang sidik jari. Galton memperkenalkan fitur-fitur kecil untuk mencocokkan sidik jari pada tahun 1888.

Keterangan Gambar 3.3 adalah sebagai berikut:

a) Sidik jari pada Pahatan di Zaman Neolitik.
b) Sidik jari pada Batu berdiri pulau goat.
c) Sidik jari pada Penutup dari tanah liat di Cina 300 SM.
d) Sidik jari pada lampu orang Palestina tahun 400 M.

Tingkat kemajuan terpenting dalam pengenalan sidik jari terjadi pada tahun 1899, dimana Edward Henry membuat "*System Henry*" yang terkenal dengan 6 klasifikasi sidik jarinya seperti terlihat pada Gambar 3.4. Sehingga pada awal abad ke dua puluh, bentuk sidik jari sudah dapat dipahami dengan baik.



a) *arch*, b) *tented,* c) *loop*, d) *left loop*, e) *right loop*, f) *whorl*

Gambar 3.4 Klasifikasi Sidik Jari

Menurut *system Henry* keenam tipe sidik jari tersebut adalah *(a) arch* atau tipe sidik jari dengan garis melengkung, *(b) tented arch* atau sidik jari dengan garis melengkung seperti tenda, *(c) loop* atau sidik jari dengan garis melingkar, *(d) left loop* atau sidik jari dengan garis melingkar ke kiri, *(e) right loop* atau sidik jari dengan

garis melingkar ke kanan*,* dan *(f) whorl* atau sidik jari dengan garis memutar.

Dasar-dasar sidik jari secara biologis dapat dijelaskan dibawah ini:

1) Punggungan dan alurnya memiliki karakteristik yang berbeda untuk setiap sidik jari.
2) Jenis konfigurasinya adalah individual akan tetapi bervariasi dan dapat diklasifikasikan secara sistematis.
3) Konfigurasi dan detil *minutiae* setiap punggungan dan alur sidik jari bersifat permanen dan invariant.

Beberapa kategori pencocokan sidik jari adalah:

1) Pencocokan berdasarkan korelasi; dua buah gambar sidik jari ditumpuk kemudian korelasi antar piksel dihitung sesuai dengan tingkat kesamaannya.
2) Pencocokan berdasarkan *minutiae*; *minutiae* diekstrak dari dua sidik jari lalu disimpan sebagai set titik dalam bidang dua dimensi. Pencocokan kategori ini sebenarnya adalah menemukan keselarasan antara data *minutiae* yang ada dengan data minutiae yang diinputkan.
3) Pencocokan berdasarkan pada fitur *ridge* (punggungan); ekstraksi *minutiae* sulit dilakukan

pada gambar sidik jari dengan resolusi rendah, sedangkan fitur lain pada sidik jari, yaitu *ridge*, dapat diekstraksi dengan lebih handal meskipun ke-khas-an nya lebih rendah. Pendekatan pencocokan ini adalah membandingkan sidik jari pada fitur yang di ekstraksi dari pola *ridge*.

Pengenalan sidik jari merupakan teknologi yang berkembang pesat dan banyak digunakan dalam dunia forensik. Teknologi ini sangat berpotensi untuk diadopsi secara luas pada aplikasi sipil maupun komersial, seperti terlihat pada Tabel 3.1.

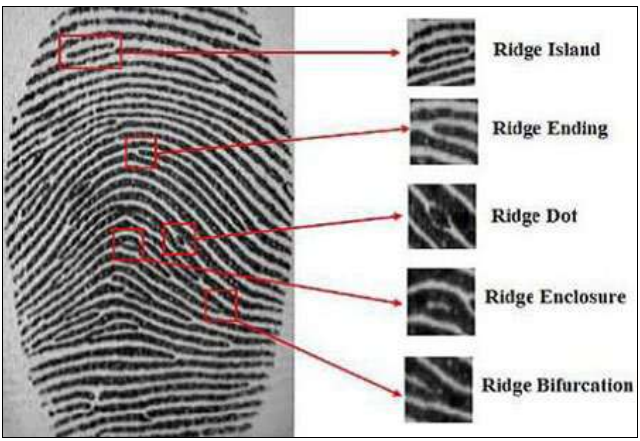
Tabel 3.1 Teknologi Pengenalan Sidik Jari

Forensik	Pemerintahan	Komersial
Identifikasi Jenazah Investigasi Kriminal Identifikasi Teroris	Kartu Tanda Penduduk Surat Izin Mengemudi Passport Kontrol Perbatasan	Login Jaringan Komputer Keamanan Data Elektronik E-Commerce Akses Internet ATM atau Kartu Kredit Akses Kontrol Telepon seluler

Saat ini, sidik jari merupakan fitur biometrik yang paling banyak digunakan dalam system verifikasi

otomatis. Sidik jari dianggap sebagai salah satu metode verifikasi pribadi yang dapat diandalkan karena keunikan dan ke-permanen-annya. Karena dua hal tersebut juga, system verifikasi sidik jari otomatis sedang diteliti secara ektensive oleh berbagai komunitas pengenalan pola dan pengenalan gambar. Kemajuan penelitian yang signifikan telah didapat namun sistem verifikasi yang handal masih merupakan masalah yang sulit.

Keunikan sidik jari telah diakui dan ditentukan oleh pola keseluruhan punggung dan lembah serta diskontinuitas punggung lokal yang disebut *"minutiae".* Secara luas dipercaya bahwa *minutiae* adalah yang paling diskriminatif dan fitur yang dapat diandalkan dalam sidik jari. Untuk alasan ini, *minutiae* adalah fitur yang paling penting dan umum digunakan dalam sistem verifikasi sidik jari otomatis. Gambar 3.5 dibawah ini adalah contoh *minutiae* secara umum.



Gambar 3.5 *Minutiae* (www.bayometric.com)

Ridge island adalah garis kecil yang menempati ruang tengah diantara dua punggungan bukit yang berbeda. Sedangkan *Ridge ending* adalah titik dimana punggungan berakhir. *Ridge dot* adalah istilah untuk punggungan-punggungan bukit yang sangat kecil. *Ridge enclosure* adalah ruang kosong yang terletak diantara dua punggung bukit yang berbeda, dan *Ridge bifurcation* adalah titik dimana sebuah punggungan mulai bercabang menjadi dua atau lebih punggungan (www.bayometric.com).

Kedua jenis *minutiae* penting yang digunakan dalam verifikasi sidik jari otomatis adalah ujung bubungan dan bifurkasi bubungan. Sebuah *minutiae* yang

terdeteksi dalam gambar sidik jari dapat ditandai dengan daftar atribut yang mencakup posisi *minutiae*, arah *minutiae*, dan jenis *minutiae* (akhir atau bifurkasi). Dengan demikian representasi pola sidik jari yang terdiri dari atribut dari semua *minutiae* yang terdeteksi dalam set *minutiae* akan menentukan apakah mereka memang mewakili jari yang sama atau tidak.

Kesulitan dapat dikaitkan dengan faktor-faktor yang biasa ditemui, diantaranya:

1) Pertama, tidak ada korespondensi minutiae antara dua set *minutiae* yang diketahui sebelumnya.
2) Kedua, set *minutiae* kemungkinan salah, disebabkan oleh kualitas sidik jari yang buruk, kualitas gambar yang tidak memadai dan ketidaksempurnaan dalam ekstraksi *minutiae*.
3) Ketiga, dua sidik jari dapat diterjemahkan, diputar, dan di-skala-kan untuk saling mencocokkan.
4) Keempat, saat jari ditekankan secara tidak merata pada sensor pembaca, maka akan terjadi deformasi nonlinear lokal karena elastisitas kulit.
5) Kelima adalah tumpang tindih antara dua sidik jari sehinga beberapa hal kecil tidak terlihat pada keduanya.

Masalah identifikasi dapat dilihat sebagai verifikasi yang dilakukan sekali setiap sidik jari dalam database. Oleh karena itu perbedaan utama dalam masalah ini adalah masalah urutan kompleksitas. Tujuan dalam masalah verifikasi adalah untuk mendapatkan hasil yang sangat tepat dan mengurangi tingkat kesalahan sebanyak mungkin. Namun, metode verifikasi yang kompleks tidak berguna untuk identifikasi karena waktu respon keseluruhan akan berlebihan.

Syarat yang dibutuhkan oleh sistem informasi sidik jari otomatis untuk menangani *database* yang besar yaitu:

1) Presisi: tingkat kesalahan harus serendah mungkin untuk mendapatkan sistem yang akurat.
2) Efisiensi: waktu yang diperlukan untuk menemukan sidik jari di *database* harus sekecil mungkin. Dalam sistem real time, misalnya, penundaan yang tinggi dapat setara dengan kegagalan system.
3) Skalabilitas: merupakan kemampuan sistem untuk menangani database dalam jumlah waktu yang wajar, menjaga persyaratan presisi.
4) Fleksibilitas: sistem harus sesuai, mudah dan efisien dengan ukuran *database*, semua fitur *database*

(seperti sidik jari yang berjejal atau *rollings*), serta konfigurasi perangkat keras apapun (berbeda arsitektur, ukuran cluster yang bervariasi, prosesor yang berbeda).

Meskipun ada beberapa solusi untuk masalah identifikasi sidik jari, struktur proses pencarian secara umum adalah masukkan pengambilan sidik jari, jalankan ekstraksi fitur, lakukan pencarian sidik jari yang serupa dalam *database*, dan yang terakhir adalah mengembalikan hasilnya.

Identifikasi sidik jari adalah metode biometrik yang paling sering digunakan untuk mengenali identitas seseorang. Sidik jari adalah sifat biometrik yang paling banyak dipelajari, dan berbagai algoritma telah diusulkan sejak tahun 1975, seperti pemrosesan, klasifikasi, dan pencocokan. Dua cara dilakukan untuk mengurangi waktu proses dengan tidak membandingkan semua pasang sidik jari menggunakan metode klasifikasi atau dengan menggunakan algoritma peng-indeks-an serta dengan mempercepat proses pencocokan menggunakan komputasi paralel. Ada beberapa algoritma pengindeksan untuk mempercepat proses

pencarian, seperti pemilihan indeks daftar kelompok. Metode ini telah terbukti lebih baik dari platform lain.

BAB IV
ALGORITMA EVOLUSI, ALGORITMA GENETIKA, ALGORITMA MEMETIKA

1. Algoritma Evolusi (AE)

Ide mengenai algoritma evolusioner diperkenalkan pertama kalinya oleh Rechenberg pada tahun 1973.

Bentuk-bentuk algoritma evolusi menggabungkan keunggulan dari teknik heuristic yang efisien dengan pendekatan pencarian berdasarkan populasi. Salah satu bentuk hibridisasi adalah digunakannya pada algoritma pencarian lokal evolusioner.

2. Algoritma Genetika (AG)

Algoritma genetika pertama kali dikembangkan oleh John Holland dari Universitas Michigan tahun 1975 yang didasari oleh proses evoluasi biologi. Dalam ilmu Biologi, sekumpulan individu yang sama, hidup dan berkembang bersama dalam suatu area, disebut dengan populasi. Konsep yang penting dalam AG adalah hereditas dan seleksi alam.

Hereditas merupakan sebuah ide bahwa sifat-sifat itu dapat diturunkan kepada generasi berikutnya. Sifat-sifat individu tersebut dikodekan kedalam sekumpulan gen-gen yang disebut kromosom. Perbedaan susunan

gen-gen dalam suatu kromosom ini akan membawa sifat yang berbeda-beda dan unit pada individu. Untuk itu dapat dikatakan bahwa sebuah kromosom melambangkan satu individu. Sedangkan ide dari seleksi alam adalah individu-individu yang tidak bisa beradaptasi akan mati, sedangkan individu-individu yang mampu beradaptasi akan hidup dan menghasilkan keturunan. John Holland mengatakan bahwa setiap masalah yang berbentuk adaptasi dapat diformulasikan dalam terminologi genetika. Algoritma genetika adalah simulasi dari proses evolusi Darwin dan operasi genetika atas kromosom. Dalam proses seleksi alam individu-individu yang dapat bertahan hidup dalam satu populasi adalah individu yang mampu beradaptasi dengan baik dalam lingkungannya. Selanjutnya individu-individu ini akan mampu menghasilkan keturunan. Setiap individu mempunyai sifat-sifat unik yang akan diwariskan kepada keturunanya. Sifat-sifat ini di kodekan kedalam barisan gen-gen yang disebut kromosom, karena itu suatu kromosom dapat dikatakan mewakili suatu individu. Dalam prosesnya, algoritma genetika bermula dari sejumlah kandidat solusi dari suatu masalah, yang kemudian secara *iterative* kualitas dari kandidat-

kandidat solusi ini terus diperbaiki. Dalam algoritma genetika, kandidat solusi direpresentasikan dalam suatu individu dimana setiap individu diwakili oleh sebuah kromosom yang merupakan barisan dari gen. Ada beberapa tahapan yang diperlukan di dalam algoritma genetika yaitu: Inisialisasi, Evaluasi, Seleksi, Rekombinasi, Mutasi. Algoritma genetika dikembangkan lebih lanjut oleh John Holland bersama murid dan rekan kerjanya dari Universitas Michigan pada tahun 1960-an dan 1970- an. Sejak itu algoritma genetika telah dipelajari, diteliti dan diaplikasikan secara luas diberbagai bidang.

Algoritma genetika telah berhasil diterapkan pada berbagai domain pencarian, optimasi dan kecerdasan buatan. Pada algoritma genetika (AG) kandidat solusi (selanjutnya disebut 'solusi') dari suatu masalah direpresentasikan sebagai kromosom. AG diawali dengan pembentukan kumpulan kromosom yang disebut dengan populasi. Masing-masing kromosom pada populasi akan dievaluasi menggunakan fungsi fitness, yaitu fungsi yang mengukur secara kuantitatif kemampuan suatu kromosom untuk bertahan dalam populasi. Kemudian secara iteratif akan dibentuk populasi baru yang lebih

baik dari populasi sebelumnya dengan menerapkan operator-operator genetika hingga mencapai kriteria berhenti. Operator dasar pada AG adalah seleksi, *crossover*, dan mutasi. Seleksi yaitu operator untuk memilih 2 kromosom sebagai orang tua. *Crossover* adalah operator untuk mengawinkan 2 kromosom orang tua, sehingga menghasilkan keturunan atau anak untuk generasi berikutnya. Dan mutasi adalah operator yang mengubah urutan gen pada kromosom.

Kelebihan dari AG adalah pada cara kerjanya yang paralel. AG bekerja dalam ruang pencarian yang menggunakan banyak individu sekaligus, sehingga kemungkinan AG untuk terjebak pada ekstrim lokal lebih kecil dibandingkan metode lain. AG juga mudah diimplementasikan. Karena jika suatu algoritma AG sudah dapat diimplementasikan, kita juga dapat menyelesaikan masalah lain hanya dengan membuat kromosom dan fungsi *fitness* yang baru, sehingga AG bisa dijalankan.

Kekurangan dari AG adalah dalam hal waktu komputasi karena harus melakukan evaluasi fitness pada semua solusi disetiap iterasinya. Sehingga AG bisa lebih lambat dibandingkan dengan metode lain. Namun karena

kita dapat mengakhiri komputasi pada waktu yang diinginkan, proses yang lama ini dapat diatasi. Sudah mashur jika AG tidak cukup baik untuk mencari solusi yang sangat dekat dengan solusi optimal.

Algoritma Genetika merupakan algoritma evolusioner untuk memecahkan masalah optimisasi berdasarkan pada ide-ide evolusi seleksi alam. Algoritma ini mensimulasikan proses evolusi dengan menghasilkan populasi solusi layak dan menerapkan beberapa operator genetik untuk menghasilkan populasi baru berdasarkan aturan seleksi. Pada awal tahun 1992 pemrograman mengenai genetika mulai diperkenalkan oleh J.H. Koza. Algoritma genetika menggunakan analogi secara langsung dari kebiasaan yang alami yaitu seleksi alam. Dalam ilmu biologi, sekumpulan individu yang sama, hidup dan berkembang bersama dalam suatu area, disebut dengan populasi. Algoritma genetika bekerja dengan suatu populasi yang terdiri atas beberapa individu, yang masing-masing individu mempresentasikan solusi yang mungkin bagi suatu permasalahan. Algoritma genetika memiliki performansi yang baik untuk masalah-masalah selain optimisasi kompleks dari satu variabel atau multivariabel.

3. Algoritma Memetika (MA)

MA didasari oleh Teori evolusi Biologi Neo-Darwinian dan pendapat Richard Dawkin tentang meme sebagai unit evolusi kultural yang mampu melakukan perbaikan terhadap dirinya sendiri. MA adalah suatu metode pencarian heuristik yang memiliki karakteristik yang sama dengan AG dikombinasikan dengan metode *'Local Search'* atau pencarian lokal, yang secara bersama-sama dapat meningkatkan kualitas pencarian solusi. Pada MA, fitur pencarian lokal bertujuan untuk melakukan perbaikan lokal yang dapat diterapkan sebelum dan atau sesudah proses seleksi, *crossover* dan mutasi. Pencarian lokal juga sangat berguna untuk mengontrol besarnya ruang pencarian solusi. MA dapat memberikan hasil yang lebih baik daripada AG, namun memerlukan waktu komputasi yang lebih lama.

MA adalah perluasan dari Algoritma Genetika. Algoritma Genetika (AG) merupakan salah satu metode heuristik yang banyak digunakan untuk menyelesaikan masalah optimisasi kombinatorial yang sulit. Menurut sebuah penelitian yang dilakukan oleh Ali dkk tahun 2013, pada kasus yang memiliki *time slot* yang terbilang besar algoritma memetika dengan baik menyelesaikan

masalah tersebut. Menurut peneliti lainnya proses pencarian lokal yang dimiliki oleh algoritma memetika juga sangat membantu dalam pencarian solusi optimal terdekat dengan meningkatkan kualitas dari individu. Algoritma memetika lebih baik daripada algoritma genetika dalam hal pencarian ruang solusinya, algoritma memetika juga memiliki perbedaan dalam hal kualitas kromosom yang dihasilkan dilihat dari nilai fitness dari tiap generasi.

Algoritma memetika merepresentasikan salah satu area riset yang sedang berkembang pada bidang komputasi evolusioner. Istilah algoritma memetika secara luas telah digunakan untuk menggambarkan peningkatan prosedur dalam masalah pencarian baik yang dilakukan dengan pendekatan individu maupun pendekatan populasi. Algoritma memetika juga biasa mengacu pada Algoritma Evolusi Baldwinian, Algoritma Evolusi Lamarckian, Algoritma Kultural, maupun algoritma genetika pencarian lokal.

Terinspirasi dari dua prinsip yaitu prinsip evolusi alami Darwin dan gagasan Dawkin tentang meme, istilah algoritma memetika dikenalkan oleh Moscato tahun 1989 dalam laporan tekniknya dimana dia menunjukkan

algoritma memetika sebagai bentuk paling dekat dan dengan algoritma genetika berdasarkan populasi digabungkan dengan prosedur pembelajaran individual. Pada konteks yang berbeda algoritma memetika saat ini digunakan dengan berbagai nama yang bervariasi termasuk Algoritma Evolusi Hybrid, Algoritma Evolusi Baldwinian, Algoritma Evolusi Lamarckian, algoritma Kultural, algoritma genetika pencarian lokal. Pada konteks beragamnya optimasi, ragam algoritma memetika telah dilaporkan sesuai dengan luasnya cakupan aplikasinya, umumnya untuk solusi berkualitas tinggi dan lebih efisien dibandingkan dengan cara-cara konvensional.

Umumnya, menggunakan ide-ide memetic dan menggabungkannya dengan framework komputasi disebut "Komputasi memetic *(Memetic Computation)*". Memetic Computation memperluas gagasan memes mencakup entitas-entitas konseptual dari prosedur pengembangan pengetahuan atau representasi.

Algoritma memetika adalah algoritma sederhana namun fleksibel dan kuat yang dapat menemukan solusi berkualitas tinggi pada banyak tantangan masalah. Masalah optimasi akan melibatkan puluhan variabel

yang dengan demikian membutuhkan perangkat pengkodean yang baik untuk mengatasi waktu proses. Algoritma ini akan sangat berguna dalam bidang data mining, seperti pengelompokan dalam analisis teks, pada bidang biomedis seperti pada simulasi DNA dan simulasi molekuler, masalah jaringan, teknik seleksi fitur, peramalan, kimia kuantum, analisis spektroskopi, analisis geofisika, penemuan obat, studi genom, dan lain sebagainya.

Dalam perkembangannya algoritma memetika mengalami beberapa tingkatan yang terbagi dalam tingkatan generasi pertama, kedua dan ketiga. Generasi-generasi algoritma memetika tersebut akan dijelaskan pada paragraph di bawah ini.

a. Generasi Pertama

Generasi pertama algortima memetika mengacu pada algoritma-algoritma hibrida, perkawinan antara pencarian global algoritma evolusi dengan sebuah tahap evolusi kultural. Generasi pertama ini walaupun meliputi karakteristik dari evolusi kultural (bentuk dari perbaikan lokal) pada siklus pencarian, hal ini mungkin tidak terkualifikasi sebagai sistem yang baik didasarkan pada prinsip Darwinisme dan menjadikan algoritma

memetika menuai banyak kontroversi dan kritikan diantara para peneliti pada masa awal diperkenalkan.

b. Generasi Kedua

Multi-meme memetic algorithm, Hyper-heuristic memetic algorithm dan Meta-Lamarckian memetic algorithm merupakan generasi kedua dari algoritma memetika yang memperlihatkan prinsip-prinsip transmisi dan seleksi memetika pada desainnya. Pada Multi-meme MA, material memetika disandikan menjadi bagian dari *genotype*. Berikutnya meme dibaca dari masing-masing individu/kromosom dan digunakan untuk melakukan perbaikan lokal. Material memetik selanjutnya ditransmisikan melalui mekanisme pewarisan yang sederhana dari orangtua ke anak. Disisi lain pada hyper-heuristik dan *meta-Lamarckian* MA, himpunan kandidat meme akan bersaing berdasarkan nilai mereka sebelumnya, menentukan meme mana yang dipilih untuk proses selanjutnya. Meme-meme dengan nilai yang besar memiliki kesempatan yang besar untuk direplikasi atau diduplikasi.

c. Generasi Ketiga

Co-evolution dan *self-generating* memetic algorithm termasuk dalam generasi ketiga algoritma memetika.

Berbeda dengan generasi kedua yang mengasumsikan bahwa semua meme yang akan digunakan dikenal dengan *a priori*, generasi ketiga ini menggunakan aturan pencarian lokal terhadap solusi dengan menggunakan sistem evolusi.

Algoritma memetika berhasil dipakai untuk menyelesaikan berbagai masalah pada kehidupan. Selain algoritma memetika, nama lainnya seperti algoritma genetika hybrid juga sering digunakan. Lebih jauh lagi beberapa orang menyebut istilah memetika yang mereka perkenalkan dengan algoritma genetika.

Para peneliti menggunakan algoritma memetika untuk mengatasi masalah kompleksitas, diantaranya partisi grafik, pewarnaan grafik, pengepakan dan masalah umum lainnya. Aplikasi-aplikasi pada implementasi algoritma memetika diantaranya adalah *artificial neural network*, *pattern recognition* (pengenalan pola), *robotic motion planning*, *beam orientation*, *circuit design*, *electric service restoration*, *medical expert systems*, penjadwalan mesin tunggal, penjadwalan otomatis, penjadwalan orang, *nurse rostering optimization*, alokasi prosesor, penjadwalan perawatan (contoh: distribusi listrik dan jaringan), multidimensional *knapsack*, desain

VLSI, *clustering of gene expression profiles*, *feature* atau *gene selection*, dan *multi-class, multi-objective feature selection*.

Secara spesifik algoritma memetika telah banyak digunakan untuk pemrosesan gambar seperti meningkatkan kecerahan gambar, mendeteksi emosi melalui wajah, serta pengenalan karakter huruf, pengenalan tulisan tangan, pengenalan retina mata, pemetaan *sub-pixel* dalam gambar, klasifikasi gambar, maupun pencocokan sidik jari.

Pada tahun 2007, Sheng dkk menggunakan algoritma memetika untuk mencocokan sidik jari melalui fitur *minutiae*-nya, namun metode ini cenderung lambat dan tidak cocok digunakan secara *real-time* sehingga pada buku ini penulis menggunakan algoritma memetika untuk pengenalan sidik jari dengan mengkonversi gambar ke kode biner.

Komposisi dalam algoritma memetika mengadopsi istilah yang biasa kita temukan dalam evolusi biologi, yaitu:

1) 'Kromosom' atau individu untuk merepresentasikan 'solusi'.
2) 'Generasi' untuk menggantikan istilah iterasi

3) 'Orang btua' adalah kromosom pada generasi sekarang yang terpilih untuk menghasilkan kromosom anak.
4) 'Anak' adalah kromosom hasil perkawinan kromosom orangtua pada generasi sekarang yang akan menjadi kromosom pada generasi berikutnya.
5) 'Proses evolusi' adalah proses yang terjadi pada tiap generasi yang meliputi proses seleksi, *crossover*, mutasi dan *local search*.

```
Memetic Algorithm Pseudocode
Begin
        INITIALIZE population;
        EVALUATE each candidate;
        Repeat Until (TERMINATION CONDITION) Do
                SELECT parents;
                RECOMBINE to produce offspring;
                MUTATE offspring;
                IMPROVE offspring via LocalSearch;
                EVALUATE offspring;
                SELECT individuals for next generation;
        EndDo
End
```

Gambar 4.1 *Pseudocode* Algoritma Memetika

Algoritma memetika mengawali prosesnya dengan menginisiasi populasi inputan kemudian mengevaluasi tiap kandidat dalam populasi. Evaluasi dilakukan terus menerus sampai kondisi yang diinginkan terpenuhi. Selanjutnya algoritma memilih *parents* dari populasi

terpilih untuk dilakukan persilangan sehingga terbentuk populasi baru, yaitu keturunan dengan kualitas lebih baik dan dilakukan mutase. Untuk menjaga kualitas keturunan algoritma melakukan pencarian lokal sehingga pada proses akhir akan didapatkan keturunan yang paling baik sebagai solusi.

Kompleksitas algoritma memetika digambarkan dengan notasi $O(n^2)$ atau kompleksitas kuadratik. Algoritma memetika menyelesaikan *n* inputan dengan n^2 langkah penyelesaian. Artinya algoritma memetika dianggap tidak efisien seperti yang telah diungkapkan oleh Maltoni dkk.

Menurut Donald E. Knuth algoritma memiliki beberapa ciri sebagai berikut:

1) Algoritma mempunyai awal dan akhir, suatu algoritma harus berhenti setelah mengerjakan serangkaian tugas. Dengan kata lain, suatu algoritma memiliki langkah terbatas.
2) Setiap langkah pada algoritma harus didefinisikan dengan tepat sehingga tidak memiliki arti ganda.
3) Algoritma memiliki masukan (input) atau kondisi awal.

4) Algoritma memiliki keluaran (output) atau kondisi akhir.
5) Algoritma harus efektif, bila diikuti benar-benar maka akan menyelesaikan persoalan.

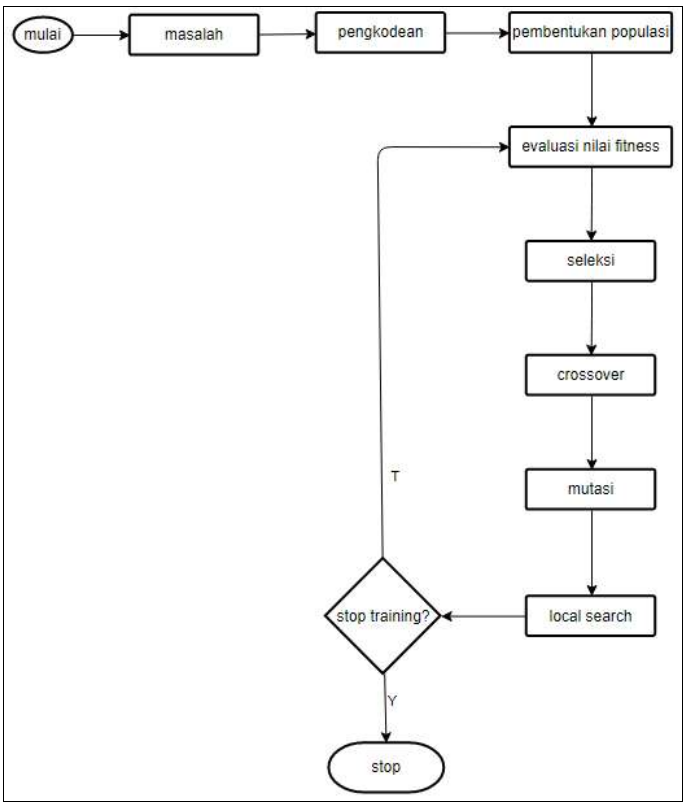
Algoritma memetika adalah perluasan dari algoritma genetika yang diperkenalkan oleh Moscato dan Norman di tahun 1992. Setelah pengenalan algoritma tersebut, Radcliffe dan Surrey merumuskan perbandingan algoritma genetika dan algoritma memetika. Algoritma memetika merepresentasikan informasi dalam meme yang setiap unitnya dapat mereplikasi diri dengan berkomunikasi antar individu. Proses replikasi disebut sebagai *process of self-reproduction* atau *the memetic life-cycle* yang dilanjutkan dengan penyebaran meme.

Proses replikasi secara vertikal dari orang tua ke anak adalah konsep algoritma genetika sedangkan proses replikasi horizontal merupakan proses yang terjadi dalam satu individu. Selanjutnya kedua konsep tersebut digabungkan menjadi dasar algoritma memetika. Dasar algoritma memetika adalah teori evolusi neo darwinian dalam ilmu biologi dan pendapat tentang meme yang merupakan unit evolusi yang dapat

memperbaiki dirinya sendiri yang dikemukakan oleh Dawkin.

Algoritma memetika merupakan metode kombinasi antara algoritma genetika dan pencarian lokal. Perbedaan algoritma genetika dan algoritma memetika lebih pada solusi yang dihasilkan. Algoritma memetika menghasilkan solusi dari hasil pencarian lokal.

Diagram alir algoritma memetika ditunjukkan pada Gambar 4.2 dibawah ini.



Gambar 4.2 Diagram Alir Algoritma Memetika

Algoritma yang diusulkan dapat dibagi menjadi dua tahap: tahap pertama adalah menghasilkan populasi awal dengan konfigurasi perataan lokasi secara efisien. Sedangkan tahap kedua adalah melakukan evolusi hibrida untuk mengidentifikasi apakah solusi yang dihasilkan sudah optimal atau mendekati optimal.

Pada tahap pertama, penulis memperkenalkan operasi pencocokan lokal untuk mendapatkan inisial populasi yang selaras dengan memeriksa fitur lokasi objek. Setelah inisialisasi, prosedur pencarian berbasis algoritma genetika dilakukan untuk mengidentifikasi optimasi secara menyeluruh sesuai dengan aturan stokastik algoritma genetika. Untuk meningkatkan efisiensi, operator lokal dirancang dan dihibridisasi dengan menggunakan fitur pencarian algoritma genetika. Kelayakan solusi dihitung menggunakan fungsi pencocokan dengan menggabungkan pasangan *minutiae* yang sesuai. Dengan kata lain, pencocokan sidik jari adalah menemukan jumlah maksimum nilai *minutiae* yang sesuai dengan rata-rata nilai yang dihasilkan dalam pencarian.

Struktur dari algoritma memetika dapat didefinisikan dengan langkah-langkah berikut.

1) Pengkodean.

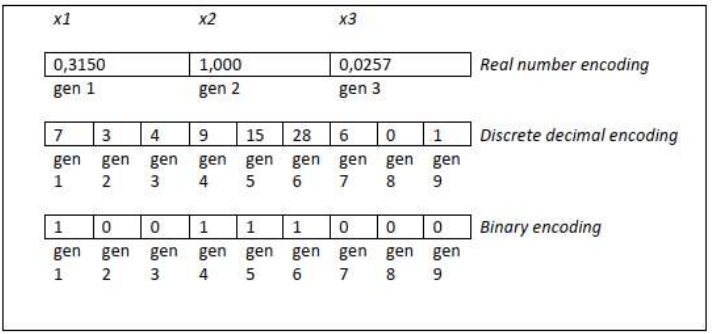
   Cara merepresentasikan solusi ke dalam kromosom. Dalam pengkodean, setiap kromosom harus dikodekan untuk merepresentasikan tepat satu solusi. Secara umum, suatu kromosom berbentuk:

   kromosom = $\{gen_1, gen_2, gen_3, gen_4 \ldots, gen_n\}$

Kromosom terdiri dari himpunan beberapa gen, dimana *n* adalah jumlah gen dan setiap gen mempunyai nilai yang mengisinya atau *allele*.

Metode pengkodean yang paling umum digunakan adalah:

a. Pengkodean dengan bilangan real *(real number).* Merupakan pengkodean gen dengan bilangan real yang bernilai [0,R]. R merupakan bilangan real positif yang biasanya bernilai 1.
b. Pengkodean dengan bilangan desimal diskret *(discrete decimal)*. Merupakan pengkodean gen dengan bilangan bulat yang bernilai [0,N].
c. Pengkodean dengan biner *(binary)*. Merupakan pengkodean yang menetapkan nilai setiap gen dengan 0 atau 1. Gambaran skema metode pengkodean terlihat pada Gambar 4.3 dibawah ini.



Gambar 4.3 Metode Pengkodean

Gambar 4.3 di atas merepresentasikan tiga variabel, *x1, x2,* dan *x3*, yang dikodekan ke dalam 3 gen dalam pengkodean bilangan real, 9 gen dalam pengkodean desimal diskret dan pengkodean biner.

2) Inisiasi atau pembentukan populasi

   Pembentukan populasi umumnya dilakukan secara acak. Algoritma memetika mencari solusi dari kumpulan kromosom yang diperbaharui setiap generasi.

3) *Fitness Function*

   Kinerja individu diukur dengan fungsi tertentu. Pada teori evolusi, individu dengan nilai fitness tertinggi yang akan hidup dan individu dengan nilai fitness yang rendah akan mati. Nilai fitness dapat dirumuskan seperti dibawah ini:

$$f = \frac{1}{(h+a)'}$$

Dimana *f* adalah fitness, *h* adalah fungsi yang di optimasi nilainya, dan *a* adalah bilangan variatif yang dianggap sangat kecil sesuai dengan permasalahan yang akan diselesaikan. Jika pencarian solusi yang dilakukan adalah dengan memaksimalkan fungsi *h,* maka nilai *fitness* sama dengan nilai fungsi *h* tersebut atau *f=h.* masalah akan terjadi jika pencarian solusi dilakukan dengan meminimalkan nilai fungsi *h,* yang karenanya fungsi *h* tidak dapat langsung digunakan. Dengan demikian digunakanlah fungsi fitness f = 1/*h*, dengan catatan *h* tidak bernilai 0.

4) Fungsi Penalti

   Fungsi ini memberikan nilai pada kromosom yang representatif untuk menentukan kualitas dan kelayakan solusi. Pada optimasi, penentuan nilai ini akan diminimalkan sehingga akan diperoleh solusi dengan kendala yang sedikit. Semakin kecil nilai akan semakin meningkatkan kualitas solusi yang dihasilkan.

5) Seleksi

Seleksi yang dimaksud adalah seleksi kandidat orang tua dengan melakukan pendekatan terhadap kandidat solusi yang diharapkan. Seleksi dilakukan dengan memilih kromosom-kromosom dari setiap generasi populasi yang akan menjadi orang tua untuk generasi berikutnya. Tahap pertama adalah mencari nilai *fitness*. Seleksi dapat dilakukan dengan:

a. Roulette wheel selection

Setiap kromosom menempati potongan roda roulette. Nilai *fitness* kromosom sebanding dengan ukuran roda. Hal pertama yang dilakukan adalah menghitung nilai *fitness* setiap kromosom kemudian dibandingkan dengan total nilai *fitness* dan akan didapat kromosom yang bertahan hidup dan menjadi calon orang tua.

b. Tournament selection

Kromosom dipilih secara acak kemudian dipilih satu yang memiliki nilai *fitness* paling tinggi dan dijadikan calon orang tua. Iterasi dilakukan sampai didapat jumlah calon orang tua sesuai keinginan.

c. Rank selection

Kromosom terlebih dahulu diurutkan berdasarkan nilai *fitness*nya, dari yang terburuk sampai yang memiliki nilai *fitness* terbaik akan tetapi teknik ini menyebabkan

lambatnya proses konvergensi karena kecilnya kesempatan setiap kromosom untuk terpilih.

d. Truncation selection

Teknik ini efektif digunakan pada populasi yang besar. Kromosom diurutkan berdasarkan nilai *fitness*nya kemudian dipilih yang terbaik sebagai calon orang tua. Parameter penilaiannya adalah nilai ambang trunk yang merupakan ukuran populasi yang akan dipilih sebagai calon orang tua. Individu dengan nilai dibawah nilai ambang tidak menghasilkan keturunan.

e. Persilangan (crossover)

Persilangan dilakukan untuk menghasilkan keturunan baru atau kromosom anak. Proses ini penting karena pada proses ini akan dicari kromosom-kromosom yang potensial sebagai kandidat solusi yang diharapkan. Beberapa jenis persilangan adalah sebagai berikut:

1) One point crossover

Merupakan jenis persilangan yang paling sederhana. Persilangan dilakukan dengan memilih satu titik secara acak untuk membagi orang tua, kemudian gen pada titik terpilih diturunkan ke kromosom anak sesuai posisinya. Proses yang sama dilakukan pada orang tua lainnya sampai semua gen diturunkan pada anak.

2) Two points crossover

Berbeda dengan jenis persilangan sebelumnya, pada persilangan jenis ini adalah memilih dua titik secara acak untuk membagi orang tua, kemudian nilai *fitness* yang ada diluar titik akan diturunkan ke kromosom anak.

3) Uniform crossover

Pemilihan gen yang akan diturunkan kepada anak dilakukan dengan menggunakan rasio.

6) Mutasi

Mutasi adalah proses perubahan informasi dalam kromosom hasil persilangan. Mutasi dilakukan pada kromosom anak dengan tingkat probabilitas mutasi yang kecil. Mutasi dilakukan untuk menggantikan gen yang hilang dari populasi karena proses seleksi. Mutasi ada dua jenis, yaitu:

1) Mutasi Biner. Mutasi biner dilakukan dengan mengganti satu atau beberapa nilai gen kromosom. Langkahnya adalah:

a. Menghitung jumlah gen pada populasi, caranya adalah mengalikan panjang kromosom dengan ukuran populasi.
b. Memilih gen secara acak.
c. Kromosom ditentukan berdasarkan gen terpilih.

d. Nilai gen dibalik, dari 0 menjadi 1 dan 1 menjadi 0.

2) Mutasi bilangan real. Mutasi bilangan real merupakan jenis mutasi yang ukuran langkahnya sulit ditentukan. Tidak ada patokan pasti tentang ukuran, karena kadang yang kecil berjalan dengan cepat kadang pula yang besar berjalan lebih cepat.

3) Pencarian lokal. Pencarian lokal dilakukan dengan menukar dua gen atau lebih tanpa mengurangi kualitas kromosom. Secara rinci algoritma pencarian lokal diterangkan pada paragraf selanjutnya.

BAB V
HIGH PERFORMANCE COMPUTING MEMETIC ALGORITHM (HPCMA)

1. Pengumpulan Data

Menggunakan data gambar sidik jari yang didapatkan dari Ivan Bartolome Gonzalez, seorang peneliti dari BiDA Lab *(Biometric and Data Pattern Analytic)* UAM, Madrid, Spanyol. Data tersebut berupa data gambar sidik jari sintetik FVC2006 sebanyak 7200 data. Walaupun data tersebut didapat dari luar negeri, bukan dari dalam negeri Indonesia, data gambar sidik jari tersebut juga dapat digunakan mengingat sifat sidik jari yang unversal dan unik serta tidak terbatas pada ras tertentu.

2. Identifikasi Data

Tahapan identifikasi data sebagai berikut, Pada dataset FVC2006 terdapat 8 *folder* yang masing-masing berisi gambar sidik jari dengan karakteristik yang berbeda. Dari ke delapan *folder* tersebut, kemudian dibagi menjadi 4 *folder* besar yang mewakili setiap karakteristik sidik jari. *Folder-folder* tersebut adalah:

1) DB1_A yang berisi 1680 *file* gambar sidik jari dan DB1_B yang berisi 120 *file* gambar sidik jari, sehingga

jumlah total *file* gambar sidik jari dalam kedua *folder* ini adalah 1.800 dan masuk dalam Kelompok 1.

2) DB2_A yang berisi 1680 *file* gambar sidik jari dan DB2_B yang berisi 120 *file* gambar sidik jari, sehingga jumlah total *file* gambar sidik jari dalam kedua *folder* ini adalah 1.800 dan masuk dalam Kelompok 2.
3) DB3_A yang berisi 1680 *file* gambar sidik jari dan DB3_B yang berisi 120 *file* gambar sidik jari, sehingga jumlah total *file* gambar sidik jari dalam kedua *folder* ini adalah 1.800 dan masuk dalam Kelompok 3.
4) DB4_A yang berisi 1680 *file* gambar sidik jari dan DB4_B yang berisi 120 *file* gambar sidik jari, sehingga jumlah total *file* gambar sidik jari dalam kedua *folder* ini adalah 1800 dan masuk dalam Kelompok 4.

Dilakukan pembagian spesimen sebanyak 15 spesimen karena untuk dapat melihat dari segala sisi data yang dimiliki, berikut adalah keterangan spesimen:

Spesimen 1: seluruh data Sidik Jari
Spesimen 2: seluruh data Sidik Jari Kelompok 1
Spesimen 3: seluruh data Sidik Jari Kelompok 2
Spesimen 4: seluruh data Sidik Jari Kelompok 3
Spesimen 5: seluruh data Sidik Jari Kelompok 4

Spesimen 6: seluruh data Sidik Jari Kelompok 1 dan Kelompok 2

Spesimen 7: seluruh data Sidik Jari Kelompok 1 dan Kelompok 3

Spesimen 8: seluruh data Sidik Jari Kelompok 1 dan Kelompok 4

Spesimen 9: seluruh data Sidik Jari Kelompok 2 dan Kelompok 3

Spesimen 10: seluruh data Sidik Jari Kelompok 2 dan Kelompok 4

Spesimen 11: seluruh data Sidik Jari Kelompok 3 dan kelompok 4

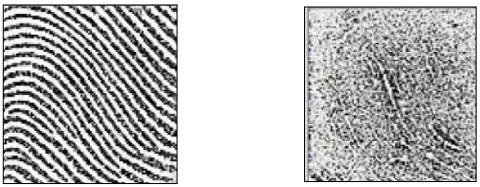
Spesimen 12: seluruh data Sidik Jari Kelompok 1, Kelompok 2, dan Kelompok 3

Spesimen 13: seluruh data Sidik Jari Kelompok 1, Kelompok 2, dan Kelompok 4

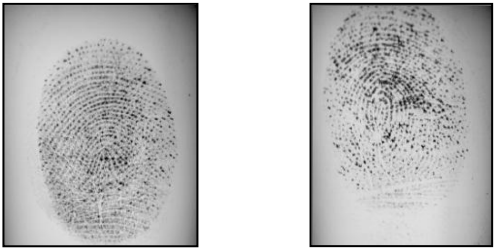
Spesimen 14: seluruh data Sidik Jari Kelompok 2, Kelompok 3, dan Kelompok 4

Spesimen 15: seluruh data Sidik Jari Kelompok 1, Kelompok 3, dan Kelompok 4

Secara visual pengelompokan data gambar sidik jari yang dilakukan adalah sebagai berikut:



Gambar 5.1 Sidik Jari Parsial



Gambar 5.2 Sidik Jari Penuh

Gambar 5.3 Sidik Jari Penuh dengan *Boundary*

Gambar 5.4 Sidik Jari Penuh dan Tidak Jelas *(unclear)*

3. Implementasi HPCMA

mengimplemetasikan algoritma memetika pada lingkungan HPC yang selanjutnya disebut dengan HPCMA (*High Performance Computing Memetic Algorithm).*

Berikut adalah struktur *database* untuk proses *training* dan proses *testing*, dimana tabel konversi gambar teks dan tabel seleksi adalah struktur *database* untuk proses *training* sedangkan tabel hasil *testing* adalah struktur *database* proses *testing*, adapun tabelnya sebagai berikut:

Tabel 5.1 Konversi Gambar ke Teks

Nama Field	Type Data
Key	int(11)
nama_folder	varchar(100)
nama_folder_parent	varchar(100)
nama_file	varchar(100)

Nama Field	Type Data
isi_file	Longtext
byte_file	Longtext
Waktu	Long
p_o	int(1)
key_parent_1	int(11)
key_parent_2	int(11)

Tabel 5.1 di atas berfungsi untuk menyimpan hasil *local search*, hasil seleksi atau *elitism*, hasil *crossover*, dan hasil mutasi.

Tabel 5.2 Seleksi

Nama Field	Type data
Key	int(11)
nama_folder_parent	varchar(100)
byte_file	Longtext

Tabel 5.2 di atas berfungsi untuk menyimpan hasil seleksi dan digunakan untuk proses seleksi berikutnya.

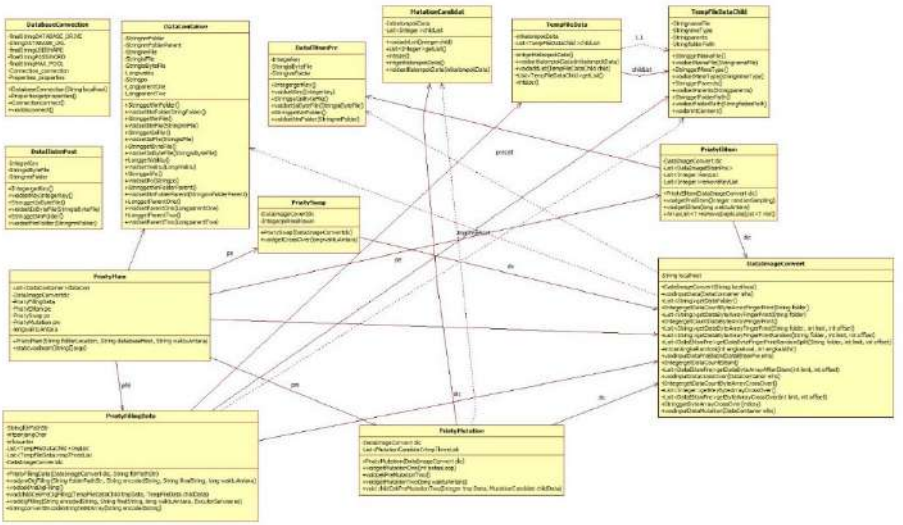
Tabel 5.3 Hasil *Testing*

Nama field	Type data
'hpcma'	char(1)
'key'	int(11)
'match'	int(11)
'timematch'	Long
'sim90'	int(11)
'timesim90'	Long

Nama field	Type data
‘sim80’	int(11)
‘timesim80’	Long
‘sim70’	int(11)
‘timesim70’	Long
‘sim60’	int(11)
‘timesim60’	Long
‘search’	int(11)
‘img_type’	int(11)

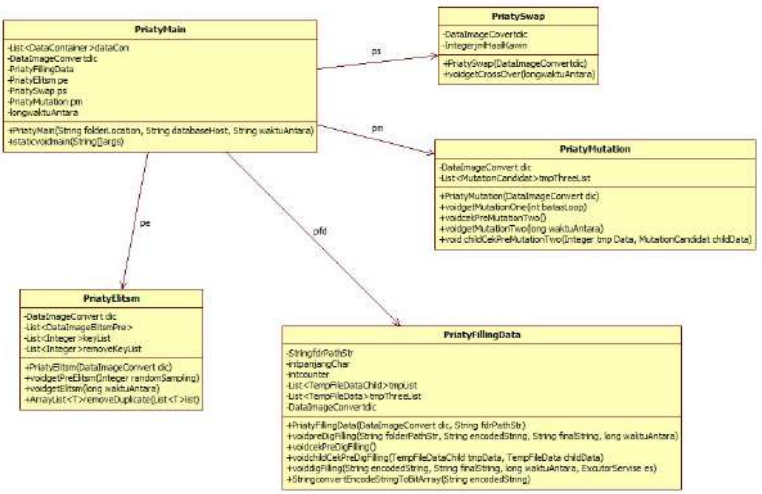
Tabel 5.3 diatas berfungsi untuk menyimpan data hasil *testing*.

Berikut adalah gambaran *class diagram*:



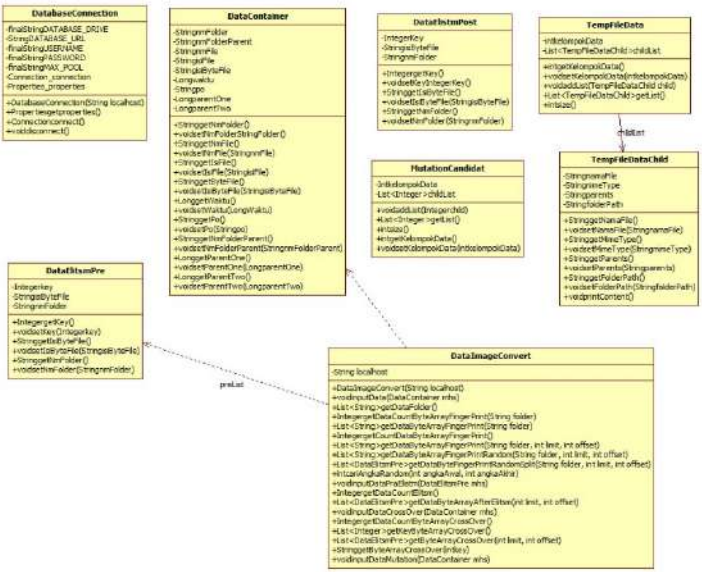
Gambar 5.5 *Class Diagram*

Class diagram keseluruhan adalah gabungan dari *class diagram* primer dan class *diagram* sekunder. Pada Class diagram ini terdapat tiga belas kelas diantaranya *DatabaseConnection, DataContainer*, *DataElitismPre*, *MutationCandidat*, *TempFileData*, *TempFileDataChild*, *PriatyElitism*, *PriatySwap*, *DataElitismPost*, *PriatyMain*, *DataImageConvert*, *PriatyMutation*, dan *PriatyFillingData.*



Gambar 5.6 *Class Diagram* Primer

Class diagram primer adalah fungsionalitas utama dalam *coding* yang berlaku pada skenario 2 dan skenario 3. *Class diagram* primer terdiri dari 5 class yaitu *PriatyMain, PriatySwap, PriatyMutation, PriatyFillingData,* dan *PriatyElitism.*



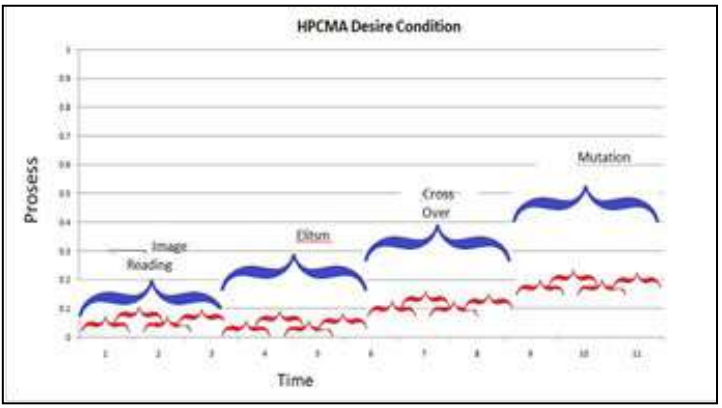
Gambar 5.7 *Class Diagram* Sekunder

Class diagram sekunder adalah fungsionalitas *pendukung* dalam *coding* yang ada pada skenario 2 dan skenario 3. *Class diagram* sekunder terdiri dari 8 *class* yaitu *DatabaseConnection, DataContainer, DataElitsmPost, MutationCandidat, TempFileData, TempFileDataChild, DataImageConvert,* dan *DataElitsmPre.*

Model HPCMA *berjalan* pada *system computer* dengan *processor* Intel i5 2540M 2.6 GHz 4 core RAM 16GB HDD Samsung SSD 850 EVO 500GB sebagai HPCMA

Machine 2 dan *system computer* dengan *processor* Intel i5 2430M 2.4 GHz 4 core RAM 8GB HDD Samsung SSD 860 EVO 250GB sebagai mesin basis data *(Database Machine)*, percobaan terhadap algoritma memetika dan *HPCMA* dilakukan menggunakan 2 PC dengan memanfaatkan jaringan *PC to PC,* tetapi berjalan pada 2 sistem operasi berbeda. Pada lingkungan percobaan pertama sistem operasi yang digunakan adalah Windows 7, sedangkan pada lingkungan percobaan kedua sistem operasi yang digunakan adalah Windows 10.

Penggunaan system operasi yang berbeda dilakukan untuk mengukur fleksibilitas *system* yang dibangun. Fleksibilitas merupakan salah satu syarat utama dalam penanganan sidik jari (Manacher, 1967). Windows 10 secara teknis lebih cepat dan stabil jika dibandingkan dengan Windows 7. Disertai dengan manajemen memori yang juga lebih baik. Karena alasan ini juga Windows 10 digunakan dalam penelitian ini.



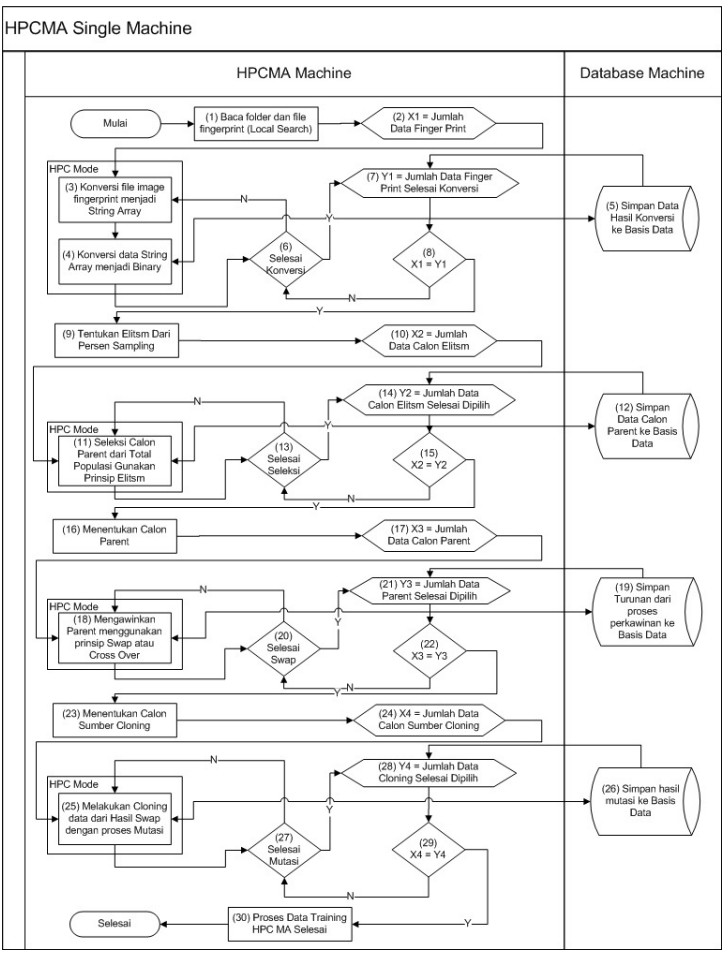
Gambar 5.8 Kondisi HPCMA

Beberapa proses yang dilakukan pada Gambar 5.8 adalah sebagai berikut:

1) *Local Search (LS);* proses mikro yang terjadi pada proses makro ini diantaranya; membaca folder dan file gambar sidik jari, konversi data gambar *string array*, konversi *string array* ke biner.
2) *Selection/Elitism;* proses mikro yang terjadi pada proses makro ini adalah proses seleksi calon *parent*, memilih 2% dari total populasi.
3) *Crossover/Swap;* proses mikro yang terjadi pada proses makro ini adalah perkawinan silang atau *crossover* setiap calon *parent* dari yang telah dipilih dari proses seleksi untuk mendapatkan turunan baru.

4) *Mutation:* proses mutasi dengan membalik posisi biner setiap turunan yang didapat dari *partial test 3*. Posisi biner 1 dirubah menjadi 0, dan begitu juga sebaliknya.

Pada Gambar 5.4 gambar garis berwarna merah adalah ilustrasi proses mikro sedangkan gambar garis berwarna biru adalah ilustrasi proses makro model *HPCMA2*. Proses model *HPCMA2* terbagi menjadi 4 proses makro dan beberapa proses mikro. Algoritma memetika pada model *HPCMA2* menjalankan proses makro secara berurutan atau sekuensial, sedangkan semua proses mikronya dijalankan secara *parallel*. Berikut ini adalah rincian proses model *HPCMA2* seperti terlihat pada diagram alir Gambar 5.5.



Gambar 5.9 Proses HPCMA2

Detil uraian Gambar 5.9 adalah sebagai berikut:

Proses Makro 1, proses mencari dan menterjemahkan data gambar sidik jari menjadi biner.

1) Melihat dan membaca *folder* dan *sub-folder* yang berisi *image* sidik jari yang sudah terbagi/ditentukan menjadi empat kelompok besar sesuai dengan karakteristiknya yang sudah dilakukan pada tahap identifikasi data yang akan diolah menjadi data *binary* kemudian disimpan dalam *class* data.
2) Menghitung jumlah *class* data yang terbentuk kemudian yang akan dijadikan data acuan jumlah data.
3) Mulai melakukan proses konversi masing-masing data image sidik jari menjadi string array dalam HPC Mode.
4) Melakukan proses konversi data string array menjadi binary dalam HPC Mode.
5) Menyimpan data binary ke tabel di *database server*.
6) *Looping* jika data gambar sidik jari nya sudah selesai diproses maka akan masuk ke proses

berikutnya, jika belum proses diulang ke proses mikro 3.

7) Mengambil data total data *image* sidik jari yang sudah dikonversi.
8) Setelah proses konversi selesai maka dihitung jumlah data yang selesai diproses dibandingkan dengan jumlah seluruh data *image* sidik jari, jika sama maka proses bisa langsung masuk ke proses *HPC* makro 2, jika belum maka proses harus menunggu proses makro 1 selesai.

Proses Makro 2, proses *selection* atau *elitsm* dengan sampling 2% dari total populasi data gambar sidik jari total.

9) Menentukan data *selection* atau *elitsm* dengan sampel 2% dari jumlah total data binary (angka 2% dibuat menjadi parameter yang dapat dirubah sesuai kebutuhan, saat ini mengambil 2% karena keterbatasan *hard disk* dalam menyimpan data seluruh *binary* ketika sudah selesai melalui proses *MA* membutuhkan sekitar 23 GB.)
10) Setelah proses mikro 9 selesai maka diketahui jumlah data *elitsm*.

11) Dengan acuan jumlah data elitsm dari proses mikro sebelumnya mulai melakukan pemilihan calon data *elitsm* secara acak sesuai dengan jumlah yang ditentukan.
12) Menyimpan hasil data elitsm ke tabel di *database server*.
13) *Looping* jika proses pemilihan elitsm selesai diproses maka akan masuk ke proses berikutnya, jika belum maka proses diulang ke proses mikro 11.
14) Mengambil data total data *elitsm* yang sudah selesai diproses.
15) Setelah proses pemilihan *elitsm* selesai maka dihitung jumlah data yang selesai diproses dibandingkan dengan jumlah *elitsm* yang ditetapkan diawal (dengan acuan sampling 2%), jika sama maka proses bisa langsung masuk ke proses *HPC* makro 3, jika belum maka akan menunggu proses makro 2 selesai.

Proses Makro 3, proses melakukan *crossover* atau *swap* atau mengawinkan data *elitsm* supaya menjadi *offspring* baru.

16) Menentukan calon *parent* dari data *elitsm* yang sudah dimiliki, dimana dari data *elitsm* tersebut akan dibagi 2 dengan jumlah sama persis untuk menetapkan *parent x* dan *y*.
17) Setelah diketahui jumlah *parent x* dan *parent y* kemudian antara x dan y dikalikan, akhirnya diketahui jumlah kemungkinan perkawinan yang dapat dilakukan.
18) Melakukan perkawinan menggunakan metode *swap* atau *crossover*, yaitu memindahkan 100 *byte* data binary dari *parent x* ke *parent y* dengan lokasi perpindahan yang sudah ditentukan dan menimpa 100 *byte* asli di *parent y*. Tanpa merubah data parent y kemudian dihasilkan menjadi *offspring* baru.
19) Menyimpan hasil data *offspring* baru ke tabel di *database server.*
20) *Looping* jika proses *swap* atau *crossover* selesai diproses maka akan masuk ke proses berikutnya, jika belum maka proses diulang ke proses mikro 18.
21) Mengambil data total data hasil *swap* atau *crossover* yang sudah selesai diproses.

22) Setelah proses *swap* dan *crossover* selesai maka dihitung jumlah data yang selesai diproses dibandingkan dengan jumlah hasil perkalian *parent x* dan *parent y* yang sudah diketahui diawal, jika sama maka proses bisa langsung masuk ke proses *HPC* makro 4, jika belum maka akan menunggu proses makro 3 selesai.

Proses Makro 4, proses melakukan *mutation* atau memutar beberapa *byte* dari data *binary offspring* hasil dari *crossover* atau *swap* supaya menjadi *offspring* baru.

23) Mengambil data *offspring* dari hasil proses *swap* atau *crossover* untuk dijadikan *parent cloning* dari proses *mutation*.
24) Menghitung jumlah calon *parent cloning* untuk proses mutasi.
25) Melakukan proses mutasi dari *parent* kemudian dari *cloning* tersebut lakukan pertukaran *byte* dari data binary hingga menjadi *offspring* baru.
26) Menyimpan hasil data *offspring* baru ke tabel di *database server*.

27) *Looping* jika proses mutasi selesai diproses maka akan masuk ke proses berikutnya, jika belum maka proses diulang ke proses mikro 25.
28) Mengambil data total data hasil mutasi yang sudah selesai diproses.
29) Setelah proses mutasi selesai maka dihitung jumlah data yang selesai diproses dibandingkan dengan jumlah *parent cloning mutation* yang sudah diketahui diawal, jika sama maka proses selesai, jika belum maka akan menunggu proses makro 4 selesai.
30) Proses menyelesaikan proses *thread* yang masih berjalan di *background*.

Psesudocode pertama pada skenario ketiga adalah *pseudocode Main.* Berikut adalah deskripsi dari *pseudocode Main*:

Variabel global dalam *class PriatyMain* merupakan media yang menyambungkan fungsi yang berada diluar *class PriatyMain* seperti *DataImageConverter* yang berisi fungsi untuk mengolah gambar menjadi biner dan menyimpannya ke dalam basis data, kemudian fungsi *PriatyFillingData* yang bertugas sebagai secara logis untuk memasukkan data gambar yang telah berbentuk

biner ke dalam basis data menggunakan prinsip kerja algoritma memetika yang berpadu dengan *thread* dan *multi-thread* pada konsep *HPC.*

Method main pada *class PriatyMain* berisi logika dan pengurutan seluruh fungsi algoritma memetika agar semua urutan proses, mulai dari pencarian lokal, penyimpanan ke dalam basis data, pemilihan kandidat pada proses seleksi, membagi hasil seleksi ke dalam kelompok *male* dan *female*, menyilangkan atau *crossover* antara *male* dan *female*, hingga proses mutasi yang berjalan dengan memutasikan keturunan hasil proses *crossover* menjadi keturunan baru, dapat berjalan dengan baik tanpa ada proses yang saling tumpang tindih atau *overlap* sehingga tidak ada data yang hilang.

Constructor pada *class PriatyMain* adalah *constructor* yang memiliki fungsi untuk mengaktifkan variabel dalam *class* yang sudah diinisiasi pada awal *class* agar dapat berjalan dengan baik serta memastikan setiap *method* yang terdapat dalam *class* tersebut dapat diakses. Berikut adalah *pseudocode* secara rinci:

Pseudocode Main

```
Pseudocode PriatyMain.java

Initiate variable globally dataCon to ArrayList
of DataContainer,
```

```
Initiate variable globally dic to
DataImageConvert,
Initiate variable globally pfd to
PriatyFillingData,
Initiate variable globally pe to PriatyElitsm,
Initiate variable globally ps to PriatySwap,
Initiate variable globally pm to PriatyMutation,
Initiate variable globally waktuAntara to long.
Main method;
     Initiate locally encodeStr to String,
     Initiate locally finalString to String,
     Constructor locally pic to PriatyMain with
parameters;
     Initiate variable locally waktuMakro to
Current Time;
     Initiate variable locally waktuMakroAll to
Current Time;
     Constructor locally es to Executor with
CacheThreadPool;
     Call function preDigFilling with parameters
from PriatyFillingData;
     Call function cekPreDigFilling with
parameters from PriatyFillingData;
     Call function digFilling with parameters
from PriatyFillingData;
     Assign value waktuMakro from CurrentTime
minus variable waktuMakro;
     while condition true;
          Initiate variable locally jmlAwal to
Data Temporary Size;
          Initiate variable locally jmlAkhir to
Current Data Size;
          If jmlAwal equal jmlAkhir Then
               Initiate variable locally
waktuMakro to Current Time;
               Call function preElitsm with
parameter value from PriatyElitsm;
               Call function elitsm with
parameter value from PriatyElitsm;
               Assign value waktuMakro from
CurrentTime minus variable waktuMakro;
               while condition true;
                    Initiate variable locally
jmlElAwal to Data Elitsm Temporary Size;
```

```
                    Initiate variable locally
jmlElAkhir to Current Data Elitsm Size;
                    If jmlElAwal equal
jmlElAkhir Then
                         Initiate variable
locally waktuMakro to Current Time;
                         Call function crossOver
with parameter value from PriatySwap;
                         Assign value waktuMakro
from CurrentTime minus variable waktuMakro;
                         while condition true;
                              Initiate variable
locally jmlCro to Data Crossover Temporary Size;
                              Initiate variable
locally jmlCroAkhir to Current Data Crossover
Size;
                              If jmlCrol equal
jmlCroAkhir Then
                                   Initiate
variable locally waktuMakro to Current Time;
                                   Call function
preMutation from PriatyMutation;
                                   Call function
                                        mutatio
                                        n with
                                        paramet
                                        er
                                        value
                                        from
                                        PriatyM
                                        utation
                                        ;
                                   Assign value
                                   waktuMakro
                                   from
                                   CurrentTime
                                   minus
                                   variable
                                   waktuMakro;
                                   break;
                              End if;
                         End while;
                         Break;
                    End if;
```

```
                End while;
                Break;
           End if;
      End while;
      Assign value waktuMakro from CurrentTime
                                          minus
                                          variabl
                                          e;
End Main Method.

Constructor PriatyMain with parameters;
      Assign value waktuAntara from parameter;
      Assign value dic from DataImageConvert with
                                          paramet
                                          er;
      Assign value pfd from PriatyFillingData with
                                          paramet
                                          er;
      Assign value pe from PriatyElitsm with
                                          paramet
                                          er;
      Assign value ps from PriatySwap with
                                          paramet
                                          er;
      Assign value pm from PriatyMutation with
                                          paramet
                                          er;
End Constructor;
```

Pseudocode kedua pada skenario ketiga adalah *Filling Data.* Deskripsi *pseudocode Filling Data* adalah sebagai berikut:

Variabel global dalam *class PriatyFillingData* adalah media yang menyambungkan fungsi yang berada diluar *class PriatyFillingData*, seperti *DataImageConverter* yang berisi fungsi untuk memasukkan data gambar ke basis

data, serta variabel lain yang berisi data yang akan menentukan alur logika dalam pengisian dan pengolahan data gambar.

Constructor dalam *class PriatyFillingData* berfungsi untuk mengaktifkan *class* variabel yang sudah diinisiasi diawal *class* agar dapat berfungsi serta mengisi variabel dengan nilai yang dibutuhkan untuk alur logika pada pengolahan data gambar dalam *class.*

Method preDigFilling adalah *method* yang berisi proses untuk memasukkan data biner ke dalam *class* temporer sebagai tempat pengumpulan data sementara sebelum didistribusikan kembali ke dalam *class container* yang sudah terbagi menurut urutan *thread.*

Mehtod cekPreDigFilling adalah *method* yang berisi proses memasukkan data biner dari *class* temporer ke dalam *class thread*. Data biner dibagi ke dalam 25 *thread class* sehingga ketika data-data tersebut dieksekusi, semua proses dapat berjalan secara paralel 25 proses sekaligus.

Method childCekPreDigFilling adalah *method* yang berisi proses memasukkan data ke dalam masing-masing *child class*, dimana proses ini merupakan sub proses dari

cekPreDigFilling untuk mengeliminasi code yang berulang untuk proses yang sama.

Method digFilling adalah *method* yang berisi proses untuk menjalankan fitur *multi-thread* yang menjalankan 25 *threads* secara paralel dengan *input* data yang berasal dari *class* temporer yang telah di kelompokkan menjadi 25 kelompok data agar masing-masing kelompok data dapat berjalan secara bersamaan menggunakan perulangan. Sedangkan method *convertEncodeStringToBitArray* adalah *method* yang berisi proses untuk merubah *array string* menjadi kode biner yang kemudian siap untuk dimasukkan ke dalam basis data. Berikut adalah pseudocode *Filling Data* secara rinci:

Pseudocode Filling Data

```
Pseudocode PriatyFillingData.java
Initiate variable globally fdrPathStr to String,
Initiate variable globally panjangChar to int,
Initiate variable globally counter to int,
Initiate variable globally tmpList to ArrayList
of Temporary File Data Child,
Initiate variable globally tmpThreeList to
ArrayList of Temporary File Data,
Initiate variable globally dic to
DataImageConvert.

Constructor PriatyFillingData with parameters;
     Assign value dic from parameter;
     Assign value fdrPathStr from parameter;
     Assign value panjangChar from parameter;
```

```
End Constructor;

Function method with name preDigFilling with
parameters;
     Initiate locally folderPath to File with
parameters.
     For each fileEntry in listFile Do
          Initiate variable locally mimeType to
String with probeContentType;
          if variable fileEntry is directory then
               Call this function preDigfilling
with parameters;
          else if
               if mimeType not null and mimeType
contain bmp then
                    Initiate locally
tempFileData to TempFileDataChild.
                    Assign value to each
variables in tempFileData,
                    Add tempFileData to tmpList
arrayList;
               end if;
          end if;
     end for;
End function method.

Function method with name cekPreDigFilling;
     Initiate locally incrementMacro to int with
value.
     Initiate locally increment to int with
value.
     Initiate locally parent to TempFileData.
     for each tmpData from TempFileDataChild in
tmpList Do
          if increment is zero then
               call function
childCekPreDigFilling with parameters;
          else if increment modulus 25 is zero
then
               Initiate locally parent to
TempFileData.
               call function
childCekPreDigFilling with parameters;
          else if increment modules 25 is 24 then
```

```
                call function
childCekPreDigFilling with parameters;
                Set value kelompokData with
incrementMakro in parent,
                Add parent to tmpThreeList array,
                Increment Variable incrementMacro.
          else if tmpList size minus one is
increment value then
                call function
childCekPreDigFilling with parameters;
                Set value kelompokData with
incrementMakro in parent,
                Add parent to tmpThreeList array,
          else
                call function
childCekPreDigFilling with parameters;
          end if;
     end for;
End function method.

Function method with name childCekPreDigFilling;
     Initiate locally tmpFileData  to
TempFileDataChild.
     Assign value to each variables in
tmpFileData,
     Add tmpFileData to childData arrayList;
End function method.

Function method with name digFilling and
paramters;
     Initiate locally counter  to int with value.
     for each tmpData from TempFileData in
tmpThreeList Do
          Thread with name thrd to Thread;
                Initiate locally encodeStringInner
to String.
                Initiate locally finalStringInner
to String.
                Inner function method with name
run
                     for each tmpDataChild from
TempFileDataChild in tmpData Do
                          if mimeType not null
and mimeType contain bmp then
```

```
                                    Initiate locally
waktu  to long with value.
                                    Initiate locally
now  to long with value.
                                    Initiate final
                                    locally
                                    fileContent  to
                                    byte array with
                                    value from
                                         function
                                    readAllByte with
                                    parameter.
                                    Assign value to
                                    variable
                                    encodeStringInner
                                    from encoder
                                    Base64,
                                    Assign value to
                                    variable
                                    finalStringInner
                                    from function
                                    method
                                    Assign value to
                                         variable
                                         waktu from
                                         Current Time
                                         minus
                                         variable now,
                                    Initiate locally
innerData  to DataContainer.
                                    Assign value to
each variables in innerData,
                                    call function
inputData with parameters from variable dic;
                               end if;
                          end for;
                     End inner function method;
               end thread;
               Paremeter es call function execute
               Parameter es call function
awaitTermination with parameters
               Increment Variable counter.
          end for;
          Parameter es call function shutdown.
```

```
End function method.

Function method with name
convertEncodeStringToBitArray with parameter
     Initiate locally innerFinalString  to
StringBuilder.
     for each i from zero to variable size of
encodedString from parameter Do
          Initiate locally tempChar from
encodedString.
          Append value innerFinalString from
tempChar.
     End for;
     return value.
End function method.
```

Pseudocode ketiga dalam skenario ketiga adalah *Elitsm.* Deskripsi secara rinci pseudocode *Elitsm* adalah sebagai berikut:

Variabel global dalam class *PriatyElitsm* merupakan media yang menyambungkan fungsi yang berada diluar *class PriatyElitsm,* seperti *DataImageConverter* yang berisi fungsi untuk memasukkan data gambar serta data binernya ke basis data, maupun variabel lain yang berisi data yang akan menjadi penentu alur logika dalam pengisian dan pengolahan data.

Constructor dalam *class PriatyElitsm* adalah *constructor* yang berfungsi untuk mengaktifkan *class* variabel yang sudah diinisiasi diawal *class* agar dapat berfungsi dan mengisi variabel dengan nilai yang dibutuhkan. *Method getPreElitsm* adalah *method* yang

berfungsi untuk mengambil data kandidat untuk proses elitism atau seleksi dari tabel penampung data biner, kemudian dibandingkan dengan presentase yang sudah ditetapkan sebesar 2%. Jika kemudian ditemukan duplikasi data dari hasil filter maka data tersebut dihapus atau di *remove*.

Method getElitsm adalah *method* yang berfungsi untuk memasukkan data hasil *elitism* atau seleksi ke dalam tabel penampun dengan prinsip *HPC*. Berikut adalah pseudocode proses *Elitsm:*

Pseudocode Elitsm

```
Pseudocode PriatyElitsm.java

Initiate variable globally dic to
DataImageConverter,
Initiate variable globally elPreList to ArrayList
of DataElitsmPre,
Initiate variable globally keyList to ArrayList
of Integer,
Initiate variable globally removeKeyList to
ArrayList of Integer,

Constructor PriatyElistm with parameters;
     Assign value dic from parameter;
End Constructor;

Function method with name getPreElitsm with
parameters;
     Initiate variable locally namaFolder to
ArrayList of String from function getDataFolder
in variable dic,
     For each nmFdr in namaFolder List Do
```

```
          Initiate variable locally hitungData
from function getdataCountByteArray with
parameter;
          Assign value to limitData from
hitungData multiply ramdomSampling persen;
          Initiate variable locally byteArray
from function getdatabytearrayrandomsplit with
parameters;
          For each byteArr in byteArrays List Do
               Assign value to nmFolder variables
in byteArr,
               Add byteArr to elPreList
arrayList;
               Add variable key at byteArr to
keyList arrayList;
          Ed for;
     Ed for;

     Call funtion sort in Collection interface;
     Assign value keyList from call function
                                          method
                                          removeD
                                          uplicat
                                          e with
                                          paramte
                                          r;

     if Sice keyList and Size elPreList not equal
                                          then
          For each pre in removeKeyList Do
               Initiate locally increment to int
with value.
               For each iterator in elPreList map
to DataElitsmPre Do
                    Assign value elPreOld from
iterator next;
                    if elPreOld key equal to
variable pre then
                         if increment equal zero
then
                              call function
remove in iterator;
                              Increment variable
increment.
```

```
                              end if;
                         end if;
                    End for;
               End for;
          end if;
End function method.

Function method with name getElitsm with
parameters;
     Initiate locally es from ExecutorService to
                                             cacheTr
                                             eadPool
                                             ;
     For each byteArr in elPreList Do
          Create thread with name thrd to Thread;
               Create inner function method with
name run
                    Initiate locally elitsPre
to DataElitsmPre.
                    Assign value to each
variables in elitsPre,
                    call function
inputDataPreElitsm with parameters from variable
dic;
                    Assign value null to
elitsPre.
               End inner function method;
          end thread;
          Paremeter es call function execute
          Parameter es call function
awaitTermination with parameters
     End for;
     Parameter es call function shutdown.
End function method.
```

Pseudocode keempat pada skenario ketiga adalah *Swap,* yang dapat dideskripsikan sebagai berikut:

Swap adalah proses persilangan antar kandidat *parent* dari proses *elitsm.* Variabel global pada class *PriatySwap* merupakan media penghubung fungsi yang

berada diluar class *PriatySwap,* seperti *DataImageConverter* yang berisi fungsi untuk memasukkan data gambar serta kode binernya ke dalam basis data, serta variabel lain yang berisi data yang menjadi penentu alur logika dalam pengisian dan pengolahan data.

Constructor dalam class *PriatySwap* berfungsi untuk mengaktifkan *class* variabel yang sudah diinisiasi diawal *class* agar dapat berfungsi serta mengisi variabel dengan nilai yang dibutuhkan. *Method getCrossOver* berfungsi untuk mengambil data hasil *elitsm* dari tabel penampungan data calon *parent* yang sudah diseleksi dan dibagi menjadi dua kelompok, *male* dan *female,* untuk dikawinsilangkan. Perkawinan silang dilakukan satu persatu dengan cara mengambil sebagian baris biner dari *male* kemudian disisipkan atau ditimpakan ke baris biner *female* untuk mendapatkan keturunan baru. Proses persilangan ini menggunakan prinsip *HPC* dengan menggunakan *multi-thread* yang jumlah *thread*-nya disesuaikan dengan jumlah *male*. Berikut adalah *pseudocode Swap* secara rinci:

Pseudocode Swap

`Pseudocode PriatySwap.java`

```
Initiate variable globally jmlHasilKawin to
Integer,
Initiate variable globally dic to
DataImageConvert.

Constructor PriatyElistm with parameters;
     Assign value dic from parameter;
End Constructor;

Function method with name getCrossOver  and
paramters;
     Initiate locally hitungData to Integer from
call function getDataCountElitsmPre.
     Assign value limit from hitungData division
by two;
     Assign value elitsmList from DataElitsmPre
     Array from call function
     getDataByteArrayAfterElitsm with parameter;
     Assign value elitsmListInner from
     DataElitsmPre Array from call function
     getDataByteArrayAfterElitsm with parameter;
     Initiate locally es from ExecutorService to
                                        cacheTr
                                        eadPool
                                        ;
     Assign value jmlHasilHasin from elitsmList
     size multiply elitsmListInner size;
     for each elitsm from DataElitsmPre in
elitsmList Do
          Assign value keyMale to int from
variable key in elitsm;
          Assign value male to String from
substring operation;
          Thread with name thrd to Thread;
               Inner function method with name
run
                    for each elitsmInner from
DataElitsmPre in elitsmList Do
                         Initiate locally now
to long with Current Time.
                         Initiate
locallystrBuild to StringBuilder
                         Assign value to
                         variablemarried from
```

```
                              male append female
                              StringArray,
                              Assign value to
                                   variablekeyFemale
                                   from key in
                                   elitsmInner;
                              Initiate locally
innerData  to DataContainer.
                              Assign value to each
variables in innerData,
                              call function
inputDataCrossOver with parameters from variable
dic;
                         end for;
                    End inner function method;
          end thread;
          Paremeter es call function execute
          Parameter es call function
awaitTermination with parameters
          Increment Variable counter.
     end for;
     Parameter es call function shutdown.
End function method.
```

Pseudocode terakhir pada skenario ketiga adalah *pseudocode Mutation*, yang dapat dideskripsikan sebagai berikut:

Variable global pada *class PriatyMutation* adalah media penyambung ke beberapa fungsi diluar *class PriatyMutation*, seperti *DataImageConverter* yang berisi fungsi untuk memasukan data gambar serta kode binernya ke dalam basis data, serta variabel lain yang berisi data yang akan menentukan alur logika dalam pengisian dan pengolahan data. *Constructor* dalam *class*

PriatyMutation berfungsi untuk mengaktifkan *class* variabel yang sudah diinisiasi diawal *class* agar berfungsi serta mengisi variabel dengan nilai yang dibutuhkan.

Method cekPreMutation adalah *method* yang berisi proses untuk memasukkan data biner dari hasil *swap* atau *crossover* ke dalam *thread class* dengan menggunakan 25 *threads*, data dibagi sesuai dengan jumlah *thread* sehingga ketika dieksekusi semua data dapat berjalan secara paralel. *Method getMutation* adalah *method* yang berisi proses untuk melakukan mutasi data hasil *swap* menjadi keturunan baru. Caranya adalah membalik nilai kode biner masing-masing hasil *swap* menjadi data mutation baru antara biner ke 0 hingga biner ke 1000. Proses ini ditangani oleh 25 *thread HPC*. Berikut adalah pseudocode *Mutation* secara rinci:

Pseudocode Mutation

```
Pseudocode PriatyMutation.java

Initiate variable globally tmpThreeList to
MutationCandidat ArrayList,
Initiate variable globally dic to
DataImageConvert.

Constructor PriatyMutation with parameters;
     Assign value dic from parameter;
End Constructor;

Function method with name cekPreMutation;
```

```
     Initiate locally keyList to Integer List
from call function getKeyByteArrayCrossOver at
dic;
     Initiate locally incrementMacro to int with
value.
     Initiate locally increment to int with
value.
     Initiate locally parent to MutationCandidat.
     for each tmpData from Integer in keyList Do
          if increment is zero then
               call function childCekPreMutation
with parameters;
          else if increment modulus 25 is zero
then
               Initiate locally parent to
MutationCandidat.
               call function childCekPreMutation
with parameters;
          else if increment modules 25 is 24 then
               call function childCekPreMutation
with parameters;
               Set value kelompokData with
incrementMakro in parent,
               Add parent to tmpThreeList array,
               Increment Variable incrementMacro.
          else if tmpList size minus one is
increment value then
               call function childCekPreMutation
with parameters;
               Set value kelompokData with
incrementMakro in parent,
               Add parent to tmpThreeList array,
          else
               call function childCekPreMutation
with parameters;
          end if;
          Increment Variable increment.
     end for;
End function method.

Function method with name getMutation  and
paramters;
     Assign value hitung as int;
```

```
     Initiate locally es from ExecutorService to
                                          cacheTr
                                          eadPool
                                          ;
     for each mutCan from MutationCandidat in
tmpThreeList Do
          Thread with name thrd to Thread;
               Assign value hitungIn to int from
variable hitung;
               Inner function method with name
run
                    for each keys from Integer
in list of mutCan Do
                         Initiate locally now
to long with Current Time.
                         Assign value to
                         mutationCand in String
                         from call function
                         getByteArrayCrossOver
                         with parameter,
                         Assign value to header
                              from mutationCand
                              first of thousand
                              character;
                         Assign value to mutant
                              from mutationCand
                              second of two
                              hundred character;
                         Assign value to footer
                              from mutationCand
                              third of last
                              character;

                         Initiate locally
strReverse to StringBuilder,
                         Initiate locally
                              reverseMutant to
                              charArray from
                              mutant,
                         for each i from Integer
in list of reverserMutant Do
                              if array of
reverseMutant contain zero then
```

```
                                        change
reverseMutant value to one;
                                   else
                                        change
reverseMutant value to zero;
                                   end if;
                              end for;

                              Initiate locally
strBuild to StringBuilder,
                              Assign value to married
                                   combine between
                                   header, new mutant
                                   and footer,
                              Initiate locally
innerData  to DataContainer.
                              Assign value to each
variables in innerData,
                              call function
inputDataMutation with parameters from variable
dic;z
                              Increment of variable
hitungIn.
                         end for;
                    End inner function method;
               end thread;
               Paremeter es call function execute
               Parameter es call function
awaitTermination with parameters
               Increment Variable counter.
     end for;
     Parameter es call function shutdown.
End function method.

Function method with name childCekPreMutation
with parameter
     Add tmpData from parameter into childData
Array
End function method
```

4. Hasil Analisis

Model *HPCMA2* adalah dengan mengatur semua proses makro berjalan secara sekuensial, proses makro kedua berjalan setelah proses makro pertama dinyatakan selesai dengan sempurna, begitu seterusnya sampai proses makro terakhir.

Tabel 5.4 Hasil Pengujian Model *HPCMA2* di Windows 7

Sps	*Algorithm*	*Macro Process (ms)*				*Total Time (ms)*	*Speed Up* (kali)	*Data Storage*	Jumlah Hasil Data Percobaan			
		1	2	3	4				*Ori.*	*Elts.*	*Swap*	*Mut.*
1	MA	1039191	37019	704345	1485208	3265763	44,80	22.8 GB	7200	140	4900	4900
	HPCMA	17106	26488	19451	9859	72904		22.8 GB	7200	140	4900	4900
2	MA	43562	1537	6355	11210	62666	7,51	237 MB	1800	35	289	289
	HPCMA	4024	2510	1181	632	8347		237 MB	1800	35	289	289
3	MA	417844	14795	68181	117249	618073	29,92	4.8 GB	1800	35	289	289
	HPCMA	4373	8676	6984	626	20659		4.8 GB	1800	35	289	289
4	MA	373987	13013	61202	104230	552432	28,88	4.3 GB	1800	35	289	289
	HPCMA	4088	8099	6316	627	19130		4.3 GB	1800	35	289	289
5	MA	218915	7548	35285	60255	322007	23,32	2.4 GB	1800	35	289	289
	HPCMA	4002	5301	3876	630	13809		2.4 GB	1800	35	289	289
6	MA	456752	16220	272223	513011	1258207	39,75	8.8 GB	3600	70	1225	1225
	HPCMA	8586	11733	8810	2521	31650		8.8 GB	3600	70	1225	1225
7	MA	425300	14891	247455	456929	1144577	39,73	7.8 GB	3600	70	1225	1225
	HPCMA	8193	10501	7582	2532	28808		7.8 GB	3600	70	1225	1225
8	MA	256802	8971	139007	259072	663853	28,23	4.5 GB	3600	70	1225	1225
	HPCMA	8068	7764	5156	2529	23517		4.5 GB	3600	70	1225	1225
9	MA	784634	28012	250508	467833	1530989	36,63	11.4 GB	3600	70	1225	1225
	HPCMA	8260	17460	13566	2515	41801		11.4 GB	3600	70	1225	1225
10	MA	640232	22444	149268	267210	1079155	28,56	8 GB	3600	70	1225	1225
	HPCMA	8000	15211	12061	2514	37786		8 GB	3600	70	1225	1225
11	MA	587155	20791	150504	268047	1026498	29,47	7.6 GB	3600	70	1225	1225
	HPCMA	8459	13488	10363	2520	34830		7.6 GB	3600	70	1225	1225
12	MA	836732	30253	571096	1086747	2524832	51,53	17.2 GB	5400	105	2704	2704

Sps	***Algorithm***	***Macro Process (ms)***				***Total Time (ms)***	***Speed Up* (kali)**	***Data Storage***	**Jumlah Hasil Data Percobaan**			
		1	2	3	4				*Ori.*	*Elts.*	*Swap*	*Mut.*
	HPCMA	12581	17856	12972	5585	48994		17.2 GB	5400	105	2704	2704
13	MA	687131	24844	423271	802183	1937431	41,00	13 GB	5400	105	2704	2704
	HPCMA	12463	17111	12341	5339	47254		13 GB	5400	105	2704	2704
14	MA	1024541	36407	403578	751388	2215919	38,81	15.7 GB	5400	105	2704	2704
	HPCMA	12335	22548	16863	5352	57098		15.7 GB	5400	105	2704	2704
15	MA	641450	22718	397468	750433	1812071	41,14	12.2 GB	5400	105	2704	2704
	HPCMA	12082	15397	10970	5601	44050		12.2 GB	5400	105	2704	2704

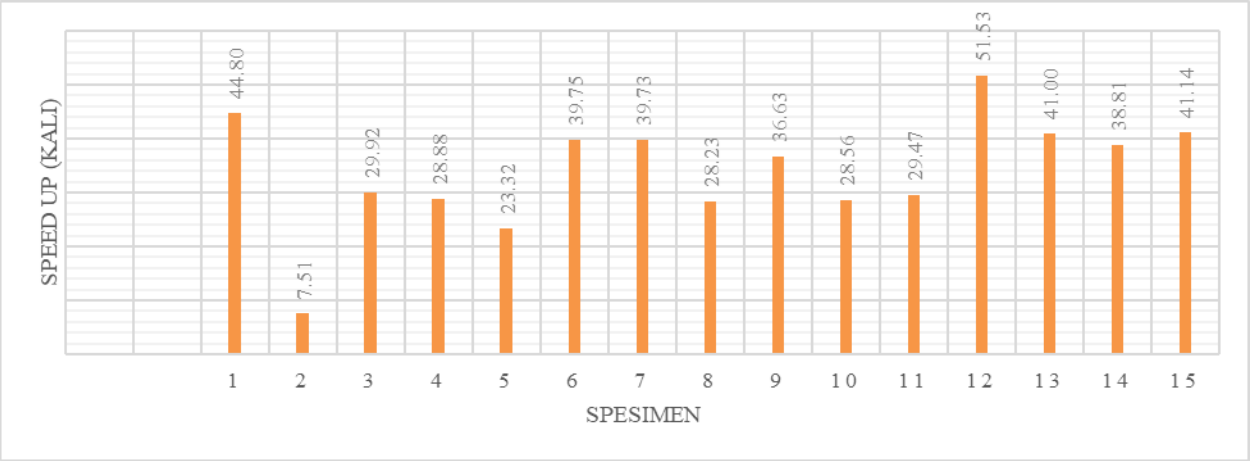
Tabel 5.4 adalah hasil pengujian model *HPCMA2.* Pada spesimen 1, proses makro 1 MA didapat waktu proses sebesar 1039191 milidetik, pada proses makro 2 MA membutuhkan waktu menghasilkan 37019 milidetik, pada makro proses 3 MA membutuhkan waktu proses 704345 milidetik, dan pada makro proses 4 MA membutuhkan waktu proses 1485208 milidetik, sehingga total waktu proses MA adalah sebesar 3265763 milidetik. Pada spesimen 1, proses makro algoritma HPCMA sebesar 17106 milidetik, pada proses makro 2 algoritma HPCMA memerlukan waktu proses sebesar 26488 milidetik, pada proses makro 3 algoritma HPCMA membutuhkan waktu proses 19451 milidetik, sedangkan pada proses makro proses 4 algoritma HPCMA membutuhkan waktu proses sebesar 9859 milidetik, sehingga total waktu proses yang dibutuhkan algoritma HPCMA adalah sebesar 72904 milidetik.

Nilai *speed up* adalah nilai yang didapat dari perbandingan waktu proses MA dengan waktu proses HPCMA setiap proses makro. Nilai *speed up* yang didapat pada spesimen 1 adalah 44,80 kali. Proses yang sama dilakukan pada spesimen 2 sampai dengan spesimen 15 yang hasilnya juga dapat dilihat secara detail pada Tabel 5.4 di atas. Tabel 5.5 adalah daftar *speed up* yang didapatkan oleh *HPCMA2* pada Windows 7.

Tabel 5.5 *Speed Up HPCMA2* di Windows 7

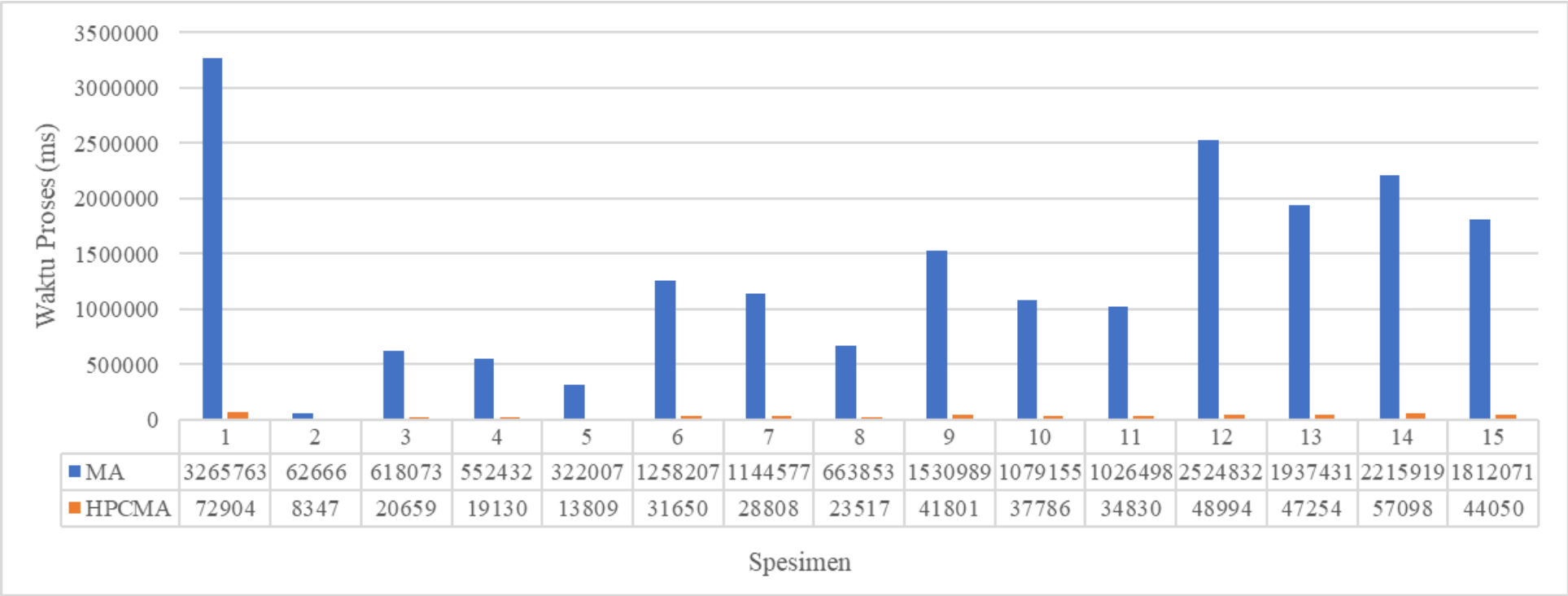
Spesimen	Ukuran Data	*Speed up* (kali)
1	22.8GB	44,80
2	0.237 GB	7,51
3	4.8 GB	29,92
4	4.3 GB	28,88
5	2.4 GB	23,32
6	8.8 GB	39,75
7	7.8 GB	39,73
8	4.5 GB	28,23
9	11.4 GB	36,63
10	8 GB	28,56
11	7.6 GB	29,47
12	17.2 GB	51,53
13	13 GB	41,00
14	15.7 GB	38,81
15	12.2 GB	41,14

Tabel 5.5 di atas adalah daftar ukuran data setiap spesimen, *speed up* atau kecepatan algoritma HPCMA untuk memprosesnya. Visualisasi *speed up* untuk setiap spesimen terdapat pada Gambar 5.6 dibawah ini.



Gambar 5.10 *Speed Up HPCMA2* di Windows 7

Gambar 5.10 adalah *speed up* yang didapatkan oleh model *HPCMA2* untuk memproses data sidik jari pada Windows 7. Pada spesimen 1 *speed up* yang didapatkan adalah 44,80 kali, pada spesimen 2 *speed up* yang didapatkan adalah 7,51 kali, pada spesimen 3 *speed up* yang didapatkan adalah 29,92 kali, serta pada spesimen 4 *speed up* yang didapatkan adalah 28,88 kali. Visualisasi nilai-nilai *speed up* pada spesimen 5 sampai dengan spesimen 15 dapat dilihat juga pada Gambar 5.10 di atas.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MA	3265763	62666	618073	552432	322007	1258207	1144577	663853	1530989	1079155	1026498	2524832	1937431	2215919	1812071
HPCMA	72904	8347	20659	19130	13809	31650	28808	23517	41801	37786	34830	48994	47254	57098	44050

Gambar 5.11 Waktu proses MA dan *HPCMA2* di Windows 7

Gambar 5.11 adalah visualisasi kinerja algoritma memetika dan algoritma HPCMA pada Windows 7. Terlihat waktu proses untuk setiap spesimen dan algoritma HPCMA dengan pengaturan yang benar dapat mengungguli waktu proses algoritma memetika asli. Pada spesimen 1 algoritma memetika membutuhkan waktu proses sebesar 3265763 milidetik sedangkan algoritma HPCMA dengan model *HPCMA2* hanya membutuhkan waktu proses sebesar 72906 milidetik. Demikian juga pada spesimen 2 sampai dengan spesimen 15.

Pengujian model HPCMA2 juga dilakukan pada sistem operasi Windows 10, dan Tabel 5.6 berikut adalah hasil pengujian untuk setiap spesimen.

Tabel 5.6 Hasil Pengujian *HPCMA2* di Windows 10

Sps	Alg.	MA				*Total Time (ms)*	*Speed Up* (kali)	*Data Storage*	Jumlah Hasil Data Percobaan			
		Macro Process (ms)							*Ori.*	*Elts.*	*Swap*	*Mut.*
		1	2	3	4							
1	MA	1184095	40340	868547	1621900	3714882	45,87	22.8 GB	7200	140	4900	4900
	HPCMA	19034	29792	20107	12049	80982		22.8 GB	7200	140	4900	4900
2	MA	46925	1675	7102	12206	67908	6,97	237 MB	1800	35	289	289
	HPCMA	4765	2822	1408	751	9746		237 MB	1800	35	289	289
3	MA	475459	15772	74645	124998	690889	31,35	4.8 GB	1800	35	289	289
	HPCMA	4801	9305	7165	766	22037		4.8 GB	1800	35	289	289
4	MA	416677	13749	65222	108888	604551	28,90	4.3 GB	1800	35	289	289
	HPCMA	5004	8634	6516	765	20919		4.3 GB	1800	35	289	289
5	MA	266665	7937	50570	63525	388697	24,99	2.4 GB	1800	35	289	289
	HPCMA	4745	5922	4119	766	15552		2.4 GB	1800	35	289	289
6	MA	520343	17187	300769	539040	1377355	43,06	8.8 GB	3600	70	1225	1225
	HPCMA	9725	11471	7695	3097	31988		8.8 GB	3600	70	1225	1225
7	MA	450076	15473	275119	484268	1224936	37,97	7.8 GB	3600	70	1225	1225
	HPCMA	9710	11598	7836	3116	32260		7.8 GB	3600	70	1225	1225
8	MA	295250	9648	152158	273889	730945	28,61	4.5 GB	3600	70	1225	1225
	HPCMA	9251	8369	4956	2969	25545		4.5 GB	3600	70	1225	1225
9	MA	878024	30005	273747	492236	1674012	37,21	11.4 GB	3600	70	1225	1225
	HPCMA	9610	18448	13833	3094	44985		11.4 GB	3600	70	1225	1225

Sps	Alg.	MA				*Total Time (ms)*	*Speed Up* (kali)	*Data Storage*	Jumlah Hasil Data Percobaan			
		Macro Process (ms)							*Ori.*	*Elts.*	*Swap*	*Mut.*
		1	2	3	4							
10	MA	725712	24275	162848	284697	1197532	31,45	8 GB	3600	70	1225	1225
	HPCMA	9755	14720	10648	2954	38077		8 GB	3600	70	1225	1225
11	MA	671166	22061	161441	183356	1138024	30,29	7.6 GB	3600	70	1225	1225
	HPCMA	9550	14630	10439	2954	37573		7.6 GB	3600	70	1225	1225
12	MA	918025	31825	609194	1120973	2688017	45,49	17.2 GB	5400	105	2704	2704
	HPCMA	14361	22597	15573	6563	59094		17.2 GB	5400	105	2704	2704
13	MA	751917	25703	450696	837397	2065793	40,21	13 GB	5400	105	2704	2704
	HPCMA	14409	18280	12151	6532	51372		13 GB	5400	105	2704	2704
14	MA	1212867	39155	446292	820903	2519297	40,09	15.7 GB	5400	105	2704	2704
	HPCMA	14508	24304	17479	6547	62838		15.7 GB	5400	105	2704	2704
15	MA	741269	24858	442187	818795	2027109	37,50	12.2 GB	5400	105	2704	2704
	HPCMA	14095	19880	13241	6845	54061		12.2 GB	5400	105	2704	2704

Tabel 5.6 adalah hasil pengujian model *HPCMA2*. Pada spesimen 1, proses makro 1 MA didapat waktu proses sebesar 1184095 milidetik, pada proses makro 2 MA membutuhkan waktu proses sebesar 40340 milidetik, pada proses makro 3 MA membutuhkan waktu proses sebesar 868547 milidetik, sedangkan pada proses makro 4 MA membutuhkan waktu proses sebesar 1621900 milidetik. Sehingga total waktu proses MA pada spesimen 1 adalah 3714882 milidetik.

Pada spesimen 1, proses makro algoritma HPCMA adalah sebesar 19034 milidetik, pada proses makro 2 algoritma HPCMA memerlukan waktu proses sebesar 29792 milidetik, pada proses makro 3 algoritma HPCMA membutuhkan waktu proses 20107

milidetik, sedangkan pada proses makro proses 4 algoritma HPCMA membutuhkan waktu proses sebesar 12049 milidetik, sehingga total waktu proses yang dibutuhkan algoritma HPCMA pada spesimen 1 adalah sebesar 80982 milidetik.

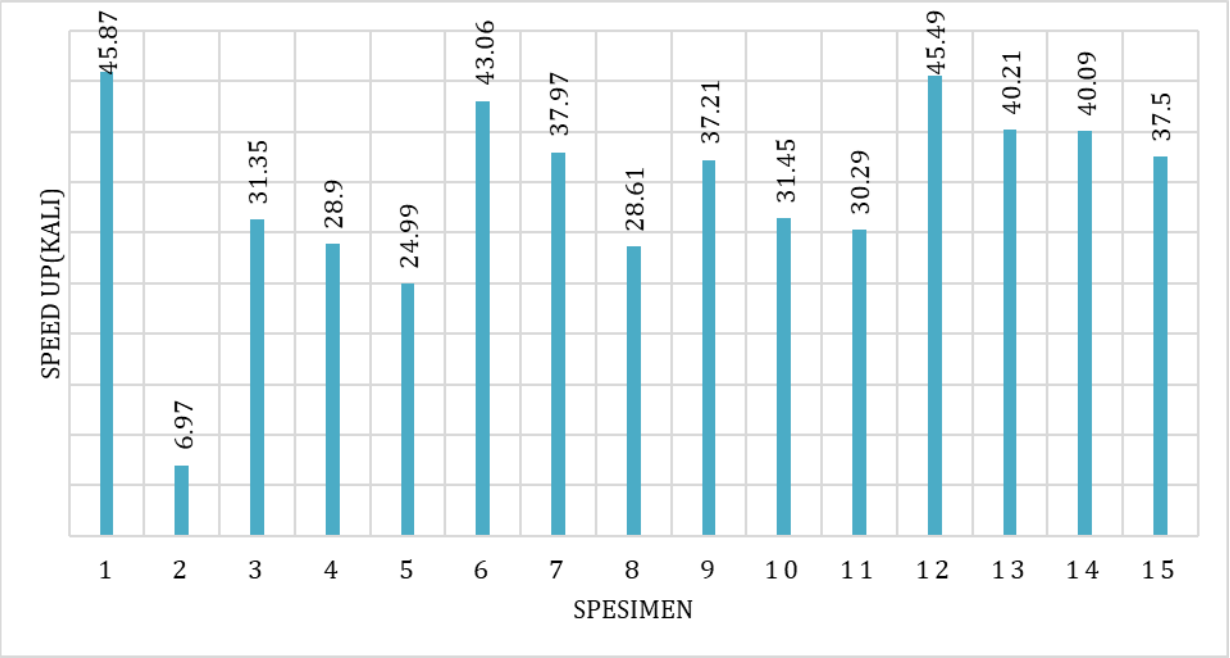
Nilai *speed up* adalah nilai yang didapat dari perbandingan waktu proses MA dengan waktu proses *HPCMA* setiap proses makro. Nilai *speed up* yang didapat pada spesimen 1 adalah 45,87 kali. Proses yang sama dilakukan pada spesimen 2 sampai dengan spesimen 15 yang hasilnya juga dapat dilihat secara detail pada Tabel 5.6 di atas.

Tabel 5.7 adalah daftar nilai *speed up* yang didapatkan oleh model *HPCMA2* pada Windows 10.

Tabel 5.7 *Speed Up HPCMA2* di Windows 10

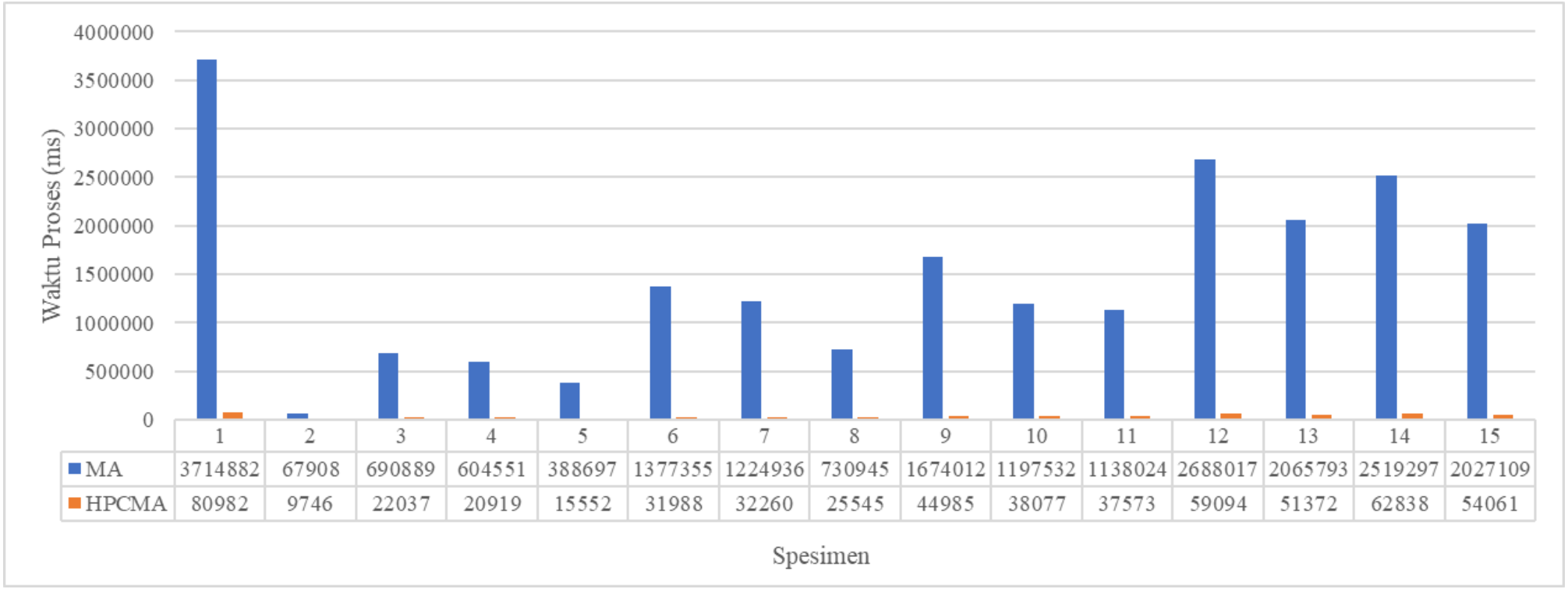
Spesimen	Ukuran Data	Speed Up (kali)
1	22.8 GB	45,87
2	0.237 GB	6,97
3	4.8 GB	31,35
4	4.3 GB	28,90
5	2.4 GB	24,99
6	8.8 GB	43,06
7	7.8 GB	37,97
8	4.5 GB	28,61
9	11.4 GB	37,21
10	8 GB	31,45
11	7.6 GB	30,29
12	17.2 GB	45.49
13	13 GB	40,21
14	15.7 GB	40,09
15	12.2 GB	37,50

Tabel 5.7 diatas adalah daftar ukuran data setiap spesimen dan *speed up* atau kecepatan algoritma *HPCMA* untuk memprosesnya. Visualisasi speedup untuk setiap spesimen terdapat pada Gambar 5.8 dibawah ini.



Gambar 5.12 *Speed Up* HPCMA2 di Windows 10

Gambar 5.12 adalah *speed up* yang didapatkan oleh model *HPCMA2* untuk memproses data sidik jari pada Windows 10. Pada spesimen 1 *speed up* yang didapatkan adalah 45,87 kali, pada spesimen 2 *speed up* yang didapatkan adalah 6,97 kali, pada spesimen 3 *speed up* yang didapatkan adalah 31,35 kali, serta pada spesimen 4 *speed up* yang didapatkan adalah 28,9 kali. Visualisasi nilai-nilai *speed up* pada spesimen 5 sampai dengan spesimen 15 dapat dilihat juga pada Gambar 5.12 di atas.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MA	3714882	67908	690889	604551	388697	1377355	1224936	730945	1674012	1197532	1138024	2688017	2065793	2519297	2027109
HPCMA	80982	9746	22037	20919	15552	31988	32260	25545	44985	38077	37573	59094	51372	62838	54061

Gambar 5.13 Waktu Proses MA dan *HPCMA2* di Windows 10

Gambar 5.13 adalah visualisasi kinerja algoritma memetika dan algoritma *HPCMA* pada Windows 10. Terlihat waktu proses untuk setiap spesimen dan algoritma *HPCMA* dengan pengaturan yang benar dapat mengungguli waktu proses algoritma memetika asli. Pada spesimen 1 algoritma memetika membutuhkan waktu proses sebesar 3714882 milidetik sedangkan algoritma *HPCMA* dengan model *HPCMA2* hanya membutuhkan waktu proses sebesar 80982 milidetik. Demikian juga pada spesimen 2 sampai dengan spesimen 15.

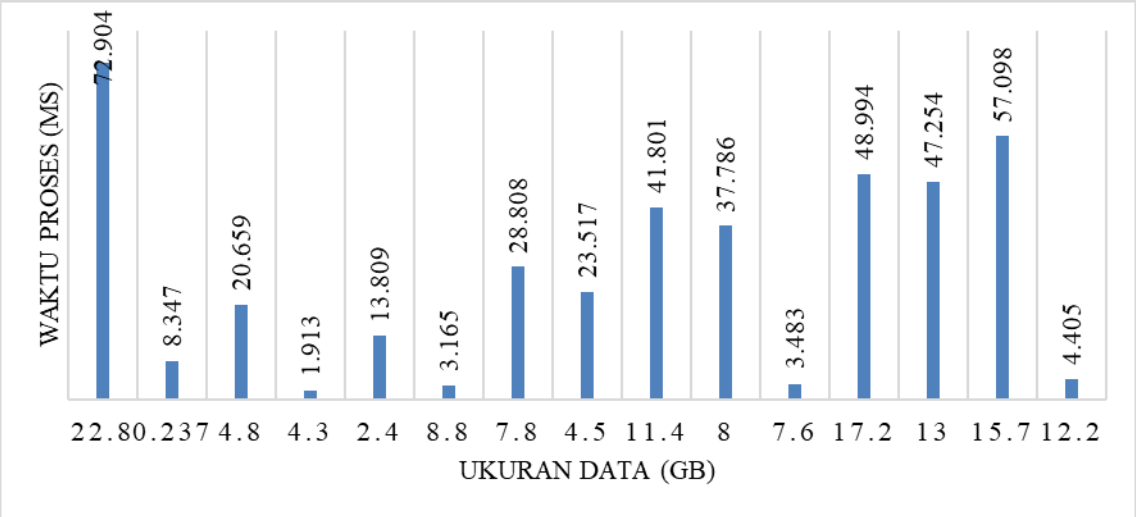
Ukuran data pada setiap spesimen yang dihasilkan oleh model *HPCMA2* pada Windows 7 kemudian dibandingkan dengan waktu prosesnya. Tabel 5.8 dan Tabel 5.9 adalah daftar ukuran data setiap spesimen beserta waktu prosesnya.

Tabel 5.8 Ukuran Data dan waktu proses *HPCMA2* di Windows 7

Spesimen	Jumlah data	Waktu proses (s)
1	22.8GB	72.904
2	0.237GB	8.347
3	4.8GB	20.659
4	4.3GB	19.130
5	2.4GB	13.809
6	8.8GB	31.650
7	7.8GB	28.808
8	4.5GB	23.517
9	11.4GB	41.801
10	8GB	37.786
11	7.6GB	34.830
12	17.2GB	48.994
13	13GB	47.254
14	15.7GB	57.098
15	12.2GB	44.050

Tabel 5.8 merupakan daftar ukuran data dan waktu proses model *HPCMA2* yang berjalan pada sistem operasi Windows 7. Model *HPCMA2* yang berjalan pada Windows 7 menghasilkan data dengan ukuran 22,8 GB pada spesimen 1 dengan waktu proses 72,904 detik, pada spesimen 2 model *HPCMA2* yang berjalan pada Windows 7 menghasilkan data dengan ukuran 0,237 GB yang diproses dalam waktu 8,347 detik, sedangkan pada spesimen 3 model *HPCMA2* yang berjalan pada Windows 7 menghasilkan data dengan ukuran 4,8 GB yang diproses dalam waktu 20,659 detik.

Model *HPCMA2* yang berjalan pada Windows 7 menghasilkan data dengan ukuran 4,3 GB pada spesimen 4 dan diproses dalam waktu 19,130 detik, sedangkan pada spesimen 5 model *HPCMA2* menghasilkan data dengan ukuran 2,4 GB yang diproses dalam waktu 13,809 detik. Ukuran data dan waktu proses pada spesimen 6 sampai dengan spesimen 15 dapat dilihat detail pada Tabel 5.8 diatas. Secara visual, ukuran data dan waktu proses pada Windows 7 dapat dilihat pada Gambar 5.14.



Gambar 5.14 Ukuran Data dan Waktu Proses *HPCMA2* di Windows 7

Gambar 5.14 adalah visualisasi ukuran data dan waktu proses model *HPCMA2* di Windows 7. Terlihat pada gambar bahwa pada data dengan ukuran yang besar waktu proses menjadi semakin lama. Hal yang sebaliknya terjadi pada data dengan ukuran lebih kecil.

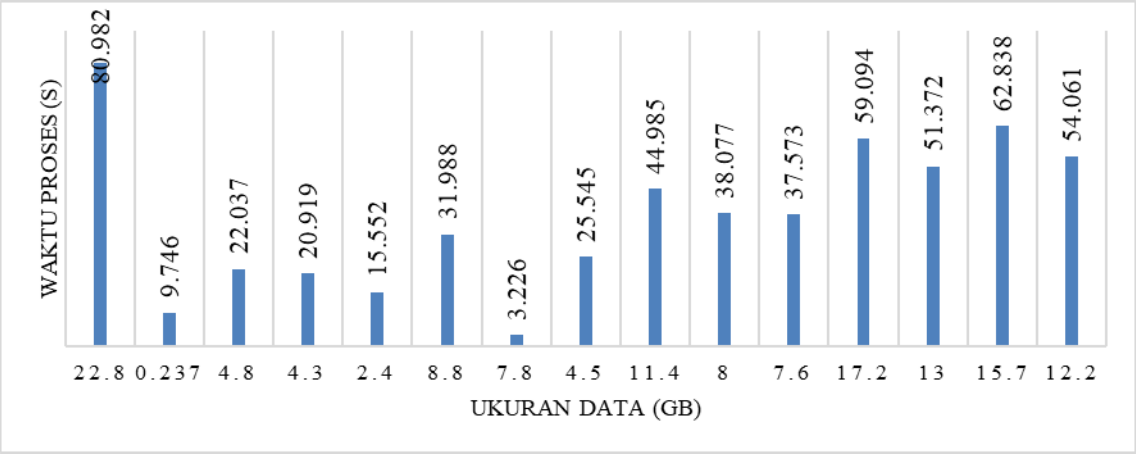
Tabel 5.9 Ukuran Data dan Waktu Proses *HPCMA2* di Windows 10

Spesimen	Ukuran Data (x)	Waktu Proses (y)-second
1	22.8GB	80,982
2	0.237GB	9,746
3	4.8GB	22,037
4	4.3GB	20,919
5	2.4GB	15,552
6	8.8GB	31,988
7	7.8GB	32,260
8	4.5GB	25,545
9	11.4GB	44,985

Spesimen	Ukuran Data (x)	Waktu Proses (y)-second
10	8GB	38,077
11	7.6GB	37,573
12	17.2GB	59,094
13	13GB	51,372
14	15.7GB	62,838
15	12.2GB	54,061

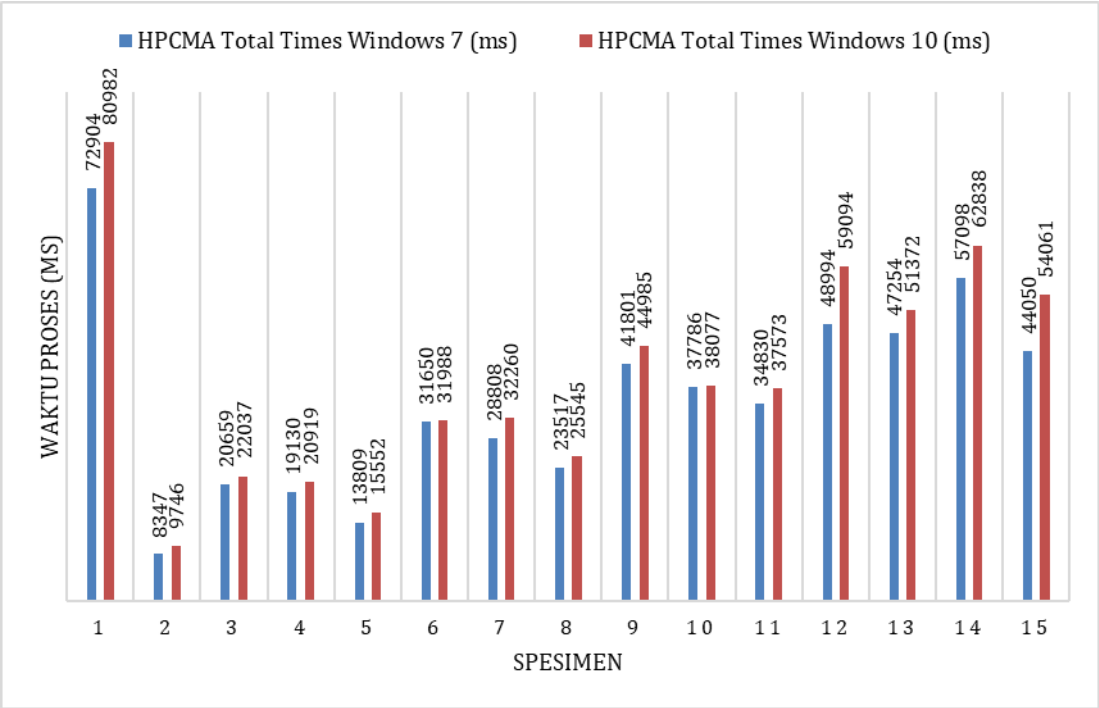
Tabel 5.9 merupakan daftar ukuran data dan waktu proses model *HPCMA2* yang berjalan pada sistem operasi Windows 10. Model *HPCMA2* yang berjalan pada Windows 10 menghasilkan data dengan ukuran 22,8 GB pada spesimen 1 dengan waktu proses 80,982 detik, pada spesimen 2 model *HPCMA2* yang berjalan pada Windows 10 menghasilkan data dengan ukuran 0,237 GB yang diproses dalam waktu 9,746 detik, sedangkan pada spesimen 3 model *HPCMA2* yang berjalan pada Windows 10 menghasilkan data dengan ukuran 4,8 GB yang diproses dalam waktu 22,037 detik.

Model *HPCMA2* yang berjalan pada Windows 10 menghasilkan data dengan ukuran 4,3 GB pada spesimen 4 dan diproses dalam waktu 20,919 detik, sedangkan pada spesimen 5 model *HPCMA2* menghasilkan data dengan ukuran 2,4 GB yang diproses dalam waktu 15,552 detik. Ukuran data dan waktu proses pada spesimen 6 sampai dengan spesimen 15 dapat dilihat detail pada Tabel 5.9 diatas. Secara visual, ukuran data dan waktu proses pada Windows 10 dapat dilihat pada Gambar 5.15.



Gambar 5.15 Ukuran Data dan Waktu Proses *HPCMA2* di Windows 10

Gambar 5.15 adalah visualisasi ukuran data dan waktu proses model *HPCMA2* di Windows 10. Terlihat pada gambar bahwa pada data dengan ukuran yang besar waktu proses menjadi semakin lama. Hal yang sebaliknya terjadi pada data dengan ukuran lebih kecil. Berikut ini adalah gambar yang dapat disimpulkan dari kinerja *HPCMA* dengan model *HPCMA2* secara keseluruhan pada Windows 7 dan Windows 10.



Gambar 5.16 Perbandingan Kinerja *HPCMA* Pada Windows 7 dan Windows 10

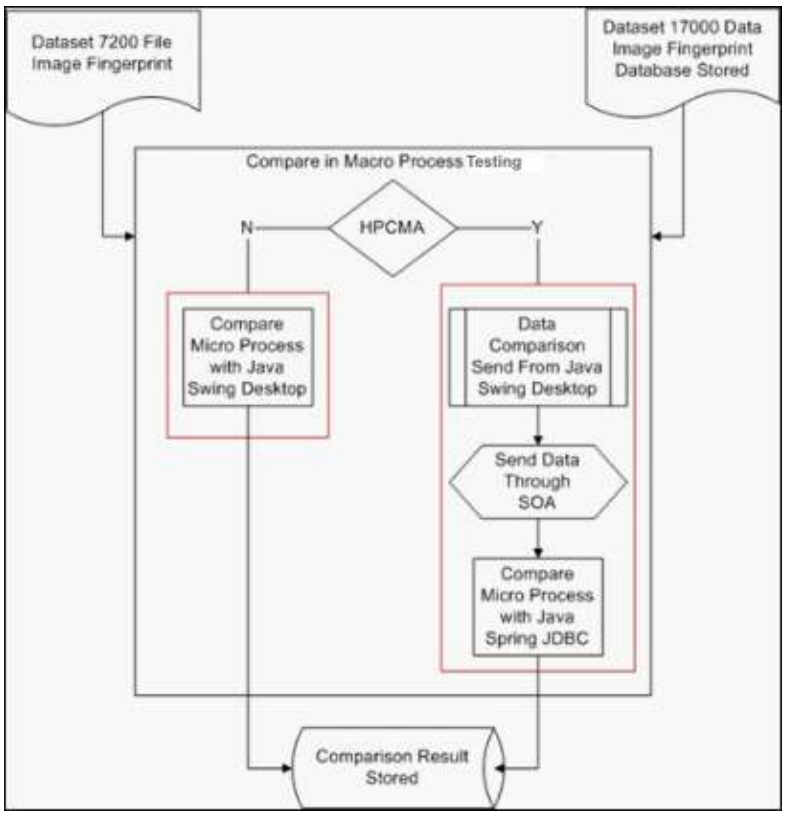
Secara garis besar kinerja algoritma *HPCMA* dengan model *HPCMA2* pada sistem operasi Windows 7 dan Windows 10 tidak jauh berbeda. Pada spesimen 1 waktu proses algoritma *HPCMA* dengan model *HPCMA2* pada sistem operasi Windows 7 adalah 72,904 detik sedangkan pada sistem operasi Windows 10 waktu proses algoritma *HPCMA* dengan model *HPCMA2* adalah 80,982 detik, demikian juga dengan spesimen yang lainnya.

5. Identifikasi Data *Testing* Sidik Jari

Struktur proses identifikasi secara umum seperti yang diuraikan oleh Maltoni, dkk. adalah masukkan pengambilan sidik jari, jalankan ekstraksi fitur, lakukan pencarian sidik jari yang serupa dalam *database,* dan yang terakhir adalah mengembalikan hasilnya sehingga pada buku ini melakukan input satu data gambar sidik jari kemudian melakukan ekstraksi fitur dengan mengelompokkan gambar sidik jari menjadi 4 bagian, yaitu sidik

jari parsial, sidik jari penuh, sidik jari penuh dengan boundary, dan sidik jari penuh tapi kurang jelas. Kemudian sidik jari dikonversi menjadi bentuk biner dan data biner kemudian dikomparasikan dengan data biner yang tersimpan dalam *database.* Kemudian untuk nilai similarity 100%, 90%, 80%, 70%, dan 60% diproses berdasarkan potongan data binary yang didapatkan dari file yang disubmit. Pada proses pengembalian hasil akan ditampilkan jumlah gambar yang ditemukan sesuai dengan *similarity*-nya dan juga waktu prosesnya.

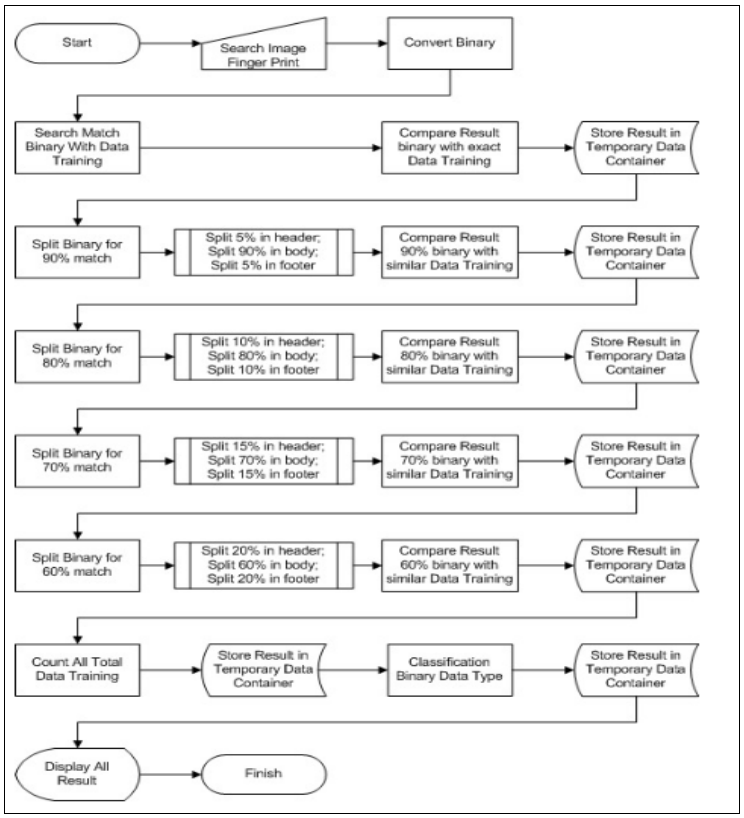
Identifikasi data testing sidik jari dilakukan dengan pemisahan *logic coding* antara MA dan HPCMA menjadi 2 program yang terpisah, dimana MA masih disimpan di sisi *Java Desktop*, kemudian HPCMA dikeluarkan menjadi aplikasi terpisah, namun aplikasi terpisah itu harus berkomunikasi dengan aplikasi utama *Java Desktop* tersebut, maka dibangun jembatan untuk menghubungkan kedua aplikasi tersebut menggunakan salah satu prinsip *Service Oriented Architecture* yaitu *Web Service* dengan protokol data menggunakan *Javascript Object Notation* atau *JSON*.



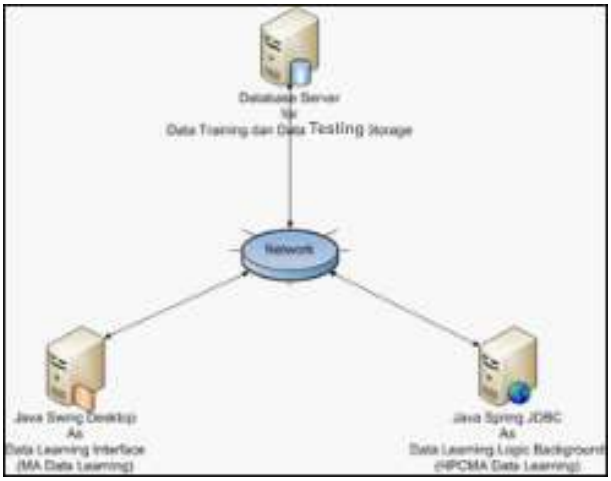
Gambar 5.17 Proses *Testing* Identifikasi Data Sidik Jari

Rincian pada Gambar 5.17 di atas adalah perbandingan *dataset* hasil yang sudah disimpan di *database* yaitu sebanyak 17000, adapun untuk yang terlihat pada gambar

yang dikotak merah adalah proses yang terjadi pada Gambar 5.18. Sedangkan ilustrasi untuk *hardware*-nya terlihat pada Gambar 5.19.



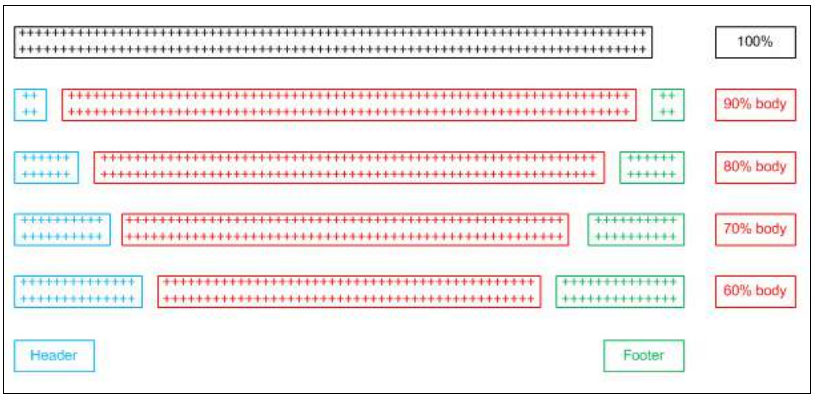
Gambar 5.18 Identifikasi *Data testing* Sidik Jari



Gambar 5.19 Ilustrasi *Hardware Data testing*

Percobaan dilakukan dengan menggunakan 3 komputer dimana masing-masing komputer memiliki fungsi tersendiri. Komputer pertama dengan spesifikasi *hardware*

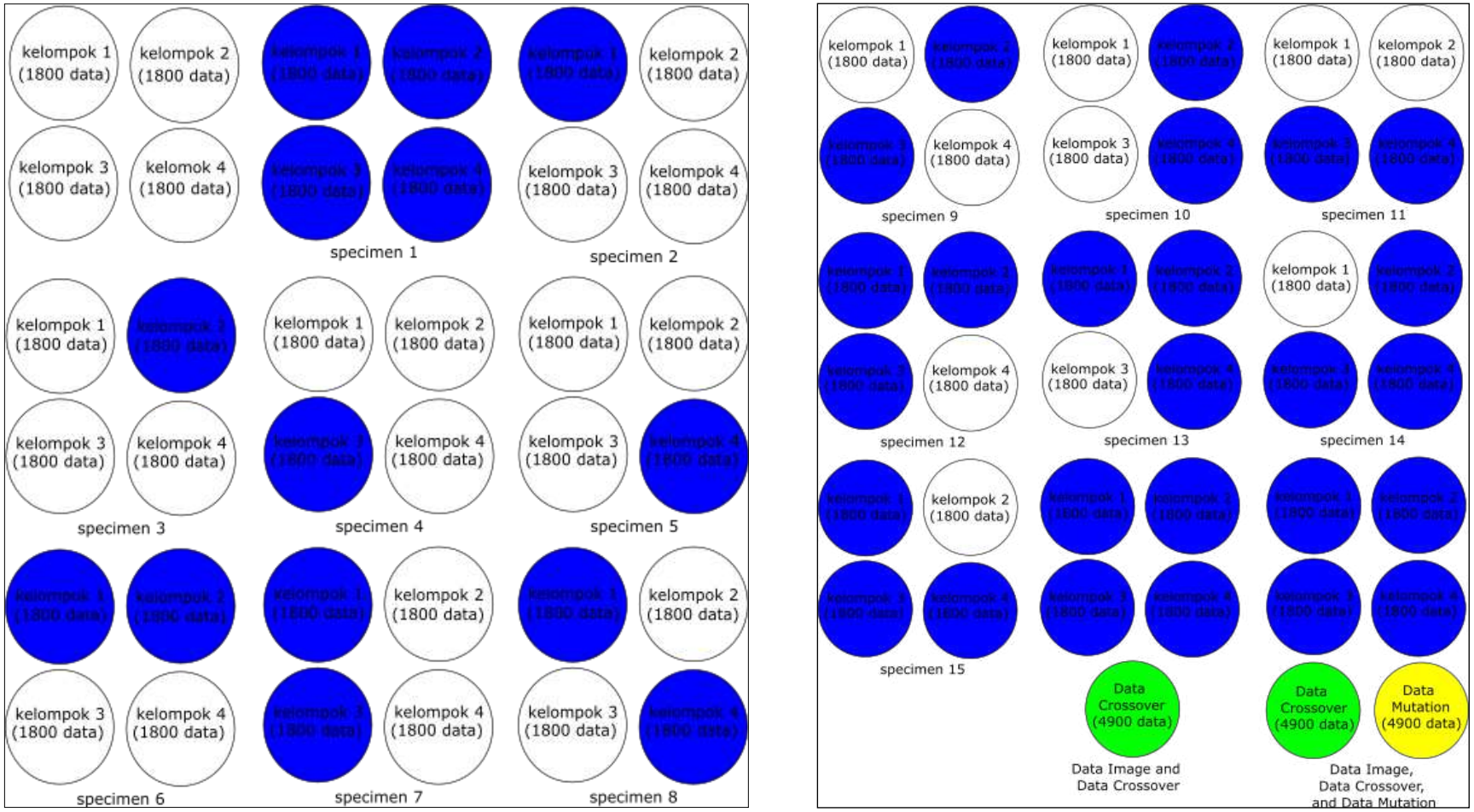
Intel i7 6600U 2.6 GHz 4 core (up to 2.8 GB) RAM 16GB SanDisk X400 M.2 2200 256GB untuk yang SOA atau Java Spring JDBC sebagai *data testing logic background (HPCMA data testing)*. Komputer kedua dengan Intel i5 2540M 2.6 GHz 4 core RAM 16GB HDD Samsung SSD 850 EVO 500GB berfungsi untuk *java swing desktop* sebagai *data testing interface (MA data testing)*, dan komputer ketiga dengan processor Intel i5 2430M 2.4 GHz 4 core RAM 8GB HDD Samsung SSD 860 EVO 250GB yang berfungsi sebagai *database server* untuk *data training* dan *data testing* seperti terlihat pada Gambar 5.19.



Gambar 5.20 Ilustrasi Pembagian *header*, *body*, dan *footer*

Gambar 5.20 adalah proses pencarian data testing yang dipecah-pecah berdasarkan jumlah karakter dari gambar sidik jari yang dikonversi menjadi *binary* kemudian dipecah menjadi 3 bagian yaitu *header*, *body* dan *footer* berdasarkan *match* 100%, 90%, 80%, 70% dan 60%.

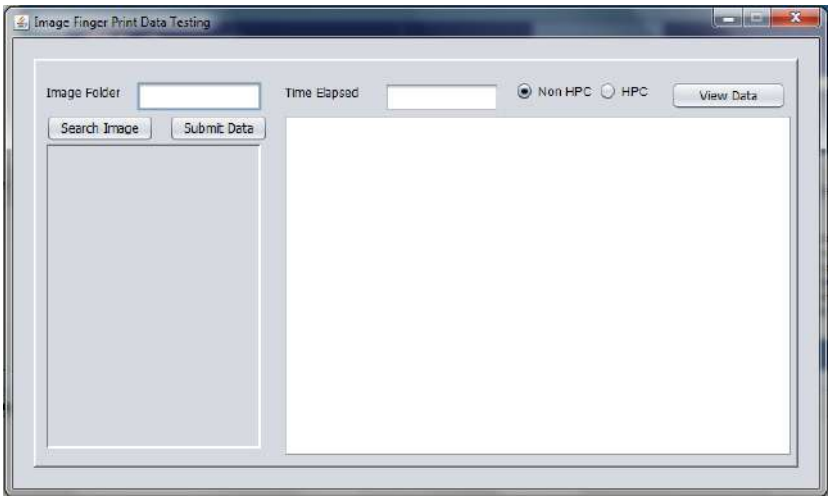
Pengaturan data pada algoritma MA menjadi HPCMA tidak ada perubahan yang signifikan, karena perubahan yang terjadi hanya dari sisi alur proses algoritma MA yang sekuensial menjadi HPCMA yang *parallel*. Pengaturan data *binary* menjadi *Header, Body* dan *Footer* disebabkan konsep *Management Data File* yang terdiri dari *Header* yang berisi metadata dari file tersebut kemudian *Body* data yang merupakan isi informasi dari data tersebut, dan *footer* yang berisi informasi penutup dari file tersebut.



Gambar 5.21 Kelompok Data dan Spesimen

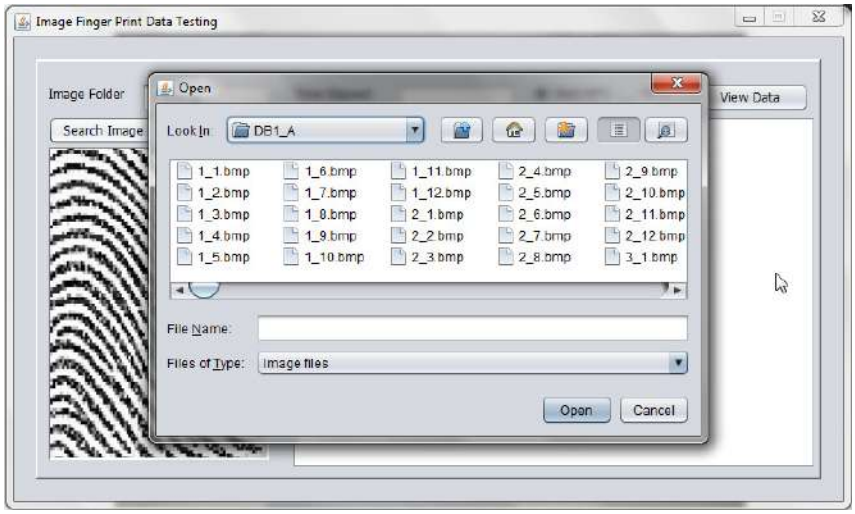
Berdasarkan Gambar 5.21, Kelompok 1, 2, 3 dan Kelompok 4 diilustrasikan dalam satu lingkaran dengan jumlah masing-masing file 1800, sehingga jumlah total data sidik jari dari keempat kelompok tersebut adalah 7200.

Lingkaran berwarna biru adalah kelompok yang digunakan dalam setiap spesimen. Lingkaran berwarna hijau adalah keturunan baru dari proses *crossover* atau persilangan yang berjumlah 4900 file, sedangkan lingkaran berwarna kuning adalah keturunan baru hasil dari proses mutasi yang berjumlah 4900 file. Sehingga total data training yang terbentuk adalah 17000 data.



Gambar 5.22 Tampilan Aplikasi *Data Testing*

Gambar 5.22 adalah tampilan aplikasi ketika pertama kali dibuka. Terdapat pencarian data gambar atau *Search Image* dan *Submit Data* untuk memproses identifikasi, serta pilihan penggunaan *Non-HPC* untuk menggunakan algoritma memetika dan *HPC* untuk menggunakan algoritma HPCMA.



Gambar 5.23 Memilih Gambar Sidik Jari

Gambar 5.23 adalah tampilan ketika menu *Search Image* di klik. Pengguna dapat memilih data gambar sidik jari yang terdapat dalam folder untuk proses identifikasi.

Gambar 5.24 Tampilan Sidik Jari Terpilih

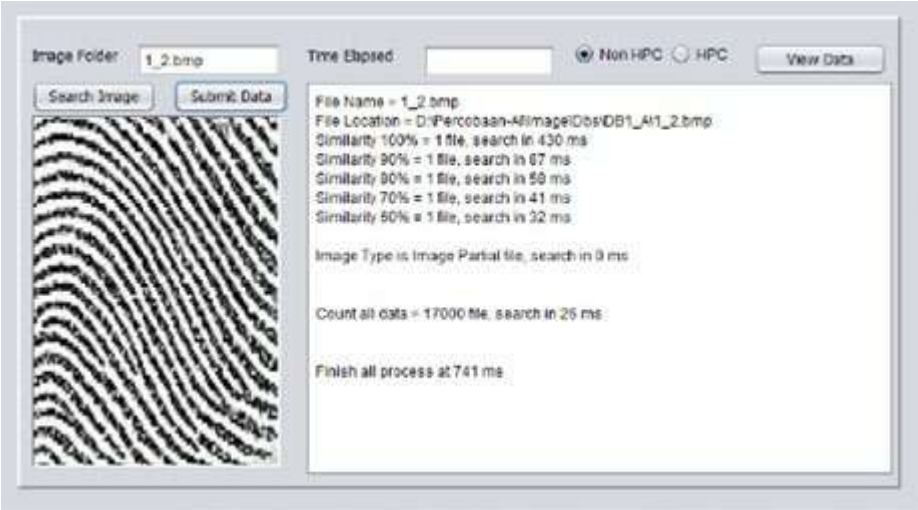
Gambar 5.24 adalah gambar sidik jari yang telah dipilih untuk dilakukan proses identifikasi. Selanjutnya pengguna dapat memilih tipe proses yang akan digunakan, *Non-HPC* dan *HPC*.

Gambar 5.25 Menjalankan Program dengan MA

Gambar 5.25 adalah tampilan ketika menu *Submit Data* di klik. Muncul konfirmasi apakah akan menjalankan program atau tidak. Gambar 5.26 adalah hasil pemrosesan menggunakan mode *Non-HPC.*

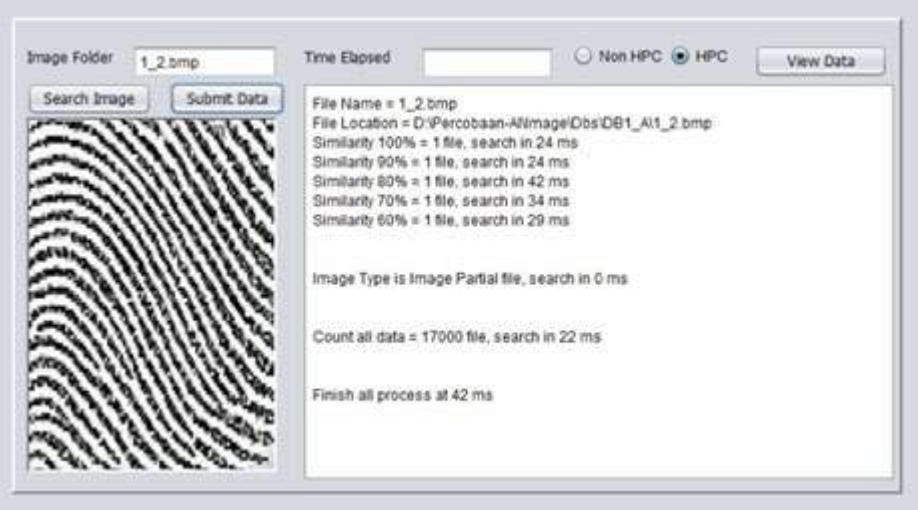
Gambar 5.26 Menjalankan Program dengan *HPCMA*

Gambar 5.26 adalah tampilan ketika menu *Submit Data* di klik. Muncul konfirmasi apakah akan menjalankan program atau tidak. Gambar 5.27 adalah hasil pemrosesan menggunakan *HPC Mode*.



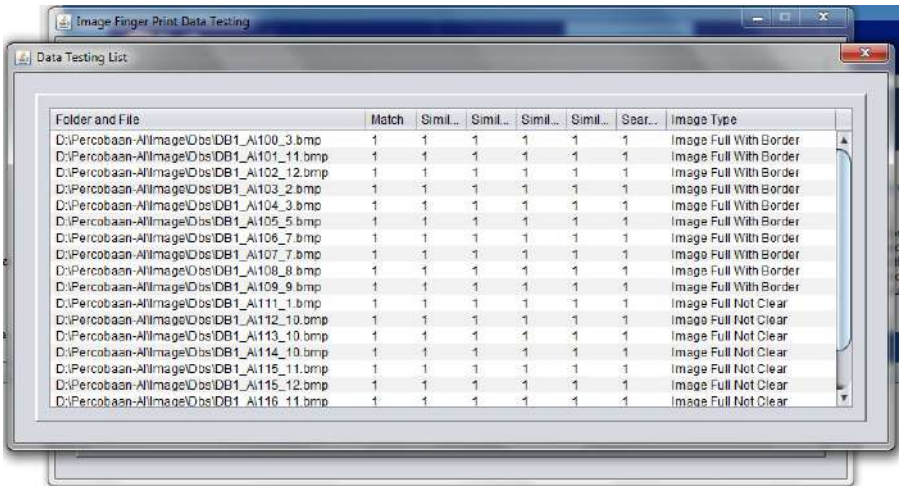
Gambar 5.27 Hasil Proses *Testing* dengan MA

Gambar 5.23 adalah tampilah hasil proses *testing* dengan mode Non-HPC atau hanya algoritma memetika. Secara detil ditampilkan semua hasil *similarity* beserta waktu pencariannya, dan total data yang ada.



Gambar 5.28 Hasil Proses *Testing* dengan *HPCMA*

Gambar 5.28 adalah tampilan hasil proses *testing* dengan *HPC Mode* atau algoritma *HPCMA*. Secara detil ditampilkan semua hasil *similarity* beserta waktu pencariannya, dan total data yang ada.



Gambar 5.29 Data *Testing List*

Gambar 5.29 adalah tampilan daftar data *testing* beserta keterangan tipe gambarnya. Hasil percobaan *data testing* ditampilkan pada Tabel 5.10 berikut:

Tabel 5.10 Hasil Percobaan Data *testing* MA dan Data *testing* HPCMA

Sps.	Nama Folder	Sample File	Algoritma Memetika (MA)							Algoritma HPCMA							*Speed up* HPCMA (kali)				
			Waktu Proses (milidetik)					Jumlah file ditemukan	Total waktu (mildetik)	Waktu Proses (milidetik)					Jumlah file ditemukan	Waktu Total (milidetik)	*Similarity*				
			100%	90%	80%	70%	60%			100%	90%	80%	70%	60%			100%	90%	80%	70%	60%
1	DB2_A	1_1.bmp	53860	521	463	405	352	1	55601	261	221	197	176	148	1	261	206.3602	2.357466	2.350254	2.301136	2.378378
2	DB1_A	1_2.bmp	244	48	51	44	76	1	463	26	32	30	2	36	1	36	9.384615	1.5	1.7	22	2.111111
3	DB2_A	1_2.bmp	58836	526	446	405	343	1	60556	266	234	213	186	160	1	266	221.188	2.247863	2.093897	2.177419	2.14375
4	DB3_A	1_2.bmp	60093	469	409	360	296	1	61625	210	195	173	152	136	1	210	286.1571	2.405128	2.364162	2.368421	2.176471
5	DB4_A	1_2.bmp	26761	271	252	214	194	1	27692	122	112	102	92	83	1	122	219.3525	2.419643	2.470588	2.326087	2.337349
6	DB2_A	1_3.bmp	58882	506	477	398	352	1	60615	255	238	214	179	163	1	255	230.9098	2.12605	2.228972	2.223464	2.159509
7	DB3_A	1_3.bmp	53516	463	435	379	310	1	55103	221	199	183	165	164	1	221	242.1538	2.326633	2.377049	2.29697	1.890244
8	DB4_A	1_3.bmp	25062	256	235	212	180	1	25945	123	113	102	92	85	1	123	203.7561	2.265487	2.303922	2.304348	2.117647
9	DB3_A	1_3.bmp	53516	463	435	379	310	1	55103	221	199	183	165	164	1	221	242.1538	2.326633	2.377049	2.29697	1.890244
10	DB2_A	1_3.bmp	58882	506	477	398	352	1	60615	255	238	214	179	163	1	255	230.9098	2.12605	2.228972	2.223464	2.159509
11	DB3_A	1_3.bmp	53516	463	435	379	310	1	25945	221	199	183	165	164	1	221	242.1538	2.326633	2.377049	2.29697	1.890244
12	DB2_A	1_4.bmp	58496	526	479	420	353	1	60274	238	223	196	173	148	1	238	245.7815	2.358744	2.443878	2.427746	2.385135
13	DB2_A	1_4.bmp	58496	526	479	420	353	1	60274	238	223	196	173	148	1	238	245.7815	2.358744	2.443878	2.427746	2.385135
14	DB3_A	1_4.bmp	56395	481	413	356	331	1	57976	221	203	183	167	150	1	221	255.181	2.369458	2.256831	2.131737	2.206667
15	DB4_A	1_4.bmp	22756	267	238	206	191	1	23658	137	122	113	93	83	1	137	166.1022	2.188525	2.106195	2.215054	2.301205

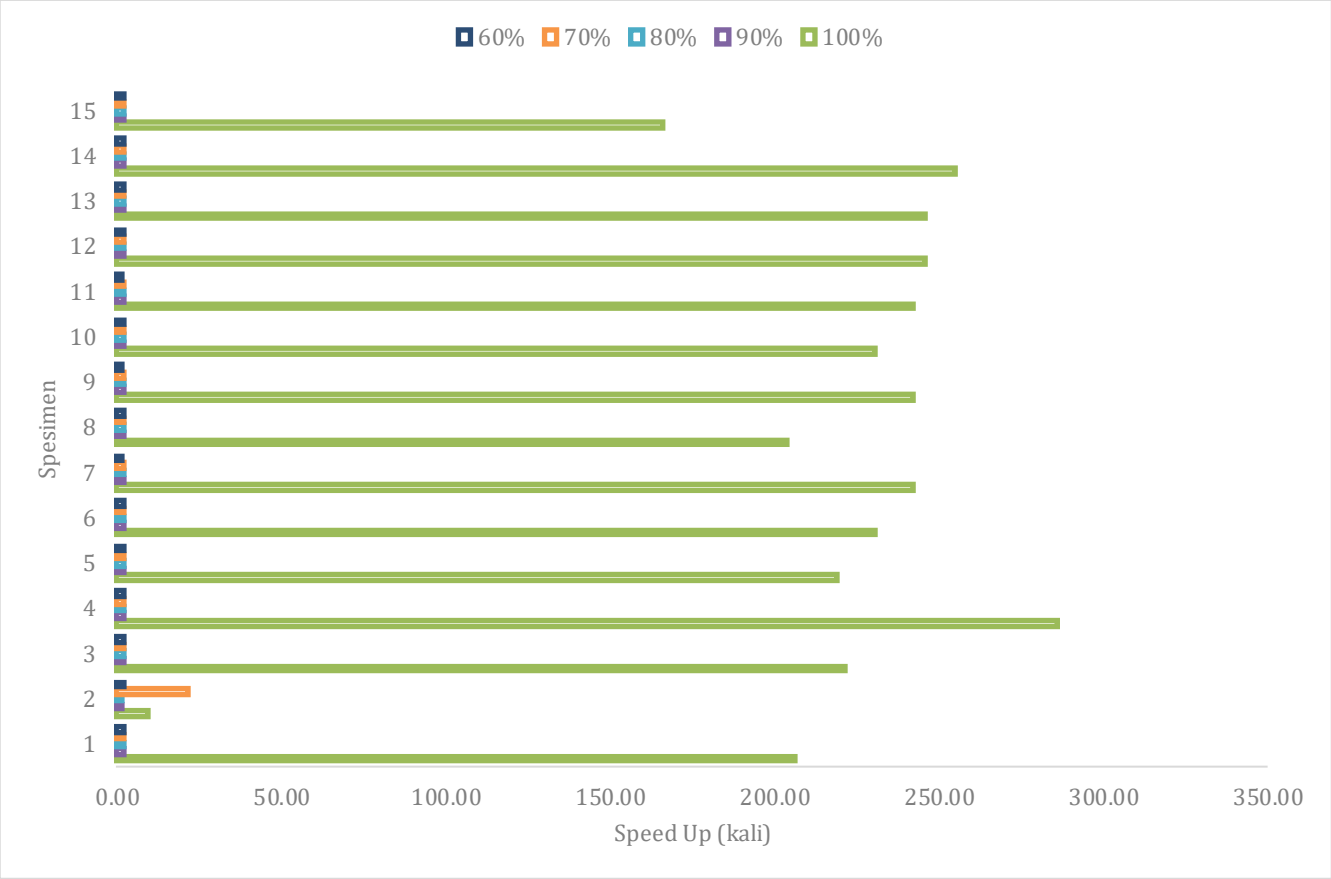
Berdasarkan Tabel 5.10 diatas, hasil percobaan *Data Testing* dengan *MA* dan *Data Testing* dengan algoritma *HPCMA* adalah sebagai berikut:

Percobaan dilakukan dengan mengambil sampel *data testing* secara acak pada setiap spesimen. Pada spesimen 1 diambil *data testing* dari folder DB2_A, dan file contoh *1_1.bmp*, pada algoritma memetika (MA) dengan akurasi 100% ditemukan 1 *file* yang sama dalam waktu 5380 milidetik, dan pada algoritma HPCMA dengan akurasi 100% ditemukan 1 *file* yang sama dalam waktu 261 milidetik. Berdasarkan rumus nomor 1, dimana *speed up* adalah perbandingan antara waktu proses serial (MA) dan waktu proses paralel (HPCMA) maka nilai *speed up* yang didapatkan adalah 206,3602 kali.

Percobaan dengan folder dan file yang sama di spesimen 1, untuk akurasi 90% pada algoritma memetika ditemukan 1 *file* yang sama dalam waktu 291 milidetik, dan pada algoritma HPCMA ditemukan 1 *file* yang sama dalam waktu 221 milidetik, sehingga *speed up* yang didapatkan sebesar 2,357466 milidetik, dan dengan akurasi 80% algoritma memetika dapat menemukan 1 *file* yang sama dalam waktu 463 milidetik, sedangkan algoritma *HPCMA* dapat menemukan 1 *file* yang sama dalam waktu 197 milidetik, sehingga *speed up* yang didapatkan adalah 2,350254 kali.

Percobaan dengan akurasi 70% pada spesimen 1 dengan sampel *file* dan folder yang sama, algoritma memetika daat menemukan 1 *file* yang sama dalam waktu 405 milidetik, sedangkan algoritma *HPCMA* dapat menemukan 1 *file* yang sama dalam waktu 176 milidetik, sehingga *speed up* yang didapatkan adalah 2,301136 kali dan dengan akurasi 60% algoritma memetika dapat menemukan 1 *file* yang sama dalam waktu 352 milidetik, sedangkan algoritma *HPCMA* dapat menemukan 1 *file* yang sama dalam waktu 148 milidetik, sehingga *speed up* yang didapatkan adalah 2,378378 kali.

Hasil percobaan didapatkan bahwa waktu proses algoritma memetika asli lebih lambat dari algoritma HPCMA. *Speed up* juga diukur sebagai salah satu tujuan utama komputasi parallel.



Gambar 5.30 *Speed Up* setiap *Similarity*

Gambar 4.40 adalah visualisasi *speed up* pada setiap spesimen pada masing-masing tingkat *similarity*. Terlihat untuk memproses tingkat similarity 100% algoritma HPCMA membutuhkan kecepatan yang tidak sedikit jika dibandingkan dengan pada saat memproses tingkat *similarity* 90%, 80%, 70%, maupun 60%.

Pada tahun 2001 Le dkk melakukan penelitian dengan menggunakan algoritma genetika untuk merancang pengenal sidik jari dengan fuzzy evolutionary programming. Metode yang diusulkan digunakan untuk mengidentifikasi sidik jari melalui fitur minutie-nya dengan menelusuri tingkat grayscale pada gambar untuk mendeteksi detail sidik jari. Metode ini berjalan baik pada data gambar sidik jari yang jelas tapi tidak pada gambar sidik jari yang kurang jelas atau buram.

Identifikasi sidik jari dengan menggunakan algoritma genetika selanjutnya dilakukan oleh Tan dan Bhanu pada tahun 2003. Algoritma genetika digunakan untuk

mengidentifikasi sidik jari melalui fitur minutiae-nya. Data sidik jari yang digunakan adalah data sidik jari dari NIST-4. Secara detail metode ini berhasil mengatasi distorsi gambar termasuk translasi dan rotasi serta dapat berkerja dengan baik pada gambar dengan kualitas yang kurang baik. Akan tetapi algoritma genetika murni tidak cocok untuk ruang pencarian yang kompleks, sehingga untuk meningkatkan efisiensi dilakukan teknik hibridisasi dengan menambahkan pencarian lokal yang disebut algoritma memetika.

Sheng dkk, pada tahun 2007 mencoba melakukan hibridisasi algoritma genetika dengan menambahkan fitur pencarian local atau *local search* untuk melakukan pencocokan sidik jari. Mereka mengusulkan metode memetic fingerprint matching algorithm (MFMA). Algoritma bekerja mencocokkan sidik jari melalui fitur *minutiae*-nya. Sheng dkk menggunakan data set dari FVC2002 yang terdiri dari 4 kategori data sidik jari yaitu DB1, DB2, DB3, dan DB4. Hasil pengujian menunjukkan waktu untuk menemukan sidik jari yang sama pada DB1 adalah 3.59 detik, sedangkan pada DB2 adalah 3.67 detik, pada DB3 adalah 3.71 detik, dan pada DB4 adalah 3.61 detik, namun metode ini cenderung lambat dan tidak cocok digunakan secara *real-time*. Seiring dengan perkembangan dan penggunaan komputasi *parallel* yang semakin *massive*, implementasi algoritma memetika secara paralel bisa dilakukan untuk meningkatkan efisiensi pencocokan.

Pada tahun 2020, peneliti memulai penelitian dengan menggunakan algoritma memetika pada lingkungan *high performance computing* dengan menggunakan *dataset* gambar sidik jari FVC2006. Berbeda dengan penelitian yang dilakukan oleh Sheng, dkk., yang memanfaatkan algoritma memetika murni dan menggunakan fitur *minutiae* untuk pencocokan sidik jari, penulis menggunakan algoritma HPCMA. Algoritma HPCMA adalah algoritma memetika yang diproses pada lingkungan HPC. Artinya, setiap tahapan pada algoritma memetika, yaitu pencarian lokal, seleksi, persilangan, dan mutasi berjalan pada Mode HPC. Mode HPC yang dimaksud adalah memanfaatkan fitur *multi-threads* yang ada pada prosesor.

Pencarian lokal yang berjalan pada Mode HPC menggunakan 25 *threads* untuk memproses 7200 data. Setiap *thread* memproses 288 data. Seleksi yang dilakukan pada HPC Mode menggunakan 2% dari jumlah total data. Proses Seleksi ini menggunakan *multiple unlimited thread*. Proses Persilangan pada Mode HPC menggunakan 25 *threads*, dan proses mutasi pada Mode HPC menggunakan 25 *threads*.

Hasil penelitian menunjukkan rata-rata waktu proses algoritma HPCMA dengan data gambar ber-format BMP pada *database* FVC2006 adalah 0.2 detik, sedangkan dengan

algoritma HPCMA dengan data gambar ber-format TIFF pada database FVC2002 adalah 0.1 detik. Detail hasil penelitian dan perbandingannya dengan penelitian Sheng, dkk. terdapat pada Tabel 2.2 berikut.

Tabel 5.11 Perbandingan Penelitian

	Sheng, dkk, 2007	Priati, 2020	Priati, 2021
Database	FVC2002	FVC2006	FVC2002
Jumlah Data	320 gambar sidik jari	7200 gambar sidik jari	320 gambar sidik jari
Format file	Tiff	BMP	Tiff
Ukuran Data	40 Mb	956 Mb	40Mb
Waktu Proses	Rata-rata 3,5 detik	Rata-rata 0,2 detik	Rata-rata 0,1 detik
Fitur	Minutiae	Pengkodean Biner	Pengkodean Biner
Algoritma	Memetika	HPCMA	HPCMA

BAB VI
PENUTUP

Algoritma memetika yang diparallelkan pada lingkungan HPC untuk pengenalan data gambar sidik jari. Pengenalan sidik jari dapat dikelompokkan dalam dua bentuk yang berbeda yaitu verifikasi dan identifikasi. Verifikasi adalah membandingkan satu sidik jari dengan satu sidik jari yang lain. Sedangkan identifikasi adalah mencocokkan satu sidik jari inputan dengan data sidik jari yang ada didalam database. Dengan demikian, identifikasi dapat diartikan sebagai perluasan dari verifikasi yang dilakukan dengan membandingkan satu sidik jari ke banyak sidik jari, dan perlu diketahui bahwa identifikasi pada dasarnya lebih komplek dari verifikasi.

Algoritma MA direkayasa menjadi algoritma HPCMA dengan cara sebagai berikut:

1. Memasukan setiap *step* dalam MA ke dalam HPC *Mode*.
2. Modifikasi proses pengolahan data dalam setiap *step* menjadi *parallel* yang sebelumnya *sequential*.
3. Memastikan setiap *step* tidak ada *overlap process*, dengan pengecekan kelengkapan data setiap akan melangkah ke *step* berikutnya.
4. Modifikasi proses *parallel* tidak membuat *processor* dan RAM menjadi penuh, dengan menggunakan rasio proses *thread* dibandingkan dengan jumlah data yang akan diproses.

Rekayasa yang terjadi adalah secara *logic* melakukan transformasi proses MA yang bersifat *sequential* menjadi *parallel*. Penggunaan *multiprocessor* itu akan dialokasikan oleh *thread* ketika kebutuhan sistem masih cukup menggunakan 1 *core* maka akan menggunakan *resource* secara minimal, namun jika dibutuhkan beberapa *core* maka thread akan membuka *core* yang lain untuk digunakan.

Kinerja algoritma diketahui dengan mengukur waktu proses dan *speed up* setiap spesimen. Selain itu, Manacher dalam penelitiannya men-syaratkan beberapa hal dalam menangani sidik jari, diantaranya adalah:

1. Presisi: Akurat dengan tingkat kesalahan yang rendah. Penelitian ini mengukur akurasi dengan menggunakan kemiripan 100%, 90%, 80%, 70%, dan 60%. Semua tingkat akurasi tersebut dapat terjawab.
2. Efisiensi: waktu yang singkat untuk menemukan sidik jari di database. Hal ini dapat dilihat dari kecepatan algoritma dalam menemukan sidik jari yang sesuai.

3. Skalabilitas: kemampuan untuk memproses *database* yang besar dengan waktu yang rasional. Penelitian ini memenuhi syarat skalabilitas karena untuk memproses *database* sidik jari berformat BMP dengan jumlah data 7200 gambar dan *database* sidik jari berformat TIF dengan jumlah data 320 gambar.
4. Fleksibilitas: mampu menyesuaikan perubahan, baik perangkat keras maupun perangkat lunak. Salah satu percobaan dilakukan dengan memanfaatkan virtual box, Windows 7 dan Windows 10 dengan menggunakan ukuran data dan karakteristik data gambar sidik jari yang berbeda yang terbagi menjadi 15 spesimen. Selain itu source code program ini dapat diuji menggunakan data gambar sidik jari berjenis BMP dan TIF.

Hasil pengujian menunjukkan algoritma memetika dapat bekerja secara paralel untuk mendapatkan waktu proses yang cepat dan efisien. Algoritma ini dapat digunakan untuk mengidentifikasi data gambar sidik jari dengan kinerja yang lebih baik dari algoritma memetika asli. Hal ini dikarenakan setiap proses makro pada algoritma *HPCMA* berjalan secara sekuensial atau berurutan sedangkan proses mikro nya berjalan secara *parallel*. Berbeda dengan algoritma memetika murni yang semua prosesnya, baik makro maupun mikro, berjalan secara sekuensial.

DAFTAR PUSTAKA

A. K. Jain, r. Bolle, S. P. (1999). Biometrics: Personal Identification in Networked Security. In *Personal Identification in Networked Society*. https://doi.org/10.1007/978-0-387-32659-7

Agarwal, D., & Merugu, S. (2007). Predictive discrete latent factor models for large scale dyadic data. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. https://doi.org/10.1145/1281192.1281199

Aguilar, J., & Colmenares, A. (1998). Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. *Pattern Analysis and Applications*. https://doi.org/10.1007/BF01238026

Aickelin, U. (2016). Nurse Rostering with Genetic Algorithms. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2832070

Alsmadi, M. K. (2017). An efficient similarity measure for content based image retrieval using memetic algorithm. *Egyptian Journal of Basic and Applied Sciences*, *4*(2), 112–122. https://doi.org/10.1016/j.ejbas.2017.02.004

Aranha, C., & Iba, H. (2009). The Memetic tree-based genetic algorithm and its application to Portfolio Optimization. *Memetic Computing*, *1*(2), 139–151. https://doi.org/10.1007/s12293-009-0010-2

Areibi, S., & Yang, Z. (2004). Effective memetic algorithms for VLSI design = genetic algorithms + local search + multi-level clustering. *Evolutionary Computation*. https://doi.org/10.1162/1063656041774947

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., & Smolinski, B. (1999). Toward a common component architecture for high-performance scientific computing. *IEEE International Symposium on High Performance Distributed Computing, Proceedings*. https://doi.org/10.1109/hpdc.1999.805289

Assiroj, P., Hananto, A. L., Fauzi, A., & Hendric Spits Warnars, H. L. (2019). High Performance Computing (HPC) Implementation: A Survey. *1st 2018 Indonesian Association for Pattern Recognition International Conference, INAPR 2018 - Proceedings*, 213–217. https://doi.org/10.1109/INAPR.2018.8627040

Assiroj, P., Warnars, H. L. H. S., Kosala, R., Ranti, B., Supangat, S., Kistijantoro, A. I., & Abdurrachman, E. (2019). The Form of High-Performance Computing: A Survey. *IOP Conference Series: Materials Science and Engineering*, *662*(5). https://doi.org/10.1088/1757-899X/662/5/052002

Augugliaro, A., Dusonchet, L., & Sanseverino, E. R. (1998). Service restoration in compensated distribution networks using a hybrid genetic algorithm. *Electric Power Systems Research*. https://doi.org/10.1016/s0378-7796(98)00025-x

Bahmann, S., & Kortus, J. (2013). EVO - Evolutionary algorithm for crystal structure prediction. *Computer Physics Communications*.

https://doi.org/10.1016/j.cpc.2013.02.007

Bai, L., Liang, J., Dang, C., & Cao, F. (2011). A novel attribute weighting algorithm for clustering high-dimensional categorical data. *Pattern Recognition*. https://doi.org/10.1016/j.patcog.2011.04.024

Bakly, A. M. El. (2018). *A memetic optimization algorithm for multi- constrained multicast routing in ad hoc networks*. 1–17. https://doi.org/10.1007/s11047-016-9545-6.Funding

Bao, Y., Hu, Z., & Xiong, T. (2013). A PSO and pattern search based memetic algorithm for SVMs parameters optimization. *Neurocomputing*, *117*, 98–106. https://doi.org/10.1016/j.neucom.2013.01.027

Barrientos, R. J., Gómez, J. I., Tenllado, C., Matias, M. P., & Marin, M. (2011). kNN query processing in metric spaces using GPUs. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-642-23400-2_35

Bereta, M. (2019). Baldwin effect and Lamarckian evolution in a memetic algorithm for Euclidean Steiner tree problem. *Memetic Computing*, *11*(1), 35–52. https://doi.org/10.1007/s12293-018-0256-7

Bhanu, B., & Tan, X. (2001). A triplet based approach for indexing of fingerprint database for identification. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/3-540-45344-x_29

Blum, J., Le Dimet, F. X., & Navon, I. M. (2009). Data Assimilation for Geophysical Fluids. In *Handbook of Numerical Analysis*. https://doi.org/10.1016/S1570-8659(08)00209-3

Botzheim, J., Toda, Y., & Kubota, N. (2012). Bacterial memetic algorithm for offline path planning of mobile robots. *Memetic Computing*, *4*(1), 73–86. https://doi.org/10.1007/s12293-012-0076-0

Bozejko, W., & Wodecki, M. (2011). The methodology of parallel memetic algorithms designing. *ICAART 2011 - Proceedings of the 3rd International Conference on Agents and Artificial Intelligence*, *1*, 643–648. https://doi.org/10.5220/0003186006430648

Branke, J. (1998). Creating robust solutions by means of evolutionary algorithms. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/bfb0056855

Buck, A. R., Keller, J. M., & Skubic, M. (2013). A memetic algorithm for matching spatial configurations with the histograms of forces. *IEEE Transactions on Evolutionary Computation*, *17*(4), 588–604. https://doi.org/10.1109/TEVC.2012.2226889

Buell, D., El-Ghazawi, T., Gaj, K., & Kindratenko, V. (2007). Guest editors' introduction: High-performance reconfigurable computing. In *Computer*. https://doi.org/10.1109/MC.2007.91

Burke, E. K., & Smith, A. J. (1999). A Memetic Algorithm to Schedule Planned Maintenance

for the National Grid. *ACM Journal of Experimental Algorithmics*. https://doi.org/10.1145/347792.347801

Burke, Edmund K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: A survey of the state of the art. In *Journal of the Operational Research Society*. https://doi.org/10.1057/jors.2013.71

Cabido, R., Montemayor, A. S., & Pantrigo, J. J. (2012). High performance memetic algorithm particle filter for multiple object tracking on modern GPUs. *Soft Computing*, *16*(2), 217–230. https://doi.org/10.1007/s00500-011-0715-2

Cao, K., Liu, E., & Jain, A. K. (2014). Segmentation and enhancement of latent fingerprints: A coarse to fine ridge structure dictionary. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(9), 1847–1859. https://doi.org/10.1109/TPAMI.2014.2302450

Caponio, A., Cascella, G. L., Neri, F., Salvatore, N., & Sumner, M. (2007). A fast adaptive memetic algorithm for online and offline control design of PMSM drives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*. https://doi.org/10.1109/TSMCB.2006.883271

Caponio, A., & Neri, F. (2012). Memetic algorithms in engineering and design. *Studies in Computational Intelligence*, *379*, 241–260. https://doi.org/10.1007/978-3-642-23247-3_15

Cappelli, R, & Maio, D. (n.d.). *Chapter 9 The State of the Art in Fin g erprint Classification*.

Cappelli, Raffaele, Ferrara, M., & Maltoni, D. (2010). Minutia Cylinder-Code: A new representation and matching technique for fingerprint recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/TPAMI.2010.52

Cappelli, Raffaele, Ferrara, M., & Maltoni, D. (2011). Fingerprint indexing based on minutia cylinder-code. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/TPAMI.2010.228

Chávez, E., & Navarro, G. (2005). A compact space decomposition for effective metric indexing. *Pattern Recognition Letters*. https://doi.org/10.1016/j.patrec.2004.11.014

Chávez, E., Navarro, G., Baeza-Yates, R., & Marroquín, J. L. (2001). Searching in metric spaces. *ACM Computing Surveys*. https://doi.org/10.1145/502807.502808

Chen, L., Fujishiro, I., & Nakajima, K. (2002). Parallel performance optimization of large-scale unstructured data visualization for the earth simulator. *Eurographics Workshop on Parallel Graphics and Visualization*.

Chen, X., Ong, Y. S., Lim, M. H., & Tan, K. C. (2011). A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation*, *15*(5), 591–607. https://doi.org/10.1109/TEVC.2011.2132725

Chi, Y., & Liu, J. (2014). Learning large-scale fuzzy cognitive maps using a hybrid of memetic algorithm and neural network. *IEEE International Conference on Fuzzy*

Systems, 1036–1040. https://doi.org/10.1109/FUZZ-IEEE.2014.6891604

Coello Coello, C. A. (2007). *Evolutionary Algorithms: Basic Concepts and Applications in Biometrics*. https://doi.org/10.1142/9789812770677_0012

Conte, D., Foggia, P., Sansone, C., & Vento, M. (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, *18*(3), 265–298. https://doi.org/10.1142/S0218001404003228

Cordon, O., & Lozano, M. (1995). *A classified r e v i e w on the combination fuzzy l o g i c - g e n e t i c algorithms bibliography: 1989-1995*. 1989–1995.

Costa, D. (1995). An Evolutionary Tabu Search Algorithm And The NHL Scheduling Problem. *INFOR: Information Systems and Operational Research*. https://doi.org/10.1080/03155986.1995.11732279

Dargan, S., & Kumar, M. (2020). A comprehensive survey on the biometric recognition systems based on physiological and behavioral modalities. In *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2019.113114

Datta, A., & Soundaralakshmi, S. (2001). Fast parallel algorithm for distance transforms. *Proceedings - 15th International Parallel and Distributed Processing Symposium, IPDPS 2001*, *33*(5), 1130–1134. https://doi.org/10.1109/IPDPS.2001.925083

De Jong, K., Fogel, L., & Schwefel, H.-P. (1997). Handbook of Evolutionary Computation. In *IOP Publishing Ltd and Oxford University Press* (Vol. 1). https://doi.org/10.1007/978-3-540-69281-2_3

de Lima Corrêa, L., & Dorn, M. (2020). A multi-population memetic algorithm for the 3-D protein structure prediction problem. *Swarm and Evolutionary Computation*, *55*, 100677. https://doi.org/10.1016/j.swevo.2020.100677

Di Gesù, V., Lo Bosco, G., Millonzi, F., & Valenti, C. (2008). A memetic algorithm for binary image reconstruction. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *4958 LNCS*, 384–395. https://doi.org/10.1007/978-3-540-78275-9-34

Dudley, J. T., Schadt, E., Sirota, M., Butte, A. J., & Ashley, E. (2010). Drug discovery in a multidimensional world: Systems, patterns, and networks. In *Journal of Cardiovascular Translational Research*. https://doi.org/10.1007/s12265-010-9214-6

Dworak, K., & Boryczka, U. (2012). *Intelligent Information and Database Systems* (J.-S. Pan, S.-M. Chen, & N. T. Nguyen (eds.); Vol. 7197). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-28490-8

El-emary, I. M. M., & El-kareem, M. M. A. (2008). On the Application of Genetic Algorithms in Finger Prints Registration. *World Applied Sciences Journal 5*, *5*(3), 276–281.

El-Ghazawi, T. (2009). *Plenary talk II Advances in high-performance computing*. https://doi.org/10.1109/icces.2008.4772952

Ergun, H., Van Hertem, D., & Belmans, R. (2012). Transmission system topology optimization for large-scale offshore wind integration. *IEEE Transactions on*

Sustainable Energy. https://doi.org/10.1109/TSTE.2012.2199341

Feng, L., Tan, A. H., Lim, M. H., & Jiang, S. W. (2016). Band selection for hyperspectral images using probabilistic memetic algorithm. *Soft Computing*, *20*(12), 4685–4693. https://doi.org/10.1007/s00500-014-1508-1

Fernandez, E., Grana, M., & Cabello, J. R. (2004). *An instantaneous memetic algorithm for illumination correction*. 1105–1110. https://doi.org/10.1109/cec.2004.1330985

Firdaus, F., Zulfadilla, Z., & Caniago, F. (2021). Research Methodology : Types in the New Perspective. *Manazhim*, *3*(1), 1–16. https://doi.org/10.36088/manazhim.v3i1.903

França, P. M., Mendes, A., & Moscato, P. (2001). A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*. https://doi.org/10.1016/S0377-2217(00)00140-5

França, P., Mendes, A., & Moscato, P. (1999). Memetic Algorithms to Minimize Tardiness on a Single Machine with Sequence-Dependent Setup Times. *Proceedings of the {DSI'} 99 - 5th International Conference of the Decision Sciences Institute*, *May 2014*, 1708–1710.

Galar, M., Derrac, J., Peralta, D., Triguero, I., Paternain, D., Lopez-Molina, C., García, S., Benítez, J. M., Pagola, M., Barrenechea, E., Bustince, H., & Herrera, F. (2015). A survey of fingerprint classification Part I: Taxonomies on feature extraction methods and learning models. *Knowledge-Based Systems*, *81*(February), 76–97. https://doi.org/10.1016/j.knosys.2015.02.008

Galinier, P., Boujbel, Z., & Fernandes, M. C. (2011). An efficient memetic algorithm for the graph partitioning problem. *Annals of Operations Research*, *191*(1), 1–22. https://doi.org/10.1007/s10479-011-0983-3

Galvez, A., & Iglesias, A. (2018). Modified Memetic Self-Adaptive Firefly Algorithm for 2D Fractal Image Reconstruction. *Proceedings - International Computer Software and Applications Conference*, *2*, 165–170. https://doi.org/10.1109/COMPSAC.2018.10222

Garg, P. (2009). A Comparison between Memetic algorithm and Genetic algorithm. *International Journal of Network Security & Its Applications (IJNSA)*.

Ghosh, M., Kundu, T., Ghosh, D., & Sarkar, R. (2019). Feature selection for facial emotion recognition using late hill-climbing based memetic algorithm. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-019-07811-x

Ghosh, M., Malakar, S., Bhowmik, S., Sarkar, R., & Nasipuri, M. (2017). *Memetic Algorithm Based Feature Selection for Handwritten City Name Recognition*. *775*, 599–613. https://doi.org/10.1007/978-981-10-6427-2

Gil-Costa, V., Barrientos, R. J., Marin, M., & Bonacic, C. (2010). Scheduling metric-space queries processing on multi-core processors. *Proceedings of the 18th Euromicro Conference on Parallel, Distributed and Network-Based Processing, PDP 2010*. https://doi.org/10.1109/PDP.2010.94

Godsell, J. (1963). Fingerprint Techniques. *Journal of the Forensic Science Society*. https://doi.org/10.1016/S0015-7368(63)70113-7

Gong, Y. J., Ge, Y. F., Li, J. J., Zhang, J., & Ip, W. H. (2016). A splicing-driven memetic algorithm for reconstructing cross-cut shredded text documents. *Applied Soft Computing Journal*, *45*, 163–172. https://doi.org/10.1016/j.asoc.2016.03.024

Haas, O. C. L., Burnham, K. J., & Mills, J. A. (1998). Optimization of beam orientation in radiotherapy using planar geometry. *Physics in Medicine and Biology*. https://doi.org/10.1088/0031-9155/43/8/013

Hakim, L., R Setya, B., & Qodariyah, N. (2015). PENERAPAN ALGORITMA MEMETIKA PADA PENENTUAN KOMPOSISI PAKAN AYAM PETELUR. *Journal of Visual Languages & Computing*, *11*(3), 55. http://repository.unmuhjember.ac.id/2162/1/ARTIKEL JURNAL.pdf

Haque, M. N., Mathieson, L., & Moscato, P. (2018). A memetic algorithm for community detection by maximising the connected cohesion. *2017 IEEE Symposium Series on Computational Intelligence, SSCI 2017 - Proceedings*, *2018-Janua*, 1–8. https://doi.org/10.1109/SSCI.2017.8285404

Harris, S. P., & Ifeachor, E. C. (1998). Automatic design of frequency sampling filters by hybrid genetic algorithm techniques. *IEEE Transactions on Signal Processing*. https://doi.org/10.1109/78.735305

Hart, W. E. (1994). *Adaptive Global Optimization with Local Search*.

Hasibuan, Z. A. (2007). Metodologi Penelitian Pada Bidang Ilmu Komputer Dan Teknologi Informasi. *Konsep, Teknik, Dan Aplikasi*.

Holland, J. H. (2019). Adaptation in Natural and Artificial Systems. In *Adaptation in Natural and Artificial Systems*. https://doi.org/10.7551/mitpress/1090.001.0001

Hong, J. H., & Cho, S. B. (2006). Efficient huge-scale feature selection with speciated genetic algorithm. *Pattern Recognition Letters*. https://doi.org/10.1016/j.patrec.2005.07.009

Hong, J. H., Min, J. K., Cho, U. K., & Cho, S. B. (2008). Fingerprint classification using one-vs-all support vector machines dynamically ordered with nai{dotless}ve Bayes classifiers. *Pattern Recognition*. https://doi.org/10.1016/j.patcog.2007.07.004

Huang, K. W., Wu, Z. X., Peng, H. W., Tsai, M. C., Hung, Y. C., & Lu, Y. C. (2018). A memetic particle gravitation optimization algorithm for solving image segmentation. *Proceedings of 4th IEEE International Conference on Applied System Innovation 2018, ICASI 2018*, 82–85. https://doi.org/10.1109/ICASI.2018.8394392

Huang, K. W., Wu, Z. X., Peng, H. W., Tsai, M. C., Hung, Y. C., & Lu, Y. C. (2019). Memetic particle gravitation optimization algorithm for solving clustering problems. *IEEE Access*, *7*(1), 80950–80968. https://doi.org/10.1109/ACCESS.2019.2923979

Ichimura, T., & Kuriyama, Y. (1998). Learning of neural networks with parallel hybrid GA using a Royal Road function. *IEEE International Conference on Neural Networks - Conference Proceedings*. https://doi.org/10.1109/ijcnn.1998.685931

Jain, A. K., & Feng, J. (2011). Latent fingerprint matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *33*(1), 88–100.

https://doi.org/10.1109/TPAMI.2010.59

Jain, A. K., Flynn, P., & Ross, A. A. (2007). *Handbook of Biometrics*. http://www.springer.com/computer/image+processing/book/978-0-387-71040-2

Jain, A. K., Hong;, L., & Bolle, R. (1997). Online fingerprint verification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *19*, 302–314. https://doi.org/10.1080/03091900500324194

Jain, A. K., Hong, L., Pankanti, S., & Bolle, R. (1997). An identity-authentication system using fingerprints. *Proceedings of the IEEE*, *85*(9), 1365–1388. https://doi.org/10.1109/5.628674

Jat, S. N., & Shengxiang, Y. (2008). A memetic algorithm for the university course timetabling problem. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*. https://doi.org/10.1109/ICTAI.2008.126

Jayaram, M. A., & Fleyeh, H. (2013). Soft Computing in Biometrics : A Pragmatic Appraisal. *American Journal of Intelligent Systems*, *3*(3), 105–112. https://doi.org/10.5923/j.ajis.20130303.01

Jiao, L., Gong, M., Wang, S., Hou, B., Zheng, Z., & Wu, Q. (2010). Natural and remote sensing image segmentation using memetic computing. *IEEE Computational Intelligence Magazine*. https://doi.org/10.1109/MCI.2010.936307

Karkavitsas, G. V., & Tsihrintzis, G. A. (2011). Automatic music Genre Classification using hybrid Genetic Algorithms. *Smart Innovation, Systems and Technologies*. https://doi.org/10.1007/978-3-642-22158-3_32

Kendall, G., Soubeiga, E., & Cowling, P. (2002). Choice Function and Random Hyperheuristics. *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning*.

Kielarova, S. W. (2017). *Advances in Swarm Intelligence* (Y. Tan, H. Takagi, Y. Shi, & B. Niu (eds.); Vol. 10386). Springer International Publishing. https://doi.org/10.1007/978-3-319-61833-3

Kolen, A., & Pesch, E. (1994). Genetic local search in combinatorial optimization. *Discrete Applied Mathematics*. https://doi.org/10.1016/0166-218X(92)00180-T

Kralev, V. (2009). A genetic and memetic algorithm for solving the university course timetable problem. . *International Journal on Information Technologies and Knowledge (IJ ITK)*, *August*.

Krasnogor, N., Krasnogor, N., & Gustafson, S. (2002). Toward Truly "Memetic" Memetic Algorithms: discussion and proofs of concept. *ADVANCES IN NATURE-INSPIRED COMPUTATION: THE PPSN VII WORKSHOPS. PEDAL (PARALLEL, EMERGENT AND DISTRIBUTED ARCHITECTURES LAB). UNIVERSITY OF READING. ISBN 0-9543481-0-9. ICALP.TEX; 9/12/2003; 16:52; P.21 22 NATALIO KRASNOGOR, STEVEN GUSTAFSON*.

Kumar, A., & Kwong, C. (2013). Towards contactless, low-cost and accurate 3D fingerprint identification. *Proceedings of the IEEE Computer Society Conference on Computer*

Vision and Pattern Recognition. https://doi.org/10.1109/CVPR.2013.441

Kumar, D., Kumar, S., & Rai, C. S. (2009). Feature selection for face recognition: a memetic algorithmic approach. *Journal of Zhejiang University-SCIENCE A*, *10*(8), 1140–1152. https://doi.org/10.1631/jzus.A0820460

Lang, S., Drouvelis, P., Tafaj, E., Bastian, P., & Sakmann, B. (2011). Fast extraction of neuron morphologies from large-scale SBFSEM image stacks. *Journal of Computational Neuroscience*. https://doi.org/10.1007/s10827-011-0316-1

Langkos, S. (2014). Recearch Methodology: Data collection method and research tools. *ResearchGate*, *September 2014*. https://doi.org/10.13140/2.1.3023.1369

Lastra, M., Molina, D., & Benítez, J. M. (2015). A high performance memetic algorithm for extremely high-dimensional problems. *Information Sciences*, *293*, 35–58. https://doi.org/10.1016/j.ins.2014.09.018

Le, H. H., Nguyen, N. H., & Nguyen, T. T. (2016). Exploiting GPU for large scale fingerprint identification. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-662-49381-6_66

Lee, H. C., & Gaensslen, R. . (2003). *Advances in Fingerprint Technology* (2nd ed., Issue 1). CRC Press. https://doi.org/10.16309/j.cnki.issn.1007-1776.2003.03.004

Li, Y., Hu, J., & Jia, Y. (2014). Automatic SAR image enhancement based on nonsubsampled contourlet transform and memetic algorithm. *Neurocomputing*, *134*, 70–78. https://doi.org/10.1016/j.neucom.2013.03.068

Liang, B., Liu, B., Zhou, F., Guo, B., Xu, X., Kang, J., Li, J., & Liu, W. (2015). *World Congress on Medical Physics and Biomedical Engineering, June 7-12, 2015, Toronto, Canada* (D. A. Jaffray (ed.); Vol. 51). Springer International Publishing. https://doi.org/10.1007/978-3-319-19387-8

Lin, J. Y., & Chen, Y. P. (2011). Analysis on the collaboration between global search and local search in memetic computation. *IEEE Transactions on Evolutionary Computation*, *15*(5), 608–623. https://doi.org/10.1109/TEVC.2011.2150754

Lu, Y., Wang, S., Li, S., & Zhou, C. (2011). Particle swarm optimizer for variable weighting in clustering high-dimensional data. *Machine Learning*. https://doi.org/10.1007/s10994-009-5154-2

Madhavi, K. V., Tamilkodi, R., & Sudha, K. J. (2016). An Innovative Method for Retrieving Relevant Images by Getting the Top-ranked Images First Using Interactive Genetic Algorithm. *Procedia Computer Science*. https://doi.org/10.1016/j.procs.2016.03.033

Majdi, M., Abdullah, S., & Jaddi, N. S. (2017). *Fuzzy Population-Based Meta-Heuristic Approaches for Attribute Reduction in Rough Set Theory*. *April*.

Maltoni, D., Maio, D., Jain, A. K., & Prabhakar, S. (2009). *Handbook of Fingerprint Recognition* (2nd ed.). Springer London. https://doi.org/https://doi.org/10.1007/978-1-84882-254-2

Manacher, G. K. (1967). Production and Stabilization of Real-Time Task Schedules. *Journal of the ACM (JACM)*. https://doi.org/10.1145/321406.321408

Manosij Ghosh, Samir Malakar, Showmik Bhowmik, R. S. and M. N. (2005). *Feature Selection for Handwritten Word Recognition Using Memetic Algorithm Manosij* (Vol. 3644, Issue November). Springer Singapore. https://doi.org/10.1007/11538059

Marin, M., Gil-Costa, V., Bonacic, C., Baeza-Yates, R., & Scherson, I. D. (2010). Sync/Async parallel search for the efficient design and construction of web search engines. *Parallel Computing*. https://doi.org/10.1016/j.parco.2010.02.001

Marksteiner, P. (1996). *Computer Physics Communications H i g h - p e r f o r m a n c e c o m p u t i n g - an o v e r v i e w*. *97*, 16–35.

Matsui, T., Katagiri, Y., Katagiri, H., & Kato, K. (2015). Automatic feature point selection through hybrid metaheauristics based on tabu search and memetic algorithm for augmented reality. *Procedia Computer Science*, *60*(1), 1120–1127. https://doi.org/10.1016/j.procs.2015.08.160

Merz, P., & Freisleben, B. (1999). Fitness Landscapes and Memetic Algorithm Design. *Electrical Engineering*.

Merz, P., & Zell, A. (2002). Clustering gene expression profiles with memetic algorithms. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/3-540-45712-7_78

Montazeri, M., & Kerman, I. (n.d.). *Memetic Algorithm Image Enhancement for Preserving Mean*. 1–12.

Moscato, P., Mendes, A., & Berretta, R. (2007). Benchmarking a memetic algorithm for ordering microarray data. *BioSystems*, *88*(1–2), 56–75. https://doi.org/10.1016/j.biosystems.2006.04.005

Moscato, P. (1989). *On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms*. *C3P 826*. citeseer.ist.psu.edu/moscato89evolution.html

Moscato, Pablo. (1999). Memetic Algorithms: A Short Introduction. In *New Ideas in Optimization*.

Mu, C., Xie, J., Liu, R., & Jiao, L. (2014). A memetic algorithm using local structural information for detecting community structure in complex networks. *Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014*, *2013*, 680–686. https://doi.org/10.1109/CEC.2014.6900336

Nagy, B., & Moisi, E. V. (2016). Memetic algorithms for reconstruction of binary images on triangular grids with 3 and 6 projections. *Applied Soft Computing Journal*, *3*. https://doi.org/10.1016/j.asoc.2016.10.014

Navarro, G., & Uribe-Paredes, R. (2011). Fully dynamic metric access methods based on hyperplane partitioning. *Information Systems*. https://doi.org/10.1016/j.is.2011.01.002

Naveen, N., & Rao, M. C. (2013). *Multi-disciplinary Trends in Artificial Intelligence*. *8271*, 153–161. https://doi.org/10.1007/978-3-642-44949-9

Nguyen, K., Lu, T., Le, T., & Tran, N. (2011). *Memetic Algorithm for a University Course Timetabling Problem*. 67–71.

Niu, D., Wang, Y., & Wu, D. D. (2010). Power load forecasting using support vector machine and ant colony optimization. *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2009.08.019

Ong, Y.-S., Lim, M., & Chen, X. (2010). Memetic Computation—Past, Present & Future [Research Frontier. *IEEE Computational Intelligence Magazine*. https://doi.org/10.1109/mci.2010.936309

Ong, Y. S., Lim, M. H., Zhu, N., & Wong, K. W. (2006). Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *36*(1), 141–152. https://doi.org/10.1109/TSMCB.2005.856143

Özcan, E. (2006). Memes, self-generation and nurse rostering. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-540-77345-0_6

Özcan, E., & Başaran, C. (2009). A case study of memetic algorithms for constraint optimization. *Soft Computing*. https://doi.org/10.1007/s00500-008-0354-4

Ozcan, E., & Mohan, C. K. (1998). *Steady state memetic algorithm for partial shape matching* (pp. 527–536). https://doi.org/10.1007/BFb0040804

Özcan, E., & Onbaşioğlu, E. (2007). Memetic algorithms for parallel code optimization. *International Journal of Parallel Programming*. https://doi.org/10.1007/s10766-006-0026-x

Pachori, V., Ansari, G., & Chaudhary, N. (2012). Improved performance of advance encryption standard using parallel computing. *International Journal of Engineering Research and Applications (IJERA)*, *2*(1), 967–971.

Pagacz, A., Hu, B., & Raidl, G. R. (2010). A Memetic Algorithm with population management for the generalized minimum vertex-biconnected network problem. *Proceedings - 2nd International Conference on Intelligent Networking and Collaborative Systems, INCOS 2010*, 356–361. https://doi.org/10.1109/INCOS.2010.22

Pankanti, S., Prabhakar, S., & Jain, A. K. (2002). On the individuality of fingerprints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *24*(8), 1010–1025. https://doi.org/10.1109/TPAMI.2002.1023799

Pant, A., Jafri, H., & Kindratenko, V. (2009). Phoenix: A runtime environment for high performance computing on chip multiprocessors. *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2009*. https://doi.org/10.1109/PDP.2009.41

Peralta, D., Triguero, I., Sanchez-Reillo, R., Herrera, F., & Benitez, J. M. (2014). Fast fingerprint identification for large databases. *Pattern Recognition*, *47*(2), 588–602.

https://doi.org/10.1016/j.patcog.2013.08.002

Peralta, Daniel, García, S., Benitez, J. M., & Herrera, F. (2017). Minutiae-based fingerprint matching decomposition: Methodology for big data frameworks. *Information Sciences*, *408*, 198–212. https://doi.org/10.1016/j.ins.2017.05.001

Potti, S., & Pothiraj, S. (2011). GPGPU Implementation of Parallel Memetic Algorithm for VLSI Floorplanning Problem. In *Communications in Computer and Information Science: Vol. 204 CCIS* (pp. 432–441). https://doi.org/10.1007/978-3-642-24043-0_44

Radtke, P. V. W., Wong, T., & Sabourin, R. (2005). *A Multi-objective Memetic Algorithm for Intelligent Feature Extraction* (pp. 767–781). https://doi.org/10.1007/978-3-540-31880-4_53

Ridao, M. A., Camacho, E. F., Riquelme, J., & Toro, M. (2001). An evolutionary and local search algorithm for motion planning of two manipulators. *Journal of Robotic Systems*. https://doi.org/10.1002/rob.1037

Roy, T. K., & Gerber, R. B. (2013). Vibrational self-consistent field calculations for spectroscopy of biological molecules: New algorithmic developments and applications. In *Physical Chemistry Chemical Physics*. https://doi.org/10.1039/c3cp50739d

Ruiz, L. G. B., Capel, M. I., & Pegalajar, M. C. (2019). Parallel memetic algorithm for training recurrent neural networks for the energy efficiency problem. *Applied Soft Computing Journal*, *76*, 356–368. https://doi.org/10.1016/j.asoc.2018.12.028

S, B. H., Samarth, B., Richa, S., & Mayank, V. (2012). Memetic approach for matching sketches with digital face images. *Indraprastha Institute of Information Technology Delhi*, *7*(5), 1–8.

Seidenberg, M. S., & McClelland, J. L. (1989). A Distributed, Developmental Model of Word Recognition and Naming. *Psychological Review*. https://doi.org/10.1037/0033-295X.96.4.523

Shao, Y., Molnar, L. F., Jung, Y., Kussmann, J., Ochsenfeld, C., Brown, S. T., Gilbert, A. T. B., Slipchenko, L. V., Levchenko, S. V., O'Neill, D. P., DiStasio, R. A., Lochan, R. C., Wang, T., Beran, G. J. O., Besley, N. A., Herbert, J. M., Yeh Lin, C., Van Voorhis, T., Hung Chien, S., … Head-Gordon, M. (2006). Advances in methods and algorithms in a modern quantum chemistry program package. In *Physical Chemistry Chemical Physics*. https://doi.org/10.1039/b517914a

Sheng, W., Howells, G., Fairhurst, M., & Deravi, F. (2007). A memetic fingerprint matching algorithm. *IEEE Transactions on Information Forensics and Security*, *2*(3), 402–411. https://doi.org/10.1109/TIFS.2007.902681

Shi, W., Wahba, G., Irizarry, R. A., Bravo, H. C., & Wright, S. J. (2012). The partitioned LASSO-patternsearch algorithm with application to gene expression data. *BMC Bioinformatics*. https://doi.org/10.1186/1471-2105-13-98

Smith, J. E. (2007). Coevolving memetic algorithms: A review and progress report. In *IEEE*

Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics. https://doi.org/10.1109/TSMCB.2006.883273

Stamatakis, A., & Ott, M. (2008). Exploiting fine-grained parallelism in the phylogenetic likelihood function with MPI, Pthreads, and OpenMP: A performance study. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-540-88436-1-36

Sugiyono. (2012). Metode Penelitian Kuantitatif, Kualitatif dan R & D.Bandung:Alfabeta. *Metode Penelitian Kuantitatif, Kualitatif Dan R & D.Bandung:Alfabeta.* https://doi.org/10.1017/CBO9781107415324.004

Suyanto. (2005). Algoritma Genetika Dalam Matlab. In *Algoritma Genetika dalam Matlab, Andi Offset, Yogyakarta, Indonesia.*

Tan, K. C., Lee, T. H., & Khor, E. F. (2001). Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. *IEEE Transactions on Evolutionary Computation*. https://doi.org/10.1109/4235.974840

Tan, X., & Bhanu, B. (2003). A robust two step approach for fingerprint identification. *Pattern Recognition Letters*, *24*(13), 2127–2134. https://doi.org/10.1016/S0167-8655(03)00084-9

Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., & Rossi, T. (2008). An Enhanced Memetic Differential Evolution in Filter Design for Defect Detection in Paper Production. *Evolutionary Computation*, *16*(4), 529–555. https://doi.org/10.1162/evco.2008.16.4.529

Tu Van Le, Ka Yeung Cheung, & Minh Ha Nguyen. (2001). A fingerprint recognizer using fuzzy evolutionary programming. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, *00*(c), 7. https://doi.org/10.1109/HICSS.2001.926322

Ulder, N. L. J., Aarts, E. H. L., Bandelt, H. J., van Laarhoven, P. J. M., & Pesch, E. (1991). Genetic local search algorithms for the traveling salesman problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/BFb0029740

Urselmann, M., Barkmann, S., Sand, G., & Engell, S. (2011). A memetic algorithm for global optimization in chemical process synthesis problems. *IEEE Transactions on Evolutionary Computation*. https://doi.org/10.1109/TEVC.2011.2150753

Varshney, K. R., & Willsky, A. S. (2011). Linear dimensionality reduction for margin-based classification: High-dimensional data and sensor networks. *IEEE Transactions on Signal Processing*. https://doi.org/10.1109/TSP.2011.2123891

Vinoth Kumar, B., Karpagam, G. R., & Zhao, Y. (2019). Evolutionary Algorithm With Memetic Search Capability for Optic Disc Localization in Retinal Fundus Images. In *Intelligent Data Analysis for Biomedical Applications* (Issue 1, pp. 191–207). Elsevier. https://doi.org/10.1016/B978-0-12-815553-0.00009-4

Wan, Y., Zhong, Y., & Ma, A. (2019). Fully Automatic Spectral-Spatial Fuzzy Clustering

Using an Adaptive Multiobjective Memetic Algorithm for Multispectral Imagery. *IEEE Transactions on Geoscience and Remote Sensing*, *57*(4), 2324–2340. https://doi.org/10.1109/TGRS.2018.2872875

Wang, Y., Wang, L., Cheung, Y. M., & Yuen, P. C. (2015). Learning compact binary codes for hash-based fingerprint indexing. *IEEE Transactions on Information Forensics and Security*, *10*(8), 1603–1616. https://doi.org/10.1109/TIFS.2015.2421332

Wehrens, R., Lucasius, C., Buydens, L., & Kateman, G. (1993). HIPS, a hybrid self-adapting expert system for nuclear magnetic resonance spectrum interpretation using genetic algorithms. *Analytica Chimica Acta*. https://doi.org/10.1016/0003-2670(93)80444-P

Welekar, R., & Thakur, N. V. (2019). *An Enhanced Approach to Memetic Algorithm Used for Character Recognition* (Vol. 768). Springer Singapore. https://doi.org/10.1007/978-981-13-0617-4

Winarno.M.E. (2012). Metodologi Penelitian dalam Pendidikan Jasmani. *Center For Human Capacity Development Jakarta, 2004*.

Witt, C. (2004). *Worst-Case and Average-Case Approximations by Simple Randomized Search Heuristics*. *September*. https://doi.org/10.1007/978-3-540-31856-9_4

Yamada, T., & Nakano, R. (1996). Scheduling by genetic local search with multi-step crossover. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/3-540-61723-X_1059

Yang, S., Cheng, K., Wang, M., Xie, D., & Jiao, L. (2013). High resolution range-reflectivity estimation of radar targets via compressive sampling and Memetic Algorithm. *Information Sciences*, *252*, 144–156. https://doi.org/10.1016/j.ins.2013.06.029

Zhang, M., Ma, J., Gong, M., Li, H., & Liu, J. (2017). Memetic algorithm based feature selection for hyperspectral images classification. *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, *2*, 495–502. https://doi.org/10.1109/CEC.2017.7969352

Zhang, Y., & Zhong, Y. (2015). Sub-pixel mapping based on memetic algorithm for hyperspectral imagery. *International Geoscience and Remote Sensing Symposium (IGARSS)*, *2015-Novem*, 393–396. https://doi.org/10.1109/IGARSS.2015.7325783

Zhao, Y., Sheong, F. K., Sun, J., Sander, P., & Huang, X. (2013). A fast parallel clustering algorithm for molecular simulation trajectories. *Journal of Computational Chemistry*. https://doi.org/10.1002/jcc.23110

Zhou, D., Fang, Y., Botzheim, J., Kubota, N., & Liu, H. (2017). Bacterial memetic algorithm based feature selection for surface EMG based hand motion recognition in long-term use. *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016*. https://doi.org/10.1109/SSCI.2016.7850241

Zhu, Z., Jia, S., & Ji, Z. (2010). Affinity propagation based memetic band selection on hyperspectral imagery datasets. *2010 IEEE World Congress on Computational Intelligence, WCCI 2010 - 2010 IEEE Congress on Evolutionary Computation, CEC 2010*.

https://doi.org/10.1109/CEC.2010.5586533

Zhu, Z., Ong, Y. S., & Dash, M. (2007a). Markov blanket-embedded genetic algorithm for gene selection. *Pattern Recognition*. https://doi.org/10.1016/j.patcog.2007.02.007

Zhu, Z., Ong, Y. S., & Dash, M. (2007b). Wrapper-filter feature selection algorithm using a memetic framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*. https://doi.org/10.1109/TSMCB.2006.883267

Zhu, Z., Ong, Y. S., & Zurada, J. M. (2010). Identification of full and partial class relevant genes. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*. https://doi.org/10.1109/TCBB.2008.105

Carey, P. (2015). Supercomputers a Hidden Power Center of Silicon Valley, The San Jose Mercury News, May 7, 2015, http://www.mercurynews.com/business/ci_28071868/supercomputershiddenpowercenter-silicon-valley.

Data from Top500.org, "The List: November 2015," http://www.top500.org/lists/2015/11/

Harmon, A. (2014). HPC Matters: Funding, Collaboration, Innovation. ScienceNode, November 26, 2014, https://sciencenode.org/feature/hpc-matters-funding-collaboration-innovation.php.

IDC. High Performance Computing in the EU: Progress on the Implementation of the European HPC Strategy (Brussels: European Commission DG Communications Networks, Content & Technology, 2015), 17, http://knowledgebase.eirg.eu/documents/243153/246094/High+Performance+Computing+in+the+EU+Progress+on+the+Implementation+of+the+European+HPC+Strategy.pdf/b0adf617-3f50-4a6f-9217- 4e0fbb5edd09.

International Biometrics Group, Biometrics Market and Industry Report 2007-2012.

Johnston, D. (2014). HPC Matters to Our Quality of Life and Prosperity. Scientific Computing, November 11, 2014, http://www.scientificcomputing.com/articles/2014/11/hpc-matters-our-quality-life-andprosperity

Phillips, J. (2007). High Performance Computing with CUDA – Case Study: Molecular Dynamics; Super Computing Workshop.

Pillarrichie, R. & Suyanto. (2012). Algoritma Genetika Dengan Lokal Search Untuk Penjadwalan Kuliah, Skripsi S1, Fakultas Teknik Informatika, Telkom University.

Sahid. (1997) A Study on University Timetabling. Thesis, Department of Mathematics, The University of Queensland Australia.

Senate Energy and Natural Resources Committee (SENRC). Next Frontier in High-Performance Computing to Usher in New Chapter in Scientific Discovery. news release, August 1, 2012, https://www.highbeam.com/doc/1P3-2725052741.html

Shimpi, A.L. (2012). Inside the Titan Supercomputer: 299K AMD x86 Cores and 18.6K

NVIDIA GPUs. Anand Tech, October 31, 2012, http://www.anandtech.com/show/6421/inside-the-titansupercomputer-299k-amd-x86-cores-and-186k-nvidia-gpu-cores

Techopedia. (2016). High-Performance Computing (HPC). (diakses 10 Januari 2020) https://www.techopedia.com/definition/4595/high-performance-computing-hpc.

The National Institute for Computational Sciences (NICS). What is HPC?. (diakses 10 Januari 2020), https://www.nics.tennessee.edu/computing-resources/what-is-hpc.

Bayometric. https://www.bayometric.com/minutiae-based-extraction-fingerprint-recognition/, (diakses 6 Desember 2020)