Penarapan Heuristik Dan Metaheuristik Pada Examination Timetabling Problem

Yusnardo Tendio[1], Helena Angelita Depari[2], dan Ahmad Muklason[3]
[1,2,3]Departemen Sistem Informasi ITS
[1]yusnardotendio@yahoo.com, [2]helenaangelita1407@gmail.com, [3]ahmad.muklason@gmail.com

***Abstrak*—Secara umum penjadwalan ujian dilakukan secara manual dan membutuhkan waktu yang cukup lama. Banyak penelitian yang telah dilakukan untuk menemukan strategi yang paling tepat untuk menyusun jadwal dengan baik. *Hill Climbing, Simulated Annealing, Great Deluge,* dan *Tabu Search* merupakan algoritma yang digunakan dalam penelitian ini. *Simple random* digunakan sebagai metode untuk memilih *low level heuristic* yang digunakan. *Dataset* Carter (Toronto) digunakan sebagai bahan pengujian. Setiap algoritma akan dijalankan sebanyak 200.000 iterasi yang akan dibandingkan. Hasil yang diperoleh menunjukan *Simulated Annealing* dengan menggunakan *Kempe Chain* sebagai *low level heuristic* mampu memberikan hasil yang lebih baik dibandingkan dengan algoritma lainnya. Namun algoritma heuristic lainnya juga mampu bersaing dengan mendominasi pada dataset tertentu.**

***Kata Kunci*—dataset carter (toronto), algoritma hill climbing, algoritma simulated annealing, algoritma great deluge, algoritma tabu search**

I. PENDAHULUAN

Penjadwalan ujian merupakan salah satu masalah yang sering ditemui baik di institusi pendidikan maupun perusahaan swasta ataupun negara. Penjadwalan merupakan salah satu masalah yang masuk dalam kategori optimasi. Optimasi sendiri dapat menyelesaikan masalah yang memiliki pertimbangan batasan atau syarat dalam penyelesaiannya. Optimasi akan menemukan solusi terbaik dengan menemukan fungsi objektifnya. Penelitian ini menggunakan konsep *single objective optimization* dengan tujuan untuk mencari apakah algoritma yang diusulkan dapat mencapai nilai minimum dari *objective function*. Banyak algoritma yang menjanjikan untuk dapat mencari nilai minimum tersebut, sehingga perlu menggunakan beberapa algoritma sehingga dapat menentukan algoritma terbaik untuk menentukan solusi terbaik [1].

Beberapa algoritma yang dipakai dan akan dibandingkan adalah *Hill Climbing, Simulated Annealing, Simple Random Great Deluge, dan Tabu Search.* Metode hill climbing terinspirasi dari langkah-langkah yang dilakukan oleh para pendaki dalam menemukan *camp* mereka yang terletak diatas lereng gunung. Para pendaki akan mencari jalan yang lebih pintas untuk mencapai tujuannya. *Simulated Annealing* merupakan metode metaheuristik untuk mencari solusi yang memadukan interaksi antara prosedur pencarian lokal dan strategi yang lebih tinggi. *Simple Random Great Deluge* adalah suatu prosedur pencarian lokal yang dapat diterapkan untuk membantu permasalahan penjadwalan ujian disuatu universitas. *Tabu Search* adalah salah satu *local search metaheuristik* yang dapat digunakan untuk menyelesaikan masalah optimasi kombinatorial [1].

Keempat algoritma tersebut akan dibandingkan nilai penalty-nya untuk mengetahui algoritma mana yang menghasilkan solusi terbaik dalam kasus ini.

Bagian lain dari penelitian ini akan dijelaskan kedalam beberapa bagian sebagai berikut: bagian II untuk menjelaskan metode, bagian III untuk menjelaskan hasil dan pembahasan, dan bagian IV berisi kesimpulan dari penelitian ini.

II. METODE

A. Deskripsi Problem Domain

Penjadwalan merupakan salah satu masalah yang sering sekali ditemui diberbagai bidang dan pekerjaan. Masalah ini membutuhkan optimasi agar dapat menemukan penyelesaian yang memenuhi syarat dan juga batasan yang ada. Penelitian ini akan mencoba menyelesaikan masalah tersebut. Data set yang dipakai adalah *dataset* Toronto, karena *dataset* tersebut terkenal akan kestabilannya dan juga variabelnya yang sangat sesuai dengan kasus yang terjadi pada dunia nyata. Instansi pada *dataset* Carter dapat dilihat pada Tabel 1.

Tabel 1. Dataset Carter

Dataset	Timeslots	Exams	Students	Conflict Density
Car91	35	682	16925	0.13
Car92	32	543	18419	0.14
Ear83	24	190	1125	0.27
Hec92	18	81	2823	0.42
Kfu93	20	461	5349	0.06
Lse92	18	381	2726	0.06
Pur93	42	2419	30029	0.03
Rye92	23	486	11483	0.07
Sta83	13	139	611	0.47
Tre92	23	261	4360	0.18
Uta92	35	622	21266	0.13
Ute92	10	184	2749	0.08
Yor83	21	181	941	0.29

Exam timetabling problem dalam *dataset* Toronto, berisi *exam* yang akan ditempatkan dalam *slot* waktu tertentu dan penempatannya harus memenuhi constraint yang ada (*soft constraint* dan *hard constraint*). Untuk *soft constraint*, pengalokasian ujian dalam rentang waktu yang cukup panjang sehingga siswa memiliki waktu yang lebih panjang untuk dapat mengerjakan perbaikan dan juga waktu belajar. Untuk hard constraint, tidak boleh ada siswa yang dijadwalkan untuk mengikuti lebih dari satu ujian pada periode waktu yang sama.

Hard constraint merupakan *constraint* yang harus dipenuhi, sementara untuk *soft constraint* sebisa mungkin dipenuhi. Ketika *hard constraint* sudah dipenuhi, lalu digunakan fungsi objektif untuk memperkecil nilai *penalty*-nya. Nilai *penalty* diperoleh dari seberapa besar *soft constraint* yang dilanggar. Rumus fungsi objektif untuk menghitung *penalty* itu adalah:

$$fitness = \frac{(\sum_{s=0}^{4} \omega_s \times N_s)}{S} \quad (1)$$

Jarak kedua *timeslot* tersebut dihitung dengan bobot $\omega_s = 2^s$. Sedangkan N_s merupakan representasi dari jumlah siswa yang melanggar *soft constraint*. Dan S adalah total siswa di dalam penjadwalan. Contohnya seorang siswa memiliki 2 ujian a dan b, dimana 2 ujian tersebut terpisah 3 rentang *timeslot* sehingga bobot penaltynya adalah 2^1, sedangkan N_s merepresentasikan jumlah siswa yang melanggar *soft constraint* pada dua exam tersebut dan S adalah total seluruh siswa. *Dataset* yang digunakan dapat diunduh pada link berikut ini http://www.cs.nott.ac.uk/~pszrq/data.htm [1].

B. Pembentukan Solusi Awal

Untuk menghasilkan inisial solusi yang layak, penulis mengacu pada penelitian yang dilakukan oleh [1]. Penelitian tersebut menggunakan algoritma *Saturation Degree with Backtracking* untuk menghasilkan inisial solusi yang layak. *Saturation degree* menentukan *timeslot* pada *exam* dengan mendahulukan *exam* yang memiliki kemungkinan masuk pada *timeslot* terkecil. Jika tidak ada *timeslot* yang layak untuk sebuah *exam*, maka algoritma *backtracking* akan dilakukan. Algoritma *Saturation Degree with Backtracking* mampu menghasilkan inisial solusi yang layak pada semua *dataset*.

C. Low Level Heuristic

Low level heuristic atau *neighbor search* merupakan salah satu komponen utama untuk mendapatkan solusi optimum. Penulis mengusulkan beberapa *low level heuristic*, yaitu:

1) ***Random Move***, pilih sebuah exam secara acak, lalu pindahkan exam tersebut secara acak dari timeslot asal ke timeslot lain.
2) ***Random Swap Two***, pilih dua exam secara acak, kemudian tukar timeslot dari dua exam tersebut.
3) ***Random Move Two***, pilih dua exam secara acak, lalu pindahkan dua exam tersebut secara acak dari timeslot asal ke timeslot lain.
4) ***Random Swap Three***, pilih tiga exam secara acak, lalu tukar timeslot dari ketiga eam tersebut. Contohnya, exam a, b, c terpilih secara acak, maka timeslot exam b diubah menjadi timeslot exam a, timeslot exam c diubah menjadi timeslot exam b, dan timeslot exam a diubah menjadi timeslot exam c.
5) ***Random Move Three***, sama seperti random move two, tetapi menggunakan tiga exam secara random.
6) ***Kempe Chain***, dua himpunan exam yang tidak konflik satu sama lain, timeslot-nya ditukar. Pada penelitian yang dilakukan oleh [1], menyimpulkan bahwa Kempe Chain merupakan pilihan yang tepat untuk mengoptimalkan solusi dari Examination Timetabling Problem.

D. Hill Climbing

Secara umum, algoritma *Hill Climbing* selalu menerima solusi yang lebih baik dari sebelumnya. *Random move* akan digunakan sebagai *low level heuristic* untuk melakukan pencarian solusi yang lebih baik. Kelemahan dari Hill Climbing adalah solusi yang dihasilkan tidak mampu keluar dari local optimum.

Pada penelitian ini, penulis mengacu pada [2] untuk membuat algoritma *Hill Climbing*. Pada paper tersebut algoritma *Hill Climbing* menggunakan delta evaluasi untuk menghemat waktu. Berikut adalah *pseudo-code* yang digunakan pada penelitian ini.

Algorithm 1: Hill Climbing

```
bestScoreHc ← initialScore;
hcSolution ← initialSolution;
counter ← 0;
maxCounter ← 200000;
while counter < maxcounter do
    generate new solution with LLH Random Move;
    newScore ← evaluate(newSolution);
    if newScore < bestscorehc then
        hcSolution ← newSolution;
        bestScoreHc ← newScore;
    end
    counter + +;
end
```

Gambar 1. Pseudo Code Hill Climbing Yang Digunakan

Hill climbing akan dilakukan sebanyak 200.000 iterasi dan *low level heuristic* yang digunakan adalah *Random Move*.

E. Simulated Annealing

Annealing merupakan proses untuk memanaskan benda padat hingga titik lelehnya dan kemudian dilakukan pendinginan. Algoritma Simulated Annealing pada prinsipnya sama dengan algoritma metaheuristik lainnya. Namun algoritma Simulated Annealing dapat keluar dari solusi *local optimum* dengan memungkinkan penerimaan pada solusi yang lebih buruk. Pada penelitian ini, penulis akan membandingkan algoritma Simulated Annealing

tanpa Kempe Chain sebagai *low level heuristic* dan algoritma Simulated Annealing yang menggunakan Kempe Chain sebagai *low level heuristic*. Pada metode Simulated Annealing ini penulis menggunakan 2 eksperimen berdasarkan *low level heuristic* untuk menguji apakah Kempe Chain mampu menghasilkan solusi lebih baik dibandingkan dengan SA yang tidak menggunakan Kempe Chain sebagai *neighbor search*. SA1 menggunakan *low level heuristic* 1 – 5, sedangkan SA2 menggunakan *low level heuristic* 1, 2, dan 6.

Pada penelitian ini suhu awal yang digunakan adalah 1, dan suhu akhir diatur agar mencapai 100.000 iterasi, selanjutnya suhu akan diulang kembali menjadi suhu awal sebanyak 2 kali yang mengakibatkan total iterasi seluruhnya adalah 200.000. Dalam mengimplementasikan algoritma ini, penulis mengacu pada paper [3]. Berikut adalah pseudo-code yang digunakan.

Algorithm 2: Simulated Annealing Tipe 1

```
saAccepted ← initialSolution;
saBest ← initialSolution;
for i=1 to 2 do
    T0 ← 1;
    T1 ← 2.4e − 322;
    while T0 > T1 do
        generate new solution with LLH Random Move;
        newScore← evaluate(newSolution)
        if random[0, 1] <= 0.2 then
            s ← RandomMove(saAccepted);
        else if random[0, 1] <= 0.4 then
            s ← RandomSwapTwo(saAccepted);
        else if random[0, 1] <= 0.6 then
            s ← RandomMoveTwo(saAccepted);
        else if random[0, 1] <= 0.8 then
            s ← RandomSwapThree(saAccepted);
        else if random[0, 1] <= 1 then
            s ← RandomMoveThree(saAccepted);
        end if
        d ← evaluate(s) − evaluate(saAccepted);
        if d < 0 then
            saAccepted ← s;
        else if random[0, 1] < exp(−d/T0) then
            saAccepted ← s;
        end if
        if evaluate(saAccepted) < evaluate(saBest) then
            saBest ← saAccepted;
        end if
    end
end
```

Gambar 2**.** Pseudo Code Simulated Annealing Yang Digunakan

F. Simple Random Great Deluge

Simple Random Great Deluge adalah suatu prosedur pencarian lokal. Algoritma ini diterapkan untuk membantu menyelesaikan masalah penjadwalan ujian disuatu universitas. Dibutuhkan dua parameter yang mewakili *user requirement*: waktu komputasi yang tersedia dan perkiraan solusi yang dibutuhkan. Metode ini akan menerima solusi yang lebih buruk jika nilai kualitas yang dihasilkan dirasa kurang dari atau sama dengan nilai batas atas yang diberikan.

Pada paper [4] yang menjadi acuan pengerjaan algoritma *Great Deluge* menggunakan batasan apabila sebanyak 50.000 iterasi algoritma *Great Deluge* tidak mampu meningkatkan hasil dari sebelumnya maka iterasi yang dihentikan. Cara ini mampu memaksimalkan solusi yang didapatkan dan mengurangi waktu dalam pencarian solusi. Namun pada penelitian ini, penulis melakukan modifikasi pada batasan tersebut apabila selama 1.000 iterasi tidak memberikan peningkatan hasil maka akan diberikan denda. Apabila denda tersebut mencapai 5 maka dilakukan perubahan pada solusi terbaik saat itu dengan melakukan *low level heuristic Random Move* dan *Random Swap*.

Langkah-langkah lebih jelasnya dapat dilihat pada Gambar 3 berupa pseudo-code yang digunakan pada algoritma *Simple Random Great Deluge*.

Algorithm 3: Great Deluge

```
bestScoreGD ← initialScore;
level ← initialScore;
bestGDSolution ← initialSolution;
gdSolution ← initialSolution;
decreasingLevel ← inputuser;
counter ← 0;
maxCounter ← 200000;
while counter < maxcounter do
    random[0, 1] <= 0.2 s ← RandomMove(gdSolution);
    random[0, 1] <= 0.6 s ← RandomSwapTwo(gdSolution);
    s ← KempeChain(gdSolution);
    evaluate(s) < bestScoreGD bestGDSolution ← s;
    bestScoreGD ← evaluate(s);
    notImprovingCounter ← 0;
    evaluate(s) < level bestGDSolution ← s;
    bestScoreGD ← evaluate(s);
    notImprovingCounter ← 0;
    notImprovingCounter + +;
    notImprovingCounter >= 1000 notImprovingCounter ← 0;
    bestScoreGD ← evaluate(s);
    denda + +;
    if denda >= 50 then
        bestGDSolution ← RandomMove(gdSolution);
        bestGDSolution ← RandomSwapTwo(gdSolution);
        bestScoreGD ← evaluate(bestGDSolution);
    end
    if theBestGDScore > bestScoreGD then
        theBestGD ← bestGDSolution;
    end
    counter + +;
end
```

Gambar 3. Pseudo Code Great Deluge Yang Digunakan

G. Tabu Search

Tabu Search adalah salah satu local search metaheuristik yang dapat digunakan untuk menyelesaikan masalah optimasi kombinatorial. Tabu search memiliki kecepatan yang lebih tinggi bila dibandingkan dengan algoritma genetika karena tabu search tidak akan kembali pada solusi yang sudah dieksplorasi. Solusi yang sudah ada sebelumnya tidak akan keluar dengan menggunakan memori yang disebut Tabu List.

Tabu List digunakan untuk menyimpan sekumpulan solusi yang baru dievaluasi. Selama proses optimasi, pada setiap iterasi, solusi yang akan dievaluasi akan dicocokkan terlebih dahulu dengan isi tabu list. Jika solusi tersebut ada pada tabu list, maka solusi tersebut tidak akan masuk dalam iterasi berikutnya. Apabila sudah tidak ada lagi solusi yang tidak menjadi anggota tabu list, maka nilai terbaik yang baru saja diperoleh merupakan solusi yang sebenarnya.

Pada pengerjaan algoritma Tabu Search ini kami mengacu pada paper [5]. Pseudo-code yang kami gunakan untuk mendapatkan solusi pada permasalahan timetabling ini dapat dilihat pada Gambar 4. Pada penelitian ini kami mengatur maksimal tabu list adalah 10 solusi.

Algorithm 4: Tabu Search

```
sBest ← initialSolution;
bestCandidate ← initialSolution;
tabuList ← [];
counter ← 0;
maxCounter ← 200000;
while counter < maxcounter do
    sNeighbor ← getNeighborByRandom(bestCandidate);
    for sCandidate in sNeighbor do
        if (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) then
            bestCandidate ← sCandidate;
        end
    end
    if fitness(bestCandidate) > fitness(sBest) then
        bestCandidate ← sCandidate;
    end
    tabuList.append(bestCandidate);
    if tabuList.size > maxTabuSize then
        tabuList.pop[0];
    end
    counter + +;
end
```

Gambar 4. Pseudo Code Tabu Search Yang Digunakan

III. HASIL DAN PEMBAHASAN

Selain melakukan pengujian pada dataset yang sama, penulis juga menggunakan batas iterasi yang sama yaitu 200.000 iterasi. Pengujian algoritma-algoritma tersebut dilakukan pada Bahasa pemrograman Python di Jupyter Notebook, system operasi Windows 10 Pro dengan spesifikasi perangkat keras sebagai berikut: Processor Intel (R) Core(TM) i7-4510U CPU @ 2.00 GHz 2.60 GHz dan RAM 8 GB. Hasil eksperimen dari tahapan-tahapan yang telah dilakukan dapat dilihat pada Tabel 2.

Tabel 2.
Perbandingan Hasil Dari Tahapan Yang Dilakukan

Dataset	IS	HC	SA1	SA2	GD	TS
Car91	9.48	7.06	6.21	**5.64**	5.84	5.72
Car92	7.76	6.27	5.17	**4.63**	4.78	4.88
Ear83	60.80	53.84	49.88	40.11	**38.81**	41.29
Hec92	18.72	15.19	14.63	**11.13**	11.19	12.52
Kfu93	31.27	21.03	16.63	**14.37**	15.13	16
Lse92	22.56	17.18	13.63	**11.71**	13.11	12.33
Pur93	10.33	7.05	6.76	5.99	5.78	**5.44**
Rye92	21.43	13.89	10.94	**9.41**	9.82	**9.41**
Sta83	174.15	171.71	158.35	157.13	**157.09**	157.59
Tre92	13.74	12.03	9.64	9.28	**9.26**	9.51
Uta92	6.19	5.09	4.15	3.87	3.84	**3.83**
Ute92	47.87	35.86	28.54	**25.66**	26.26	29.16
Yor83	51.95	45.66	41.99	41.64	40.35	**40.21**

Note: IS= Initial Solution, HC= Hill Climbing, SA1= Simulated Annealing 1, SA2= Simulated Annealing 2, GD= Great Deluge, TS= Tabu Search

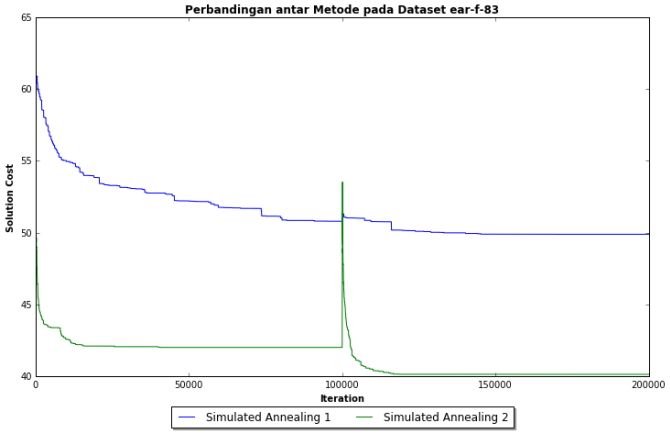
Simulated Annealing yang menggunakan Kempe Chain sebagai salah satu *low level heuristic* dapat menghasilkan solusi yang lebih baik dibanding tiga algoritma lainnya pada dataset Carleton91, Carleton92, EdHEC92 , KingFahd93, LSE91, Rye-f-92, dan TorontoE92. Sementara algoritma Simple Random Great Deluge dapat menghasilkan solusi yang lebih baik dibanding tiga algoritma lainnya untuk dataset EarlHaig83, StAndrews83, Trent92, sedangkan Tabu Search dapat menghasilkan solusi lebih baik dibanding tiga algoritma lainnya untuk dataset Rye-f-92, TorontoAS92, dan YorkMills83. Tetapi dari keempat algoritma tersebut Simulated Annealing dengan Kempe Chain sebagai salah satu low level heuristic dapat dikatakan yang terbaik diantara tiga algoritma lainnya, karena dapat menghasilkan solusi terbaik di tujuh dataset, sementara Simple Random Great Deluge hanya dapat menghasilkan solusi terbaik pada tiga dataset, begitu juga dengan Tabu Search. Perbandingan *Simulated Annealing* tanpa Kempe Chain sebagai *low level heuristic* dengan Simulated Annealing yang menggunakan Kempe Chain dapat dilihat pada Gambar 5 dan Gambar 6. Selain itu terlihat hasil dari Hill Climbing tidak lebih baik apabila dibandingkan dengan algoritma lainnya pada seluruh dataset. Perbandingan hasil dari algoritma-algoritma tersebut dapat dilihat pada Gambar 7 dan Gambar 8. Waktu untuk menyelesaikan seluruh algoritma dapat dikatakan memakan waktu. Hal ini diakibatkan karena kompleksitas dari dataset yang dijalankan. Durasi waktu yang dibutuhkan untuk menjalankan seluruh algoritma dapat dilihat pada Tabel 3 yang ditampilkan dalam satuan detik.

Tabel 3.
Perbandingan Durasi Dari Taip Tahapan Yang Dilakukan
Note: IS= Initial Solution, HC= Hill Climbing, SA1= Simulated Annealing 1, SA2= Simulated Annealing 2, GD= Great Deluge, TS= Tabu Search

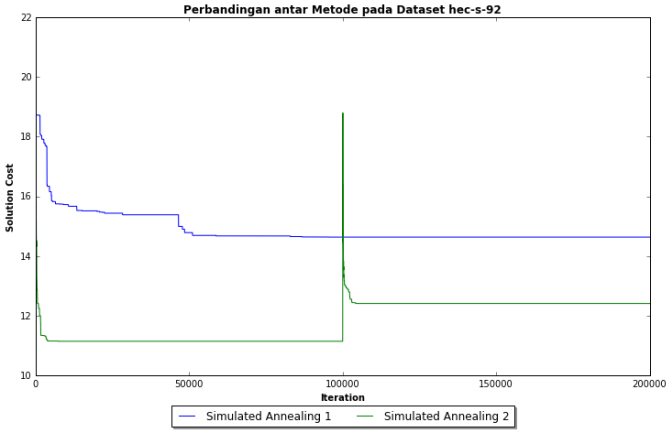
Dataset	IS	HC	SA1	SA2	GD	TS
Car91	5.28	323.3	3027.8	2540.2	2589	5069.1
Car92	5.02	454.4	3709.9	2125.4	2419.1	2505.6
Ear83	0.28	255.4	763.3	631.6	576.7	1320.6
Hec92	0.02	155.9	406.9	359.4	319.6	555.1
Kfu93	6.95	472.4	2869.2	1530.3	1572.9	4309
Lse92	0.56	187.7	997.9	627.5	618.1	1547.7
Pur93	364	2508.6	21034	9407.8	20250	46491
Rye92	2.5	391.8	2939.2	1930.7	2016	5272
Sta83	0.09	203.7	639.8	460.7	522.5	1085.9
Tre92	0.46	275.5	1409.7	849.1	924.1	2232.82
Uta92	9.19	527	3811.8	2585.8	2761.3	7792.4
Ute92	1.43	240.24	2789	282.6	269.46	679.8
Yor83	0.23	182.89	763.3	569.5	538.5	1364.5

Terlihat pada tabel tersebut bahwa dataset Pur93 merupakan dataset yang paling menghabiskan waktu untuk menjalankan semua tahapan dan dataset Hec92 membutuhkan waktu paling singkat untuk menjalankan semua tahapan tersebut.

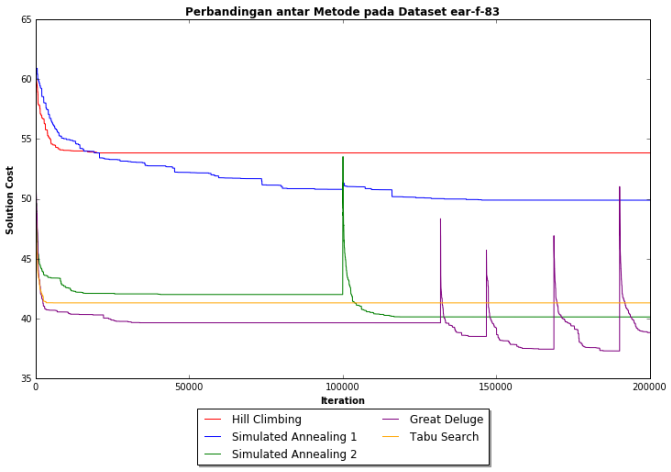
Berdasarkan hasil yang didapatkan, penulis menyimpulkan bahwa pemilihan low level heuristic sangat berpengaruh terhadap solusi yang didapatkan. Simulated Annealing dengan menggunakan Kempe Chain mampu mendominasi solusi yang dihasilkan. Hal ini diakibatkan karena Simulated Annealing mampu keluar dari local optimum.



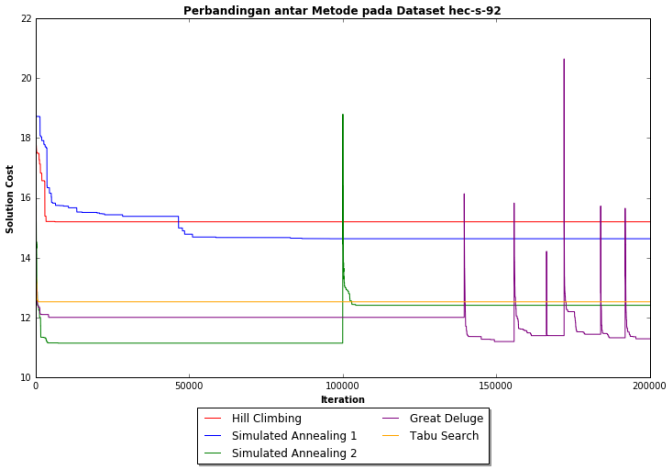
Gambar 5. Perbandingan Simulated Annealing 1 dengan Simulated Annealing 2 pada Dataset Ear83



Gambar 6. Perbandingan Simulated Annealing 1 dengan Simulated Annealing 2 pada Dataset Hec92

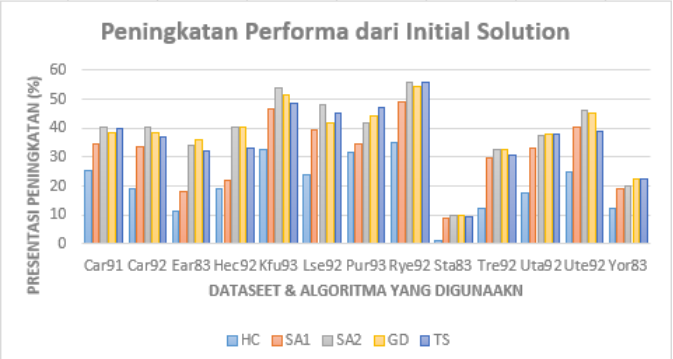


Gambar 7. Perbandingan Fungsi Objektif Menggunakan Beberapa Algoritma pada Dataset Ear83



Gambar 8. Perbandingan Fungsi Objektif Menggunakan Beberapa Algoritma pada Dataset Hec92

Algoritma yang digunakan mampu meningkatkan fungsi objektif secara signifikan dari inisial solusi. Peningkatan fungsi objektif tersebut dapat dilihat pada Gambar 9.



Gambar 9. Peningkatan Fungsi Objektif dari Inisial Solusi

IV. KESIMPULAN

Dari penelitian diatas dapat disimpulkan bahwa algoritma Simulated Annealing dengan menggunakan Kempe Chain sebagai low level heuristic dapat menghasilkan performa yang mendominasi dibandingkan yang lain karena dapat menghasilkan solusi terbaik di tujuh dataset, sementara Simple Random Great Deluge hanya dapat menghasilkan solusi terbaik pada tiga dataset, begitu juga dengan Tabu Search.Solusi terbaik untuk dataset Carleton91 adalah 5.64, untuk dataset Carleton92 adalah 4.63, untuk dataset EarlHaig83 adalah 38.36, untuk dataset EdHEC92 adalah 11.15, untuk dataset KingFahd93 adalah 14.37, untuk dataset LSE91 adalah 11.71, untuk dataset Rye-f-92 adalah 9.41, untuk dataset St.Andrews83 adalah 157.09, untuk dataset TorontoAS92 adalah 3.83, untuk dataset TorontoE92 adalah 25.66, untuk dataset Trent92 adalah 9.26, untuk dataset YorkMills83 adalah 40.21,dan untuk dataset Pur-s-93 adalah 5.44. Penelitian ini tidak menggunakan batasan waktu, tetapi menggunakan batasan iterasi sampai 200.000 iterasi.

REFERENSI

[1] V. A. Supoyo dan A. Muklason, "PENDEKATAN HYPER HEURISTIC DENGAN KOMBINASI ALGORITMA PADA EXAMINATION TIMETABLING PROBLEM," *ILKOM Jurnal Ilmiah*, vol. 11, no. 1, pp. 34-44, 2019.

[2] E. K. Burke dan J. P. Newall, "A Multistage Evolutionary Algorithm for the Timetable Problem," *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION*, vol. 3, no. 1, pp. 63-74, 1999.

[3] M. Cheraitia dan S. Haddadi, "Simulated annealing for the uncapacitated exam scheduling problem," *International Journal Metaheuristics*, vol. 5, no. 2, pp. 156-170, 2016.

[4] H. Turabieh dan S. Abdullah, "An integrated hybrid approach to the examination timetabling problem," *Omega*, vol. 39, pp. 598-607, 2011.

[5] L. D. Gaspero dan A. Schaerf, "Tabu search techniques for examination," *Practice and Theory of Automated Timetabling III*, pp. 104-117, 2001.